IBM

System i

System i Access for Windows: プログラミング

バージョン 6 リリース 1

IBM

System i

System i Access for Windows: プログラミング

バージョン 6 リリース 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、 635 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、System i Access for Windows (プロダクト番号 5761-XE1) バージョン 6、リリース 1、モディフィケーション 0 に適用されます。また、改訂版で断りがない限り、それ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。

IBM 発行のマニュアルに関する情報のページ

`http://www.ibm.com/jp/manuals/`

こちらから、日本語版および英語版のオンライン・ライブラリーをご利用いただけます。また、マニュアルに関するご意見やご感想を、上記ページよりお送りください。今後の参考にさせていただきます。

(URL は、変更になる場合があります)

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： System i
System i Access for Windows: Programming
Version 6 Release 1

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2008.2

目次

System i Access for Windows: プログラミング

アプリケーション開発者は、このトピックを調べて System i™ Access for Windows® の技術的なプログラミング情報、ツール、および手法を参照し、使用してください。

ここには、System i の資源にアクセスするアプリケーションを作成する際に役立つ、プログラミング概念、機能、例などといった情報が含まれています。このトピックを使用することにより、ユーザーのビジネス・ニーズに合わせたクライアント/サーバー・アプリケーションの開発や、調整を行います。このサーバーで提供される豊富な機能に接続し、それらを管理し、利用することができるように、さまざまなプログラミング手法が説明されています。以下の各トピックを選択することにより、それぞれの情報にアクセスすることができます。

System i Access for Windows のフィーチャーに関する基本的な作業知識が必要な場合には、System i Access for Windows 製品に同梱されている **ウェルカム・ウィザード**および**ユーザーズ・ガイド**を参照してください。

注: 各フィーチャーを Windows PC から起動するには、「**スタート**」 → 「**プログラム**」 → 「**System i Access for Windows**」と選択し、構成要素を選択してください。

注: コード例を使用することで、 633 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』の条件に同意します。

V6R1 の新機能

このページでは、System i Access for Windows V6R1 のプログラミング・トピックに加えられた変更内容を中心に説明します。

V6R1 からは、.NET Data Provider で以下の内容がサポートされるようになりました。

- ADO.NET 2.0 基底クラス・モデル
- System.Transactions 名前空間を使用する分散トランザクション
- 複数行のブロック化 INSERT
- 64 ビット・モデルでの実行
- 10 進浮動小数点データ・タイプとデータ・リンク・データ・タイプ
- ブロック化した LOB データの送受信に対するコントロールの改善
- SQL 照会ストレージの制限
- 新規 SQL 特殊クライアント情報レジスター

V6R1 では、以下の内容が OLE DB Data Provider でサポートされます。

- SQL 照会での最大ストレージの設定
- 10 進浮動小数点データの使用
- 数値データで生じたエラーの処理
- 入力データからの末尾ブランクの切り捨て
- クライアント情報 (クライアント・アカウンティング、プログラム ID、ユーザー ID、アプリケーション名、ワークステーション名など) の設定、およびデータベース・ホストとの相互の受け渡し。

V6R1 でサポートされる ODBC の機能は、以下のとおりです。

- SQL 照会ストレージの制限
- ODBC アプリケーションと QZDASOINIT システム・ジョブの関連付け
- 128 バイトのカーソル名
- 10 進浮動小数点 (DECFLOAT) データ・タイプ
- ストアード・プロシージャーの日時形式の追加

新機能や変更点の確認方法

技術的な変更が行われた個所を探すには、次の情報を利用してください。

- » というイメージは、新規または変更済み情報の開始個所を示しています。
- « というイメージは、新規または変更済み情報の終了個所を示しています。

今回のリリースで追加または変更された内容に関するその他の情報については、『プログラム資料説明書』を参照してください。

トピックの印刷

System i Access for Windows 情報の PDF を表示および印刷することができます。

この文書の PDF 版を表示またはダウンロードするには、「System i Access for Windows プログラミング」を選択します。

PDF ファイルの保存

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

1. ご使用のブラウザーで PDF のリンクを右クリックする。
2. ローカルに PDF を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必要があります。 Adobe Reader は、Adobe の Web サイト (www.adobe.com/products/acrobat/readstep.html) から無償でダウンロードすることができます。

System i Access for Windows の C/C++ API

System i Access for Windows の C/C++ アプリケーション・プログラミング・インターフェース (API) を使用して、System i 資源にアクセスします。

この API は、主として C/C++ プログラマーを対象としたものです。これは、C 形式の API をサポートするその他の言語からも呼び出されます。

注: コード例を使用することで、 633 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』の条件に同意します。

System i Access for Windows の C/C++ API の概要

System i Access for Windows の C/C++ API の概要情報については、以下のトピックを参照してください。

API グループ、ヘッダー・ファイル、インポート・ライブラリー、および DLL

System i Access for Windows のすべての C/C++ API グループについては、System i Access for Windows **Programmer's Toolkit** のインターフェース定義ファイルにアクセスして、参照してください。

System i Access for Windows の各 C/C++ API グループについて、以下の情報が後出の表に示されています。

- API ドキュメンテーション資料へのリンク
- 必要なインターフェース定義 (ヘッダー) ファイル (該当する場合)
- 関連するインポート・ライブラリー・ファイル (該当する場合)
- 関連するダイナミック・リンク・ライブラリー (DLL) ファイル

Toolkit の System i Access for Windows ヘッダー・ファイルにアクセスする方法

1. System i Access for Windows プログラム・ディレクトリーで **Programmer's Toolkit** アイコンを見つけて起動します。 Programmer's Toolkit がプログラム・ディレクトリーにない場合は、Toolkit をインストールします。
2. 左側のナビゲーション・パネルで、該当する API グループを選択します。

   注: Programmer's Toolkit の API カテゴリー名のうち、System i Access for Windows プログラミングで使用した名前と異なるものがあります。

検索する System i Access for Windows プログラミング API グループ・ヘッダー・ファイル	**選択する Programmer's Toolkit トピック**
管理	クライアント情報
データ形式変更	データ操作
各国語サポート	
LDAP	ディレクトリー
保守容易性	エラー処理
AS/400® オブジェクト	AS/400 オペレーション
システム・オブジェクト・アクセス	

3. 左側のナビゲーション・パネルで、「**C/C++ API**」サブトピックを選択します。
4. 右側の表示パネルで、ヘッダー・ファイル (.h) を見付けて、それを選択します。

注: Toolkit の System i Access for Windows API グループ・トピックには、インターフェースの記述および定義に加えて、他の情報源に対するリンクが含まれています。

インポート・ライブラリーについて

Programmer's Toolkit に同梱されているインポート・ライブラリーは、 Microsoft® Visual C++ コンパイラーを使用して作成されています。そのため、共通オブジェクト・ファイル形式 (COFF) になっています。Borland の C コンパイラーなど、一部のコンパイラーは COFF をサポートしていません。そのようなコン

パイラーから System i Access for Windows の C/C++ API にアクセスするには、IMPLIB ツールを使用して、オブジェクト・モデル形式 (OMF) のインポート・ライブラリーを作成する必要があります。例えば、以下のとおりです。

```
implib cwbdq.lib %windir%¥system32¥cwbdq.dll
```

注: V5R1 で、ファイル・サイズを小さくするために cwbapi.lib インポート・ライブラリーの形式が変更されました。このライブラリーは、Microsoft Visual C++ 5.0 以前では作動しません。 Microsoft Visual C++ 5.0 以前から API を呼び出す必要がある場合は、古い形式を使用して作成されたインポート・ライブラリーを、「Import Libraries」 (www.ibm.com/eserver/iseries/access/toolkit/importlibraries.htm) から入手することができます。

表 1. System i Access for Windows の C/C++ API グループ、ヘッダー・ファイル、ライブラリー・ファイル、および DLL ファイル

API グループ	ヘッダー・ファイル	インポート・ライブラリー	DLL
管理	cwbad.h	cwbapi.lib	cwbad.dll
通信およびセキュリティー	cwbcosys.h cwbco.h cwb.h	cwbapi.lib	cwbco.dll
データ待ち行列	cwbdq.h	cwbapi.lib	cwbdq.dll
データ形式変更	cwbdt.h	cwbapi.lib	cwbdt.dll
ディレクトリー更新	cwbup.h	cwbapi.lib	cwbup.dll
エミュレーション (標準 HLLAPI インターフェース)	hapi_c.h	pscal32.lib	pcshll.dll pcshll32.dll
エミュレーション (拡張 HLLAPI インターフェース)	ehlapi32.h	ehlapi32.lib	ehlapi32.dll
エミュレーション (Windows EHLLAPI インターフェース)	whllapi.h	whllapi.lib	whllapi.dll
		whlapi32.lib	whllapi32.dll
エミュレーション (HACL インターフェース)	eclall.hpp	pcseclva.lib	pcseclva.dll
		pcseclvc.lib	pcseclvc.dll
エミュレーション (PCSAPI インターフェース)	pcsapi.h	pcscal32.lib	pcsapi.dll pcsapi32.dll
各国語サポート (汎用 NLS)	cwbnl.h	cwbapi.lib	cwbnl.dll
各国語サポート (変換 NLS)	cwbnlcnv.h	cwbapi.lib	cwbnl1.dll
各国語サポート (ダイアログ・ボックス NLS)	cwbnldlg.h	cwbapi.lib	cwbnldlg.dll
System i オブジェクト	cwbobj.h	cwbapi.lib	cwbobj.dll

表 1. System i Access for Windows の C/C++ API グループ、ヘッダー・ファイル、ライブラリー・ファイル、および DLL ファイル (続き)

API グループ	ヘッダー・ファイル	インポート・ライブラリー	DLL
ODBC	sql.h sqlext.h sqltypes.h sqlucode.h	odbc32.lib	odbc32.dll
データベース API (最適化 SQL) **注:** これらの API は、サポートされなくなりました。	cwbdb.h	cwbapi.lib	cwbdb.dll
OLE DB Provider	ad400.h da400.h		cwbzzodb.dll 詳しくは、Microsoft Universal Data Access Web サイトの『OLE DB』セクション (英語) を参照してください。
リモート・コマンド / 分散プログラム呼び出し	cwbrc.h	cwbapi.lib	cwbrc.dll
保守容易性	cwbsv.h	cwbapi.lib	cwbsv.dll
システム・オブジェクト・アクセス	cwbsoapi.h	cwbapi.lib	cwbsoapi.dll

関連資料

8 ページの『OEM、ANSI、およびユニコードの考慮事項』
System i Access for Windows の、ストリング・パラメーターを受け入れる C/C++ API の大部分は、OEM、ANSI、Unicode の 3 つのうち、いずれかの形式になっています。

10 ページの『単一の System i Access for Windows API タイプの使用』
特定のタイプの System i Access for Windows API のみをアプリケーションで使用するように制限するには、単一のプリプロセッサー定義のみを定義する必要があります。

Programmer's Toolkit

System i Access for Windows アプリケーションを開発するための、ヘッダー・ファイルおよびあらゆる情報を検索します。

System i Access for Windows Programmer's Toolkit は、System i Access for Windows 製品のインストール可能な構成要素であり、System i Access for Windows アプリケーションの開発に不可欠な、主要な情報ソースとなります。 System i Access for Windows の ActiveX オートメーション・オブジェクト、ADO/OLE DB、.NET、および Java™ によるプログラミングが含まれています。 Programmer's Toolkit には、ヘッダー・ファイル、サンプル・プログラム、およびすべてのドキュメンテーション資料へのリンクが含まれています。

注:

- Toolkit や System i Access forWindows 製品のいずれの部分も、作成したアプリケーションと一緒に再配布することはできません。
- コード例を使用することで、 633 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』

の条件に同意します。

Programmer's Toolkit は、次の 2 つの部分から構成されています。

- System i Access for Windows Programmer's Toolkit の構成要素には、以下のものが含まれます。
  - Toolkit のオンライン・ヘルプ情報、および、製品のその他のオンライン・ヘルプ
  - C/C++ ヘッダー・ファイル
  - C インポート・ライブラリー
  - ActiveX オートメーション・タイプ・ライブラリー
- Programmer's Toolkit の Web サイト。System i Access for Windows アプリケーションの開発に役立つ、サンプル・アプリケーションやツールを入手できます。このサイトは定期的に更新されます。定期的に新しい情報を確認してください。

関連情報


System i Access for Windows データベース API (英語)

Programmer's Toolkit のインストール:

Programmer's Toolkit は、System i Access for Windows 製品のフィーチャーの 1 つとしてインストールされます。

本製品の Programmer's Toolkit およびその他のフィーチャーを追加または除去するには、ご使用の PC のコントロール・パネルにある「プログラムの追加と削除」を使用します。

1. 「スタート」>「コントロール パネル」>「プログラムの追加と削除」>「IBM® System i Access for Windows」>「変更」とクリックします。
2. 画面の指示に従って、「変更」ボタンを選択します。
3. フィーチャー名 (Programmer's Toolkit) をクリックし、次の中から該当するものを選択します。
   - **このフィーチャーをローカルのハード・ディスクにインストールします。(This feature will be installed on local hard drive.)**(フィーチャーを単一でインストールする場合)
   - **このフィーチャーとすべてのサブフィーチャーをローカルのハード・ディスクにインストールします。(This feature, and all subfeatures, will be installed on local hard drive.)**(フィーチャーを複数インストールする場合)
   - **このフィーチャーを使用しません。(This feature will not be available.)**(フィーチャーを除去する場合)
4. 「インストール」をクリックしてインストール済みのフィーチャーを変更し、インストール・ウィザードが完了するまで作業を続行します。

関連資料

631 ページの『ActiveX プログラミング』
ActiveX オートメーションは、Microsoft によって定義されたプログラミング・テクノロジーであり、System i Access for Windows 製品でサポートされています。

Programmer's Toolkit の立ち上げ:

Programmer's Toolkit は、System i Access for Windows 製品のフィーチャーの 1 つとして立ち上げられます。

1. ご使用のパーソナル・コンピューターに、Programmer's Toolkit フィーチャーをインストールします。

2. 「**スタート**」 → 「**すべてのプログラム**」 → 「**IBM System i Access for Windows**」 → 「**Programmer's Toolkit**」と選択します。

注: パーソナル・コンピューターに Programmer's Toolkit がインストールされると、Toolkit のアイコンが表示されます。

関連資料

631 ページの『ActiveX プログラミング』
ActiveX オートメーションは、Microsoft によって定義されたプログラミング・テクノロジーであり、System i Access for Windows 製品でサポートされています。

接続 API 用の System i 名の形式

パラメーターとして System i 名を取得する API では、3 つの異なる形式の名前を使用できます。

有効な形式は次のとおりです。

- TCP/IP ネットワーク名 (system.network.com)
- ネットワーク ID なしのシステム名 (SYSTEM)
- IP アドレス (1.2.3.4)

関連資料

34 ページの『System i Access for Windows 管理 API』
これらの API には、PC にインストールされている System i Access for Windows のコードに関する情報にアクセスするための機能が備わっています。

46 ページの『System i Access for Windows の通信およびセキュリティー API』
『System i Access forWindows 通信およびセキュリティー』トピックでは、System i Access for Windows アプリケーション・プログラミング・インターフェース (API) の使用方法について説明します。

133 ページの『System i データ待ち行列 API』
System i Access for Windows のデータ待ち行列アプリケーション・プログラミング・インターフェース (API) を使用すると、System i のデータ待ち行列に簡単にアクセスできます。データ待ち行列を使用することによって、通信 API を使う必要のないクライアント/サーバー・アプリケーションを作成することができるようになります。

185 ページの『System i Access for Windows データ形式変更 API』
System i Access for Windows の**データ形式変換**アプリケーション・プログラミング・インターフェース (API) を使用すると、クライアント/サーバー・アプリケーションで System i 数値データの形式変更 (システム形式と PC 形式との相互変更) を行えるようになります。 System i 数値データの送受信をシステムとの間で行う場合に、形式変更が必要になることがあります。データ形式変更 API は、数多くの数値形式の変換をサポートします。

204 ページの『System i Access for Windows 各国語サポート (NLS) API』
各国語サポート API によって、各国の言語バージョンに関連した System i Access for Windows の設定値の取得および保存 (照会および変更) を、アプリケーションで行えるようになります。

235 ページの『System i Access for Windows ディレクトリー更新 API』
System i Access for Windows ディレクトリー更新機能を使用する PC ディレクトリー更新を指定します。

258 ページの『System i Access for Windows 用システム・オブジェクト API』
System i Access for Windows 用システム・オブジェクトのアプリケーション・プログラミング・イン

ターフェース (API) を使用して、システム上にある印刷関連のオブジェクトを処理できます。これらの API は、System i スプール・ファイル、書き出しプログラム・ジョブ、出力待ち行列、プリンターなどを使った作業を可能にします。

369 ページの『System i Access for Windows のリモート・コマンド/分散プログラム呼び出し API』
PC アプリケーション・プログラマーは、System i Access for Windows リモート・コマンド/分散プログラム呼び出し API を使用することで、System i の機能にアクセスできます。ユーザー・プログラムとシステム・コマンドを、エミュレーション・セッションなしに呼び出します。コマンドとプログラムは単一の System i プログラムによって扱われるため、コマンドとプログラムの両方に対して、1 つのシステム・ジョブのみが開始されます。

451 ページの『System i Access for Windows システム・オブジェクト・アクセス (SOA) API』
システム・オブジェクト・アクセスを使用することで、グラフィカル・ユーザー・インターフェースを介して、システム・オブジェクトの表示および操作を行うことができます。

OEM、ANSI、およびユニコードの考慮事項

System i Access for Windows の、ストリング・パラメーターを受け入れる C/C++ API の大部分は、OEM、ANSI、Unicode の 3 つのうち、いずれかの形式になっています。

System i Access for Windows の C/C++ API の汎用版は、デフォルトの OEM 版と同じ形式を取ります。この情報では、それぞれの関数ごとに 1 つの名前のみが示されていますが、異なる 3 つのシステム・エントリー・ポイントがあります。例えば、以下のとおりです。

```
cwbNL_GetLang();

 compiles to:

cwbNL_GetLang();    //CWB_OEM or undefined

  または:

cwbNL_GetLangA();   //CWB_ANSI defined

  または:

cwbNL_GetLangW();   //CWB_UNICODE defined
```

表 2. API タイプ、名前の形式、およびプリプロセッサー定義

API タイプ	**API 名の形式 (使用されている場合)**	**プリプロセッサー定義**
OEM	cwbXX_xxx	なし (明示的に、CWB_OEM を指定可能)
ANSI	cwbXX_xxxA	CWB_ANSI
UNICODE	cwbXX_xxxW	CWB_UNICODE

注:

- データ転送 API **(cwbDT_xxx)** は、「A」および「W」のサフィックスの規則には従いません。 API の汎用バージョンは「String」を関数名の一部として使用しています。 ANSI/OEM バージョンは「ASCII」を関数名の一部として使用しています。ユニコード・バージョンは「Wide」を関数名の一部として使用しています。数値ストリングを取り扱う **cwbDT_xxx** API の OEM および ANSI 文字セットの間に違いはありません。したがって、関連する API の ANSI および OEM バージョンはいずれも同じです。例えば、以下のとおりです。

  ```
  cwbDT_HexToString();

   compiles to:
  ```

```
cwbDT_HexToASCII();   //CWB_UNICODE not defined
```

または:

```
cwbDT_HexToWide();    //CWB_UNICODE defined
```

詳細については、データ形式変更ヘッダー・ファイル **cwbdt.h** の関連リンクを選択して、参照してください。

- ストリングを渡すためのバッファーと長さ (例えば、**cwbCO_GetUserIDExW**) を持つユニコード API の場合、その長さはバイト数として扱われます。文字数としては扱われません。

関連資料

34 ページの『System i Access for Windows 管理 API』
これらの API には、PC にインストールされている System i Access for Windows のコードに関する情報にアクセスするための機能が備わっています。

46 ページの『System i Access for Windows の通信およびセキュリティー API』
『System i Access forWindows 通信およびセキュリティー』トピックでは、System i Access for Windows アプリケーション・プログラミング・インターフェース (API) の使用方法について説明します。

133 ページの『System i データ待ち行列 API』
System i Access for Windows のデータ待ち行列アプリケーション・プログラミング・インターフェース (API) を使用すると、System i のデータ待ち行列に簡単にアクセスできます。データ待ち行列を使用することによって、通信 API を使う必要のないクライアント/サーバー・アプリケーションを作成することができるようになります。

185 ページの『System i Access for Windows データ形式変更 API』
System i Access for Windows の**データ形式変換**アプリケーション・プログラミング・インターフェース (API) を使用すると、クライアント/サーバー・アプリケーションで System i 数値データの形式変更 (システム形式と PC 形式との相互変更) を行えるようになります。 System i 数値データの送受信をシステムとの間で行う場合に、形式変更が必要になることがあります。データ形式変更 API は、数多くの数値形式の変換をサポートします。

204 ページの『System i Access for Windows 各国語サポート (NLS) API』
各国語サポート API によって、各国の言語バージョンに関連した System i Access for Windows の設定値の取得および保存 (照会および変更) を、アプリケーションで行えるようになります。

235 ページの『System i Access for Windows ディレクトリー更新 API』
System i Access for Windows ディレクトリー更新機能を使用する PC ディレクトリー更新を指定します。

258 ページの『System i Access for Windows 用システム・オブジェクト API』
System i Access for Windows 用システム・オブジェクトのアプリケーション・プログラミング・インターフェース (API) を使用して、システム上にある印刷関連のオブジェクトを処理できます。これらの API は、System i スプール・ファイル、書き出しプログラム・ジョブ、出力待ち行列、プリンターなどを使った作業を可能にします。

369 ページの『System i Access for Windows のリモート・コマンド/分散プログラム呼び出し API』
PC アプリケーション・プログラマーは、System i Access for Windows リモート・コマンド/分散プログラム呼び出し API を使用することで、System i の機能にアクセスできます。ユーザー・プログラムとシステム・コマンドを、エミュレーション・セッションなしに呼び出します。コマンドとプログラムは単一の System i プログラムによって扱われるため、コマンドとプログラムの両方に対して、1 つのシステム・ジョブのみが開始されます。

3 ページの『API グループ、ヘッダー・ファイル、インポート・ライブラリー、および DLL』
System i Access for Windows のすべての C/C++ API グループについては、System i Access for Windows **Programmer's Toolkit** のインターフェース定義ファイルにアクセスして、参照してください。

単一の System i Access for Windows API タイプの使用:

特定のタイプの System i Access for Windows API のみをアプリケーションで使用するように制限するには、単一のプリプロセッサー定義のみを定義する必要があります。

プリプロセッサー定義には以下のものがあります。

- CWB_OEM_ONLY
- CWB_ANSI_ONLY
- CWB_UNICODE_ONLY

例えば、純粋な ANSI アプリケーションを作成する場合は、CWB_ANSI_ONLY および CWB_ANSI の両方を指定します。これらのプリプロセッサー定義および API 名の詳細については、個々の Programmer's Toolkit のヘッダー・ファイルを参照してください。詳しくは、『API グループ、ヘッダー・ファイル、インポート・ライブラリー、および DLL』のトピック集の関連リンク (以下参照) を参照してください。

関連資料

3 ページの『API グループ、ヘッダー・ファイル、インポート・ライブラリー、および DLL』
System i Access for Windows のすべての C/C++ API グループについては、System i Access for Windows **Programmer's Toolkit** のインターフェース定義ファイルにアクセスして、参照してください。

System i Access for Windows API タイプの混合使用:

System i Access for Windows の API 名を明示的に使用することによって、ANSI、OEM、およびユニコードの API を混合して使用することができます。

例えば、CWB_ANSI プリプロセッサー定義を指定することによって、System i Access for Windows の ANSI アプリケーションを作成することができます。この場合にも、接尾部「W」を使用することで、ユニコード・バージョンの API を呼び出すことが可能です。

System i Access for Windows 汎用アプリケーションの作成:

System i Access for Windows 汎用アプリケーションでは、同じソース・コードを OEM、ANSI、およびユニコード用にコンパイルすることができるため、移植性が大幅に向上します。

汎用アプリケーションを作成するには、異なるプリプロセッサー定義を指定し、System i Access for Windows API の汎用版 (サフィックス「A」または「W」がないもの) を使用します。汎用アプリケーションを作成する場合のガイドラインの要約を以下に示します。

- ストリングの操作用に通常の <string.h> を組み込む代わりに、 <TCHAR.H> を組み込みます。
- 文字およびストリングに汎用データ・タイプを使用します。ソース・コードで、'char' の代わりに 'TCHAR' を使用します。
- リテラル文字およびストリング用に _TEXT マクロを使用します。例えば、`TCHAR A[]=_TEXT("A Generic String")` とします。
- 汎用ストリング処理関数を使用します。例えば、`_tcscpy` の代わりに `strcpy` を使用します。

- 'sizeof' 演算子を使用する場合は特に注意してください。ユニコード文字が常に 2 バイトを占有することを忘れないでください。汎用 TCHAR 配列 A の文字数を決定する場合、単純な `sizeof(A)` の代わりに、 `sizeof(A)/sizeof(TCHAR)` を使用してください。
- コンパイルには、適切なプリプロセッサー定義を使用してください。ユニコード用のソースを Visual C++ でコンパイルする場合は、プリプロセッサー定義 `UNICODE` および `_UNICODE` を使用してください。MAK ファイルに `_UNICODE` を定義する代わりに、ソース・コードの先頭で、次のように定義することもできます。

```
#ifdef UNICODE
  #define _UNICODE
#endif
```

これらのガイドラインの詳細については、以下の資料を参照してください。

1. Richter, J. *Advanced Windows: The Developer's Guide to the Win32 API for Windows NT® 3.5 and Windows 95*, Microsoft Press, Redmond, WA, 1995.
2. Kano, Nadine *Developing International Software for Windows 95 and Windows NT: a handbook for software design*, Microsoft Press, Redmond, WA, 1995.
3. Microsoft 知識ベースの記事 (関連リンクを参照。)
4. MSDN ライブラリー (関連リンクを参照。)

関連情報

知識ベース

MSDN ライブラリー

廃止された System i Access for Windows API

Client Access で提供されていた API の一部は、新しい API に置き換えられました。廃止された古い API も引き続きサポートされますが、新規の System i Access for Windows API を使用することをお勧めします。

廃止された Client Access API と System i Access for Windows API を、以下のリストで関数別に示します。廃止された各 Client Access API に代わる新しい System i Access for Windows API がある場合は、そのリンクも掲載します。

注: 以下の関数の API はすべて廃止され、 System i Access for Windows でのサポートは行われません。

- **APPC**
- **ライセンス管理**
- **アルチメディア・システム・ファシリティー (USF)**
- **Messaging Application Programming Interface (MAPI)**

廃止された System i Access for Windows API のリスト

廃止された通信 API:

System i Access for Windows では、廃止された通信 API があります。

cwbCO_IsSystemConfigured

System i Access for Windows 関数では、System i 接続を使用するための事前構成は必要ありません。このため、System i 接続が (cwbCO_Connect を呼び出して明示的に、もしくは

cwbRC_RunCmd など別の API への呼び出しの結果として暗黙的に) 必要なプログラムでも、接続が事前構成されているかどうかを確認するためのチェックは必要ありません。したがって、上記の API はもはや必要ではありません。

cwbCO_IsSystemConnected
59 ページの『cwbCO_IsConnected』を使用してください。

System i Access for Windows API のほとんどが、System i 名ではなく、システム・オブジェクトを処理の対象にします。同一のプロセスで、システム・オブジェクトを複数作成し、同一のシステムに接続させることが可能です。 cwbCO_IsSystemConnected API は、現行のプロセス内でシステムに接続されているシステム・オブジェクトが、少なくとも 1 つ存在するかどうかを示す指標を戻します。 cwbCO_IsConnected API を使用すると、特定のシステム上のシステム・オブジェクトが接続されているかどうかを判別できます。

cwbCO_GetUserID
71 ページの『cwbCO_GetUserIDEx』を使用してください。

System i Access for Windows API のほとんどが、System i 名ではなく、システム・オブジェクトを処理の対象にします。同一のプロセスで異なるユーザー ID を使用して、システム・オブジェクトを複数作成し、同じシステムに接続させることが可能です。 cwbCO_GetUserID API は、指定されたシステムについて、現行のプロセスにおける最初のシステム・オブジェクトのユーザー ID を戻します。 cwbCO_GetUserIDEx API は、特定のシステム上にあるシステム・オブジェクトのユーザー ID を戻します。

cwbCO_GetHostVersion
92 ページの『cwbCO_GetHostVersionEx』を使用してください。

これらの API の動作は同じです。ただし、cwbCO_GetHostVersionEx API を使用すると、より効率的です。

廃止されたデータ待ち行列 API:

System i Access for Windows では、廃止されたデータ待ち行列 API があります。

cwbDQ_Create
135 ページの『cwbDQ_CreateEx』を使用してください。

cwbDQ_Delete
138 ページの『cwbDQ_DeleteEx』を使用してください。

cwbDQ_Open
139 ページの『cwbDQ_OpenEx』を使用してください。

cwbDQ_StartSystem
54 ページの『cwbCO_Connect』を使用してください。

注: cwbCO_Connect を使用する場合に、cwbDQ_StartSystem と同じような効果を得るには、データ待ち行列のサービスに接続する必要があります。詳細については、 54 ページの『cwbCO_Connect』を参照してください。

cwbDQ_StopSystem
56 ページの『cwbCO_Disconnect』を使用してください。

注: cwbCO_Disconnect を使用する場合に、cwbDQ_StopSystem と同じような効果を得るには、データ待ち行列サービスから切断する必要があります。詳細については、 56 ページの『cwbCO_Disconnect』を参照してください。

廃止されたリモート・コマンド/分散プログラム呼び出し API:

System i Access for Windows では、廃止されたリモート・コマンド/分散プログラム呼び出し API があります。

cwbRC_StartSys
374 ページの『cwbRC_StartSysEx』を使用してください。

cwbRC_GetSysName
95 ページの『cwbCO_GetSystemName』を使用してください。

廃止されたセキュリティー API:

System i Access for Windows では、廃止されたセキュリティー API があります。

cwbSY_CreateSecurityObj
51 ページの『cwbCO_CreateSystem』を使用してください。

cwbSY_DeleteSecurityObj
53 ページの『cwbCO_DeleteSystem』を使用してください。

cwbSY_SetSys
51 ページの『cwbCO_CreateSystem』 を使用し、システム名を呼び出しで渡す

cwbSY_VerifyUserIDPwd
84 ページの『cwbCO_VerifyUserIDPassword』を使用してください。

cwbSY_ChangePwd
63 ページの『cwbCO_ChangePassword』を使用してください。

cwbSY_GetUserID
71 ページの『cwbCO_GetUserIDEx』を使用してください。

cwbSY_Logon
82 ページの『cwbCO_Signon』を使用してください。

cwbSY_LogonUser
79 ページの『cwbCO_SetUserIDEx』、 76 ページの『cwbCO_SetPassword』、または 82 ページの『cwbCO_Signon』を使用してください。

cwbSY_GetDateTimeCurrentSignon
70 ページの『cwbCO_GetSignonDate』を使用してください。

cwbSY_GetDateTimeLastSignon
68 ページの『cwbCO_GetPrevSignonDate』を使用してください。

cwbSY_GetDateTimePwdExpires
67 ページの『cwbCO_GetPasswordExpireDate』を使用してください。

cwbSY_GetFailedAttempts
66 ページの『cwbCO_GetFailedSignons』を使用してください。

廃止された保守容易性 API:

System i Access for Windows では、廃止された保守容易性 API があります。

以下に示す、問題ログ・サービス・レコードを読むための保守容易性 API が廃止されました。

cwbSV_GetCreatedBy
選択不可

cwbSV_GetCurrentFix
選択不可

cwbSV_GetFailMethod
選択不可

cwbSV_GetFailModule
選択不可

cwbSV_GetFailPathName
選択不可

cwbSV_GetFailProductID
選択不可

cwbSV_GetFailVersion
選択不可

cwbSV_GetOriginSystemID
選択不可

cwbSV_GetOriginSystemIPAddr
選択不可

cwbSV_GetPreviousFix
選択不可

cwbSV_GetProblemID
選択不可

cwbSV_GetProblemStatus
選択不可

cwbSV_GetProblemText
選択不可

cwbSV_GetProblemType
選択不可

cwbSV_GetSeverity
選択不可

cwbSV_GetSymptomString
選択不可

廃止されたシステム・オブジェクト・アクセス (SOA) API:

System i Access for Windows では、廃止された SOA API があります。

CWBSO_CreateListHandle
467 ページの『CWBSO_CreateListHandleEx』を使用してください。

廃止された各国語サポート (NLS) API:

System i Access for Windows では、いくつかの NLS API が廃止されています。

cwbNL_CreateConverter
221 ページの『cwbNL_CreateConverterEx』を使用してください。

cwbNL_ConvertCodePages
216 ページの『cwbNL_ConvertCodePagesEx』を使用してください。

廃止されたデータベース API:

System i Access for Windows では、いくつかのデータベース API が廃止されています。

注: コード例を使用することで、 633 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』の条件に同意します。

戻りコードおよびエラー・メッセージ

System i Access for Windows の C/C++ アプリケーション・プログラミング・インターフェース (API) では、多くの関数で整数の戻りコードの戻りがサポートされています。戻りコードは、関数がどのように完了したのかを示しています。

System i Access for Windows のエラー・メッセージは、ヒストリー・ログに記録されると共に、システムにも記録されます。

ヒストリー・ログのエラー・メッセージ

ヒストリー・ログの開始

デフォルトでは、ヒストリー・ログは活動状態になっていません。エラー・メッセージが必ずこのファイルに書き込まれるようにするには、ヒストリー・ログを開始する必要があります。ヒストリー・ログの開始については、製品に同梱されている System i Access for Windows ユーザーズ・ガイドを参照してください。

記録されたメッセージの表示

ヒストリー・ログに記録されているメッセージを表示するには、「**スタート**」 → 「**プログラム**」 → 「**System i Access for Windows**」 → 「**サービス**」 → 「**ヒストリー・ログ**」と選択します。

ヒストリー・ログの中の項目は、メッセージ ID 付きとメッセージ ID なしのメッセージで構成されています。メッセージ ID 付きのメッセージでは、オンライン・ヘルプを使用することができます。メッセージ ID なしのメッセージでは、オンライン・ヘルプを使用することはできません。メッセージ ID 付きのメッセージをダブルクリックすると、そのメッセージに関連した原因と回復情報が表示されます。また、System i Access for Windows のオンライン・ユーザーズ・ガイドでメッセージのトピックを選択すると、メッセージ ID 付きのメッセージをすべて表示させることができます。

System i のエラー・メッセージ

System i Access for Windows メッセージの一部は、システムにも記録されます。これらのメッセージは、PWS または IWS で始まります。特定の PWSxxxx または IWSxxxx メッセージを表示させるには、該当するコマンドを、コマンド行プロンプトで入力します (xxxx はメッセージ番号)。

```
DSPMSGD RANGE(IWSxxxx) MSGF(QIWS/QIWSMSG)

DSPMSGD RANGE(PWSxxxx) MSGF(QIWS/QIWSMSG)
```

System i Access for Windows のオペレーティング・システム・エラーに対応する戻りコード:

System i Access for Windows の戻りコードとシステム・エラー・メッセージとの関係を示します。

```
0       CWB_OK
          正常終了。
1       CWB_INVALID_FUNCTION
          サポートされない関数。
2       CWB_FILE_NOT_FOUND
```

ファイルが見付からない。
3 CWB_PATH_NOT_FOUND
パスが見付からない。
4 CWB_TOO_MANY_OPEN_FILES
システムはファイルをオープンできない。
5 CWB_ACCESS_DENIED
アクセスが拒否された。
6 CWB_INVALID_HANDLE
リスト・ハンドルが無効。
8 CWB_NOT_ENOUGH_MEMORY
メモリーが不足、一時バッファーの割り振りが失敗した可能性がある。
15 CWB_INVALID_DRIVE
システムは指定のドライブを見付けることができない。
18 CWB_NO_MORE_FILES
これ以上ファイルは見付からない。
21 CWB_DRIVE_NOT_READY
装置が作動可能でない。
31 CWB_GENERAL_FAILURE
一般エラー発生。
32 CWB_SHARING_VIOLATION
他のプロセスが使用しているためこのプロセスはファイルにアクセスできない。
33 CWB_LOCK_VIOLATION
他のプロセスがファイルの一部をロックしたためこのプロセスはファイルにアクセスできない。
38 CWB_END_OF_FILE
ファイルの終わりに達しました。
50 CWB_NOT_SUPPORTED
ネットワーク要求はサポートされない。
53 CWB_BAD_NETWORK_PATH
ネットワーク・パスが見付からない。
54 CWB_NETWORK_BUSY
ネットワークが使用中。
55 CWB_DEVICE_NOT_EXIST
指定のネットワーク資源または装置がもう使用可能でない。
59 CWB_UNEXPECTED_NETWORK_ERROR
予期しないネットワーク・エラーが発生。
65 CWB_NETWORK_ACCESS_DENIED
ネットワーク・アクセスが拒否された。
80 CWB_FILE_EXISTS
ファイルが存在する。
85 CWB_ALREADY_ASSIGNED
そのローカル装置名は既に使用されている。
87 CWB_INVALID_PARAMETER
パラメーターが無効。
88 CWB_NETWORK_WRITE_FAULT
書き込み障害がネットワークで発生。
110 CWB_OPEN_FAILED
システムは指定の装置またはファイルをオープンできない。
111 CWB_BUFFER_OVERFLOW
出力バッファーのスペースが不足。*bufferSize を使用して適正なサイズを決める必要がある。
112 CWB_DISK_FULL
ディスクに十分なスペースがない。
115 CWB_PROTECTION_VIOLATION
アクセスが拒否された。
124 CWB_INVALID_LEVEL
システム呼び出しのレベルが正しくない。
142 CWB_BUSY_DRIVE
この時点では、システムは JOIN または SUBST を実行できない。
252 CWB_INVALID_FSD_NAME
装置名が誤り。
253 CWB_INVALID_PATH
指定のネットワーク・パスが正しいものではない。

System i Access for Windows の戻りコード:

System i Access for Windows には、グローバルおよび固有の戻りコードがあります。

System i Access for Windows のグローバル戻りコード:

System i Access for Windows のグローバル戻りコードを紹介します。

4000 CWB_USER_CANCELLED_COMMAND
ユーザーがコマンドを取り消した。

4001 CWB_CONFIG_ERROR
構成エラーが発生。

4002 CWB_LICENSE_ERROR
ライセンス・エラーが発生。

4003 CWB_PROD_OR_COMP_NOT_SET
プロダクトまたは構成要素の登録と使用に関する障害による内部エラー。

4004 CWB_SECURITY_ERROR
セキュリティー・エラー発生。

4005 CWB_GLOBAL_CFG_FAILED
グローバル構成を試みたが失敗した。

4006 CWB_PROD_RETRIEVE_FAILED
プロダクトの検索が失敗した。

4007 CWB_COMP_RETRIEVE_FAILED
コンピューターの検索が失敗した。

4008 CWB_COMP_CFG_FAILED
コンピューターの構成が失敗した。

4009 CWB_COMP_FIX_LEVEL_UPDATE_FAILED
コンピューターの修正レベル更新が失敗した。

4010 CWB_INVALID_API_HANDLE
要求ハンドルが無効。

4011 CWB_INVALID_API_PARAMETER
指定のパラメーターが無効。

4012 CWB_HOST_NOT_FOUND
サーバーが非活動中であるかまたは存在しない。

4013 CWB_NOT_COMPATIBLE
System i Access のプログラムまたは関数が正しいレベルでない。

4014 CWB_INVALID_POINTER
ポインターが NULL。

4015 CWB_SERVER_PROGRAM_NOT_FOUND
サーバー・アプリケーションが見付からない。

4016 CWB_API_ERROR
一般 API 障害。

4017 CWB_CA_NOT_STARTED
System i Access プログラムが開始していない。

4018 CWB_FILE_IO_ERROR
レコードが読み取れない。

4019 CWB_COMMUNICATIONS_ERROR
通信エラー発生。

4020 CWB_RUNTIME_CONSTRUCTOR_FAILED
C ランタイム・コンストラクターが失敗した。

4021 CWB_DIAGNOSTIC
予期しないエラー。
メッセージ番号とメッセージ中のデータを記録して IBM サポートに
連絡してください。

4022 CWB_COMM_VERSION_ERROR
データ待ち行列は、このバージョンの通信では稼働しない。

4023 CWB_NO_VIEWER
System i Access 関数のビューアー・サポートがインストールされていない。

4024 CWB_MODULE_NOT_LOADABLE
フィルター DLL がロードできない。

4025 CWB_ALREADY_SETUP
オブジェクトが既にセットアップされている。

4026 CWB_CANNOT_START_PROCESS
プロセスの開始が失敗。他のエラー・コードも参照。

4027 CWB_NON_REPRESENTABLE_UNICODE_CHAR
1 つ以上の入力 UNICODE 文字が、使用中のコード・ページにない。

8998 CWB_UNSUPPORTED_FUNCTION
サポートされない機能。

8999 CWB_INTERNAL_ERROR
内部エラーが発生しました。

関連資料

46 ページの『System i Access for Windows の通信およびセキュリティー API』
『System i Access forWindows 通信およびセキュリティー』トピックでは、System i Access for Windows アプリケーション・プログラミング・インターフェース (API) の使用方法について説明します。

System i Access for Windows 固有の戻りコード:

System i Access for Windows に固有の戻りコードです。

セキュリティーの戻りコード:

System i Access for Windows のセキュリティーの戻りコードです。

```
8001    CWB_UNKNOWN_USERID
8002    CWB_WRONG_PASSWORD
8003    CWB_PASSWORD_EXPIRED
8004    CWB_INVALID_PASSWORD
8006    CWB_INCORRECT_DATA_FORMAT
8007    CWB_GENERAL_SECURITY_ERROR
8011    CWB_USER_PROFILE_DISABLED
8013    CWB_USER_CANCELLED
8014    CWB_INVALID_SYSNAME
8015    CWB_INVALID_USERID
8016    CWB_LIMITED_CAPABILITIES_USERID
8019    CWB_INVALID_TP_ON_HOST
8022    CWB_NOT_LOGGED_ON
8026    CWB_EXIT_PGM_ERROR
8027    CWB_EXIT_PGM_DENIED_REQUEST
8050    CWB_TIMESTAMPS_NOT_SET
8051    CWB_KERB_CLIENT_CREDENTIALS_NOT_FOUND
8052    CWB_KERB_SERVICE_TICKET_NOT_FOUND
8053    CWB_KERB_SERVER_CANNOT_BE_CONTACTED
8054    CWB_KERB_UNSUPPORTED_BY_HOST
8055    CWB_KERB_NOT_AVAILABLE
8056    CWB_KERB_SERVER_NOT_CONFIGURED
8057    CWB_KERB_CREDENTIALS_NOT_VALID
8058    CWB_KERB_MAPPED_USERID_FAILURE
8059    CWB_KERB_MAPPED_USERID_SUCCESS
8070    CWB_PROFILE_TOKEN_INVALID
8071    CWB_PROFILE_TOKEN_MAXIMUM
8072    CWB_PROFILE_TOKEN_NOT_REGENERABLE
8257    CWB_PW_TOO_LONG
8258    CWB_PW_TOO_SHORT
8259    CWB_PW_REPEAT_CHARACTER
8260    CWB_PW_ADJACENT_DIGITS
8261    CWB_PW_CONSECUTIVE_CHARS
8262    CWB_PW_PREVIOUSLY_USED
8263    CWB_PW_DISALLOWED_CHAR
8264    CWB_PW_NEED_NUMERIC
8266    CWB_PW_MATCHES_OLD
8267    CWB_PW_NOT_ALLOWED
8268    CWB_PW_CONTAINS_USERID
8270    CWB_PW_LAST_INVALID_PWD
8271    CWB_PW_STAR_NONE
8272    CWB_PW_QPWDVLDPGM
```

通信の戻りコード:

System i Access for Windows の通信の戻りコードです。

```
8400    CWB_INV_AFTER_SIGNON
8401    CWB_INV_WHEN_CONNECTED
8402    CWB_INV_BEFORE_VALIDATE
```

```
8403    CWB_SECURE_SOCKETS_NOTAVAIL
8404    CWB_RESERVED1
8405    CWB_RECEIVE_ERROR
8406    CWB_SERVICE_NAME_ERROR
8407    CWB_GETPORT_ERROR
8408    CWB_SUCCESS_WARNING
8409    CWB_NOT_CONNECTED
8410    CWB_DEFAULT_HOST_CCSID_USED
8411    CWB_USER_TIMEOUT
8412    CWB_SSL_JAVA_ERROR
8413    CWB_USER_TIMEOUT_SENDRCV
8414    CWB_FIPS_UNAVAILABLE
```

構成の戻りコード:

System i Access for Windows の構成の戻りコードです。

```
8500   CWB_RESTRICTED_BY_POLICY
8501   CWB_POLICY_MODIFY_MANDATED_ENV
8502   CWB_POLICY_MODIFY_CURRENT_ENV
8503   CWB_POLICY_MODIFY_ENV_LIST
8504   CWB_SYSTEM_NOT_FOUND
8505   CWB_ENVIRONMENT_NOT_FOUND
8506   CWB_ENVIRONMENT_EXISTS
8507   CWB_SYSTEM_EXISTS
8508   CWB_NO_SYSTEMS_CONFIGURED
8580   CWB_CONFIGERR_RESERVED_START
8599   CWB_CONFIGERR_RESERVED_END
```

オートメーション・オブジェクトの戻りコード:

System i Access for Windows のオートメーション・オブジェクトの戻りコードです。

```
8600   CWB_INVALID_METHOD_PARM
8601   CWB_INVALID_PROPERTY_PARM
8602   CWB_INVALID_PROPERTY_VALUE
8603   CWB_OBJECT_NOT_INITIALIZED
8604   CWB_OBJECT_ALREADY_INITIALIZED
8605   CWB_INVALID_DQ_ORDER
8606  CWB_DATA_TRANSFER_REQUIRED
8607   CWB_UNSUPPORTED_XFER_REQUEST
8608  CWB_ASYNC_REQUEST_ACTIVE
8609   CWB_REQUEST_TIMED_OUT
8610   CWB_CANNOT_SET_PROP_NOW
8611  CWB_OBJ_STATE_NO_LONGER_VALID
```

WINSOCK 戻りコード:

System i Access for Windows の WINSOCK 戻りコードを示します。

```
10024    CWB_TOO_MANY_OPEN_SOCKETS
10035    CWB_RESOURCE_TEMPORARILY_UNAVAILABLE
10038    CWB_SOCKET_OPERATION_ON_NON_SOCKET
10047    CWB_PROTOCOL_NOT_INSTALLED
10050    CWB_NETWORK_IS_DOWN
10051    CWB_NETWORK_IS_UNREACHABLE
10052    CWB_NETWORK_DROPPED_CONNECTION_ON_RESET
10053    CWB_SOFTWARE_CAUSED_CONNECTION_ABORT
10054    CWB_CONNECTION_RESET_BY_PEER
10055    CWB_NO_BUFFER_SPACE_AVAILABLE
10057    CWB_SOCKET_IS_NOT_CONNECTED
10058    CWB_CANNOT_SEND_AFTER_SOCKET_SHUTDOWN
10060    CWB_CONNECTION_TIMED_OUT
10061    CWB_CONNECTION_REFUSED
10064    CWB_HOST_IS_DOWN
10065    CWB_NO_ROUTE_TO_HOST
```

10091 CWB_NETWORK_SUBSYSTEM_IS_UNAVAILABLE
10092 CWB_WINSOCK_VERSION_NOT_SUPPORTED
11001 CWB_HOST_DEFINITELY_NOT_FOUND
TCP/IP アドレスのルックアップでシステム名が見付からない。
11002 CWB_HOST_NOT_FOUND_BUT_WE_ARE_NOT_SURE
TCP/IP アドレスのルックアップでシステム名が見付からない。
11004 CWB_VALID_NAME_BUT_NO_DATA_RECORD
ローカルの SERVICES ファイルでサービス名が見付からない。

SSL 戻りコード:

System i Access for Windows の SSL 戻りコードを示します。

キー・データベースのエラー・コード

20001 - 不明エラーが発生。
20002 - asn.1 エンコード/デコード・エラーが発生。
20003 - asn.1 エンコーダー/デコーダーの初期化時にエラーが発生。
20004 - 索引が範囲外であるかオプション・フィールドが存在しないために
asn.1 エンコード/デコード・エラーが発生。
20005 - データベース・エラーが発生。
20006 - データベース・ファイルのオープン時にエラーが発生。
20007 - データベース・ファイルの再オープン時にエラーが発生。
20008 - データベースの作成に失敗。
20009 - データベースが既に存在する。
20010 - データベース・ファイルの削除時にエラーが発生。
20011 - データベースがオープンされていない。
20012 - データベース・ファイルの読み取り時にエラーが発生。
20013 - データベース・ファイルへのデータの書き込み時にエラーが発生。
20014 - データベース妥当性検査エラーが発生。
20015 - 無効なデータベース・バージョンを検出。
20016 - 無効なデータベース・パスワードを検出。
20017 - 無効なデータベース・ファイル・タイプを検出。
20018 - データベースが破壊された。
20019 - 無効なパスワードを検出、またはデータベースが無効である。
20020 - データベース・キー項目整合性エラーが発生。
20021 - データベースに重複するキーが既にある。
20022 - データベースに重複するキーが既にある (レコード ID)。
20023 - データベースに重複するキーが既にある (ラベル)。
20024 - データベースに重複するキーが既にある (署名)。
20025 - データベースに重複するキーが既にある (未署名の証明書)。
20026 - データベースに重複するキーが既にある (発行者およびシリアル番号)。
20027 - データベースに重複するキーが既にある (サブジェクト公開鍵情報)。
20028 - データベースに重複するキーが既にある (未署名の CRL)。
20029 - ラベルがデータベースで使用されている。
20030 - パスワード暗号化エラーが発生。
20031 - LDAP 関連エラーが発生。
20032 - 暗号エラーが発生。
20033 - 暗号機能エラーが発生。
20034 - 無効な暗号アルゴリズムを検出。
20035 - データの署名時にエラーが発生。
20036 - データの検査時にエラーが発生。
20037 - データのダイジェストの計算時にエラーが発生。
20038 - 無効な暗号パラメーターを検出。
20039 - サポートされない暗号アルゴリズムを検出。
20040 - 指定された入力サイズがサポートされている係数サイズより大きい。
20041 - サポートされない係数サイズを検出。
20042 - データベース妥当性検査エラーが発生。
20043 - キー項目妥当性検査に失敗。
20044 - 重複する拡張フィールドが存在する。
20045 - キーのバージョンが間違っている。
20046 - 必須の拡張フィールドが存在しない。
20047 - 有効期間に本日の日付が含まれていない、または有効期間がその発行者の有効期間に含まれていない。
20048 - 有効期間に本日の日付が含まれていない、または有効期間がその発行者の有効期間に含まれていない。
20049 - 秘密鍵使用拡張の有効期間の妥当性検査時にエラーが発生。
20050 - キーの発行者が見つからない。
20051 - 必須の証明書拡張がない。

20052 - キー署名の妥当性検査が失敗した。
20053 - キー署名の妥当性検査が失敗した。
20054 - キーのルート・キーが信頼されない。
20055 - キーが失効している。
20056 - 権限キー ID 拡張の妥当性検査時にエラーが発生。
20057 - 秘密鍵使用拡張の妥当性検査時にエラーが発生。
20058 - サブジェクト代替名拡張の妥当性検査時にエラーが発生。
20059 - 発行者代替名拡張の妥当性検査時にエラーが発生。
20060 - キー使用拡張の妥当性検査時にエラーが発生。
20061 - 不明な重大拡張を検出。
20062 - 鍵ペア項目の妥当性検査時にエラーが発生。
20063 - CRL の妥当性検査時にエラーが発生。
20064 - mutex エラーが発生。
20065 - 無効なパラメーターを検出。
20066 - ヌルのパラメーターまたはメモリー割り振りエラーを検出。
20067 - サイズが大きすぎる、または小さすぎる。
20068 - 古いパスワードは無効。
20069 - 新しいパスワードは無効。
20070 - パスワードが期限切れである。
20071 - スレッド関連のエラーが発生。
20072 - スレッドの作成時にエラーが発生。
20073 - スレッドが終了を待機している間にエラーが発生。
20074 - 入出力エラーが発生。
20075 - CMS のロード時にエラーが発生。
20076 - 暗号化ハードウェア関連のエラーが発生。
20077 - ライブラリー初期化ルーチンが正しく呼び出されなかった。
20078 - 内部データベース・ハンドル・テーブルが破壊された。
20079 - メモリー割り振りエラーが発生。
20080 - 認識されないオプションを検出。
20081 - 時間情報の取得時にエラーが発生。
20082 - mutex 作成エラーが発生。
20083 - メッセージ・カタログのオープン時にエラーが発生。
20084 - エラー・メッセージ・カタログのオープン時にエラーが発生。
20085 - ヌルのファイル名を検出。
20086 - ファイルのオープン時にエラーが発生したので、ファイルの存在と許可をチェックする。
20087 - 読み取るファイルのオープン時にエラーが発生。
20088 - 書き込むファイルのオープン時にエラーが発生。
20089 - このようなファイルはない。
20090 - 許可設定が原因でファイルをオープンできない。
20091 - ファイルへのデータの書き込み時にエラーが発生。
20092 - ファイルの削除時にエラーが発生。
20093 - 無効な Base64 エンコード・データを検出。
20094 - 無効な Base64 メッセージ・タイプを検出。
20095 - Base64 エンコード規則でデータをエンコードしているときにエラーが発生。
20096 - Base64 エンコード・データのデコード時にエラーが発生。
20097 - 識別名タグの取得時にエラーが発生。
20098 - 必須の共通名フィールドが空。
20099 - 必須の国名フィールドが空。
20100 - 無効なデータベース・ハンドルを検出。
20101 - キー・データベースが存在しない。
20102 - 要求鍵ペア・データベースが存在しない。
20103 - パスワード・ファイルが存在しない。
20104 - 新規パスワードが古いパスワードと等しい。
20105 - キー・データベース内にキーが見つからない。
20106 - 要求キーが見つからない。
20107 - トラステッド CA が見つからない。
20108 - 証明書に対する要求キーが見つからない。
20109 - キー・データベース内に秘密鍵がない。
20110 - キー・データベース内にデフォルト・キーがない。
20111 - キー・レコードに秘密鍵がない。
20112 - キー・レコードに証明書がない。
20113 - CRL 項目がない。
20114 - 無効なキー・データベース・ファイル名を検出。
20115 - 認識されない秘密鍵タイプを検出。
20116 - 無効な識別名入力を検出。
20117 - 指定されたキー・ラベルを持つキー項目が見つからない。
20118 - キー・ラベル・リストが破壊された。

20119 - 入力データが有効な PKCS12 データではない。
20120 - パスワードが無効である、または PKCS12 データが破壊されたか PKCS12 の
後続のバージョンで作成されている。
20121 - 認識されないキー・エクスポート・タイプを検出。
20122 - サポートされないパスワード・ベースの暗号化アルゴリズムを検出。
20123 - 鍵リング・ファイルから CMS キー・データベースへの変換時にエラーが発生。
20124 - CMS キー・データベースから鍵リング・ファイルへの変換時にエラーが発生。
20125 - 証明書要求に対して証明書を作成しているときにエラーが発生。
20126 - 完全な発行者チェーンを作成できない。
20127 - 無効な WEBDB データを検出。
20128 - 鍵リング・ファイルに書き込むデータがない。
20129 - 入力した日数が許容されている有効期間を超えている。
20130 - パスワードが短すぎる。最低文字数に満たない。
20131 - パスワードには少なくとも数字が 1 つ含まれていなければならない。
20132 - パスワード内の文字がすべて英字または数字のいずれかである。
20133 - 認識されないまたはサポートされない署名アルゴリズムが指定された。
20134 - 無効なキー・データベース・タイプが指定された。
20135 - 現在、2 次キー・データベースは別の基本キー・データベースに対する 2 次キー・データベースである。
20136 - キー・データベースに、関連付けられた 2 次キー・データベースがない。
20137 - ラベルを持つ暗号トークンが見つからない。
20138 - 暗号トークン・パスワードが指定されていないが、これは必須である。
20139 - 暗号トークン・パスワードが指定されたが、これは必須ではない。
20140 - 暗号モジュールをロードできない。暗号トークンのサポートを受けられない。
20141 - 暗号トークンの機能がサポートされていない。
20142 - 暗号トークン機能が失敗した。

SSL エラー・コード

25001 - ハンドルが無効。
25002 - ダイナミック・リンク・ライブラリーが使用できない。
25003 - 内部エラーが発生。
25004 - メイン・メモリーに操作を実行するだけのスペースがない。
25005 - ハンドルが操作に対して有効な状態ではない。
25006 - キー・ラベルが見つからない。
25007 - 証明書が使用できない。
25008 - 証明書妥当性検査エラー。
25009 - 暗号化処理エラー。
25010 - 証明書内の ASN フィールドの妥当性検査エラー。
25011 - LDAP サーバー接続エラー。
25012 - 不明な内部エラー。問題をサービスに報告する。
25101 - 暗号の処理でエラーが発生。
25102 - キー・ファイルの読み取りで入出力エラーが発生。
25103 - キー・ファイルの内部形式が無効。キー・ファイルを再作成する。
25104 - キー・ファイルに同じキーを持つ項目が 2 つある。iKeyman を使用して重複するキーを除去する。
25105 - キー・ファイルに同じラベルを持つ項目が 2 つある。iKeyman を使用して重複するラベルを除去する。
25106 - キー・ファイルのパスワードが整合性チェックとして使用されている。
キー・ファイルが破壊されているか、パスワード ID が誤っている。
25107 - キー・ファイル内のデフォルト・キーに対する証明書の有効期限が切れている。
iKeyman を使用して有効期限切れの証明書を除去する。
25108 - ダイナミック・リンク・ライブラリーの 1 つをロードしているときにエラーが発生。
25109 - 環境がクローズした後で接続を確立しようとしている。
25201 - キー・ファイルを初期化できなかった。
25202 - キー・ファイルを開くことができない。パスの指定が誤っているか、
ファイル権限でファイルを開くことが許可されていなかった。
25203 - 一時的な鍵ペアを生成できない。
25204 - ユーザー名オブジェクトが指定されたが、見つからない。
25205 - LDAP 照会に使用するパスワードが正しくない。
25206 - LDAP サーバーのフェイルオーバー・リストに対する索引が正しくない。
25301 - クローズ時にエラーが発生。
25401 - システム日付が無効な値に設定された。
25402 - SSLV2 も SSLV3 も使用可能ではない。
25403 - 必要な証明書をパートナーから受け取らなかった。
25404 - 受け取った証明書の形式が正しくない。
25405 - 受け取った証明書のタイプはサポートされていない。
25406 - データの読み取りまたは書き込み時に入出力エラーが発生。
25407 - キー・ファイルで指定されたラベルが見つからない。

25408 - 指定されたキー・ファイルのパスワードが誤っている。キー・ファイルを使用できなかった。
キー・ファイルが壊れている可能性もある。
25409 - 制限された暗号化環境で、キー・サイズが長すぎてサポートされない。
25410 - 誤った形式の SSL メッセージをパートナーから受け取った。
25411 - メッセージ確認コード (MAC) が正常に検査されなかった。
25412 - この操作はサポートされない。
25413 - 受け取った証明書に誤った署名が入っていた。
25414 - サーバー証明書が信頼されない。通常、これは、サーバー証明書の認証局をダウンロードしなかった場合に発生する。デジタル証明書マネージャーを使用して認証局を取得し、PC の IBM 鍵管理ユーティリティーを使用してローカル・キー・データベースにその認証局を置く。詳細については、CWBCO1050 を参照。
25415 - リモート・システム情報が無効。
25416 - アクセスが拒否された。
25417 - 自己署名証明書が無効。
25418 - 読み取りが失敗した。
25419 - 書き込みが失敗した。
25420 - プロトコルが完了する前にパートナーがソケットを閉じた。つまり、パートナーは SSL クライアント認証用に構成されているのに、クライアント証明書がパートナーに送信されなかった可能性がある。
25421 - 指定された V2 暗号が無効。
25422 - 指定された V3 暗号が無効。
25425 - ハンドルを作成できない。
25426 - 初期化に失敗した。
25427 - 証明書を妥当性検査しているときに、指定された LDAP ディレクトリーにアクセスできない。
25428 - 指定されたキーに秘密鍵が入っていなかった。
25429 - 指定された PKCS11 共用ライブラリーをロードする試みが失敗した。
25430 - PKCS #11 ドライバーが、呼び出し元で指定されたトークンを見つけられなかった。
25431 - PKCS #11 トークンがスロット内にない。
25432 - PKCS #11 トークンにアクセスするパスワード/ピンが無効。
25433 - 受け取った SSL ヘッダーが正しい SSLV2 形式のヘッダーではなかった。
25434 - ハードウェア・ベースの暗号サービス・プロバイダー (CSP) にアクセスできない。
25435 - 属性の設定が矛盾する。
25436 - アプリケーションが実行されているプラットフォームで要求された機能がサポートされていない。
25437 - IPv6 接続を検出。
25438 - リセット・セッション・タイプのコールバック機能から誤った値が戻された。
25501 - バッファー・サイズが負または 0 である。
25502 - 非ブロッキング I/O で使用した。
25601 - reset_cipher では SSLV3 を必要とし、接続は SSLV2 を使用する。
25602 - 関数呼び出しで無効な ID が指定された。
25701 - 関数呼び出しの ID が無効。
25702 - 属性が負の長さで、無効。
25703 - 指定された列挙タイプに対し列挙値が無効。
25704 - SID キャッシュ・ルーチンを置き換えるためのパラメーター・リストが無効。
25705 - 数値属性を設定するときに、指定された値が設定対象の具体的な属性に対して無効である。
25706 - 追加の証明書妥当性検査で矛盾するパラメーターが設定された。
25707 - 暗号仕様に、実行のシステムでサポートされていない AES 暗号仕様が含まれていた。
25708 - ピア ID の長さが誤っている。16 バイト以下でなければならない。

System i Access for Windows の構成要素に固有の戻りコード:

System i Access for Windows の API タイプの戻りコードを示します。

管理 API の戻りコード:

System i Access for Windows の管理の戻りコードです。

6001 CWBAD_INVALID_COMPONENT_ID
構成要素 ID が無効。

関連資料

34 ページの『System i Access for Windows 管理 API』
これらの API には、PC にインストールされている System i Access for Windows のコードに関する情報にアクセスするための機能が備わっています。

通信 API の戻りコード:

System i Access for Windows 通信 API の戻りコードを示します。

6001 CWBCO_END_OF_LIST
システム・リストの終わりに達した。システム名が、戻されませんでした。
6002 CWBCO_DEFAULT_SYSTEM_NOT_DEFINED
デフォルト・システムの設定が未定義。
6003 CWBCO_DEFAULT_SYSTEM_NOT_CONFIGURED
デフォルト・システムは定義されているが、接続が構成されていない。
6004 CWBCO_SYSTEM_NOT_CONNECTED
現行プロセスでは、指定のシステムは現在接続されていない。
6005 CWBCO_SYSTEM_NOT_CONFIGURED
指定のシステムは、現在構成されていない。
6007 CWBCO_INTERNAL_ERROR
内部エラー
6008 CWBCO_NO_SUCH_ENVIRONMENT
指定の環境は存在しません。

関連資料

46 ページの『System i Access for Windows の通信およびセキュリティー API』
『System i Access forWindows 通信およびセキュリティー』トピックでは、System i Access for Windows アプリケーション・プログラミング・インターフェース (API) の使用方法について説明します。

データベース API の戻りコード:

System i Access for Windows のデータベース API の戻りコードを示します。

注: データベース API に関する重要な情報については、System i Access for Windows のデータベース API に関するトピックを参照してください。

6001 CWBDB_CANNOT_CONTACT_SERVER
データ・アクセス・サーバーの開始を阻むエラーが発生。
6002 CWBDB_ATTRIBUTES_FAILURE
データ・アクセス・サーバーの属性設定中にエラー発生。
6003 CWBDB_SERVER_ALREADY_STARTED
有効なサーバーの稼働中にデータ・アクセス・サーバーを開始しようとした。
再始動の前にそれを停止してください。
6004 CWBDB_INVALID_DRDA_PKG_SIZE
サブミットされた DRDA 圧縮サイズが無効。
6005 CWBDB_REQUEST_MEMORY_ALLOCATION_FAILURE
要求ハンドルによるメモリー割り振りが失敗。
6006 CWBDB_REQUEST_INVALID_CONVERSION
要求ハンドルがデータ変換に失敗。
6007 CWBDB_SERVER_NOT_ACTIVE
データ・アクセス・サーバーが開始していない。
サーバーを開始してから、継続してください。
6008 CWBDB_PARAMETER_ERROR
パラメーター設定が失敗。再度試してみてください。エラーが直らないようであれば、
使用可能メモリーが不足している可能性があります。
6009 CWBDB_CLONE_CREATION_ERROR
複製要求を作成できない。
6010 CWBDB_INVALID_DATA_FORMAT_FOR_CONNECTION
この接続に対するデータ形式オブジェクトが無効。
6011 CWBDB_DATA_FORMAT_IN_USE
データ形式オブジェクトが既に別の要求により使用中。
6012 CWBDB_INVALID_DATA_FORMAT_FOR_DATA
データ形式オブジェクトがデータの形式に一致しない。
6013 CWBDB_STRING_ARG_TOO_LONG
与えられたストリングは、パラメーターとして長すぎる。
6014 CWBDB_INVALID_INTERNAL_ARG
(ユーザーが与えたものでない) 内部的に生成された引数が無効。
6015 CWBDB_INVALID_NUMERIC_ARG
数値引数の値が無効。
6016 CWBDB_INVALID_ARG
引数の値が無効。

6017 CWBDB_STMT_NOT_SELECT
与えられたステートメントが SELECT ステートメントでない。
この呼び出しでは、SELECT ステートメントが必要。
6018 CWBDB_STREAM_FETCH_NOT_COMPLETE
この接続は、ストリーム・フェッチ方式。ストリーム・フェッチが終わるまで
要求の操作はできない。
6019 CWBDB_STREAM_FETCH_NOT_ACTIVE
この接続は、ストリーム・フェッチ方式でない。要求の
操作のためには、ストリーム・フェッチ方式でなければならない。
6020 CWBDB_MISSING_DATA_PROCESSOR
要求オブジェクト中のデータ処理装置を指すポインターが NULL。
6021 CWBDB_ILLEGAL_CLONE_REQUEST_TYPE
属性要求の複製を作成できない。
6022 CWBDB_UNSOLICITED_DATA
サーバーからデータを受け取ったが、要求したものでない。
6023 CWBDB_MISSING_DATA
サーバーにデータを要求したが、一部未受領。
6024 CWBDB_PARM_INVALID_BITSTREAM
パラメーター中のビット・ストリームが無効。
6025 CWBDB_CONSISTENCY_TOKEN_ERROR
システムからのデータの解釈に使用したデータ形式が、戻されたデータと一致しない。
6026 CWBDB_INVALID_FUNCTION
このタイプの要求ではこの関数は無効。
6027 CWBDB_FORMAT_INVALID_ARG
API に渡されたパラメーター値が無効。
6028 CWBDB_INVALID_COLUMN_POSITION
API に渡された列位置が無効。
6029 CWBDB_INVALID_COLUMN_TYPE
API に渡された列タイプが無効。
6030 CWBDB_ROW_VECTOR_NOT_EMPTY
使用した形式が無効または間違っている。
6031 CWBDB_ROW_VECTOR_EMPTY
使用した形式が無効または間違っている。
6032 CWBDB_MEMORY_ALLOCATION_FAILURE
メモリー割り振り中にエラー発生。
6033 CWBDB_INVALID_CONVERSION
無効なタイプ変換を試みた。
6034 CWBDB_DATASTREAM_TOO_SHORT
ホストから受け取ったデータ・ストリームが短すぎる。
6035 CWBDB_SQL_WARNING
データベース・サーバーが SQL 操作から警告を受け取った。
6036 CWBDB_SQL_ERROR
データベース・サーバーが SQL 操作からエラーを受け取った。
6037 CWBDB_SQL_PARAMETER_WARNING
データベース・サーバーが SQL 操作に使用されているパラメーターに関する
警告を受け取った。
6038 CWBDB_SQL_PARAMETER_ERROR
データベース・サーバーが SQL 操作に使用されているパラメーターに関する
警告を受け取った。
6039 CWBDB_LIST_SERVER_WARNING
データベース・サーバーがカタログ操作から警告を戻した。
6040 CWBDB_LIST_SERVER_ERROR
データベース・サーバーがカタログ操作からエラーを戻した。
6041 CWBDB_LIST_PARAMETER_WARNING
データベース・サーバーがカタログ操作に使用したパラメーター
に関する警告を戻した。
6042 CWBDB_LIST_PARAMETER_ERROR
データベース・サーバーがカタログ操作に使用したパラメーター
に関するエラーを戻した。
6043 CWBDB_NDB_FILE_SERVER_WARNING
データベース・サーバーがファイル処理操作から
警告を戻した。
6044 CWBDB_NDB_FILE_SERVER_ERROR
データベース・サーバーがファイル処理操作からエラーを戻した。
6045 CWBDB_FILE_PARAMETER_WARNING
データベース・サーバーがファイル処理操作に使用したパラメーター
に関する警告を戻した。

6046 CWBDB_FILE_PARAMETER_ERROR
データベース・サーバーがファイル処理操作に使用したパラメーター
に関するエラーを戻した。
6047 CWBDB_GENERAL_SERVER_WARNING
データベース・サーバーが一般警告を戻した。
6048 CWBDB_GENERAL_SERVER_ERROR
データベース・サーバーが一般エラーを戻した。
6049 CWBDB_EXIT_PROGRAM_WARNING
データベース・サーバーが出口プログラムから警告を戻した。
6050 CWBDB_EXIT_PROGRAM_ERROR
データベース・サーバーが出口プログラムからエラーを戻した。
6051 CWBDB_DATA_BUFFER_TOO_SMALL
ターゲットのデータ・バッファーが移動元のバッファーよりも小さい。
6052 CWBDB_NL_CONVERSION_ERROR
PiNlConverter からエラーを受け取った。
6053 CWBDB_COMMUNICATIONS_ERROR
処理中に通信エラーを受け取った。
6054 CWBDB_INVALID_ARG_API
引数の値が無効 - API レベル。
6055 CWBDB_MISSING_DATA_HANDLER
データ・ハンドラーがデータ・ハンドラー・リストにない。
6056 CWBDB_REQUEST_DATASTREAM_NOT_VALID
カタログ要求中に無効なデータ・ストリームがある。
6057 CWBDB_SERVER_UNABLE
サーバーは要求の関数を実行できない。

以下の戻りコードは、cwbDB_StartServerDetailed API によって戻されます。

6058 CWBDB_WORK_QUEUE_START_ERROR
クライアント作業待ち行列の問題のため、サーバーを開始できない。
6059 CWBDB_WORK_QUEUE_CREATE_ERROR
クライアント作業待ち行列の問題のため、サーバーを開始できない。
6060 CWBDB_INITIALIZATION_ERROR
クライアント初期設定の問題のため、サーバーを開始できない。
6061 CWBDB_SERVER_ATTRIBS_ERROR
サーバー属性の問題のため、サーバーを開始できない。
6062 CWBDB_CLIENT_LEVEL_ERROR
クライアント・レベル設定の問題のため、サーバーを開始できない。
6063 CWBDB_CLIENT_LFC_ERROR
クライアント言語機能コード設定の問題のため、サーバーを開始できない。
6064 CWBDB_CLIENT_CCSID_ERROR
クライアント CCSID の問題のため、サーバーを開始できない。
6065 CWBDB_TRANSLATION_INDICATOR_ERROR
変換標識設定エラーのため、サーバーを開始できない。
6066 CWBDB_RETURN_SERVER_ATTRIBS_ERROR
サーバー属性の戻しの問題のため、サーバーを開始できない。
6067 CWBDB_SERVER_ATTRIBS_REQUEST
サーバー属性要求オブジェクトの欠落のため、
サーバーを開始できない。
6068 CWBDB_RETURN_ATTRIBS_ERROR
属性の戻り値の問題のため、サーバーを開始できない。
6069 CWBDB_SERVER_ATTRIBS_MISSING
戻されたサーバーの属性不足で、サーバーを開始できない。
(データの欠落)
6070 CWBDB_SERVER_LFC_CONVERSION_ERROR
サーバー属性のサーバー言語フィーチャー・コード・フィールドの
データ変換エラーのため、サーバーを開始できない。
6071 CWBDB_SERVER_LEVEL_CONVERSION_ERROR
サーバー属性のサーバー機能レベル・フィールドの
データ変換エラーのため、サーバーを開始できない。
6072 CWBDB_SERVER_LANGUAGE_TABLE_ERROR
サーバー属性のサーバー言語テーブル ID フィールドの
データ変換エラーのため、サーバーを開始できない。
6073 CWBDB_SERVER_LANGUAGE_LIBRARY_ERROR
サーバー属性のサーバー言語ライブラリー ID フィールドの
データ変換エラーのため、サーバーを開始できない。

6074 CWBDB_SERVER_LANGUAGE_ID_ERROR
サーバー属性のサーバー言語 ID フィールドの
データ変換エラーのため、サーバーを開始できない。
6075 CWBDB_COMM_DEQUEUE_ERROR
通信エラーのためサーバーを開始できない。
6076 CWBDB_COMM_ENQUEUE_ERROR
通信エラーのためサーバーを開始できない。
6077 CWBDB_UNSUPPORTED_COLUMN_TYPE
データにサポートされない列タイプが見付かった。
6078 CWBDB_SERVER_IN_USE
所定の接続ハンドルでのデータベース・サーバーへの接続が、
同じシステム・オブジェクト・ハンドルで作成された別の
接続ハンドルで既に使用されている。
6079 CWBDB_SERVER_REL_DB_CONVERSION_ERROR
サーバー属性のサーバー言語 ID フィールドの
データ変換エラーのため、サーバーを開始できない。
この戻りコードには、メッセージやヘルプ・テキストはありません。
6080 CWBDB_SERVER_FUNCTION_NOT_AVAILABLE
この機能はこのバージョンのホスト・サーバーでは利用できない。
6081 CWBDB_FUNCTION_NOT_VALID_AFTER_CONNECT
ホスト・サーバーへの接続後、この機能は無効。
6082 CWBDB_INVALID_INITIAL_REL_DB_NAME
初期リレーショナル DB 名 (IASP) が無効。
6099 CWBDB_LAST_STREAM_CHUNK
ストリーム・フェッチが完了。
注: 通知メッセージであり、エラーではありません。この戻りコードには、メッセージや
ヘルプ・テキストはありません。

関連資料

631 ページの『System i Access データベースの API』
System i Access for Windows の専有 C/C++ データベース API で提供されていた機能のうち、現在はサポート対象外となっているテクノロジーを使用します。

データ待ち行列 API の戻りコード:

System i Access for Windows のデータ待ち行列 API の戻りコードを示します。

6000 CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。
6001 CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。
6002 CWBDQ_INVALID_QUEUE_HANDLE
待ち行列ハンドルが無効。
6003 CWBDQ_INVALID_READ_HANDLE
データ待ち行列読み取りハンドルが無効。
6004 CWBDQ_INVALID_QUEUE_LENGTH
データ待ち行列の最大レコード長が無効。
6005 CWBDQ_INVALID_KEY_LENGTH
キーの長さが無効。
6006 CWBDQ_INVALID_ORDER
待ち行列の順序が無効。
6007 CWBDQ_INVALID_AUTHORITY
待ち行列の権限が無効。
6008 CWBDQ_INVALID_QUEUE_TITLE
待ち行列の表題 (記述) が長すぎるか変換できない。
6009 CWBDQ_BAD_QUEUE_NAME
待ち行列名が長すぎるか変換できない。
6010 CWBDQ_BAD_LIBRARY_NAME
ライブラリー名が長すぎるか変換できない。
6011 CWBDQ_BAD_SYSTEM_NAME
システム名が長すぎるか変換できない。
6012 CWBDQ_BAD_KEY_LENGTH
このデータ待ち行列のキーの長さが正しくないか、キーの長さが
LIFO または FIFO データ待ち行列に対して 0 よりも大きい。
6013 CWBDQ_BAD_DATA_LENGTH
このデータ待ち行列のデータの長さが適正でない。

データの長さが、ゼロか許容最大値 31744 バイトよりも大きい
(OS/400 の V4R5 以降のバージョンでは 64512 バイト)。
注: V4R5M0 以降の OS/400 システムに接続する場合の
データ長の許容最大値は、64512 バイトに増えています。
それよりも前のリリースの OS/400 に接続される場合、
データ待ち行列に 64512 バイトのデータを書き込むことはできても、
データ待ち行列から読み取ることができるデータの最大値は 31744 バイトになります。

6014 CWBDQ_INVALID_TIME
待ち時間が正しくない。

6015 CWBDQ_INVALID_SEARCH
サーチ順序が正しくない。

6016 CWBDQ_DATA_TRUNCATED
戻りデータが切り捨てられた。

6017 CWBDQ_TIMED_OUT
待ち時間が満了したがデータが戻されなかった。

6018 CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

6019 CWBDQ_USER_EXIT_ERROR
出口プログラムのエラーまたは出口プログラムの数が無効。

6020 CWBDQ_LIBRARY_NOT_FOUND
システムにライブラリーがない。

6021 CWBDQ_QUEUE_NOT_FOUND
システムに待ち行列がない。

6022 CWBDQ_NO_AUTHORITY
ライブラリーかデータ待ち行列に対して権限がない。

6023 CWBDQ_DAMAGED_QUEUE
データ待ち行列が使用不可状態。

6024 CWBDQ_QUEUE_EXISTS
データ待ち行列が既に存在する。

6025 CWBDQ_INVALID_MESSAGE_LENGTH
メッセージ長が無効 - 待ち行列の最大レコード長よりも大きい。

6026 CWBDQ_QUEUE_DESTROYED
レコード読み取りのため待機中または読み取り中に待ち行列が壊された。

6027 CWBDQ_NO_DATA
データを 1 つも受け取らなかった。

6028 CWBDQ_CANNOT_CONVERT
このデータ待ち行列のデータが変換できない。このデータ待ち行列は
使用できるが、ASCII と EBCDIC との間でデータを変換できない。
このデータ・オブジェクトの変換フラグは無視される。

6029 CWBDQ_QUEUE_SYNTAX
データ待ち行列名の構文が正しくない。待ち行列名は、
システム・オブジェクト構文の後に続く。最初の文字は英字で、
後に続く文字は英数字でなければなりません。

6030 CWBDQ_LIBRARY_SYNTAX
ライブラリー名の構文が正しくない。ライブラリー名は、
システム・オブジェクト構文の後に続く。最初の文字は英字で、
後に続く文字は英数字でなければなりません。

6031 CWBDQ_ADDRESS_NOT_SET
アドレスが設定されていない。データ・オブジェクトが cwbDQ_SetDataAddr()
で設定されていないので、アドレスが検索できない。cwbDQ_GetData()
を使用し、cwbDQ_GetDataAddr() は使用しない。

6032 CWBDQ_HOST_ERROR
戻りコードが定義されていないホスト・エラーが発生。
メッセージ・テキストについてはエラー・ハンドルを参照。

6033 CWBDQ_INVALID_SYSTEM_HANDLE
システム・ハンドルが無効。

6099 CWBDQ_UNEXPECTED_ERROR
予期しないエラー。

関連資料

133 ページの『System i データ待ち行列 API』

System i Access for Windows のデータ待ち行列アプリケーション・プログラミング・インターフェース (API) を使用すると、System i のデータ待ち行列に簡単にアクセスできます。データ待ち行列を使用することによって、通信 API を使う必要のないクライアント/サーバー・アプリケーションを作成することができるようになります。

ディレクトリー更新 API の戻りコード:

System i Access for Windows のディレクトリー更新 API の戻りコードを示します。

6000 CWBUP_ENTRY_NOT_FOUND
検索値に一致する更新項目がない。
6001 CWBUP_SEARCH_POSITION_ERROR
検索開始位置が無効。
6002 CWBUP_PACKAGE_NOT_FOUND
パッケージ・ファイルが見付からない。
6003 CWBUP_POSITION_INVALID
与えられた位置が範囲内ではない。
6004 CWBUP_TOO_MANY_ENTRIES
既に存在する更新項目の最大数。No more can be
これ以上作成できない。
6005 CWBUP_TOO_MANY_PACKAGES
この項目に対するパッケージ・ファイルの許容最大数が既にある。
6006 CWBUP_STRING_TOO_LONG
渡されたテキスト・ストリング・パラメーターが CWBUP_MAX_LENGTH より長い。
6007 CWBUP_ENTRY_IS_LOCKED
現在、別のアプリケーションが更新項目リストを変更中です。
No
この時点で、変更は許可されない。
6008 CWBUP_UNLOCK_WARNING
アプリケーションは、更新項目をロックしていません。

関連資料

235 ページの『System i Access for Windows ディレクトリー更新 API』
System i Access for Windows ディレクトリー更新機能を使用する PC ディレクトリー更新を指定します。

各国語サポート API の戻りコード:

System i Access for Windows の NLS API の戻りコードを示します。

6101 CWBNL_ERR_CNV_UNSUPPORTED
あるコード・ページから別のコード・ページへの文字データの変換が試みられたが、
この変換はサポートされていない。
6102 CWBNL_ERR_CNV_TBL_INVALID
変換テーブルの形式が認知されないものである。
6103 CWBNL_ERR_CNV_TBL_MISSING
変換テーブルを使用しようとしたが、テーブルが見付からない。
6104 CWBNL_ERR_CNV_ERR_GET
サーバーからコード・ページ変換テーブル検索中にエラーが発生。
6105 CWBNL_ERR_CNV_ERR_COMM
サーバーからコード・ページ変換テーブル検索中に通信エラーが発生。
6106 CWBNL_ERR_CNV_ERR_SERVER
サーバーからコード・ページ変換テーブル検索中サーバー・エラーが発生。
6107 CWBNL_ERR_CNV_ERR_STATUS
文字データをあるコード・ページから別のコード・ページに変換中、
変換不可能な文字が見付かった。
6108 CWBNL_ERROR_CONVERSION_INCOMPLETE_MULTIBYTE_INPUT_CHARACTER
文字データを変換中に不完全なマルチバイト文字が見付かった。
6109 CWBNL_ERR_CNV_INVALID_SISO_STATUS
SISO パラメーターが正しくない。
6110 CWBNL_ERR_CNV_INVALID_PAD_LENGTH
埋め込み長さパラメーターが正しくない。

以下の戻りコードは、言語 API 用です。

6201 CWBNL_ERR_STR_TBL_INVALID
メッセージ・ファイルの形式が認知されない。破壊されている。
6202 CWBNL_ERR_STR_TBL_MISSING
メッセージ・ファイルが見付からない。
6203 CWBNL_ERR_STR_NOT_FOUND
メッセージ・ファイルにメッセージがない。

6204 CWBNL_ERR_NLV_NO_CONFIG
言語構成がない。
6205 CWBNL_ERR_NLV_NO_SUBDIR
言語サブディレクトリーがない。
6206 CWBNL_DEFAULT_HOST_CCSID_USED
デフォルトのサーバー CCSID (500) を使用。

以下の戻りコードは、ロケール API 用です。

6301 CWBNL_ERR_LOC_TBL_INVALID
6302 CWBNL_ERR_LOC_TBL_MISSING
6303 CWBNL_ERR_LOC_NO_CONFIG
6304 CWBNL_ERR_LOC_NO_LOCPATH

システム・オブジェクト API の戻りコード:

System i Access for Windows システム・オブジェクト API の戻りコードを示します。

6000 CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。
6001 CWBOBJ_RC_INVALID_TYPE
オブジェクト・タイプが正しくない。
6002 CWBOBJ_RC_INVALID_KEY
キーが正しくない。
6003 CWBOBJ_RC_INVALID_INDEX
リストへの索引が正しくない。
6004 CWBOBJ_RC_LIST_OPEN
リストは既にオープンされている。
6005 CWBOBJ_RC_LIST_NOT_OPEN
リストがオープンされていない。
6006 CWBOBJ_RC_SEEKOUTOFRANGE
シークのオフセットが範囲外。
6007 CWBOBJ_RC_SPLFNOTOPEN
スプール・ファイルがオープンされていない。
6007 CWBOBJ_RC_RSCNOTOPEN
資源がオープンされていない。
6008 CWBOBJ_RC_SPLFENDOFFILE
ファイルの終わりに達した。
6008 CWBOBJ_RC_ENDOFFILE
ファイルの終わりに達した。
6009 CWBOBJ_RC_SPLFNOMESSAGE
スプール・ファイルがメッセージを待っていない。
6010 CWBOBJ_RC_KEY_NOT_FOUND
パラメーター・リストに指定のキーがない。
6011 CWBOBJ_RC_NO_EXIT_PGM
出口プログラムが登録されていない。
6012 CWBOBJ_RC_NOHOSTSUPPORT
ホストは、関数をサポートしない。

関連資料

258 ページの『System i Access for Windows 用システム・オブジェクト API』
System i Access for Windows 用システム・オブジェクトのアプリケーション・プログラミング・インターフェース (API) を使用して、システム上にある印刷関連のオブジェクトを処理できます。これらの API は、System i スプール・ファイル、書き出しプログラム・ジョブ、出力待ち行列、プリンターなどを使った作業を可能にします。

リモート・コマンド/分散プログラム呼び出し API 戻りコード:

System i Access for Windows のリモート・コマンド/分散プログラム呼び出し API の戻りコードを示します。

6000 CWBRC_INVALID_SYSTEM_HANDLE
システム・ハンドルが無効。
6001 CWBRC_INVALID_PROGRAM

プログラム・ハンドルが無効。
6002 CWBRC_SYSTEM_NAME
システム名が長すぎるか変換できない。
6003 CWBRC_COMMAND_STRING
コマンド・ストリングが長すぎるか、変換できない。
6004 CWBRC_PROGRAM_NAME
プログラム名が長すぎるか、変換できない。
6005 CWBRC_LIBRARY_NAME
ライブラリー名が長すぎるか変換できない。
6006 CWBRC_INVALID_TYPE
指定のパラメーター・タイプが無効。
6007 CWBRC_INVALID_PARM_LENGTH
パラメーターの長さが無効。
6008 CWBRC_INVALID_PARM
指定のパラメーターが無効。
6009 CWBRC_TOO_MANY_PARMS
プログラムに追加しようとしたパラメーターが多すぎる。
6010 CWBRC_INDEX_RANGE_ERROR
索引がこのプログラムの範囲外。
6011 CWBRC_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。
6012 CWBRC_USER_EXIT_ERROR
ユーザー出口プログラムのエラー。
6013 CWBRC_COMMAND_FAILED
コマンドは失敗しました。
6014 CWBRC_PROGRAM_NOT_FOUND
プログラムが見付からないかアクセスできない。
6015 CWBRC_PROGRAM_ERROR
プログラム呼び出し時に、エラーが発生。
6016 CWBRC_COMMAND_TOO_LONG
コマンド・ストリングが長すぎます。
6099 CWBRC_UNEXPECTED_ERROR
予期しないエラー。

関連資料

369 ページの『System i Access for Windows のリモート・コマンド/分散プログラム呼び出し API』
PC アプリケーション・プログラマーは、System i Access for Windows リモート・コマンド/分散プログラム呼び出し API を使用することで、System i の機能にアクセスできます。ユーザー・プログラムとシステム・コマンドを、エミュレーション・セッションなしに呼び出します。コマンドとプログラムは単一の System i プログラムによって扱われるため、コマンドとプログラムの両方に対して、1 つのシステム・ジョブのみが開始されます。

セキュリティー API の戻りコード:

System i Access for Windows のセキュリティー API の戻りコードを示します。

6000 CWBSY_UNKNOWN_USERID
ユーザー ID が存在しない。
6002 CWBSY_WRONG_PASSWORD
指定の ユーザー ID のパスワードが正しくない。
6003 CWBSY_PASSWORD_EXPIRED
パスワードの有効期限切れ。
6004 CWBSY_INVALID_PASSWORD
パスワードの中の 1 つまたは複数の文字が無効であるか、パスワードが長すぎる。
6007 CWBSY_GENERAL_SECURITY_ERROR
一般セキュリティー・エラーが起こりました。ユーザー・プロファイルにパスワードがないか、
パスワード検証プログラムがパスワードにエラーを検出しました。
6009 CWBSY_INVALID_PROFILE
サーバーのユーザー・プロファイルが無効。
6011 CWBSY_USER_PROFILE_DISABLED
System i ユーザー・プロファイル (ユーザー ID) が
使用不可に設定されている。
6013 CWBSY_USER_CANCELLED
ユーザーがユーザー ID / パスワードのプロンプトを取り消した。

6015 CWBSY_INVALID_USERID
ユーザー ID 中の 1 つまたは複数の文字が無効であるか、
ユーザー ID が長すぎる。
6016 CWBSY_UNKNOWN_SYSTEM
指定のシステムが不明。
6019 CWBSY_TP_NOT_VALID
PC が System i セキュリティー・サーバーを検証できなかった。
システム上の IBM 提供のセキュリティー・サーバー・プログラムの
改ざんの可能性がある。
6022 CWBSY_NOT_LOGGED_ON
現在、指定のシステムにログオンしているユーザーはない。
6025 CWBSY_SYSTEM_NOT_CONFIGURED
セキュリティー・オブジェクトに指定のシステムは構成されていない。
6026 CWBSY_NOT_VERIFIED
オブジェクトに定義されているユーザー ID とパスワードは、まだ検証されていない。
cwbSY_VerifyUserIDPwd API を使用して検証しなければならない。
6255 CWBSY_INTERNAL_ERROR
内部エラー。IBM サービスに連絡してください。

以下の戻りコードは、パスワード変更 API 用です。

6257 CWBSY_PWD_TOO_LONG
新規パスワードの文字数が多すぎる。最大許容文字数は、
iSeries システム値 QPWDMAXLEN で定義されている。
6258 CWBSY_PWD_TOO_SHORT
新規パスワードの文字数が少なすぎる。最小許容文字数は、
iSeries システム値 QPWDMINLEN で定義されている。
QPWDMINLEN.
6259 CWBSY_PWD_REPEAT_CHARACTER
新規パスワードに 2 回以上使用されている文字がある。iSeries の構成
(システム値 QPWDLMTREP) は、パスワードに文字の繰り返し使用を
許可しない。
6260 CWBSY_PWD_ADJACENT_DIGITS
新規パスワードに 2 つの隣り合わせの数字がある。iSeries の構成
(システム値 QPWDLMTAJC) は、パスワードに隣り合わせの数字の使用を
許可しない。
6261 CWBSY_PWD_CONSECUTIVE_CHARS
新規パスワードに連続して繰り返し使われている文字がある。iSeries の構成
(システム値 QPWDLMTREP) は、パスワードに文字の連続繰り返し使用を
許可しない。
6262 CWBSY_PWD_PREVIOUSLY_USED
新規パスワードは、以前に使われたパスワードと同じ。iSeries の構成
(システム値 QPWDRQDDIF) は、以前に使われたすべてのパスワードと
異なるものを要求する。
6263 CWBSY_PWD_DISALLOWED_CHAR
新規パスワードは、導入システムが禁止している文字を使用している。iSeries
構成 (システム値 QPWDLMTCHR) は、新規パスワードで特定の文字が使われる
ことを制限している。
6264 CWBSY_PWD_NEED_NUMERIC
新規パスワードは、数字を含まなければならない。iSeries 構成 (システム値
QPWDRQDDGT) は、新規パスワードに 1 つまたは複数の数字を含むことを必要
としている。
6266 CWBSY_PWD_MATCHES_OLD
新規パスワードは旧パスワードと 1 つまたは複数の文字で一致する。
AS/400 構成 (システム値 QPWDPOSDIF) では、前のパスワードと
同じ位置に同じ文字を使用できない。
6267 CWBSY_PWD_NOT_ALLOWED
パスワードが拒否された。
6268 CWBSY_PWD_MATCHES_USERID
パスワードがユーザー ID と同じ。
6269 CWBSY_PWD_PRE_V3
旧パスワードは、異なる暗号化技法を使用した V3 以前のシステムで作られている。
AS/400 で、手動でパスワードを作り直す必要がある。
6270 CWBSY_LAST_INVALID_PASSWORD
次の無効入力は、ユーザー・プロファイルを使用禁止にする。

関連資料

46 ページの『System i Access for Windows の通信およびセキュリティー API』
『System i Access forWindows 通信およびセキュリティー』トピックでは、System i Access for Windows アプリケーション・プログラミング・インターフェース (API) の使用方法について説明します。

保守容易性 API の戻りコード:

System i Access for Windows の保守容易性 API の戻りコードを示します。

```
6000    CWBSV_INVALID_FILE_TYPE
          渡されたファイル・タイプが使用できない。
6001    CWBSV_INVALID_RECORD_TYPE
          渡されたレコード・タイプが使用できない。
6002    CWBSV_INVALID_EVENT_TYPE
          使用できないイベント・タイプが見付かった。
6003    CWBSV_NO_ERROR_MESSAGES
          エラー・ハンドルに関連するエラー・メッセージがない。
6004    CWBSV_ATTRIBUTE_NOT_SET
          属性が現行メッセージ内に設定されていない。
6005    CWBSV_INVALID_MSG_CLASS
          使用できないメッセージ・クラスが渡された。
6006    CWBSV_LOG_NOT_STARTED
          要求されたログを開始できない。
```

関連資料

392 ページの『System i Access for Windows の保守容易性 API』
System i Access for Windows の保守容易性アプリケーション・プログラミング・インターフェース (API) を使用すると、プログラム内のメッセージおよびイベントを保守ファイルに記録することができます。

システム・オブジェクト・アクセス API 戻りコード:

System i Access for Windows の SOA API 戻りコードを示します。

```
0    CWBSO_NO_ERROR
          エラーはなし。
1    CWBSO_ERROR_OCCURRED
          エラーが発生しました。
          詳細な情報を入手するには、エラー・ハンドルを使用してください。
2    CWBSO_LOW_MEMORY
          要求に対する十分なメモリーがない。
3    CWBSO_BAD_LISTTYPE
          リストのタイプに指定した値が無効。
4    CWBSO_BAD_HANDLE
          指定したハンドルが無効。
5    CWBSO_BAD_LIST_HANDLE
          指定したリスト・ハンドルが無効。
6    CWBSO_BAD_OBJ_HANDLE
          指定したオブジェクト・ハンドルが無効。
7    CWBSO_BAD_PARMOBJ_HANDLE
          指定したパラメーター・オブジェクト・ハンドルが無効。
8    CWBSO_BAD_ERR_HANDLE
          指定したエラー・ハンドルが無効。
9    CWBSO_BAD_LIST_POSITION
          リストに指定した位置は存在しない。
10    CWBSO_BAD_ACTION_ID
          指定されたアクション ID がこのリストのタイプに対して無効。
11    CWBSO_NOT_ALLOWED_NOW
          要求のアクションはこの時点で許可されない。
12    CWBSO_BAD_INCLUDE_ID
          指定のフィルター ID はこのリストに対して無効。
13    CWBSO_DISP_MSG_FAILED
          メッセージの表示の要求が失敗。
```

14 CWBSO_GET_MSG_FAILED
エラー・メッセージのテキストを検索できなかった。
15 CWBSO_BAD_SORT_ID
指定のソート ID がこのリストのタイプに対して無効。
16 CWBSO_INTERNAL_ERROR
内部処理エラーが発生。
17 CWBSO_NO_ERROR_MESSAGE
指定のエラー・ハンドルにエラー・メッセージがない。
18 CWBSO_BAD_ATTRIBUTE_ID
属性キーがこのオブジェクトに対して無効。
19 CWBSO_BAD_TITLE
指定の表題が無効。
20 CWBSO_BAD_FILTER_VALUE
指定したフィルター値が無効。
21 CWBSO_BAD_PROFILE_NAME
指定したプロファイル名が無効。
22 CWBSO_DISPLAY_FAILED
ウィンドウが作成できない。
23 CWBSO_SORT_NOT_ALLOWED
このリストのタイプに対するソートは許可されていない。
24 CWBSO_CANNOT_CHANGE_ATTR
属性は現時点では変更できない。
25 CWBSO_CANNOT_READ_PROFILE
指定のプロファイル・ファイルから読み取りができない。
26 CWBSO_CANNOT_WRITE_PROFILE
指定のプロファイル・ファイルに書き込みができない。
27 CWBSO_BAD_SYSTEM_NAME
指定されたシステム名が有効なシステム名ではない。
28 CWBSO_SYSTEM_NAME_DEFAULTED
リストの "CWBSO_CreateListHandle" 呼び出しでシステム名が指定されなかった。
29 CWBSO_BAD_FILTER_ID
指定のフィルター ID がこのリストのタイプに対して無効です。

関連資料

451 ページの『System i Access for Windows システム・オブジェクト・アクセス (SOA) API』
システム・オブジェクト・アクセスを使用することで、グラフィカル・ユーザー・インターフェースを介して、システム・オブジェクトの表示および操作を行うことができます。

461 ページの『システム・オブジェクト・アクセスのエラー』
System i Access for Windows 関数では、戻りコードを使用してエラー条件を報告するシステム・オブジェクト・アクセス API を、すべてサポートしています。

System i Access for Windows 管理 API

これらの API には、PC にインストールされている System i Access for Windows のコードに関する情報にアクセスするための機能が備わっています。

管理 API を使用して、以下のことが判別できます。

- System i Access for Windows のバージョンとサービス・レベル
- 個々のフィーチャーのインストール状況
- System i ナビゲーターのプラグインのインストール状況

System i Access for Windows 管理 API に必要なファイル

ヘッダー・ファイル	インポート・ライブラリー	ダイナミック・リンク・ライブラリー
cwbad.h	cwbapi.lib	cwbad.dll

Programmer's Toolkit:

System i Access for Windows Programmer's Toolkit には、管理 API 資料、cwbad.h ヘッダー・ファイルへのアクセス、およびプログラム例へのリンクが用意されています。この情報にアクセスするには、Programmer's Toolkit をオープンして、「**クライアント情報**」 → 「**C/C++ API**」と選択します。

System i Access for Windows 管理 API のトピック

注: コード例を使用することで、 633 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』の条件に同意します。

関連資料

23 ページの『管理 API の戻りコード』
System i Access for Windows の管理の戻りコードです。

7 ページの『接続 API 用の System i 名の形式』
パラメーターとして System i 名を取得する API では、3 つの異なる形式の名前を使用できます。

8 ページの『OEM、ANSI、およびユニコードの考慮事項』
System i Access for Windows の、ストリング・パラメーターを受け入れる C/C++ API の大部分は、OEM、ANSI、Unicode の 3 つのうち、いずれかの形式になっています。

管理 API のリスト

以下は、System i Access for Windows 管理で使用する API です。

cwbAD_GetClientVersion:
目的

現在 PC にインストールされている System i Access for Windows 製品のバージョンを調べます。

構文

```
unsigned int CWB_ENTRY cwbAD_GetClientVersion(
                            unsigned long      *version
                            unsigned long      *release
                            unsigned long      *modificationLevel);
```

パラメーター

unsigned long *version - output
System i Access for Windows 製品のバージョン・レベルが戻されるバッファーを指すポインター。

unsigned long *release - output
System i Access for Windows 製品のリリース・レベルが戻されるバッファーを指すポインター。

unsigned long *modificationLevel - output
System i Access for Windows 製品のモディフィケーション・レベルが戻されるバッファーを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
1 つまたは複数のポインター・パラメーターが NULL です。

使用法

戻りコードが CWB_OK ではない場合は、バージョン、リリース、およびモディフィケーション・レベルの値は無意味です。

cwbAD_GetProductFixLevel:
目的

System i Access for Windows の現在の修正レベルを戻します。

構文

```
unsigned int CWB_ENTRY cwbAD_GetProductFixLevel(
                            char              *szBuffer
                            unsigned long     *ulBufLen);
```

パラメーター

char *szBuffer - output
プロダクトの修正レベル・ストリングが書き込まれるバッファー。

unsigned long * ulBufLen - input/output
szBuffer のサイズ。NULL 終了文字のスペースを含みます。出力時には、NULL 終了文字のスペースと共に、修正レベル・ストリングの長さを含みます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
バッファー・オーバーフロー。必要な長さを ulBufLen に戻します。

CWB_INVALID_POINTER
無効なポインター。

使用法

System i Access for Windows 製品の修正レベルを戻します。修正が適用されていない場合は、空ストリングが戻されます。

cwbAD_IsComponentInstalled:

V6R1 以降、System i Access for Windows の構成要素はフィーチャーと呼ばれるようになりました。この API は、製品のインストール済みフィーチャーを識別するために使用します。

目的

System i Access for Windows の特定のフィーチャーがインストールされているかどうかを示します。

構文

```
unsigned long CWB_ENTRY cwbAD_IsComponentInstalled(
                            unsigned long      ulComponentID,
                            cwb_Boolean       *bIndicator);
```

パラメーター

unsigned long ulComponentID - input
以下のいずれかの構成要素 ID に設定されている必要があります。

CWBAD_COMP_SSL
Secure Sockets Layer (セキュア・ソケット・レイヤー)。

CWBAD_COMP_SSL_128_BIT
128 ビットの Secure Sockets Layer (セキュア・ソケット・レイヤー)。

注: この定数は、 CWBAD_COMP_SSL と同じになるように定義します。

CWBAD_COMP_SSL_56_BIT
56 ビットの Secure Sockets Layer (セキュア・ソケット・レイヤー)。

注: この定数は、 CWBAD_COMP_SSL と同じになるように定義します。

CWBAD_COMP_SSL_40_BIT
40 ビットの Secure Sockets Layer (セキュア・ソケット・レイヤー)。

注: この定数は、 CWBAD_COMP_SSL と同じになるように定義します。

CWB_COMP_BASESUPPORT
System i Access for Windows に必要なプログラム

CWBAD_COMP_OPTIONAL_COMPS
System i Access for Windows のオプション・フィーチャー

CWBAD_COMP_DIRECTORYUPDATE
ディレクトリー更新

CWBAD_COMP_IRC
着信リモート・コマンド

CWBAD_COMP_OUG
ユーザーズ・ガイド

CWBAD_COMP_OPNAV
System i ナビゲーター

CWBAD_COMP_DATA_ACCESS
データ・アクセス

CWBAD_COMP_DATA_TRANSFER
データ転送

CWBAD_COMP_DT_BASESUPPORT
データ転送の基本サポート

CWBAD_COMP_DT_EXCEL_ADDIN
データ転送の Excel アドイン

CWBAD_COMP_DT_WK4SUPPORT
データ転送の WK4 ファイル・サポート

CWBAD_COMP_ODBC
ODBC

CWBAD_COMP_OLEDB
OLE DB Provider

CWBAD_COMP_MP
.NET Data Provider

CWBAD_COMP_AFP_VIEWER
AFP™ ワークベンチ・ビューアー

CWBAD_COMP_JAVA_TOOLBOX
Java Toolbox

CWBAD_COMP_PC5250
PC5250 ディスプレイおよびプリンター・エミュレーター

PC5250 ディスプレイおよびプリンター・エミュレーター・サブコンポーネント

- CWBAD_COMP_PC5250_BASE_KOREAN
- CWBAD_COMP_PC5250_PDFPDT_KOREAN
- CWBAD_COMP_PC5250_BASE_SIMPCHIN
- CWBAD_COMP_PC5250_PDFPDT_SIMPCHIN
- CWBAD_COMP_PC5250_BASE_TRADCHIN
- CWBAD_COMP_PC5250_PDFPDT_TRADCHIN
- CWBAD_COMP_PC5250_BASE_STANDARD
- CWBAD_COMP_PC5250_PDFPDT_STANDARD
- CWBAD_COMP_PC5250_FONT_ARABIC
- CWBAD_COMP_PC5250_FONT_BALTIC
- CWBAD_COMP_PC5250_FONT_LATIN2
- CWBAD_COMP_PC5250_FONT_CYRILLIC
- CWBAD_COMP_PC5250_FONT_GREEK
- CWBAD_COMP_PC5250_FONT_HEBREW
- CWBAD_COMP_PC5250_FONT_LAO
- CWBAD_COMP_PC5250_FONT_THAI
- CWBAD_COMP_PC5250_FONT_TURKISH
- CWBAD_COMP_PC5250_FONT_VIET
- CWBAD_COMP_PC5250_FONT_HINDI

CWBAD_COMP_PRINTERDRIVERS
プリンター・ドライバー

CWBAD_COMP_AFP_DRIVER
AFP プリンター・ドライバー

CWBAD_COMP_SCS_DRIVER
SCS プリンター・ドライバー

CWBAD_COMP_OP_CONSOLE
オペレーション・コンソール

CWBAD_COMP_TOOLKIT
Programmer's Toolkit

CWBAD_COMP_TOOLKIT_BASE
ヘッダー、ライブラリー、および資料

CWBAD_COMP_TOOLKIT_VBW
Visual Basic ウィザード

CWBAD_COMP_EZSETUP
簡単セットアップ

CWBAD_COMP_TOOLKIT_JAVA_TOOLS
Programmer's Toolkit Tools for Java

CWBAD_COMP_SCREEN_CUSTOMIZER_ENABLER
Screen Customizer Enabler

CWBAD_COMP_OPNAV_BASESUPPORT
System i ナビゲーターの基本サポート

CWBAD_COMP_OPNAV_BASE_OPS
System i ナビゲーターの基本操作

CWBAD_COMP_OPNAV_JOB_MGMT
System i ナビゲーターのジョブ管理

CWBAD_COMP_OPNAV_SYS_CFG
System i ナビゲーターのシステム構成

CWBAD_COMP_OPNAV_NETWORK
System i ナビゲーターのネットワーク

CWBAD_COMP_OPNAV_SECURITY
System i ナビゲーターのセキュリティー

CWBAD_COMP_OPNAV_USERS_GROUPS
System i ナビゲーターのユーザーとグループ

CWBAD_COMP_OPNAV_DATABASE
System i ナビゲーターのデータベース

CWBAD_COMP_OPNAV_BACKUP
System i ナビゲーターのバックアップ

CWBAD_COMP_OPNAV_APP_DEV
System i ナビゲーターのアプリケーション開発

CWBAD_COMP_OPNAV_APP_ADMIN
System i ナビゲーターのアプリケーション管理

CWBAD_COMP_OPNAV_FILE_SYSTEMS
System i ナビゲーターのファイル・システム

CWBAD_COMP_OPNAV_MGMT_CENTRAL
System i ナビゲーターのマネージメント・セントラル

CWBAD_COMP_OPNAV_MGMT_COMMANDS
System i ナビゲーターのマネージメント・セントラル - コマンド

CWBAD_COMP_OPNAV_MGMT_PACK_PROD
System i ナビゲーターのマネージメント・セントラル - パッケージおよび製品

CWBAD_COMP_OPNAV_MGMT_MONITORS
System i ナビゲーターのマネージメント・セントラル - モニター

CWBAD_COMP_OPNAV_LOGICAL_SYS
System i ナビゲーターの論理システム

CWBAD_COMP_OPNAV_ADV_FUNC_PRES
System i ナビゲーターの拡張機能表示™

cwb_Boolean *bIndicator - output
構成要素がインストールされている場合は、CWB_TRUE が入っている。構成要素がインストールされていない場合は、CWB_FALSE が戻される。エラーが生じた場合は、なにも設定されない。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
無効なポインター。

CWB_INVALID_COMPONENT_ID
このリリースでは、構成要素 ID が無効。

cwbAD_IsOpNavPluginInstalled:

目的

System i ナビゲーターの特定のプラグインがインストールされているかどうかを示します。

構文

```
unsigned long CWB_ENTRY cwbAD_IsOpNavPluginInstalled(
                            const char        *szPluginName,
                            cwb_Boolean       *bIndicator);
```

パラメーター

const char* szPluginName - input
プラグインの名前が含まれている、NULL 文字で終わるストリングを指すポインター。

cwb_Boolean *bIndicator - output
プラグインがインストールされている場合は、CWB_TRUE が入っている。構成要素がインストールされていない場合は、CWB_FALSE が戻される。エラーが生じた場合は、なにも設定されない。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインター・パラメーターのいずれかが NULL です。

使用法

戻り値が CWB_OK ではない場合は、bIndicator の値は無意味。

例: 管理 API

アプリケーションにおける System i Access for Windows 管理 API の使用例です。

この例では、API を使用して、以下の情報の取得と表示を行います。

- 現行の System i Access for Windows のバージョン/リリース/モディフィケーションのレベル
- 現行のサービス・パック (修正) レベル
- 現在 PC にインストールされているフィーチャー

その後で、ユーザーは、System i ナビゲーターのプラグインの名前を入力することができ、また、そのプラグインがインストールされているかどうかが通知されます。

使用上の注意

cwbad.h * を組み込む。

cwbapi.lib とリンクする。

例

```
#include <windows.h>
#include <stdio.h>

#include "cwbad.h"


/*
 *  This is the highest numbered component ID known (it is
 *  the ID of the last component defined in cwbad.h).
 */
#define LAST_COMPID_WE_KNOW_ABOUT       (CWBAD_COMP_SSL_40_BIT)


/*
 *  Array of component names, taken from comments for component IDs
 *  in cwbad.h, so human-readable component descriptions are displayed .
 *  In the compDescr array, the component ID for a component must match
 *  the index in the array of that component's description.
 *
 *  For a blank or unknown component name, a string is provided to display
 *  an indication that the component ID is unknown, and what that ID is.
 */
static char* compDescr[ LAST_COMPID_WE_KNOW_ABOUT + 1 ] = {
                              "",                               // #0 is not used
                              "Required programs",
                              "Optional Features",
                              "Directory Update",
                              "Incoming Remote Command",
                              "", // not used,
                              "Online User's Guide",
                              "System i Navigator",
                              "Data Access",
                              "Data Transfer",
                              "Data Transfer Base Support",
```

```
                    "Data Transfer Excel Add-in",
                    "Data Transfer WK4 file support",
                    "ODBC",
                    "OLE DB Provider",
                    "AFP Workbench Viewer",
                    "System i Java Toolbox",
                    "5250 Display and Printer Emulator",
                    "Printer Drivers",
                    "AFP printer driver",
                    "SCS printer driver",
                    "System i Operations Console",
                    "System i Access Programmer's Toolkit",
                    "Headers, Libraries, and Documentation",
                    "", // not used,
"", // not used,
                    "Java Toolkit",
                    "Screen customizer",
                    ".NET Data Provider",
                    "",                         //-------------#29
                    "", "", "", "", "",         //             #30-34
                    "", "", "", "", "",         //             #35-39
                    "", "", "", "", "",         //             #40-44
                    "", "", "", "", "",         //             #45-49
                    "", "", "", "", "",         //   not       #50-54
                    "", "", "", "", "",         //             #55-59
                    "", "", "", "", "",         //             #60-64
                    "", "", "", "", "",         //             #65-69
                    "", "", "", "", "",         //   used      #70-74
                    "", "", "", "", "",         //             #75-79
                    "", "", "", "", "",         //             #80-84
                    "", "", "", "", "",         //             #85-89
                    "", "", "", "", "",         //             #90-94
                    "", "", "", "", "",         //------------ #95-99
                    "System i Navigator Base Support",
                    "System i Navigator Basic Operations",
                    "System i Navigator Job Management",
                    "System i Navigator System Configuration",
                    "System i Navigator Networks",
                    "System i Navigator Security",
                    "System i Navigator Users and Groups",
                    "System i Navigator Database",
                    "",                         // not used      #108
                    "System i Navigator Backup",
                    "System i Navigator Application Development",
                    "System i Navigator Application Administrat",
                    "System i Navigator File Systems",
                    "System i Navigator Management Central",
                    "System i Navigator Management Central - Commands",
                    "System i Navigator Management Central - Packages and Products",
                    "System i Navigator Logical Systems",
                    "System i Navigator Advanced Function Presentation",
                    "","                    //-------------#118-119
                    "", "", "", "", "",     //  not        #120-124
                    "", "", "", "", "",     //             #125-129
                    "", "", "", "", "",     //             #130-134
                    "", "", "", "", "",     //  used       #135-139
                    "", "", "", "", "",     //             #140-144
                    "", "", "", "", "",     //------------ #145-149
                    "PC5250: BASE_KOREAN",
                    "PC5250: PDFPDT_KOREAN",
                    "PC5250: BASE_SIMPCHIN",
                    "PC5250: PDFPDT_SIMPCHIN",
                    "PC5250: BASE_TRADCHIN",
                    "PC5250: PDFPDT_TRADCHIN",
                    "PC5250: BASE_STANDARD",
                    "PC5250: PDFPDT_STANDARD",
                    "PC5250: FONT_ARABIC",
```

```
                                "PC5250: FONT_BALTIC",
                                "PC5250: FONT_LATIN2",
                                "PC5250: FONT_CYRILLIC",
                                "PC5250: FONT_GREEK",
                                "PC5250: FONT_HEBREW",
                                "PC5250: FONT_LAO",
                                "PC5250: FONT_THAI",
                                "PC5250: FONT_TURKISH",
                                "PC5250: FONT_VIET",
                                "PC5250: FONT_HINDI",
                                "",                     //------------ #169
                                "", "", "", "", "",     //             #170-174
                                "", "", "", "", "",     //  not        #175-179
                                "", "", "", "", "",     //             #180-184
                                "", "", "", "", "",     //  used       #185-189
                                "", "", "", "", "",     //             #190-194
                                "", "", "", "", "",     //------------ #195-199
                                "Secure Sockets Layer (SSL)" } ;       // last one defined
static char  unknownComp[] = "unknown, ID=          ";
static char* pInsertID     = &amp;( unknownComp[12] );          // insert ID here!



/*****************************************************************************
 *  Show the System i Access for Windows Version/Release/Modification level
 *****************************************************************************/
void showCA_VRM()
{
   ULONG  caVer, caRel, caMod;
   UINT         rc;
   char   fixlevelBuf[ MAX_PATH ];
   ULONG  fixlevelBufLen = sizeof( fixlevelBuf );

   printf( "System i Access level installed:\n\n" );

   rc = cwbAD_GetClientVersion( &caVer;, &caRel;, &caMod; );
   if ( rc != CWB_OK )
   {
      printf( "   Error %u occurred when calling cwbAD_GetClientVersion()¥n¥n",
         rc );
   }
   else
   {
      printf( "   Version %lu, Release %lu, Modification %lu¥n¥n",
         caVer, caRel, caMod );

      printf( "System i Access service pack level installed:\n\n" );
      rc = cwbAD_GetProductFixLevel( fixlevelBuf, &fixlevelBufLen; );
      if ( rc != CWB_OK )
      {
         printf( "   Error %u occurred when calling "
            "cwbAD_GetProduceFixLevel()¥n¥n", rc );
      }
      else if ( fixlevelBuf[0] == '¥0' )  // empty, no service packs applied
      {
         printf( "   None¥n¥n" );
      }
      else
      {
         printf( "   %s¥n¥n", fixlevelBuf );
      }
   }
}
```

```
/*****************************************************************************
 *  Call System i Access for Windows API to determine if the component is installed,
 *  and pass back:
 *      NULL if the component is not installed or an error occurs,
 *             OR
 *      A string indicating the component name is unknown if the
 *      component ID is higher than we know about OR the component
 *      description is blank,
 *             OR
 *      The human-readable component description if known.
 *****************************************************************************/
char* isCompInstalled( ULONG compID )
{
   cwb_Boolean bIsInstalled;
   char*       pCompName;

   UINT rc = cwbAD_IsComponentInstalled( compID, &bIsInstalled; );

   /*
    *  Case 1:  Error OR component not installed, return NULL to
    *           indicate not installed.
    */
   if ( ( rc != CWB_OK ) || ( bIsInstalled == CWB_FALSE ) )
   {
      pCompName = NULL;
   }

   /*
    *  Case 2:  Component IS installed, but its name is not known,
    *           return component name unknown string.
    */
   else if ( ( compID > LAST_COMPID_WE_KNOW_ABOUT ) ||
             ( compDescr[ compID ][ 0 ] == '¥0' ) )
   {
      pCompName = unknownComp;
      sprintf( pInsertID, "%lu", compID );
   }

   /*
    *  Case 3:  Component IS installed, and a name is known, return it
    */
   else
   {
      pCompName = compDescr[ compID ];
   }

   return pCompName;
}



/*****************************************************************************
 *  List the System i Access for Windows features that currently are installed.
 *****************************************************************************/
void showCA_CompInstalled()
{
   ULONG compID;
   char* compName;

   printf( "System i Access features installed:\n\n" );

   /*
    *  Try all known features, plus a bunch more in case some
    *  have been added (via service pack).
    */
   for ( compID = 0;
         compID &lt; (LAST_COMPID_WE_KNOW_ABOUT + 50);
```

```
         compID++ )
   {
      compName = isCompInstalled( compID );
      if ( compName != NULL )
      {
         printf( "   %s¥n", compName );
      }
   }

   printf( "¥n" );
}



/*****************************************************************************
 *  MAIN PROGRAM BODY
 *****************************************************************************/
void main(void)
{
   UINT         rc;
   char         pluginName[ MAX_PATH ];
   cwb_Boolean  bPluginInstalled;

   printf( "=======================================¥n");
   printf( "System i Access What's Installed Reporter\n" );
   printf( "=======================================¥n¥n");

   showCA_VRM();
   showCA_CompInstalled();

   /*
    *  Allow user to ask by name what plug-ins are installed.
    */
   while ( TRUE )  /* REMINDER: requires a break to exit the loop! */
   {
      printf( "Enter plug-in to check for, or DONE to quit:¥n" );
      gets( pluginName );
      if ( stricmp( pluginName, "DONE" ) == 0 )
      {
         break;    /* exit from the while loop, DONE at user's request */
      }

      rc = cwbAD_IsOpNavPluginInstalled( pluginName, &bPluginInstalled; );
      if ( rc == CWB_OK )
      {
         if ( bPluginInstalled == CWB_TRUE )
         {
            printf( "The plug-in '%s' is installed.¥n¥n", pluginName );
         }
         else
         {
            printf( "The plug-in '%s' is NOT installed.¥n¥n", pluginName );
         }
      }
      else
      {
         printf(
           "Error %u occurred when calling cwbAD_IsOpNavPluginInstalled.¥n¥n",
           rc );
      }
   }  // end while (TRUE)

   printf( "¥nEnd of program.¥n¥n" );
}
```

System i Access for Windows の通信およびセキュリティー API

『System i Access forWindows 通信およびセキュリティー』トピックでは、System i Access for Windows アプリケーション・プログラミング・インターフェース (API) の使用方法について説明します。

これらの API は、以下のことに使用することができます。

- System i **システム・オブジェクト**を取得、使用、および削除する。システム・オブジェクトは、System i Access for Windows のさまざまな API で必要とされます。システム・オブジェクトは、System i のセキュリティー・オブジェクト (ユーザー ID、パスワード、サインオン日時など) に対する接続や検証に関する情報を保有しています。
- System i Access for Windows の機能を使用する際には、**システム・リスト**で構成されている、環境や接続に関する情報を入手します。システム・リストとは、現在、構成されているすべての環境のリストであり、これらの環境内でのシステムのリストです。システム・リストは「ユーザーごとに」保管および管理され、他のユーザーが使用することはできません。

  注: ユーザーが新規システムを明示的に構成して、それをシステム・リストに追加する必要はありません。新規システムは、ユーザーが新規システムに接続したときに、自動的にシステム・リストに追加されます。

System i Access for Windows の通信およびセキュリティー API に必要なファイル

ヘッダー・ファイル		インポート・ライブラリー	ダイナミック・リンク・ライブラリー
システム・オブジェクト API	**システム・リスト API**	cwbapi.lib	cwbco.dll
cwbcosys.h	cwbco.h		

Programmer's Toolkit:

Programmer's Toolkit では、通信およびセキュリティーに関する資料、cwbco.h ヘッダー・ファイルおよび cwbcosys.h ヘッダー・ファイルへのアクセス、およびプログラム例へのリンクが提供されます。この情報にアクセスするには、Programmer's Toolkit をオープンして、「**通信およびセキュリティー**」 → 「**C/C++ API**」と選択します。

System i Access for Windows の通信およびセキュリティーに関するトピック

注: コード例を使用することで、 633 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』の条件に同意します。

関連資料

23 ページの『通信 API の戻りコード』
System i Access for Windows 通信 API の戻りコードを示します。

31 ページの『セキュリティー API の戻りコード』
System i Access for Windows のセキュリティー API の戻りコードを示します。

17 ページの『System i Access for Windows のグローバル戻りコード』
System i Access for Windows のグローバル戻りコードを紹介します。

7 ページの『接続 API 用の System i 名の形式』
パラメーターとして System i 名を取得する API では、3 つの異なる形式の名前を使用できます。

8 ページの『OEM、ANSI、およびユニコードの考慮事項』
System i Access for Windows の、ストリング・パラメーターを受け入れる C/C++ API の大部分は、OEM、ANSI、Unicode の 3 つのうち、いずれかの形式になっています。

システム・オブジェクトの属性

System i プラットフォームにおける**システム・オブジェクトの属性**は、システム・オブジェクトが表すシステムへのサインオンおよび通信の動作に影響します。

cwbCO_Signon または cwbCO_Connect のいずれかを使用してサインオンが正常終了した後では、ほとんどの属性は変更不可になります。正常なサインオンの後で変更可能な属性は、ウィンドウ・ハンドルと接続タイムアウトの 2 つだけです。正常なサインオンの後で、その他の属性値を変更する API を呼び出すと、戻りコード CWB_INV_AFTER_SIGNON で失敗します。

一部の値および値を変更する機能を、**ポリシー**によって制御することができます。ポリシーとは、システム管理者がセットアップしてデフォルトの属性値を指示し、属性の変更を禁止できるようにする制御情報です。『**システム・オブジェクトの属性のリスト**』のトピックに指定されているデフォルト値 (以下にリンクする) は、以下の条件の下で使用されます。

- ポリシーが異なる別の値を指定または示唆しない場合。
- そのような属性の値が、システム・リストに指定されたシステムで明示的に構成されていない場合。

属性のデフォルト値がポリシーによって設定できる場合には、このことが示されます。属性の値の変更がポリシーによって禁止できる場合は、以下のようになります。

- 属性が変更可能かどうかをチェックするための API が用意される。
- そのポリシーのために設定が失敗した場合は、属性の設定方式によって特定の戻りコードが提供される。

関連資料

82 ページの『cwbCO_Signon』
System i Access for Windows の cwbCO_Signon コマンドを使用します。

54 ページの『cwbCO_Connect』
System i Access for Windows の cwbCO_Connect コマンドを使用します。

システム・オブジェクトの属性のリスト:

以下のリストには、System i におけるシステム・オブジェクト属性の記述、要件、および考慮事項が掲載されています。

それぞれの属性には、以下のものが示されています。

- 属性を取得して、設定するために使用可能な API
- システム・オブジェクトが作成されるときのデフォルト値

注: 属性の設定値は、設定対象になっているシステム・オブジェクトに対してのみ適用されます。たとえシステム名が同じであっても、その他のいかなるシステム・オブジェクトにも適用されません。

System i 名:
システム・オブジェクトのこのインスタンスによって定義されている、通信相手のシステム。これは、cwbCO_CreateSystem または cwbCO_CreateSystemLike が呼び出された時点でのみ、設定することができます。特定のユーザー ID のセキュリティー情報を検証する際に、システム名が固有の識別コードとして使用されることに注意してください。 2 つの異なるシステム・オブジェクトに、同じ物理装置を表す異なるシステム名が含まれる場合、2 つのシステム・オブジェクトに対す

るユーザー ID およびパスワードの検証が、個別に必要となります。例えば、システム名 ″SYS1″ および ″SYS1.ACME.COM″ が同じ System i 装置を表す場合に、これが適用されます。この結果、プロンプトが二重になり、接続時に別々のデフォルトのユーザー ID を使用することになります。

cwbCO_GetSystemName を使用して取得。

デフォルト:

システム・オブジェクトが作成されるときに明示的に設定されるため、デフォルト値はありません。

説明 System i の構成済み接続の記述。

System i ナビゲーターを使用して設定します。

cwbCO_GetDescription を使用して検索します。

記述は各システム・オブジェクトとともに保管され、そのシステム・オブジェクトのために変更されることはありません。 System i ナビゲーターを使用して記述を変更しても、変更前に存在していたそのシステムのシステム・オブジェクトは変更されません。新規のシステム・オブジェクトのみに新規の記述が含まれます。

デフォルト:

ブランク。これはポリシーで指定変更可能です。

ユーザー ID:

システムで使用される System i ユーザー ID。

cwbCO_GetUserIDEx を使用して取得。

cwbCO_SetUserIDEx を使用して設定。

デフォルト:

システム・オブジェクトで指定されているシステムに最初に接続したときに、以下のプロンプトが出されます。

- デフォルトのユーザー ID の指定。
- デフォルトのユーザー ID を Windows のユーザー ID と同一にすることの指定。
- デフォルト値を使用しないこと。

後で接続しようとした場合、使用されるデフォルトのユーザー ID は、最初に接続しようとしたときに出されたプロンプトで、どのオプションを選択したかによって決まります。

パスワード

システムへのサインオン時に使用する System i パスワード。

cwbCO_SetPassword を使用して設定。

デフォルト:

システム・オブジェクトで設定されたユーザー ID が、システム・オブジェクトで指定されたシステムにサインオンしたことがない場合には、ブランク (パスワード設定なし)。以前に、システム・オブジェクトで指定されたシステムへのサインオンまたは接続が正常に行われている場合は、次回のサインオンまたは接続で、そのパスワードを使用できます。cwbCO_SetPassword() API を介してパスワードが入力される場合、システムは System i Access for Windows の揮発性パスワード・キャッシュにパスワードを入れなくなりました。以前、このパスワードは揮発性 (つまり、セッション) パスワード・キャッシュに入っていました。

デフォルトのユーザー・モード

デフォルトのユーザー ID をどこから取得するか、それを使用するかどうかも含めた、デフォルトのユーザー ID に関連した動作を制御します。設定されていない場合 (値が CWBCO_DEFAULT_USER_MODE_NOT_SET) は、サインオンしようとした時点での希望する動作を選択するようにプロンプトが出されます。

cwbCO_GetDefaultUserMode を使用して取得。

cwbCO_SetDefaultUserMode を使用して設定。

cwbCO_CanModifyDefaultUserMode を使用して、変更の制限のチェック。

デフォルト:

CWBCO_DEFAULT_USER_MODE_NOT_SET

注: デフォルト値はポリシーで指定変更可能です。

プロンプト・モード

System i Access for Windows のユーザー ID およびパスワードを求めるプロンプトを制御します。指定できる値と関連する動作については、cwbCO_SetPromptMode の宣言の注釈を参照してください。

cwbCO_GetPromptMode を使用して取得。

cwbCO_SetPromptMode を使用して設定。

デフォルト:

CWBCO_PROMPT_IF_NECESSARY

ウィンドウ・ハンドル

呼び出し側アプリケーションのウィンドウ・ハンドル。これが設定されている場合には、System i Access for Windows が発行する、System i のサインオン関連のプロンプトは、いずれもウィンドウ・ハンドルを使用し、関連するウィンドウに対してモーダルになります。このことは、そのハンドルがシステム・オブジェクトに関連している場合、メインのアプリケーション・ウィンドウの下にプロンプトが隠れることは決してないということを意味します。ウィンドウ・ハンドルがなにも設定されていない場合は、プロンプトが存在してもメインのアプリケーション・ウィンドウの下に隠れてしまう場合があります。

cwbCO_GetWindowHandle を使用して取得。

cwbCO_SetWindowHandle を使用して設定。

デフォルト:

NULL (設定しない)

検証モード

ユーザー ID とパスワードを検証する際に、この検証を実際に実行するために System i 通信を行うかどうかを指定します。指定できる値と関連する動作については、cwbCO_SetValidateMode および cwbCO_GetValidateMode の宣言の注釈を参照してください。

cwbCO_GetValidateMode を使用して取得。

cwbCO_SetValidateMode を使用して設定。

デフォルト:

CWBCO_VALIDATE_IF_NECESSARY

セキュア・ソケットの使用法

システムを認証し、送受信されるデータを暗号化するために、System i Access for Windows ソケ

ットを使用するかどうかを指定します。セキュア・ソケットが使用できないようなケース (例えば、セキュア・ソケットのソフトウェア・サポートが PC にインストールされていない場合) があります。その場合は、セキュア・ソケットを使用するアプリケーションまたはユーザー要求が、cwbCO_UseSecureSockets API が呼び出された時点か、または接続時のいずれかで失敗することがあります。そのような失敗が起こらない場合は、セキュア・ソケットが使用され、cwbCO_IsSecureSockets は CWB_TRUE を戻します。

cwbCO_IsSecureSockets を使用して取得。

cwbCO_UseSecureSockets を使用して設定。

cwbCO_CanModifyUseSecureSockets を使用して、変更の制限のチェック。

デフォルト:

System i で構成されたものが何であれ、このシステムのシステム・リストが使用されます。このシステムに System i 構成が存在しない場合、または System i Access のデフォルト値を使用するように構成で指定されている場合、セキュア・ソケットは使用されません (CWB_FALSE)。

注: デフォルト値はポリシーで指定変更可能です。

ポート・ルックアップ・モード

i5/OS® ホスト・サービス用のリモート・ポートの検索方法を指定します。ローカル (PC 上) で検索するか、i5/OS ホストで検索するか、あるいは単に指定されたサービスのデフォルト (「標準」) のポートを使用するかを指定します。ローカルのルックアップが選択された場合、PC の SERVICES ファイルにあるルックアップの標準 TCP/IP 方式が使用されます。サーバー・ルックアップが指定されている場合は、System i マッパーへの接続が行われ、System i サービス・テーブルからのルックアップによって、ポート番号が検索されます。ローカルもしくはサーバーのルックアップ方式のいずれかが失敗した場合には、サービスへの接続は失敗します。詳細および指定できる値については、cwbCO_SetPortLookupMode の API 宣言を参照してください。

cwbCO_GetPortLookupMode を使用して取得。

cwbCO_SetPortLookupMode を使用して設定。

cwbCO_CanModifyPortLookupMode を使用して、変更の制限のチェック。

デフォルト:

System i リストでこのシステム用に構成されたものが、すべて使用されます。このシステムに System i 構成が存在しない場合、デフォルト値は CWBCO_PORT_LOOKUP_SERVER になります。

注: デフォルト値はポリシーで指定変更可能です。

パーシスタンス・モード

cwbCO_Connect への呼び出しが正常に終了した後に、このシステム・オブジェクトで指定されたシステムを (まだリストにない場合に) System i リストに追加するかどうかを指定します。詳細および指定できる値については、cwbCO_SetPersistenceMode を参照してください。

cwbCO_GetPersistenceMode を使用して取得。

cwbCO_SetPersistenceMode を使用して設定。

cwbCO_CanModifyPersistenceMode を使用して、変更の制限のチェック。

デフォルト:

CWBCO_MAY_MAKE_PERSISTENT

注: デフォルト値はポリシーで指定変更可能です。

接続タイムアウト

接続試行が完了するまでの System i Access for Windows の待機時間を指定します。この設定値は、TCP/IP 通信スタックが試行を放棄するまで待機する時間に影響しません。 TCP/IP 通信スタックは、System i Access の接続タイムアウトの有効期限が切れる前にタイムアウトになる可能性があります。詳細および指定できる値については、cwbCO_SetConnectTimeout を参照してください。この値はシステム・オブジェクトに合わせていつでも変更することができます。

cwbCO_GetConnectTimeout を使用して取得

cwbCO_SetConnectTimeout を使用して設定

デフォルト:

CWBCO_CONNECT_TIMEOUT_DEFAULT

注: デフォルト値はポリシーで指定変更可能です。

通信およびセキュリティー: 作成および削除の API

これらの API は、System i オブジェクトの作成および削除に使用されます。

cwbCO_CreateSystem:

System i Access for Windows の cwbCO_CreateSystem コマンドを使用します。

目的

新規のシステム・オブジェクトを作成し、そのシステム・オブジェクトに後続の呼び出しで使用できるハンドルを戻します。システム・オブジェクトは、設定し、検索することができる多くの属性を持っています。詳しくは、 47 ページの『システム・オブジェクトの属性』を参照してください。

構文

```
UINT CWB_ENTRY cwbCO_CreateSystem(
                            LPCSTR              systemName,
                            cwbCO_SysHandle    *system);
```

パラメーター

LPCSTR systemName - input

ヌル終了する System i 名を含んでいるバッファーを指すポインター。ホスト名、または System i の IP アドレス (小数点付き 10 進数) 自体を指定することができます。長さがゼロであってはならず、またブランクを含んでいてはなりません。指定された名前が有効な i5/OS ホスト名または IP アドレス・ストリング (「nnn.nnn.nnn.nnn」の形式) ではない場合、接続やセキュリティー検証の試みはすべて失敗します。

cwbCO_SysHandle *system - output

システム・オブジェクト・ハンドルがこのパラメーターへ戻されます。

戻りコード

以下は、共通の戻り値です。

CWB_OK

正常終了。

CWB_INVALID_POINTER
ポインター・パラメーターのいずれかが NULL です。

CWB_INVALID_SYSNAME
システム名が無効です。

CWB_RESTRICTED_BY_POLICY
ユーザーが、システム・リストにまだ定義されていないシステムのシステム・オブジェクトを作成することを禁止するポリシーが存在します。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

使用法

システム・オブジェクトの使用が終了した後に、cwbCO_DeleteSystem を呼び出し、システム・オブジェクトが使用していた資源を解放する必要があります。既存のものと同じようなシステム・オブジェクトを作成したい場合は、cwbCO_CreateSystemLike を使用します。

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbCO_CreateSystemLike:

System i Access for Windows の cwbCO_CreateSystemLike コマンドを使用します。

目的

所定のシステム・オブジェクトと似ているシステム・オブジェクトを作成します。新規システム・オブジェクトに特定のシステム名を与えることも、NULL を指定して所定のシステム・オブジェクトの名前を使用することも、いずれも可能です。所定のシステム・オブジェクトのすべての属性は、以下の例外を除いて、新規システム・オブジェクトへコピーされます。

- ユーザー ID
- パスワード
- システム名 (別のシステム名が指定されていた場合)
- IP アドレス (システム名が異なる場合)

システム・オブジェクトの属性のリストについては、47 ページの『システム・オブジェクトの属性のリスト』を参照してください。

構文

```
UINT CWB_ENTRY cwbCO_CreateSystemLike(
                            cwbCO_SysHandle     systemToCopy,
                            LPCSTR              systemName
                            cwbCO_SysHandle    *system);
```

パラメーター

cwbCO_SysHandle systemToCopy - input
以前の、cwbCO_CreateSystem もしくは cwbCO_CreateSystemLike への呼び出しによって戻されたハンドル。これは System i の ID です。これが「コピー」されるオブジェクトです。

LPCSTR systemName - input
新規システム・オブジェクトで使用する System i 名 (ヌル終了のもの) を含んでいるバッファーを指すポインター。 NULL または空のストリングが渡された場合は、所定のシステム・オブジェクトからの名前が新規システム・オブジェクトにコピーされます。システム名を指定する場合は、ホスト名または System i の IP アドレス (小数点付き 10 進数) のいずれを指定することもできます。指定された名前が有効な i5/OS ホスト名または IP アドレス・ストリング (「nnn.nnn.nnn.nnn」の形式) ではない場合、接続やセキュリティー検証の試みはすべて失敗します。

cwbCO_SysHandle *newSystem - output
新規システム・オブジェクトのシステム・オブジェクト・ハンドルがこのパラメーターに戻されます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
API に与えられたポインターが無効です。

CWB_INVALID_SYSNAME
システム名が無効です。

CWB_RESTRICTED_BY_POLICY
ユーザーが、システム・リストにまだ定義されていないシステムのシステム・オブジェクトを作成することを禁止するポリシーが存在します。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

使用法

新規システム・オブジェクトの使用が終了した後で、cwbCO_DeleteSystem を呼び出し、システム・オブジェクトが使用していた資源を解放する必要があります。

ユーザー ID とパスワードの検証が、新しいものについてはまだ行われていないため、新規システム・オブジェクトの状態は所定のシステム・オブジェクトの状態と同じではない可能性があります。また、新規システム・オブジェクトはそれに関連した接続は持っていないのに対して、所定のシステム・オブジェクトでは持っている可能性があります。このため、所定のシステム・オブジェクトの属性を、その状態のために変更できない場合であっても、新規システム・オブジェクトの属性はその状態が異なっている可能性があるために、変更できることがあります。

cwbCO_DeleteSystem:

System i Access for Windows の cwbCO_DeleteSystem コマンドを使用します。

目的

そのハンドルで指定されたシステム・オブジェクトを削除し、システム・オブジェクトが使用していたすべての資源を解放します。

構文

```
UINT CWB_ENTRY cwbCO_DeleteSystem(
                            cwbCO_SysHandle     system);
```

パラメーター

cwbCO_SysHandle system - input
以前の、cwbCO_CreateSystem もしくは cwbCO_CreateSystemLike への呼び出しによって戻されたハンドル。これは System i の ID です。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

使用法

システム・オブジェクト資源が解放される前に、指定されたシステム・オブジェクトを使用して行われた接続が 1 つでもある場合には、必要であれば強制的に、接続を終了させます。活動状態にある接続があるかどうかを判別するには、cwbCO_IsConnected を呼び出します。既存の接続の切断がいずれも正常終了したかどうかを知りたい場合は、この API を呼び出す前に cwbCO_Disconnect を明示的に呼び出します。

通信およびセキュリティー: 接続および切断の API

System i の接続および切断や、その他の関連する動作をサポートする API です。

cwbCO_Connect:

System i Access for Windows の cwbCO_Connect コマンドを使用します。

目的

指定された i5/OS ホスト・サービスに接続します。

構文

```
UINT CWB_ENTRY cwbCO_Connect(
                            cwbCO_SysHandle     system,
                            cwbCO_Service       service,
                            cwbSV_ErrHandle     errorHandle );
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは、接続に使用される System i の ID です。

cwbCO_Service service - input
System i の接続用サービス。有効な値は、CWBCO_SERVICE_ANY および CWBCO_SERVICE_ALL の値を除く、 103 ページの『cwbCO_Service の定義』でリストされている値。この API には、複数のサービスを一度に切断できる cwbCO_Disconnect とは異なり、 1 つのサービスしか指定できません。

cwbSV_ErrHandle errorHandle - input/output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合、または errorHandle が無効な場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_SERVICE_NAME_ERROR
サービス識別コードが有効な値ではないか、値を組み合わせたものとなっていました (この API では、単一の値しか許されていません)。

CWB_CONNECTION_TIMED_OUT
システムを検出するのに時間がかかりすぎて、タイムアウトになりました。

CWB_CONNECTION_REFUSED
システムが、接続の受け入れを拒否しました。

CWB_NETWORK_IS_DOWN
ネットワーク・エラーが発生しました。あるいは TCP/IP が、PC 上で正しく構成されていません。

CWB_NETWORK_IS_UNREACHABLE
現在システムが接続されているネットワーク・セグメントは、PC が接続されているセグメントから到達できません。

CWB_USER_TIMEOUT
システム・オブジェクトに関連した接続タイムアウト値の有効期限が、接続が確立される前に切れたため、待機を終了しました。

CWB_FIPS_UNAVAILABLE
この接続は SSL 用に構成され、FIPS 準拠モードが使用可能ですが、FIPS サポートが利用不可のため、SSL は使用できません。リカバリー情報については、以下のパスを使用してメッセージ CWBCO1060 を参照してください。

「スタート」 → 「System i Access for Windows フォルダー」 → 「サービス」 → 「エラーおよびトレース・メッセージのヘルプ」 → 「System i Access for Windows メッセージ (System i Access for Windows messages)」 → 「CWBCO1060」

注: セキュリティー検証を行って失敗した結果として、その他の共通の戻りコードが戻されることがあります。 cwbCO_Signon の注釈の共通戻りコードを参照してください。

使用法

System i サインオンがまだ行われていない場合、cwbCO_Connect の呼び出し時に、まずサインオンが行われます。サインオンを別のときに実行させたい場合は、先に cwbCO_Signon を呼び出してから、後で cwbCO_Connect を呼び出します。サインオンとその動作については、cwbCO_Signon の注釈を参照してください。サインオンの試行が失敗した場合は、指定されたサービスへの接続は確立されません。

指定のシステム・オブジェクトで指定されたシステムがシステム・リストに存在せず、かつ、システム・オブジェクト・パーシスタンス・モードが適切に設定されている場合、 cwbCO_Connect または cwbCO_Signon の呼び出しが最初に正常に行われた際に、システム・オブジェクトで指定されたシステムがシステム・リストに追加されます。パーシスタンス・モードの詳細については、cwbCO_SetPersistenceMode の注釈を参照してください。

指定されたサービスへの接続が既に存在している場合は、接続は新たには設定されず、CWB_OK が戻されます。この API の呼び出しが正常に行われるごとに、指定されたサービスへの接続の使用回数が増やされます。

cwbCO_Disconnect が同じサービスのために呼び出されるごとに、使用回数は減らされます。使用回数がゼロに達すると、接続が実際に終了します。

したがって、cwbCO_Connect API へのすべての呼び出しについて、接続が適切な時間に終了することができるようにするために、後で cwbCO_Disconnect API への対の呼び出しがあるということは、きわめて重要なことです。別の方法としては、CWBCO_SERVICE_ALL を指定して cwbCO_Disconnect API を呼び出し (指定されたシステム・オブジェクトを通じて行われた全サービスに対して既存の接続をすべて切断する)、使用回数を全部 0 にリセットするというものがあります。

戻りコードが CWB_USER_TIMEOUT の場合、 cwbCO_SetConnectTimeout を呼び出してこのシステム・オブジェクトの接続タイムアウト値を大きくし、接続を再度試行することができます。 TCP/IP 通信スタックによって放棄されるまで、System i Access に放棄させたくない場合は、接続タイムアウト値を CWBCO_CONNECT_TIMEOUT_NONE に設定して、接続を再度試行します。

関連資料

47 ページの『システム・オブジェクトの属性』

System i プラットフォームにおける**システム・オブジェクトの属性**は、システム・オブジェクトが表すシステムへのサインオンおよび通信の動作に影響します。

cwbCO_Disconnect:

System i Access for Windows の cwbCO_Disconnect コマンドを使用します。

目的

指定された i5/OS ホスト・サービスからの切断を行います。

構文

```
UINT CWB_ENTRY cwbCO_Disconnect(
                            cwbCO_SysHandle     system,
                            cwbCO_Service       service,
                            cwbSV_ErrHandle     errorHandle );
```

パラメーター

cwbCO_SysHandle system - input
以前の、cwbCO_CreateSystem もしくは cwbCO_CreateSystemLike への呼び出しによって戻されたハンドル。これは、切断に使用される System i の ID です。

cwbCO_Service service - input
System i の切断用サービス。有効な値は、CWBCO_SERVICE_ANY の値を除き、このファイルの冒頭にリストされています。CWBCO_SERVICE_ALL が指定されている場合は、すべての接続されたサービスへの接続は終了し、接続使用回数はすべてリセットされてゼロに戻ります。

cwbSV_ErrHandle errorHandle - input/output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合、または errorHandle が無効な場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_SERVICE_NAME_ERROR
サービス識別コードが無効です。

CWB_NOT_CONNECTED
単一サービスが接続されませんでした。

使用法

cwbCO_Connect を使用して確立された接続がもはや必要なくなった際に、この関数を呼び出してください。

指定されたサービスが切断できない場合、戻りコードはこのエラーを戻します。複数のエラーが生じた場合、最初の戻りコードだけが API 戻りコードとして戻されます。

個別サービスの切断に関する使用上の注意:
この関数によって、このシステム・オブジェクトで指定したサービスの使用回数が減らされ、接続は実際に終了する場合も、終了しない場合もあります。詳しくは、cwbCO_Connect API の使用上の注意を参照してください。

現在、接続されていないサービスを切断すると CWB_NOT_CONNECTED になります。

個別サービスは、安全に切断されます。

CWBCO_SERVICE_ALL に関する使用上の注意:
戻りコード CWB_NOT_CONNECTED は、接続されたサービスの数に関係なく、CWBCO_SERVICE_ALL が指定されている場合には戻されません。

活動状態のサービスすべてに対する切断を要求すると、System i 切断メッセージが生成されます。

cwbCO_GetConnectTimeout:

System i Access for Windows の cwbCO_GetConnectTimeout コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトについて、現在設定されている秒単位の接続タイムアウト値を取得します。

構文

```
UINT CWB_ENTRY cwbCO_GetConnectTimeout(
                            cwbCO_SysHandle             system,
                            PULONG                      timeout );
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

PULONG timeout - output
秒単位のタイムアウト値を戻します。この値は CWBCO_CONNECT_TIMEOUT_MIN から CWBCO_CONNECT_TIMEOUT_MAX の範囲になります。あるいは、接続タイムアウト値が必要でない場合には、CWBCO_CONNECT_TIMEOUT_NONE になります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
タイムアウト・ポインターが NULL です。

使用法

なし

cwbCO_GetPersistenceMode:

System i Access for Windows の cwbCO_GetPersistenceMode コマンドを使用します。

目的

指定されたシステム・オブジェクトについて、それが表すシステムがサインオンに成功した後で、システム・リストにその属性とともにそのシステムが追加されるかどうか (まだリストにない場合) についての情報を取得します。

構文

```
UINT CWB_ENTRY cwbCO_GetPersistenceMode(
                            cwbCO_SysHandle        system,
                            cwbCO_PersistenceMode *mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_PersistenceMode * mode - output
パーシスタンス・モードを戻します。指定できる値とその意味については、cwbCO_SetPersistenceMode の注釈を参照してください。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
モード・ポインターが NULL です。

使用法

なし

cwbCO_IsConnected:

System i Access for Windows の cwbCO_IsConnected コマンドを使用します。

目的

現行のシステム・オブジェクトのうち、特定のものを使用している System i 接続があるかどうかを検出し、ある場合にはその数も検出します。

構文

```
UINT CWB_ENTRY cwbCO_IsConnected(
                           cwbCO_SysHandle     system,
                           cwbCO_Service       service,
                           PULONG              numberOfConnections );
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_Service service - input
接続をチェックするサービス。 103 ページの『cwbCO_Service の定義』にリストされている cwbCO_Service のいずれの値も有効です。何らかのサービスが接続されているかどうかを検出するには、CWBCO_SERVICE_ANY を指定します。このシステム・オブジェクトを使用して接続されているサービスの数を検出するには、CWBCO_SERVICE_ALL を指定します。

PULONG numberOfConnections - output
指定されたサービス (1 つまたは複数) について活動中の接続の数を戻すのに使用されます。指定され

たサービスが CWBCO_SERVICE_ALL ではない場合は、システム・オブジェクトごとに、1 つのサービスにつき活動中の接続は、多くても 1 つしか認められないため、戻される値は 0 または 1 のいずれかです。 CWBCO_SERVICE_ALL が指定されている場合は、サービスごとに 1 つの接続が活動中である可能性があるため、この値は 0 から可能なサービスの数までになります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。指定されたすべてのサービスが接続、あるいは CWBCO_SERVICE_ANY が指定されている場合は 1 つまたは複数のサービスが接続されています。

CWB_NOT_CONNECTED
単一のサービスが指定されていた場合は、そのサービスは接続されません。
CWBCO_SERVICE_ANY の値が指定されていた場合は、活動中の接続はなにもありません。
CWBCO_SERVICE_ALL の値が指定されていた場合は、接続されていないサービスが少なくとも 1 つは存在します。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_SERVICE_NAME_ERROR
サービス識別コードが無効です。

CWB_INVALID_POINTER
numberOfConnections パラメーターが NULL です。

使用法

CWBCO_SERVICE_ALL が指定されており CWB_NOT_CONNECTED が戻された場合には、活動接続がいくつか存在する可能性があり、依然として、活動接続の回数が戻されます。指定されたシステム・オブジェクトを通じて接続がいくつ存在するかを検出する場合は、この API を呼び出し、CWBCO_SERVICE_ALL を指定します。戻りコードが、CWB_OK または CWB_NOT_CONNECTED のいずれかの場合でも、存在する接続の数は numberOfConnections に保管されます。

cwbCO_SetConnectTimeout:

System i Access for Windows の cwbCO_SetConnectTimeout コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトについて、System i Access for Windows が接続の試行を放棄してエラーを戻すまでに待機する時間 (秒単位) を指定します。

構文

```
UINT CWB_ENTRY cwbCO_SetConnectTimeout(
                            cwbCO_SysHandle           system,
                            ULONG                     timeout );
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

ULONG timeout - input
接続タイムアウト値を秒単位で指定します。この値は CWBCO_CONNECT_TIMEOUT_MIN から CWBCO_CONNECT_TIMEOUT_MAX の範囲の値でなければなりません。あるいは、タイムアウトが必要でない場合には、CWBCO_CONNECT_TIMEOUT_NONE を使用します。値が最小値より小さい場合は CWBCO_CONNECT_TIMEOUT_MIN を、値が最大値より大きい場合は CWBCO_CONNECT_TIMEOUT_MAX を使用します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

使用法

ポリシーによりタイムアウト値が示されておらず、かつ、この API を使用して明示的に設定されていない場合には、使用される接続タイムアウト値は CWBCO_CONNECT_TIMEOUT_DEFAULT になります。

cwbCO_SetPersistenceMode:

System i Access for Windows の cwbCO_SetPersistenceMode コマンドを使用します。

目的

この関数は、サインオンが正常終了したならば、システム・オブジェクトが表すシステム (システム・オブジェクトで指名) を、その属性とともに、システム・リストに追加できるかどうか (まだリストにない場合) を設定します。

構文

```
UINT CWB_ENTRY cwbCO_SetPersistenceMode(
                              cwbCO_SysHandle        system,
                              cwbCO_PersistenceMode  mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_PersistenceMode mode - input
パーシスタンス・モードを指定します。指定できる値は以下のとおりです。

CWBCO_MAY_MAKE_PERSISTENT
指定されたシステム・オブジェクトで指名されているシステムがまだシステム・リストにない場合は、サインオンが正常終了したならばそのシステムをリストに追加します。これによって、システム・オブジェクトで定義されているシステムを、現在および将来にわたって、このパーソナル・コンピューター上で実行されるこのアプリケーションおよびその他のアプリケーションが (システムがこのリストから削除されるまで) 選択できるようになります。

CWBCO_MAY_NOT_MAKE_PERSISTENT
指定されたシステム・オブジェクトで (その属性とともに) 指名されたシステムを、システム・リストに追加することはできません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_PARAMETER
モード・パラメーターの値が無効。

CWB_RESTRICTED_BY_POLICY
この値の変更を禁止するポリシーが存在します。

CWB_INV_AFTER_SIGNON
指定されたシステム・オブジェクトを使用して、サインオンが正常に行われたため、この設定値は変更されることはありません。

使用法

この API は、指定されたシステム・オブジェクトについて、サインオンが正常に行われた後では使用することができません。このシステム・オブジェクトについて、 cwbCO_Signon または cwbCO_Connect のいずれかの呼び出しが正常に行われた場合には、サインオンが行われています。

システム・オブジェクトで指定されたシステムが既にシステム・リストにある場合は、この設定は無効です。

cwbCO_Verify:

System i Access for Windows の cwbCO_Verify コマンドを使用します。

目的

特定の i5/OS ホスト・サービスに接続できるかどうかを検証します。

構文

```
UINT CWB_ENTRY cwbCO_Verify(
                            cwbCO_SysHandle     system,
                            cwbCO_Service       service,
                            cwbSV_ErrHandle     errorHandle );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは、接続可能性が検証された System i ID です。

cwbCO_Service service - input
接続可能性がされた System i サービスです。有効な値は、CWBCO_SERVICE_ANY の値を除き、
103 ページの

103 ページの『cwbCO_Service の定義』にリストされています。すべてのサービスの接続可能性を検証する場合は、CWBCO_SERVICE_ALL を指定します。

cwbSV_ErrHandle errorHandle - input/output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合、または errorHandle が無効な場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_SERVICE_NAME_ERROR
サービス識別コードが無効です。

CWB_USER_TIMEOUT
システム・オブジェクトに関連した接続タイムアウト値が、接続の検証が完了する前に有効期限が切れました。そのため、待機を停止します。

CWB_COMMUNICATIONS_ERROR
サービスへの接続を検証しようとしてエラーが起こりました。

使用法

この API ではユーザー ID とパスワードが設定されている必要はありません。また、サインオンを引き起こすこともないため、この情報についてプロンプトを出すこともありません。いずれにせよ、システム・オブジェクトの状態を変更することはありません。

指定されたサービスへの接続が既に存在している場合は、新たな接続が設定されることはなく、接続可能性がそのサービスについて検証されたと見なされます。

CWBCO_SERVICE_ALL が検証のために指定された場合は、すべてのサービスが接続できる場合のみ、戻りコードが CWB_OK になります。何らかの検査を行おうとして失敗した場合、他のサービスの検証が引き続き試行されていても、戻りコードは最初に失敗したものからの戻りコードになります。

この API は使用可能な接続を確立しないため、検証が完了した場合には自動的に切断されます。したがって、接続を終了させるために cwbCO_Disconnect を呼び出さないでください。

通信およびセキュリティー: セキュリティー妥当性検査とデータの API

セキュリティー妥当性検査およびデータを提供する System i API です。

cwbCO_ChangePassword:

System i Access for Windows の cwbCO_ChangePassword コマンドを使用します。

目的

指定した System i ユーザーのパスワードを、指定した古い値から、指定した新しい値に変更します。この API は、所定のシステム・オブジェクトで現在設定されているユーザー ID とパスワードは使用せず、またこれらの値の変更も行いません。

構文

```
UINT CWB_ENTRY cwbCO_ChangePassword(
                            cwbCO_SysHandle       system,
                            LPCSTR                userID,
                            LPCSTR                oldPassword,
                            LPCSTR                newPassword,
                            cwbSV_ErrHandle       errorHandle);
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

LPCSTR userID - input
ユーザー ID が含まれている ASCIIZ ストリングを指すポインター。最大長は、NULL 終了文字を含めて、CWBCO_MAX_USER_ID + 1 文字です。

LPCSTR oldPassword - input
旧パスワードを含むバッファーを指すポインター。最大長は、ヌル終了文字を含めて、CWBCO_MAX_PASSWORD + 1 バイトです。

LPCSTR newPassword - input
新規パスワードを含むバッファーを指すポインター。最大長は、ヌル終了文字を含めて、CWBCO_MAX_PASSWORD + 1 バイトです。

cwbSV_ErrHandle errorHandle - input/output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合、または errorHandle が無効な場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
ポインター・パラメーターが NULL。

CWB_GENERAL_SECURITY_ERROR
一般セキュリティー・エラーが起こりました。ユーザー・プロファイルにパスワードがないか、パスワード検証プログラムがパスワードにエラーを検出しました。

CWB_INVALID_PASSWORD
新規パスワード中の 1 つまたは複数の文字が無効であるか、パスワードが長すぎます。

CWB_INVALID_USERID
ユーザー ID 中の 1 つまたは複数の文字が無効であるか、ユーザー ID が長すぎます。

CWB_UNKNOWN_USERID
与えられたユーザー ID がこのシステムでは認知されていません。

CWB_WRONG_PASSWORD
パスワードが正しくありません。

CWB_USER_PROFILE_DISABLED
このユーザー ID は使用不可になっています。

CWB_PW_TOO_LONG
新規パスワードが許容最大長を超えています。

CWB_PW_TOO_SHORT
新規パスワードが許容最小長に至っていません。

CWB_PW_REPEAT_CHARACTER
新規パスワードに 2 回以上使用された文字が含まれています。

CWB_PW_ADJACENT_DIGITS
新規パスワードでは数字同士が隣接しています。

CWB_PW_CONSECUTIVE_CHARS
新規パスワードでは、ある文字が連続して繰り返し使用されています。

CWB_PW_PREVIOUSLY_USED
新規パスワードは以前使用されています。

CWB_PW_DISALLOWED_CHAR
新規パスワードには、システムで使用禁止の文字が使用されています。

CWB_PW_NEED_NUMERIC
新規パスワードは、少なくとも 1 つの数字が含まれていなければなりません。

CWB_PW_MATCHES_OLD
新規パスワードは、1 つまたは複数の文字位置で旧パスワードと一致しています。

CWB_PW_NOT_ALLOWED
新規パスワードは、使用禁止パスワードの辞書の中に存在します。

CWB_PW_CONTAINS_USERID
新規パスワードには、パスワードの一部としてユーザー ID が含まれています。

CWB_PW_LAST_INVALID_PWD
無効なパスワードをもう一度使用すると、そのユーザー・プロファイルは使用不可になります。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

有効なパスワードの長さは、System i のパスワード・レベルの現在の設定に応じて決まります。パスワード・レベル 0 および 1 では、最高 10 文字までの長さのパスワードを許可します。パスワード・レベル 2 および 3 では、最高 128 文字までの長さのパスワードを許可します。

cwbCO_GetDefaultUserMode:

System i Access for Windows の cwbCO_GetDefaultUserMode コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトについて、現在設定されているデフォルトのユーザー・モードを取得します。

構文

```
UINT CWB_ENTRY cwbCO_GetDefaultUserMode(
                            cwbCO_SysHandle         system,
                            cwbCO_DefaultUserMode  *mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_DefaultUserMode * mode - output
このシステム・オブジェクトについてのデフォルトのユーザー・モードを戻します。指定できる値とその意味のリストについては、cwbCO_SetDefaultUserMode の注釈を参照してください。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
モード・ポインターが NULL です。

使用法

なし

cwbCO_GetFailedSignons:

System i Access for Windows の cwbCO_GetFailedSignons コマンドを使用します。

目的

セキュリティー検証の試行が、前回成功して以来、これまでに成功しなかった回数を検索します。

構文

```
UINT CWB_ENTRY cwbCO_GetFailedSignons(
                            cwbCO_SysHandle       system,
                            PUSHORT               numberFailedAttempts);
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

PUSHORT numberFailedAttempts - output
失敗したログオン試行の回数が含まれる短精度整数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
numberFailedAttempts ポインターが NULL です。

CWB_INV_BEFORE_VALIDATE
指定されたシステム・オブジェクトで設定されているユーザー ID とパスワードがまだ検証されていないため、この情報は利用できません。

使用法

この API を使用する前に、cwbCO_VerifyUserIDPassword、cwbCO_Signon、または cwbCO_Connect の呼び出しに成功している必要があります。戻された値が最近のものであることを確認したい場合は、cwbCO_VerifyUserIDPassword を明示的に呼び出すか、もしくは cwbCO_Signon または cwbCO_Connect を呼び出す前に、Validate Mode を CWBCO_VALIDATE_ALWAYS に設定する必要があります。

cwbCO_GetPasswordExpireDate:

System i Access for Windows の cwbCO_GetPasswordExpireDate コマンドを使用します。

目的

システム・オブジェクトで指定されたシステムに関して、System i ユーザー ID のパスワードが期限切れになった日時を検索します。

構文

```
UINT CWB_ENTRY cwbCO_GetPasswordExpireDate(
                            cwbCO_SysHandle       system,
                            cwb_DateTime         *expirationDateTime);
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwb_DateTime * expirationDateTime - output
現行のユーザー ID について、以下の形式により、パスワードの有効期限が切れる日時が入っている構造を指すポインター。

バイト	内容
1 - 2	年 (例: 1998 = 0x07CF)
3	月 (1 月 = 0x01)
4	日 (1 日 = 0x01、31 日 = 0x1F)
5	時 (午前 0 時 = 0x00、23 時 = 0x17)
6	分 (0 分 = 0x00、59 分 = 0x3B)
7	秒 (0 秒 = 0x00、59 秒 = 0x3B)
8	0.01 秒 (0.00 秒 = 0x00、0.99 秒 = 0x63)

注: 1 日の最大時刻は、23 時 59 分 59.99 秒です。(午前 0 時は、次の日の 0 時 0 分 0.0 秒です。)

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
cwb_DateTime 構造を指すポインターが NULL です。

CWB_INV_BEFORE_VALIDATE
指定されたシステム・オブジェクトで設定されているユーザー ID とパスワードがまだ検証されていないため、パスワードの有効期限情報が利用できないか、あるいは検証が行われ、ユーザー・プロファイルのパスワード有効期限の間隔が *NOMAX に設定されています。

使用法

この API を使用する前に、cwbCO_VerifyUserIDPassword、cwbCO_Signon、または cwbCO_Connect の呼び出しに成功している必要があります。戻された値が最近のものであることを確認したい場合は、cwbCO_VerifyUserIDPassword を明示的に呼び出すか、もしくは cwbCO_Signon または cwbCO_Connect を呼び出す前に、Validate Mode を CWBCO_VALIDATE_ALWAYS に設定する必要があります。

ユーザー・プロファイルのパスワードの有効期限間隔が *NOMAX に設定されている場合、パスワードの有効期限が切れる日は存在しません。この事例を検出するには、まず上記の方法で ユーザー ID とパスワードを検証し、検証に成功した後 cwbCO_GetPasswordExpireDate を呼び出します。CWBCO_INV_BEFORE_VALIDATE の戻りコードは、パスワードの有効期限間隔が *NOMAX に設定されていることを意味します。

cwbCO_GetPrevSignonDate:

System i Access for Windows の cwbCO_GetPrevSignonDate コマンドを使用します。

目的

前回の、正常終了したセキュリティー検証の日時を検索します。

構文

```
UINT CWB_ENTRY cwbCO_GetPrevSignonDate(
                              cwbCO_SysHandle       system,
                              cwb_DateTime         *signonDateTime);
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwb_DateTime * signonDateTime - output
以下の形式で、前回に行われたサインオンの日時が入っている構造を指すポインター。

バイト	内容
1 - 2	年 (例: 1998 = 0x07CF)
3	月 (1 月 = 0x01)
4	日 (1 日 = 0x01、31 日 = 0x1F)
5	時 (午前 0 時 = 0x00、23 時 = 0x17)
6	分 (0 分 = 0x00、59 分 = 0x3B)
7	秒 (0 秒 = 0x00、59 秒 = 0x3B)
8	0.01 秒 (0.00 秒 = 0x00、0.99 秒 = 0x63)

注: 1 日の最大時刻は、23 時 59 分 59.99 秒です。(午前 0 時は、次の日の 0 時 0 分 0.0 秒です。)

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
cwb_DateTime 構造を指すポインターが NULL です。

CWB_INV_BEFORE_VALIDATE
指定されたシステム・オブジェクトで設定されているユーザー ID とパスワードがまだ検証されていないため、この情報は利用できません。

使用法

この API を使用する前に、cwbCO_VerifyUserIDPassword、cwbCO_Signon、または cwbCO_Connect の呼び出しに成功している必要があります。戻された値が最近のものであることを確認したい場合は、cwbCO_VerifyUserIDPassword を明示的に呼び出すか、もしくは cwbCO_Signon または cwbCO_Connect を

呼び出す前に、Validate Mode を CWBCO_VALIDATE_ALWAYS に設定する必要があります。

cwbCO_GetPromptMode:

System i Access for Windows の cwbCO_GetPromptMode コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトについて、現在、設定されているプロンプト・モードを取得します。

構文

```
UINT CWB_ENTRY cwbCO_GetPromptMode(
                            cwbCO_SysHandle       system,
                            cwbCO_PromptMode     *mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_PromptMode * mode - output
プロンプト・モードを戻します。指定できる値とその意味については、cwbCO_SetPromptMode の注釈を参照してください。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
モード・ポインターが NULL です。

使用法

なし

cwbCO_GetSignonDate:

System i Access for Windows の cwbCO_GetSignonDate コマンドを使用します。

目的

現行の、正常終了したセキュリティー検証の日時を検索します。

構文

```
UINT CWB_ENTRY cwbCO_GetSignonDate(
                            cwbCO_SysHandle       system,
                            cwb_DateTime         *signonDateTime);
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwb_DateTime * signonDateTime - output
以下の形式で、現行のサインオンが行われた日時が入っている構造を指すポインター。

バイト	内容
1 - 2	年 (例: 1998 = 0x07CF)
3	月 (1 月 = 0x01)
4	日 (1 日 = 0x01、31 日 = 0x1F)
5	時 (午前 0 時 = 0x00、23 時 = 0x17)
6	分 (0 分 = 0x00、59 分 = 0x3B)
7	秒 (0 秒 = 0x00、59 秒 = 0x3B)
8	0.01 秒 (0.00 秒 = 0x00、0.99 秒 = 0x63)

注: 1 日の最大時刻は、23 時 59 分 59.99 秒です。(午前 0 時は、次の日の 0 時 0 分 0.0 秒です。)

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
cwb_DateTime 構造を指すポインターが NULL です。

CWB_INV_BEFORE_VALIDATE
指定されたシステム・オブジェクトで設定されているユーザー ID とパスワードがまだ検証されていないため、この情報は利用できません。

使用法

この API を使用する前に、cwbCO_VerifyUserIDPassword、cwbCO_Signon、または cwbCO_Connect の呼び出しに成功している必要があります。戻された値が最近のものであることを確認したい場合は、cwbCO_VerifyUserIDPassword を明示的に呼び出すか、もしくは cwbCO_Signon または cwbCO_Connect を呼び出す前に、Validate Mode を CWBCO_VALIDATE_ALWAYS に設定する必要があります。

cwbCO_GetUserIDEx:

System i Access for Windows の cwbCO_GetUserIDEx コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトに関連したユーザー ID を取得します。このユーザー ID は、System i 接続に使用されます。

構文

```
UINT CWB_ENTRY cwbCO_GetUserIDEx(
                             cwbCO_SysHandle    system,
                             LPSTR              userID,
                             PULONG             length );
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

LPSTR userID - output
NULL で終わるユーザー ID が含まれているバッファーを指すポインター。ユーザー ID の長さは、最大で CWBCO_MAX_USER_ID 文字になります。

PULONG length - input/output
ユーザー ID バッファーの長さを指すポインター。バッファーが、終了の NULL のスペースを含めて、ユーザー ID を含めるには小さすぎる場合は、必要とするバッファーのサイズがこのパラメーターに入れられます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
渡されたポインター・パラメーターのいずれかが NULL です。

CWB_BUFFER_OVERFLOW
ユーザー ID バッファーが、ユーザー ID 全体を保持するには十分な大きさではありません。

使用法

System i のユーザー ID は、すでに検証されている場合もありますし、未検証の場合もあります。検証済みであることを確認するためには、この API を呼び出す前に、 cwbCO_Signon または cwbCO_Connect を呼び出します。

ユーザー ID が設定されておらず、システム・オブジェクトへのサインオンが行われていない場合に戻されるユーザー ID は、System i のデフォルトのユーザー ID が構成済みであっても、空ストリングになります。

cwbCO_GetValidateMode:

System i Access for Windows の cwbCO_GetValidateMode コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトについて、現在、設定されている検証モードを取得します。

構文

```
UINT CWB_ENTRY cwbCO_GetValidateMode(
                            cwbCO_SysHandle      system,
                            cwbCO_ValidateMode  *mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_ValidateMode * mode - output
検証モードを戻します。指定できる値とその意味については、cwbCO_SetValidateMode の注釈を参照してください。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
モード・ポインターが NULL です。

使用法

なし

cwbCO_GetWindowHandle:

System i Access for Windows の cwbCO_GetWindowHandle コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトについて、現在、関連付けられているウィンドウ・ハンドルがあれば、それを取得します。

構文

```
UINT CWB_ENTRY cwbCO_GetWindowHandle(
                            cwbCO_SysHandle       system,
                            HWND                 *windowHandle );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

HWND * pWindowHandle - output
システム・オブジェクトに関連したウィンドウ・ハンドルを戻します。あるいは、それに関連したウィンドウ・ハンドルが無い場合は、NULL を戻します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
windowHandle ポインターが NULL です。

使用法

なし

cwbCO_HasSignedOn:

System i Access for Windows の cwbCO_HasSignedOn コマンドを使用します。

目的

指定されたシステム・オブジェクトが、「サインオン」されたかどうか (ユーザー ID とパスワードが、指定されたシステム・オブジェクトの存続期間内のある時点で検証されたかどうか) の指示を戻します。

構文

```
UINT CWB_ENTRY cwbCO_HasSignedOn(
                            cwbCO_SysHandle       system,
                            cwb_Boolean          *signedOn );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwb_Boolean * signedOn - output
「サインオンの有無」の指示が保管されている cwb_Boolean を指すポインター。指定されたシステム・オブジェクトがサインオンされている場合は、これは CWB_TRUE に設定され、そうでない場合は CWB_FALSE に設定されます。 (エラーの場合は、同じく CWB_FALSE に設定されます。)

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
signedOn ポインターが NULL です。

使用法

戻された CWB_TRUE の指示は、ユーザー ID とパスワードがある一定時間枠内に検証されたことを意味するものではなく、システム・オブジェクトの作成以降にサインオンが行われたということを意味しているに過ぎません。このようなサインオンによって、System i 接続やセキュリティー検証フローが発生することはなく、これらが組み込まれることもありません。このことは、次のことを意味しています。すなわち、たとえ CWB_TRUE が戻された場合でも、正常なサインオンを必要とするシステム・オブジェクトに対して次回に呼び出しを行うと、接続を行い、ユーザー ID とパスワードを再度、検証しようとする可能性があり、さらにその検証、したがってサインオンは、失敗する可能性があるということです。 signedOn 標識は、最新のユーザー ID とパスワードの検証結果を反映しています。ユーザー ID とパスワードの検証 (サインオン) が、ある時点では成功していたとしても、その時点以降に検証が失敗すれば、signedOn は CWB_FALSE に設定されます。

cwbCO_SetDefaultUserMode:

System i Access for Windows の cwbCO_SetDefaultUserMode コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトについて、構成済みのデフォルトのユーザー ID に関して動作を設定します。

構文

```
UINT CWB_ENTRY cwbCO_SetDefaultUserMode(
                            cwbCO_SysHandle         system,
                            cwbCO_DefaultUserMode   mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_DefaultUserMode mode - input
デフォルトのユーザー ID について行うことを指定します。指定できる値は以下のとおりです。

CWBCO_DEFAULT_USER_MODE_NOT_SET
現在、デフォルトのユーザー・モードは使用されていません。このモードが活動中で、かつプロンプト・モード設定でプロンプトを禁止していない場合は、それ以降は残りのデフォルトのユーザー・モードのいずれを使用するのか、サインオン時または接続時にプロンプトが出されます。これらの他のモード値のうちのいずれか 1 つが選択されるまでは、サインオンまたは接続を成功させることはできません。デフォルトのユーザー・モードの設定をこの値に戻すと、次回 System Access がデフォルトのユーザー ID を必要とするときに、プロンプトが出されます。

CWBCO_DEFAULT_USER_USE
明示的に (cwbCO_SetUserIDEx を使用して) 設定されたユーザー ID がない場合にサインオンを行うには、System i 用に構成されているデフォルトのユーザー ID を、システム・オブジェクトで指定されたとおりに使用します。

CWBCO_DEFAULT_USER_IGNORE
デフォルトのユーザー ID を絶対に使用しないことを指定します。サインオンが生じたのに、このシステム・オブジェクト・インスタンス用にユーザー ID が明示的に設定されていない場

合は、プロンプト・モードで許可されていれば (cwbCO_SetPromptMode の注釈参照)、ユーザー ID を入力するようにプロンプトが出されます。なお、プロンプトにはこのユーザー ID の初期値は入っていません。

CWBCO_DEFAULT_USER_USEWINLOGON
このシステム・オブジェクトについて、明示的にユーザー ID が (cwbCO_SetUserIDEx を使用して) 設定されていない場合は、 Windows へログオンしたときに使用したユーザー ID がデフォルトとして使用されます。

CWBCO_DEFAULT_USER_USE_KERBEROS
このシステム・オブジェクトについて、明示的にユーザー ID が (cwbCO_SetUserIDEx を使用して) 設定されていない場合には、 Windows ドメインにログオンした際に作成された Kerberos プリンシパルがデフォルト値として使用されます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_PARAMETER
モード・パラメーターの値が無効。

CWB_RESTRICTED_BY_POLICY
この値の変更を禁止するポリシーが存在します。

CWB_INV_AFTER_SIGNON
指定されたシステム・オブジェクトを使用して、サインオンが正常に行われたため、この設定値は変更されることはありません。

CWB_KERB_NOT_AVAILABLE
このバージョンの Windows では、Kerberos セキュリティー・パッケージは利用できません。

使用法

この API は、指定されたシステム・オブジェクトについて、サインオンが正常に行われた後では使用することができません。このシステム・オブジェクトについて、 cwbCO_Signon または cwbCO_Connect のいずれかの呼び出しが正常に行われた場合には、サインオンが行われています。ユーザー ID が cwbCO_SetUserIDEx API で明示的に設定された場合には、この API で設定されたデフォルトのユーザー・モードは無視されます。

Kerberos をサポートしない Windows プラットフォームで CWBCO_DEFAULT_USER_USE_KERBEROS を設定しようとした場合には、エラー・コード CWB_KERB_NOT_AVAILABLE が戻されます。

cwbCO_SetPassword:

System i Access for Windows の cwbCO_SetPassword コマンドを使用します。

目的

この関数は、パスワードを設定して、指定されたシステム・オブジェクトに関連付けます。 cwbCO_Signon 呼び出しまたは cwbCO_Connect 呼び出しを使用して System i に接続する場合や、cwbCO_SetUserIDEx 呼

び出しを使用してユーザー ID を設定する場合に、このパスワードが使用されます。

構文

```
UINT CWB_ENTRY cwbCO_SetPassword(
                              cwbCO_SysHandle    system,
                              LPCSTR             password );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

LPCSTR password - input
NULL で終わるパスワードが含まれているバッファーを指すポインター。最大長は、NULL 終了文字を含めて、CWBCO_MAX_PASSWORD + 1 バイトです。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
パスワード・ポインターが NULL です。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_INV_AFTER_SIGNON
指定されたシステム・オブジェクトを使用して、サインオンが正常に行われたため、この設定値は変更されることはありません。

使用法

この API は、指定されたシステム・オブジェクトについて、サインオンが正常に行われた後では使用することができません。このシステム・オブジェクトについて、 cwbCO_Signon または cwbCO_Connect のいずれかの呼び出しが正常に行われた場合には、サインオンが行われています。この API で設定されたパスワードは、対応するユーザー ID が cwbCO_SetUserIDEx を使用して設定されていない場合には、使用されません。

有効なパスワードの長さは、System i のパスワード・レベルの現在の設定に応じて決まります。パスワード・レベル 0 および 1 では、最高 10 文字までの長さのパスワードを許可します。パスワード・レベル 2 および 3 では、最高 128 文字までの長さのパスワードを許可します。

cwbCO_SetPromptMode:

System i Access for Windows の cwbCO_SetPromptMode コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトについて、プロンプト・モードを設定します。これは、サインオンを行うときにユーザー ID とパスワードまたはその他の情報のプロンプトをユーザーに出すかどうか、さらにいつ出すかについて指定するものです。

構文

```
UINT CWB_ENTRY cwbCO_SetPromptMode(
                            cwbCO_SysHandle     system,
                            cwbCO_PromptMode    mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_PromptMode - input
プロンプト・モードを指定します。指定できる値は以下のとおりです。

CWBCO_PROMPT_IF_NECESSARY
ユーザー ID またはパスワードのいずれかが明示的に設定されていないか、またはこのシステムの永続的構成、パスワード・キャッシュ (使用可能な場合) などといった方法を使っても検索できない場合に、System i Access for Windows によってプロンプトが出されます。

デフォルトのユーザー・モードが設定されており、デフォルトのユーザー ID を求める System i プロンプトがまだ出されていない場合、cwbCO_Connect または cwbCO_Signon の実行時に System i がこのプロンプトを出します。

CWBCO_PROMPT_ALWAYS
指定したシステム・オブジェクトにサインオンが行われるたびに (たとえ、同じシステムに対する同じユーザー ID を使用した System i サインオンが、異なるシステム・オブジェクトを使用して正常に行われた場合であっても)、System i Access for Windows によって必ずプロンプトが出されます。サインオンは 1 つのシステム・オブジェクトについて一度しか行わないため、このことは、システム・オブジェクトごとに、厳密に 1 回のプロンプトが行われることを意味しています。明示的な追加のサインオン呼び出しを行っても、(プロンプトも含めて) なにも実行されません。下記の使用法に記載されている、このモードを使用する際の 2 つの例外を参照してください。

CWBCO_PROMPT_NEVER
ユーザー ID およびパスワード、あるいはデフォルトのユーザー ID に関するプロンプトが、System i Access for Windows では出されなくなります。このモードを使用すると、ユーザー ID またはパスワードのいずれかが設定されておらず、 (System i パスワード・キャッシュから) プログラマチックに検索できない場合には、実行にサインオンが必要な API (例えば、cwbCO_Signon または cwbCO_Connect) に対する呼び出しは、いずれも失敗することになります。このモードは、以下のいずれかの場合に使用します。

- 無人の PC、または何らかの理由によりエンド・ユーザーとの対話をサポートできない PC で、System i Access for Windows 製品が実行されている場合。
- ユーザー ID とパスワードについて、アプリケーション自体でプロンプトを出しているか、その他の方法でアプリケーションが取り出しており、さらに cwbCO_SetUserIDEx および cwbCO_SetPassword を使用して明示的に設定している場合。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_PARAMETER
モード・パラメーターの値が無効。

CWB_RESTRICTED_BY_POLICY
この値の変更を禁止するポリシーが存在します。

CWB_INV_AFTER_SIGNON
指定されたシステム・オブジェクトを使用して、サインオンが正常に行われたため、この設定値は変更されることはありません。

使用法

この API は、指定されたシステム・オブジェクトについて、サインオンが正常に行われた後では使用することができません。このシステム・オブジェクトについて、 cwbCO_Signon または cwbCO_Connect のいずれかの呼び出しが正常に行われた場合には、サインオンが行われています。プロンプト・モードを CWBCO_PROMPT_ALWAYS に設定しても、以下の 2 つのケースではプロンプトが出されません。

- ユーザー ID とパスワードが、cwbCO_setUserIDEx および cwbCO_SetPassword API を使用して明示的に設定されている。
- Windows ログオン情報 (CWBCO_DEFAULT_USER_USEWINLOGON) の使用が、cwbCO_SetDefaultUserMode API を使用して設定されている。

cwbCO_SetUserIDEx:

System i Access for Windows の cwbCO_SetUserIDEx コマンドを使用します。

目的

この関数は、ユーザー ID を設定して、指定されたシステム・オブジェクトに関連付けます。cwbCO_Signon 呼び出しまたは cwbCO_Connect 呼び出しのいずれかを使用する System i 接続で、このユーザー ID が使用されます。

構文

```
UINT CWB_ENTRY cwbCO_SetUserIDEx(
                            cwbCO_SysHandle    system,
                            LPCSTR             userID );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

LPCSTR userID - input
NULL で終わるユーザー ID が入っているバッファーを指すポインター。ユーザー ID は、終了の NULL 文字を含まずに、CWBCO_MAX_USER_ID 文字よりも長くなってはなりません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
ユーザー ID ポインターが NULL です。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_INV_AFTER_SIGNON
指定されたシステム・オブジェクトを使用して、サインオンが正常に行われたため、この設定値は変更されることはありません。

使用法

この API は、指定されたシステム・オブジェクトについて、サインオンが正常に行われた後では使用することができません。このシステム・オブジェクトについて、 cwbCO_Signon または cwbCO_Connect のいずれかの呼び出しが正常に行われた場合には、サインオンが行われています。この API を使用してユーザー ID を明示的に設定すると、cwbCO_SetDefaultUserMode API を使用して設定されたデフォルトのユーザー・モードは、いずれも無視されます。

cwbCO_SetWindowHandle:

System i Access for Windows の cwbCO_SetWindowHandle コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトについて、システム・オブジェクトに関連する何らかのプロンプト (例えば、ユーザー ID とパスワードについてのプロンプト) が出される場合に、使用するウィンドウ・ハンドルを設定します。そのように設定された場合 (NULL ではないウィンドウ・ハンドルに対して)、このプロンプトはメインのアプリケーション・ウィンドウに対して「モーダル」で表示されるため、そのウィンドウの下に隠れてしまうことはありません。

構文

```
UINT CWB_ENTRY cwbCO_SetWindowHandle(
                            cwbCO_SysHandle     system,
                            HWND                windowHandle );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

HWND windowHandle - input
システム・オブジェクトに関連付けるウィンドウ・ハンドルを指定します。 NULL の場合は、ウィンドウ・ハンドルはシステム・オブジェクトに関連付けられません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

使用法

この API は、指定されたシステム・オブジェクトのウィンドウ・ハンドルを変更するために、たとえサインオンが正常に行われた後であっても、いつでも使用することができます。

cwbCO_SetValidateMode:

System i Access for Windows の cwbCO_SetValidateMode コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトについて、検証モードを設定します。このモードは、ユーザー ID とパスワードを検証する際に、動作に影響を与えます。

構文

```
UINT CWB_ENTRY cwbCO_SetValidateMode(
                            cwbCO_SysHandle     system,
                            cwbCO_ValidateMode  mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_ValidateMode mode - input
検証モードを指定します。指定できる値は以下のとおりです。

CWBCO_VALIDATE_IF_NECESSARY
この PC から過去 24 時間以内に行われた System i ユーザー ID の検証が成功している場合、その最新の検証結果を使用し、この時点での検証には接続しません。別のシナリオで再検証を行う場合もあります。必要に応じて、System i Access for Windows の再検証が行われます。

CWBCO_VALIDATE_ALWAYS
この検証が要求される (必要とされる) ごとに、ユーザー ID とパスワードを検証するための System i 通信が行われます。このモードを設定すると、強制的に検証が行われます (システム・オブジェクトがまだサインオンされていない場合)。システム・オブジェクトがサインオンされてしまうと、この設定は無視されます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_PARAMETER
モード・パラメーターの値が無効。

CWB_RESTRICTED_BY_POLICY
この値の変更を禁止するポリシーが存在します。

CWB_INV_AFTER_SIGNON
指定されたシステム・オブジェクトを使用して、サインオンが正常に行われたため、この設定値は変更されることはありません。

使用法

この API は、指定されたシステム・オブジェクトについて、サインオンが正常に行われた後では使用することができません。このシステム・オブジェクトについて、 cwbCO_Signon または cwbCO_Connect のいずれかの呼び出しが正常に行われた場合には、サインオンが行われています。

cwbCO_Signon:

System i Access for Windows の cwbCO_Signon コマンドを使用します。

目的

ユーザー ID と パスワードを使用して、System i 指定のオブジェクトで表されるシステムに、ユーザーをサインオンさせます。

注: cwbCO_Signon API に誤ったパスワードを渡した場合は、指定されたユーザーの無効サインオン試行カウンターが増やされます。無効なパスワードをホストにあまり多く送信すると、そのユーザー・プロファイルは使用できなくなります。

構文

```
UINT CWB_ENTRY cwbCO_Signon(
                            cwbCO_SysHandle       system,
                            cwbSV_ErrHandle       errorHandle );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbSV_ErrHandle errorHandle - input/output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合、または errorHandle が無効な場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_UNKNOWN_USERID
与えられたユーザー ID がこのシステムでは認知されていません。

CWB_WRONG_PASSWORD
パスワードが正しくありません。

CWB_PASSWORD_EXPIRED
パスワードの有効期限切れ。

CWB_USER_PROFILE_DISABLED
このユーザー ID は使用不可になっています。

CWB_INVALID_PASSWORD
パスワード中の 1 つまたは複数の文字が無効であるか、パスワードが長すぎます。

CWB_INVALID_USERID
ユーザー ID 中の 1 つまたは複数の文字が無効であるか、ユーザー ID が長すぎます。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_API_ERROR
一般 API 障害。

CWB_USER_CANCELLED
ユーザーがサインオン処理を取り消しました。

その他の共通の戻りコードが、サインオン・サーバーへ接続しようとして失敗した結果として戻されることがよくあります。そのような戻りコードのリストについては、cwbCO_Connect の注釈を参照してください。

使用法

ユーザー検証時に System i がユーザーのパスワードを求めるプロンプトと、実際に System i への問い合わせを行うプロンプトを出すかどうかは、いずれも現行のシステム・オブジェクトの設定 (ユーザー ID、パスワード、プロンプト・モード、デフォルトのユーザー・モード、検証モードなど) に左右されます。詳細については、これらの属性の取得と設定についての API の宣言を参照してください。指定のシステム・オブジェクトの System i 名がシステム・リストに存在しない場合、システム・オブジェクトのパーシスタンス・モードが適切に設定されていれば、cwbCO_Connect または cwbCO_Signon の最初の呼び出しが正常に行われた際に、このシステム・オブジェクトの System i 名がシステム・リストに追加されます。

パーシスタンス・モードの詳細については、cwbCO_SetPersistenceMode の注釈を参照してください。呼び出しが正常に行われて、System i のパスワード・キャッシングが使用可能になっている場合、パスワードが結果のユーザー ID 用パスワードとして、PC の System i パスワード・キャッシュに保管されます。

以下の項も参照してください。

- 103 ページの『cwbCO_Signon と cwbCO_VerifyUserIDPassword の相違点』

- 104 ページの『cwbCO_Signon と cwbCO_VerifyUserIDPassword の類似点』

関連資料

47 ページの『システム・オブジェクトの属性』
System i プラットフォームにおける**システム・オブジェクトの属性**は、システム・オブジェクトが表すシステムへのサインオンおよび通信の動作に影響します。

cwbCO_VerifyUserIDPassword:

System i Access for Windows の cwbCO_VerifyUserIDPassword コマンドを使用します。

目的

この関数は、指定のシステム・オブジェクトで表されるシステムの System i ユーザー ID およびパスワードの正確さを検証します。ユーザー ID とパスワードが正しい場合は、この関数は、サインオンの試行とパスワードの有効期限に関するデータも検索します。

注: cwbCO_VerifyUserIDPassword API に誤ったパスワードを渡した場合は、指定されたユーザーの無効サインオン試行カウンターが増やされます。無効なパスワードをホストにあまり多く送信すると、そのユーザー・プロファイルは使用できなくなります。

構文

```
UINT CWB_ENTRY cwbCO_VerifyUserIDPassword(
                            cwbCO_SysHandle       system,
                            LPCSTR                userID,
                            LPCSTR                password,
                            cwbSV_ErrHandle       errorHandle );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

LPCSTR userID - input
NULL で終わるユーザー ID が含まれているバッファーを指すポインター。これは、終了の NULL を含めずに、長さが CWBCO_MAX_USER_ID 文字を超えてはなりません。

LPCSTR password - input
NULL で終わるパスワードが含まれているバッファーを指すポインター。最大長は、NULL 終了文字を含めて、CWBCO_MAX_PASSWORD + 1 バイトです。

cwbSV_ErrHandle errorHandle - input/output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合、または errorHandle が無効な場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
API に与えられたポインターが無効。

CWB_UNKNOWN_USERID
与えられたユーザー ID がこのシステムでは認知されていません。

CWB_WRONG_PASSWORD
パスワードが正しくありません。

CWB_PASSWORD_EXPIRED
パスワードの有効期限切れ。

CWB_USER_PROFILE_DISABLED
このユーザー ID は使用不可になっています。

CWB_INVALID_PASSWORD
パスワード中の 1 つまたは複数の文字が無効であるか、パスワードが長すぎます。

CWB_INVALID_USERID
ユーザー ID 中の 1 つまたは複数の文字が無効であるか、ユーザー ID が長すぎます。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_API_ERROR
一般 API 障害。

使用法

有効なパスワードの長さは、System i のパスワード・レベルの現在の設定に応じて決まります。パスワード・レベル 0 および 1 では、最高 10 文字までの長さのパスワードを許可します。パスワード・レベル 2 および 3 では、最高 128 文字までの長さのパスワードを許可します。

103 ページの『cwbCO_Signon と cwbCO_VerifyUserIDPassword の相違点』および 104 ページの『cwbCO_Signon と cwbCO_VerifyUserIDPassword の類似点』を参照してください。

通信およびセキュリティー: 属性取得および属性設定の API

この System i Access forWindows の API を使用して、他のシステム・オブジェクトの属性を取得および設定したり、属性がポリシーで制限されているかどうかを判別したりします。

cwbCO_CanModifyDefaultUserMode:

System i Access for Windows の cwbCO_CanModifyDefaultUserMode コマンドを使用します。

目的

指定されたシステム・オブジェクトのデフォルトのユーザー・モードが変更可能であるかどうかを示します。

構文

```
UINT CWB_ENTRY cwbCO_CanModifyDefaultUserMode(
                            cwbCO_SysHandle         system,
                            cwb_Boolean            *canModify );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwb_Boolean *canModify - output
このモードが変更可能であれば CWB_TRUE に設定し、そうでない場合は CWB_FALSE に設定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
canModify ポインターが NULL です。

使用法

ポリシー設定で変更を禁止している場合、あるいは指定されたシステム・オブジェクトを使用しているサインオンまたは接続が既に成功している場合には、この値は変更できません。これらのケースでは、canModify は CWB_FALSE に設定されます。この API から戻された結果が正しいのは、呼び出し時点のみです。

ポリシー設定が変更されるか、このシステム・オブジェクトを使用したサインオンや接続が行われた場合には、この API の結果は誤ったものになる可能性があります。特にマルチスレッドのアプリケーションの場合、この点を考慮に入れて、対処する必要があります。

cwbCO_CanModifyIPAddress:

System i Access for Windows の cwbCO_CanModifyIPAddress コマンドを使用します。

目的

接続に使用される IP アドレスがこのシステム・オブジェクト用に変更可能であるかどうかを示します。

構文

```
UINT CWB_ENTRY cwbCO_CanModifyIPAddress(
                            cwbCO_SysHandle         system,
                            cwb_Boolean            *canModify );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwb_Boolean *canModify - output
IP アドレスが変更可能であれば CWB_TRUE に設定し、そうでない場合は CWB_FALSE に設定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
canModify ポインターが NULL です。

使用法

ポリシー設定で変更を禁止している場合、あるいは指定されたシステム・オブジェクトを使用しているサインオンまたは接続が既に成功している場合には、この値は変更できません。これらのケースでは、canModify は CWB_FALSE に設定されます。 IP アドレス・ルックアップ・モードが CWBCO_IPADDR_LOOKUP_NEVER ではなく、ポリシー設定で IP アドレス・ルックアップ・モードの変更が禁止されている場合は、この値は変更できません。そのケースでは、canModify は CWB_FALSE に設定されます。この API から戻された結果が正しいのは、呼び出し時点のみです。ポリシー設定が変更されるか、このシステム・オブジェクトを使用したサインオンや接続が行われた場合には、この API の結果は誤ったものになる可能性があります。特にマルチスレッドのアプリケーションの場合、この点を考慮に入れて、対処する必要があります。

cwbCO_CanModifyIPAddressLookupMode:
目的

IP アドレス・ルックアップ・モードがこのシステム・オブジェクト用に変更可能であるかどうかを示します。

構文

```
UINT CWB_ENTRY cwbCO_CanModifyIPAddressLookupMode(
                              cwbCO_SysHandle         system,
                              cwb_Boolean            *canModify );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwb_Boolean *canModify - output
このモードが変更可能であれば CWB_TRUE に設定し、そうでない場合は CWB_FALSE に設定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
canModify ポインターが NULL です。

使用法

ポリシー設定で変更を禁止している場合、あるいは指定されたシステム・オブジェクトを使用しているサインオンまたは接続が既に成功している場合には、この値は変更できません。これらのケースでは、canModify は CWB_FALSE に設定されます。この API から戻された結果が正しいのは、呼び出し時点のみです。

ポリシー設定が変更されるか、このシステム・オブジェクトを使用したサインオンや接続が行われた場合には、この API の結果は誤ったものになる可能性があります。特にマルチスレッドのアプリケーションの場合、この点を考慮に入れて、対処する必要があります。

cwbCO_CanModifyPersistenceMode:

System i Access for Windows の cwbCO_CanModifyPersistenceMode コマンドを使用します。

目的

指定されたシステム・オブジェクトのパーシスタンス・モードが変更可能であるかどうかを示します。

構文

```
UINT CWB_ENTRY cwbCO_CanModifyPersistenceMode(
                            cwbCO_SysHandle         system,
                            cwb_Boolean            *canModify );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwb_Boolean *canModify - output
このモードが変更可能であれば CWB_TRUE に設定し、そうでない場合は CWB_FALSE に設定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
canModify ポインターが NULL です。

使用法

ポリシー設定で変更を禁止している場合、あるいは指定されたシステム・オブジェクトを使用しているサインオンまたは接続が既に成功している場合には、この値は変更できません。これらのケースでは、canModify は CWB_FALSE に設定されます。この API から戻された結果が正しいのは、呼び出し時点のみです。ポリシー設定が変更されるか、このシステム・オブジェクトを使用したサインオンや接続が行われた場合には、この API の結果は誤ったものになる可能性があります。特にマルチスレッドのアプリケーションの場合、この点を考慮に入れて、対処する必要があります。

cwbCO_CanModifyPortLookupMode:

System i Access for Windows の cwbCO_CanModifyPortLookupMode コマンドを使用します。

目的

指定されたシステム・オブジェクトのポート・ルックアップ・モードが変更可能であるかどうかを示します。

構文

```
UINT CWB_ENTRY cwbCO_CanModifyPortLookupMode(
                            cwbCO_SysHandle         system,
                            cwb_Boolean            *canModify );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwb_Boolean *canModify - output
このモードが変更可能であれば CWB_TRUE に設定し、そうでない場合は CWB_FALSE に設定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
canModify ポインターが NULL です。

使用法

ポリシー設定で変更を禁止している場合、あるいは指定されたシステム・オブジェクトを使用しているサインオンまたは接続が既に成功している場合には、この値は変更できません。これらのケースでは、canModify は CWB_FALSE に設定されます。この API から戻された結果が正しいのは、呼び出し時点のみです。ポリシー設定が変更されるか、このシステム・オブジェクトを使用したサインオンや接続が行われた場合には、この API の結果は誤ったものになる可能性があります。特にマルチスレッドのアプリケーションの場合、この点を考慮に入れて、対処する必要があります。

cwbCO_CanModifyUseSecureSockets:

System i Access for Windows の cwbCO_CanModifyUseSecureSockets コマンドを使用します。

目的

セキュア・ソケット使用の設定値がこのシステム・オブジェクト用に変更可能であるかどうかを示します。

構文

```
UINT CWB_ENTRY cwbCO_CanModifyUseSecureSockets(
                            cwbCO_SysHandle         system,
                            cwb_Boolean            *canModify );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwb_Boolean *canModify - output
セキュア・ソケット使用設定値が変更可能であれば CWB_TRUE に設定し、そうでない場合は CWB_FALSE に設定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
canModify ポインターが NULL です。

使用法

ポリシー設定で変更を禁止している場合、あるいは指定されたシステム・オブジェクトを使用しているサインオンまたは接続が既に成功している場合には、この値は変更できません。これらのケースでは、canModify は CWB_FALSE に設定されます。この API から戻された結果が正しいのは、呼び出し時点のみです。ポリシー設定が変更されるか、このシステム・オブジェクトを使用したサインオンや接続が行われた場合には、この API の結果は誤ったものになる可能性があります。特にマルチスレッドのアプリケーションの場合、この点を考慮に入れて、対処する必要があります。

cwbCO_GetDescription:

System i Access for Windows の cwbCO_GetDescription コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトに関連したテキスト記述を取得します。

構文

```
UINT CWB_ENTRY cwbCO_GetDescription(
                              cwbCO_SysHandle    system,
                              LPSTR              description,
                              PULONG             length );
```

パラメーター

cwbCO_SysHandle system - input
以前に、cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

LPSTR description - output
NULL で終わる記述が含まれているバッファーを指すポインター。記述の長さは、終了文字の NULL を含まずに、最大 CWBCO_MAX_SYS_DESCRIPTION 文字までです。

PULONG length - input/output
記述バッファーの長さを指すポインター。バッファーが、終了文字の NULL のスペースを含めて、記述を含めるためには小さすぎる場合は、必要とするバッファーのサイズがこのパラメーターに入れられます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
渡されたポインター・パラメーターのいずれかが NULL です。

CWB_BUFFER_OVERFLOW
記述バッファーが、記述全体を保持するには十分な大きさではありません。

cwbCO_GetHostCCSID:

System i Access for Windows の cwbCO_GetHostCCSID コマンドを使用します。

目的

システム・オブジェクト内のユーザー ID によって表される、System i に関連付けられた CCSID のうち、システムへのサインオンが行われたときに使用中だったものを戻します。

構文

```
UINT CWB_ENTRY cwbCO_GetHostCCSID(
                              cwbCO_SysHandle         system,
                              PULONG                  pCCSID );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

PULONG pCCSID - output
成功すれば、ホスト CCSID はここへコピーされます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
CCSID ポインターが NULL です。

CWB_DEFAULT_HOST_CCSID_USED
この API は、システム・オブジェクトに設定されているユーザー ID に適切なホスト CCSID を判別できないため、ホスト CCSID 500 が戻されます。

CWB_USER_TIMEOUT

CWB_SSL_JAVA_ERROR

CWB_USER_TIMEOUT_SENDRCV

使用法

この API は、関連する CCSID 値を検索するのに、ホスト・システムへの活動接続を行わず、またそれが必要でもありません。しかしながら、この検索は、指定されたシステム・オブジェクトで設定されているものと同じユーザー ID を使用することによって、前回成功したホスト・システムへの接続に依存しています。これは、System i のデフォルト CCSID ではなく、特定のユーザー・プロファイルの CCSID が戻されるためです。ユーザー ID を必要とせずにホスト CCSID を検索するには、cwbNL_GetHostCCSID を呼び出します。

cwbCO_GetHostVersionEx:

System i Access for Windows の cwbCO_GetHostVersionEx コマンドを使用します。

目的

ホストのバージョンとリリース・レベルを取得します。

構文

```
UINT CWB_ENTRY cwbCO_GetHostVersionEx(
                            cwbCO_SysHandle         system,
                            PULONG                  version,
                            PULONG                  release);
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

PULONG version - output
システムのバージョン・レベルが戻されるバッファーを指すポインター。

PULONG release - output
システムのリリース・レベルが戻されるバッファーを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_CONNECTED
現在活動中である環境の使用中に、このシステムは一度も接続されていません。

CWB_INVALID_POINTER
渡されたポインターの 1 つが NULL です。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

使用法

System i 接続が行われると必ず、ホストのバージョンが検索され、保管されます。現在活動中の環境に System i 接続が存在しない場合、この情報は使用できず、エラー・コード CWB_NOT_CONNECTED が戻されます。 System i への接続が正常に完了したことが分かっている場合は、戻されたバージョンとリリース・レベルはおそらく現行のものになります。この値が使用可能であり、最近検索されたものであることを確認したい場合は、このシステム・オブジェクトの cwbCO_Signon または cwbCO_Connect を最初に呼び出し、その後、cwbCO_GetHostVersionEx を呼び出します。

cwbCO_GetIPAddress:

System i Access for Windows の cwbCO_GetIPAddress コマンドを使用します。

目的

この関数は、指定のシステム・オブジェクトによって表される、System i の IP アドレスを取得します。この IP アドレスは、System i 接続で使用されていた (あるいは、cwbCO_SetIPAddress を使用するなどの方法で設定された) ものであり、指定のシステム・オブジェクトを使用する場合に、今後の接続用に使用されます。

構文

```
UINT CWB_ENTRY cwbCO_GetIPAddress(
                              cwbCO_SysHandle    system,
                              LPSTR              IPAddress,
                              PULONG             length );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

LPSTR IPAddress - output
ドット表記法 (″nnn.nnn.nnn.nnn″ の形式、ここでそれぞれの ″nnn″ は 0 から 255 までの範囲) で表した NULL で終わる IP アドレスが含まれているバッファーを指すポインター。

PULONG length - input/output
IPAddress バッファーの長さを指すポインター。バッファーが、終了の NULL のスペースを含めて、出力を入れるには小さすぎる場合は、必要とするバッファーのサイズがこのパラメーターに入れられ、CWB_BUFFER_OVERFLOW が戻されます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
入力ポインターの 1 つが NULL です。

CWB_BUFFER_OVERFLOW
IPAddress バッファーが、IPAddress のストリング全体を含めるには十分な大きさではありません。

使用法

なし

cwbCO_GetIPAddressLookupMode:

System i Access for Windows の cwbCO_GetIPAddressLookupMode コマンドを使用します。

目的

この関数は、指定のシステム・オブジェクトによって表される System i IP アドレスが動的にルックアップされる場合、そのルックアップがいつ行われるかを示す指示を取得します。

構文

```
UINT CWB_ENTRY cwbCO_GetIPAddressLookupMode(
                            cwbCO_SysHandle              system,
                            cwbCO_IPAddressLookupMode   *mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_IPAddressLookupMode * mode - output
現在、使用中の IP アドレス・ルックアップ・モードを戻します。指定できる値とその意味については、 98 ページの『cwbCO_SetIPAddressLookupMode』の注釈を参照してください。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
モード・ポインターが NULL です。

使用法

なし

cwbCO_GetPortLookupMode:

System i Access for Windows の cwbCO_GetPortLookupMode コマンドを使用します。

目的

この関数は、指定のシステム・オブジェクトに関して、System i Access for Windows のサービス接続を確立するためにホスト・サービス・ポートが必要になったときに、これらのポートをルックアップするモードや方式を取得します。

構文

```
UINT CWB_ENTRY cwbCO_GetPortLookupMode(
                            cwbCO_SysHandle         system,
                            cwbCO_PortLookupMode   *mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_PortLookupMode * mode - output
ホスト・サービス・ポートのルックアップ・モードを戻します。指定できる値とその意味については、cwbCO_SetPortLookupMode の注釈を参照してください。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
モード・ポインターが NULL です。

使用法

なし

cwbCO_GetSystemName:

System i Access for Windows の cwbCO_GetSystemName コマンドを使用します。

目的

この関数は、指定のシステム・オブジェクトに関連付けられた System i 名を取得します。

構文

```
UINT CWB_ENTRY cwbCO_GetSystemName(
                              cwbCO_SysHandle    system,
                              LPSTR              sysName,
                              PULONG             length );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

LPSTR sysName - output
NULL で終わるシステム名が含まれるバッファーを指すポインター。名前の長さは、終了の NULL を含まずに、最大で、CWBCO_MAX_SYS_NAME 文字になります。

PULONG length - input/output
sysName バッファーの長さを指すポインター。バッファーが、終了の NULL のスペースを含めて、システム名を含めるには小さすぎる場合は、必要とするバッファーのサイズがこのパラメーターに入れられ、 CWB_BUFFER_OVERFLOW が戻されます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
渡されたポインター・パラメーターのいずれかが NULL です。

CWB_BUFFER_OVERFLOW
sysName バッファーが、システム名全体を保持するには十分な大きさではありません。

使用法

なし

cwbCO_IsSecureSockets:

System i Access for Windows の cwbCO_IsSecureSockets コマンドを使用します。

目的

この関数は、(指定されたシステム・オブジェクトについて) セキュア・ソケットが使用されているかどうか (接続されている場合)、あるいはセキュア・ソケットを使用して接続しようとしているかどうか (現在は接続されていない場合) についての情報を取得します。

構文

```
UINT CWB_ENTRY cwbCO_IsSecureSockets(
                            cwbCO_SysHandle    system,
                            cwb_Boolean       *inUse );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwb_Boolean * inUse - output
System i Access が通信にセキュア・ソケットを使用しているかどうか (または使用しようとしているかどうか) の情報を戻します。

CWB_TRUE
接続が活動中である場合、現在使用中、あるいは使用する予定です。

CWB_FALSE
使用中ではなく、今後も使用する予定はありません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
inUse ポインターが NULL です。

使用法

このフラグは、System i Access for Windows が今後の通信に関して試行する内容を示しています。CWB_TRUE が戻される場合、System i の通信に対する試みのうち、セキュア・ソケットを使用して実行できないものは、すべて失敗します。

System i Access for Windows 製品で SSL が使用される場合、Federal Information Processing Standards (FIPS) に強制的に準拠することになりますが (一定の制限付き)、FIPS 準拠がオンまたはオフのいずれになっているかについては、この API では戻されません。 FIPS 準拠がオンまたはオフのいずれであるかを確認する唯一の方法は、System i Access for Windows プロパティーの FIPS 準拠のチェック・ボックスを確認することです。 FIPS とその使用法について詳しくは、「System i Access for Windows ユーザーズ・ガイド」(本製品と同時にインストールされます) を参照してください。

cwbCO_SetIPAddress:

System i Access for Windows の cwbCO_SetIPAddress コマンドを使用します。

目的

この関数は、指定のシステム・オブジェクトに、System i 接続で使用される IP アドレスを設定します。また、この関数は、システム・オブジェクトの IP アドレス・ルックアップ・モードを

CWBCO_IPADDR_LOOKUP_NEVER に変更します。この変更は、既存の、あるいは後で作成される他のシステム・オブジェクトのいずれにも影響することはありません。

構文

```
UINT CWB_ENTRY cwbCO_SetIPAddress(
                              cwbCO_SysHandle    system,
                              LPCSTR             IPAddress );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

LPCSTR IPAddress - input
IP アドレスを文字ストリングとして、ドット表記法 (″nnn.nnn.nnn.nnn″) で指定します。ここで、それぞれの ″nnn″ は、0 から 255 までの 10 進数です。 IPAddress は、終了の NULL 文字を含まずに、CWBCO_MAX_IP_ADDRESS 文字よりも長くなってはなりません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_PARAMETER
IPAddress パラメーターが有効な IP アドレスを含んでいません。

CWB_RESTRICTED_BY_POLICY
この値の変更を禁止するポリシーが存在します。

CWB_INV_AFTER_SIGNON
指定されたシステム・オブジェクトを使用して、サインオンが正常に行われたため、この設定値は変更されることはありません。

使用法

この API は、指定されたシステム・オブジェクトについて、サインオンが正常に行われた後では使用することができません。このシステム・オブジェクトについて、 cwbCO_Signon または cwbCO_Connect のいずれかの呼び出しが正常に行われた場合には、サインオンが行われています。

指定されたシステム・オブジェクトを使用して何らかの接続が行われる場合にはいつでも、特定の IP アドレスを強制的に使用させるために、この API を使用します。 IP アドレス・ルックアップ・モードが IP アドレスをルックアップしないように設定されているため、接続やサインオンが行われる前に、 IP アドレス・ルックアップ・モードが cwbCO_SetIPAddressLookupMode 呼び出しによって変更されない限り、指定されたアドレスが常に使用されます。

cwbCO_SetIPAddressLookupMode:

System i Access for Windows の cwbCO_SetIPAddressLookupMode コマンドを使用します。

目的

この関数は、指定のシステム・オブジェクトについて、このオブジェクトが表すシステムに接続を行う際に、System i の IP アドレスの動的ルックアップが行われる時点を指定します。 cwbCO_CreateSystem または cwbCO_CreateSystemLike の呼び出し時に指定されるシステム名が、実際の IP アドレスである場合、System i Access for Windows 製品でアドレスをルックアップする必要はないため、この設定は無視されます。

構文

```
UINT CWB_ENTRY cwbCO_SetIPAddressLookupMode(
                            cwbCO_SysHandle            system,
                            cwbCO_IPAddressLookupMode  mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_IPAddressLookupMode mode - input
動的アドレス・ルックアップをいつ実行するかを指定します。指定できる値は以下のとおりです。

CWBCO_IPADDR_LOOKUP_ALWAYS
接続が行われるたびに、System i の IP アドレスを動的にルックアップします。

CWBCO_IPADDR_LOOKUP_1HOUR
このシステムでの前回のルックアップから 1 時間以上経過している場合には、IP アドレスを動的にルックアップします。

CWBCO_IPADDR_LOOKUP_1DAY
このシステムでの前回のルックアップから 1 日以上経過している場合には、IP アドレスを動的にルックアップします。

CWBCO_IPADDR_LOOKUP_1WEEK
このシステムでの前回のルックアップから 1 週間以上経過している場合には、IP アドレスを動的にルックアップします。

CWBCO_IPADDR_LOOKUP_NEVER
このシステムの System i IP アドレスの動的ルックアップを行いません。この PC で最近使用した IP アドレスが、常にシステムで使用されます。

CWBCO_IPADDR_LOOKUP_AFTER_STARTUP
この システムでの前回のルックアップ以降、Windows が再始動された場合には、IP アドレスを動的にルックアップします。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_PARAMETER
モード・パラメーターの値が無効。

CWB_RESTRICTED_BY_POLICY
この値の変更を禁止するポリシーが存在します。

CWB_INV_AFTER_SIGNON
指定されたシステム・オブジェクトを使用して、サインオンが正常に行われたため、この設定値は変更されることはありません。

使用法

この API は、指定されたシステム・オブジェクトについて、サインオンが正常に行われた後では使用することができません。このシステム・オブジェクトについて、 cwbCO_Signon または cwbCO_Connect のいずれかの呼び出しが正常に行われた場合には、サインオンが行われています。

CWB_IPADDR_LOOKUP_ALWAYS 以外の値に設定すると、System i の接続時間を短縮できる場合があります。動的ルックアップによって、ネットワーク・トラフィックが増大し、完了までに時間がかかることがあるためです。動的ルックアップを行わない場合、System i の IP アドレスが変更されて、接続が失敗したり、間違ったシステムに接続されたりするというリスクが発生します。

cwbCO_SetPortLookupMode:

System i Access for Windows の cwbCO_SetPortLookupMode コマンドを使用します。

目的

この関数は、指定されたシステム・オブジェクトについて、ホスト・サーバーのポート・ルックアップがどのように行われるかを設定します。

構文

```
UINT CWB_ENTRY cwbCO_SetPortLookupMode(
                            cwbCO_SysHandle       system,
                            cwbCO_PortLookupMode  mode );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。これは System i の ID です。

cwbCO_PortLookupMode mode - input
ポート・ルックアップ方式を指定します。指定できる値は以下のとおりです。

CWBCO_PORT_LOOKUP_SERVER
まだ確立されていないサービスへの接続を行う場合は、その都度 i5/OS ホスト・サーバー・マッパーに連絡が取られて、ホスト・サーバー・ポートのルックアップが行われます。サーバー・マッパーは、希望の System i サービスに接続するために後で使用されるポート番号を戻します。

CWBCO_PORT_LOOKUP_LOCAL
ホスト・サーバー・ポートのルックアップは、PC 自体の SERVICES ファイルをルックアップすることによって行われます。

CWBCO_PORT_LOOKUP_STANDARD
標準ポートを使用して、希望するサーバーに接続します。標準ポートは、所定のホスト・サーバーにデフォルトで設定されており、そのサービスの System i サービス・テーブルに変更が一切加えられていない場合に使用されるポートです。

後の 2 つのモードは、System i マッパー接続とそれに関連した遅れ、ネットワーク・トラフィック、およびシステムの負荷を取り除きます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_PARAMETER
モード・パラメーターの値が無効。

CWB_RESTRICTED_BY_POLICY
この値の変更を禁止するポリシーが存在します。

CWB_INV_AFTER_SIGNON
指定されたシステム・オブジェクトを使用して、サインオンが正常に行われたため、この設定値は変更されることはありません。

使用法

この API は、指定されたシステム・オブジェクトについて、サインオンが正常に行われた後では使用することができません。このシステム・オブジェクトについて、 cwbCO_Signon または cwbCO_Connect のいずれかの呼び出しが正常に行われた場合には、サインオンが行われています。

サービス用のポート番号をきわめて正確なものにするためには、CWBCO_PORT_LOOKUP_SERVER を使用します。ただし、この場合、サービスへの新たな接続が行われるたびに、システムのサーバー・マッパーへの余分な接続が必要になります。

CWBCO_PORT_LOOKUP_STANDARD を使用すると最良のパフォーマンスが得られます。ただし、システム管理者がシステムのサービス・テーブルで、いずれかの i5/OS ホスト・サービスのポートを変更した場合、このモードは機能しません。

システム・オブジェクトによって表されるシステムで System i Access ホスト・サービスのポートが変更されている場合、最良のパフォーマンスを得るためには CWBCO_PORT_LOOKUP_LOCAL を使用します。これを機能させるためには、それぞれのホスト・サービス・ポートごとの記入項目を、SERVICES という名の PC のファイルに追加する必要があります。そのような各記入項目では、まず最初に、ホスト・サービスの標準名 (例えば、引用符なしの ″as-rmtcmd″) を含み、その後にスペースおよびそのサービスのポート番号が続いている必要があります。 SERVICES ファイルは、Windows インストール・ディレクトリーの下の **system32\drivers\etc** というサブディレクトリーにあります。

cwbCO_UseSecureSockets:

System i Access for Windows の cwbCO_UseSecureSockets コマンドを使用します。

目的

システム・オブジェクトで表されるシステムに対するすべての System i 通信で、セキュア・ソケットを使用するか使用しないかを指定します。

構文

```
UINT CWB_ENTRY cwbCO_UseSecureSockets(
                            cwbCO_SysHandle    system,
                            cwb_Boolean        useSecureSockets );
```

パラメーター

cwbCO_SysHandle system - input
以前に cwbCO_CreateSystem または cwbCO_CreateSystemLike から戻されたハンドル。 iServer システムを識別します。

cwb_Boolean useSecureSockets - input
指定のシステム・オブジェクト・ハンドルが表しているシステムと通信を行う際に、セキュア・ソケットの使用が必要かどうかを指定します。次のうち、適切な値を使用します。

CWB_TRUE
通信にセキュア・ソケットの使用が必要。

CWB_FALSE
通信にセキュア・ソケットは使用しない。

CWB_USER_TIMEOUT
システム・オブジェクトに関連した接続タイムアウト値が、接続の検証が完了する前に有効期限が切れました。そのため、待機を停止します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_SECURE_SOCKETS_NOTAVAIL
セキュア・ソケットが使用不可。セキュア・ソケットが、PC にインストールされていないか、このユーザーでは禁止されているか、あるいは iServer システムでは使用できません。

CWB_RESTRICTED_BY_POLICY
この値の変更を禁止するポリシーが存在します。

CWB_INV_AFTER_SIGNON
指定されたシステム・オブジェクトを使用して、サインオンが正常に行われたため、この設定値は変更されることはありません。

使用法

指定されたサービスへの接続が所定のシステム・オブジェクト用に既に存在している場合でも、新規の接続は試行されます。所定のシステム・オブジェクトの属性 (セキュア・ソケットを使用するかどうかなど) が、この接続の試行に使用されます。その結果、渡されたシステム・オブジェクトを指定した接続の検証は

失敗する可能性がありますが、属性の設定が異なるシステム・オブジェクトを指定した同じシステムでは成功する場合があります。これが最も明確に現れるのは、セキュア・ソケットの使用が関係してくる場合です。これは、非セキュア・ソケット・バージョンのサービスはシステムで稼働する可能性があるものの、セキュア・ソケット・バージョンのサービスは稼働しない可能性がある、またはその逆の場合が考えられるためです。

この API が呼び出された時、System i Access for Windows 製品において、System i 接続時に使用可能なセキュア・ソケットを検出できない場合があります。 CWB_SECURE_SOCKETS_NOTAVAIL が戻されない場合であっても、後でセキュア・ソケットが使用不可であることが判明する場合もあります。

System i Access for Windows 製品で SSL が使用される場合、Federal Information Processing Standards (FIPS) に強制的に準拠することになりますが (一定の制限付き)、FIPS 準拠がオンまたはオフのいずれになっているかについては、この API では戻されません。 FIPS 準拠がオンまたはオフのいずれであるかを確認する唯一の方法は、System i Access for Windows プロパティーの FIPS 準拠のチェック・ボックスを確認することです。 FIPS とその使用法について詳しくは、「System i Access for Windows ユーザーズ・ガイド」(本製品と同時にインストールされます) を参照してください。

cwbCO_Service の定義

System i Access for Windows の cwbCO_Service を定義する値を、以下に示します。

- CWBCO_SERVICE_CENTRAL
- CWBCO_SERVICE_NETFILE
- CWBCO_SERVICE_NETPRINT
- CWBCO_SERVICE_DATABASE
- CWBCO_SERVICE_ODBC
- CWBCO_SERVICE_DATAQUEUES
- CWBCO_SERVICE_REMOTECMD
- CWBCO_SERVICE_SECURITY
- CWBCO_SERVICE_DDM
- CWBCO_SERVICE_WEB_ADMIN
- CWBCO_SERVICE_TELNET
- CWBCO_SERVICE_MGMT_CENTRAL
- CWBCO_SERVICE_ANY
- CWBCO_SERVICE_ALL

cwbCO_Signon と cwbCO_VerifyUserIDPassword の相違点

以下に挙げるのは、System i Access for Windows の cwbCO_Signon コマンドと cwbCO_VerifyUserIDPassword コマンドとの主要な相違点です。

- cwbCO_VerifyUserIDPassword は、ユーザー ID とパスワードが渡されることが必要であり (これらのためのシステム・オブジェクト値は使用されません)、この情報についてのプロンプトを出すことはありません。 cwbCO_Signon は、他のシステム・オブジェクトの設定次第ではプロンプトを出すことがあります。その場合には、その検証を行おうとしたときにユーザーから与えられたユーザー ID とパスワードの値はすべて使用します。
- cwbCO_VerifyUserIDPassword は、ユーザー ID とパスワードを求めてプロンプトを出すことはないため、指定されたシステム・オブジェクトの設定値がこの呼び出しの結果、変更されることはありませ

ん。しかし、cwbCO_Signon への呼び出しでは、この情報に対するプロンプトが出される可能性があり、その結果として、システム・オブジェクトのユーザー ID とパスワードは変更されることがあります。

- cwbCO_VerifyUserIDPassword を使用すると、System i 接続の確立、ユーザー ID とパスワードの検証、およびサインオンの試行に関連する現行値 (前回のサインオンが正常に行われた日時など) の検索が、常に実行されるようになります。しかし、cwbCO_Signon はユーザー ID とパスワードを検証するために接続しない可能性があり、その代わりに先行の検証における最近の結果を使用することがあります。これは、所定のシステム・オブジェクトの検証モード属性のほか、先行の検証結果がどの程度新しいかによって影響を受けます。
- cwbCO_Signon が正常終了した場合にのみ、パスワードが System i パスワード・キャッシュに入れられます。cwbCO_VerifyUserIDPassword の呼び出しの結果として、キャッシュに入れられることはありません。
- cwbCO_VerifyUserIDPassword は、システム・オブジェクトの状態を「サインオン」に設定することはありません。それに対して、cwbCO_Signon は成功すれば状態を「サインオン」に変更します。システム・オブジェクトが「サインオン」状態にある場合、その属性の大部分はもはや変更されないため、このことは重要です。

cwbCO_Signon と cwbCO_VerifyUserIDPassword の類似点

以下の情報は、System i Access for Windows の cwbCO_Signon コマンドと cwbCO_VerifyUserIDPassword コマンドとの類似点に関するものです。

この両方の API は、ユーザー ID とパスワードの検証を行うために接続を使用する場合、サインオンの試行に関連した現行データの検索も行います。このデータは、以下の API を使用して検索することができます。

- cwbCO_GetSignonDate
- cwbCO_GetPrevSignonDate
- cwbCO_GetPasswordExpireDate
- cwbCO_GetFailedSignons

通信: 作成および削除の API

これらの System i Access for Windows API を使用して、現在活動中の環境もしくは別の環境で、構成済みシステムのリストを作成します。リストの項目数の検索と、各項目の順序どおりの検索を行います。

cwbCO_CreateSysListHandle:

System i Access for Windows の cwbCO_CreateSysListHandle コマンドを使用します。

目的

活動中の環境の構成済みシステム名リストへのハンドルを作成します。

構文

```
unsigned int CWB_ENTRY cwbCO_CreateSysListHandle(
                            cwbCO_SysListHandle *listHandle,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbCO_SysListHandle *listHandle - output
出力の際に返されるリスト・ハンドルを指すポインター。このハンドルは、他の呼び出しでリストを使用する時に必要になります。

cwbSV_ErrorHandle errorHandle - input
API 呼び出しが失敗した場合、このハンドルに関連したメッセージ・オブジェクトに、エラーを説明するメッセージ・テキストが書き込まれます。このパラメーターがゼロの場合は、メッセージの利用はできません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_POINTER
リスト・ハンドルを指すポインターが NULL です。

使用法

cwbCO_DeleteSysListHandle を呼び出して、この API で割り振られた資源を解放する必要があります。

cwbCO_CreateSysListHandleEnv:

System i Access for Windows の cwbCO_CreateSysListHandleEnv コマンドを使用します。

目的

この関数は、指定された環境の、構成済みシステム名のリストへのハンドルを作成します。

構文

```
unsigned int CWB_ENTRY cwbCO_CreateSysListHandleEnv(
                            cwbCO_SysListHandle *listHandle,
                            cwbSV_ErrHandle      errorHandle,
                            LPCSTR               pEnvironment );
```

パラメーター

cwbCO_SysListHandle *listHandle - output
出力の際に返されるリスト・ハンドルを指すポインター。このハンドルは、他の呼び出しでリストを使用する際に必要になります。

cwbSV_ErrorHandle errorHandle - input
API 呼び出しが失敗した場合、このハンドルに関連したメッセージ・オブジェクトに、エラーを説明するメッセージ・テキストが書き込まれます。このパラメーターがゼロの場合は、メッセージの利用はできません。

LPCSTR pEnvironment
必要な環境名が入っているストリングを指すポインター。 pEnvironment が NULL ポインター、すなわち NULL ストリング (″¥0″) を指すポインターである場合、現在活動中の環境のシステム・リストが戻ります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_INVALID_POINTER
リスト・ハンドルを指すポインターが NULL です。

CWBCO_NO_SUCH_ENVIRONMENT
指定の環境は存在しません。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

cwbCO_DeleteSysListHandle を呼び出して、この API で割り振られた資源を解放する必要があります。

cwbCO_DeleteSysListHandle:

System i Access for Windows の cwbCO_DeleteSysListHandle コマンドを使用します。

目的

構成済みシステム名のリストへのハンドルを削除します。この関数は、システム名のリストの使用を終了した際に、呼び出す必要があります。

構文

```
unsigned int CWB_ENTRY cwbCO_DeleteSysListHandle(
                            cwbCO_SysListHandle listHandle);
```

パラメーター

cwbCO_SysListHandle - listHandle
削除するシステム名へのハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

使用法

cwbCO_CreateSysListHandle または cwbCO_CreateSysListHandleEnv の API で作成されたリストを削除するために、この API を使用します。

cwbCO_GetNextSysName:

System i Access for Windows の cwbCO_GetNextSysName コマンドを使用します。

目的

システムのリストから、次の順番のシステムの名前を取得します。

構文

```
unsigned int CWB_ENTRY cwbCO_GetNextSysName(
                            cwbCO_SysListHandle listHandle,
                            char               *systemName,
                            unsigned long       bufferSize,
                            unsigned long      *needed);
```

パラメーター

cwbCO_SysListHandle handleList - input
システムのリストへのハンドル。

char *systemName - output
システム名が含まれているバッファーを指すポインター。このバッファーは、終了の NULL 文字を含めて、少なくとも、CWBCO_MAX_SYS_NAME + 1 文字を保持することのできる大きさになっている必要があります。

unsigned long bufferSize - input
systemName が指定するバッファーのサイズ。

unsigned long *needed - output
システム名全体を保持するために必要なバイト数。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
システム名を指すポインター、または必要なバッファー・サイズを指すポインターが NULL です。ヒストリー・ログのメッセージをチェックして、どれが NULL かを判別してください。

CWB_BUFFER_OVERFLOW
出力バッファーにシステム名全体を保持することのできる十分なスペースがありません。*needed

を使用して、正しいサイズを判別してください。呼び出し側が、このエラーから回復して継続する処理を行う予定であるため、ヒストリー・ログにエラー・メッセージは記録されません。

CWBCO_END_OF_LIST
システムのリストの最後まで到達しました。システム名が、戻されませんでした。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_API_ERROR
一般 API 障害。

使用法

渡されたシステム・リストが API cwbCO_CreateSystemListHandle を使用して作成された場合、この API 呼び出しの間にユーザーがその環境を除去するか、別の環境へ切り換えることをしない限り、戻されるシステムは現在活動中である環境で構成されたものです。システム・リストを作成するために cwbCO_CreateSysListHandleEnv が呼び出された場合、その環境をユーザーがそれ以降除去したのではない限り、戻されるシステムはその API に渡される環境で構成されます。

cwbCO_GetSysListSize:

System i Access for Windows の cwbCO_GetSysListSize コマンドを使用します。

目的

リストにあるシステム名の数を取得します。

構文

```
unsigned int CWB_ENTRY cwbCO_GetSysListSize(
                            cwbCO_SysListHandle  listHandle,
                            unsigned long       *listSize);
```

パラメーター

cwbCO_SysListHandle listHandle - input
システムのリストのハンドル。

unsigned long *listSize - output
このパラメーターは、出力時に、リスト内のシステムの数に設定されます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_API_HANDLE
システム・ハンドルが無効。

CWB_INVALID_POINTER
リスト・サイズを指すポインターが NULL です。

使用法

なし

通信: システム情報 API

これらの System i Access for Windows API を使用して、現行のプロセスで構成または接続された個々のシステムに関する情報を入手します。環境名がパラメーターとして渡されるのではない限り、これらの API は、現在、活動中の環境でのみ機能します。

cwbCO_GetActiveConversations:

System i Access for Windows の cwbCO_GetActiveConversations コマンドを使用します。

目的

システムの活動中の会話の数を取得します。

構文

```
int CWB_ENTRY cwbCO_GetActiveConversations(
                                        LPCSTR systemName);
```

パラメーター

LPCSTR systemName - input
システム名が含まれているバッファーを指すポインター。

戻りコード

活動中の会話がある場合には、その数が戻されます。systemName ポインターが NULL であるか、空ストリングを指しているか、システムが現在接続されていないか、あるいは変換できない 1 つまたは複数のユニコード文字が含まれている場合は、0 が戻されます。

使用法

この API は、現行のプロセス内に限り、指定されたシステムで活動中の会話の数を戻します。 PC で実行されている他のプロセス内で、その他の会話が活動中である可能性があります。

cwbCO_GetConnectedSysName:

System i Access for Windows の cwbCO_GetConnectedSysName コマンドを使用します。

目的

索引に対応する、接続されたシステムの名前を取得します。

構文

```
unsigned int CWB_ENTRY cwbCO_GetConnectedSysName(
                              char              *systemName,
                              unsigned long     *bufferSize,
                              unsigned long      index);
```

パラメーター

char *systemName - output
システム名が含まれているバッファーを指すポインター。このバッファーは、終了の NULL 文字を含めて、少なくとも、CWBCO_MAX_SYS_NAME + 1 文字を保持することのできる大きさになっている必要があります。

unsigned long * bufferSize - input/output

入力 *systemName が指すバッファーのサイズ。

出力 必要なバッファー・サイズ。

unsigned long index
名前を検索する、接続されたシステムを示します。最初に接続されたシステムの索引は 0、2 番目の索引は 1、以下同様です。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
システム名を指すポインター、または必要なバッファー・サイズを指すポインターが NULL です。ヒストリー・ログのメッセージをチェックして、どれが NULL かを判別してください。

CWB_BUFFER_OVERFLOW
出力バッファーにシステム名全体を保持することのできる十分なスペースがありません。*bufferSize を使用して、正しいサイズを判別してください。呼び出し側が、このエラーから回復して継続する処理を行う予定であるため、ヒストリー・ログにエラー・メッセージは記録されません。

CWBCO_END_OF_LIST
接続されたシステムのリストの最後まで到達しました。システム名が、戻されませんでした。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_API_ERROR
一般 API 障害。

使用法

システム名を検索することができる接続は、現行プロセス内のものに限られます。

cwbCO_GetDefaultSysName:

System i Access for Windows の cwbCO_GetDefaultSysName コマンドを使用します。

目的

活動中の環境のデフォルト・システムの名前を取得します。

構文

```
unsigned int CWB_ENTRY cwbCO_GetDefaultSysName(
                            char               *defaultSystemName,
                            unsigned long       bufferSize,
                            unsigned long      *needed,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

char *defaultSystemName - output
NULL で終わるシステム名が含まれるバッファーを指すポインター。このバッファーは、終了の NULL 文字を含めて、少なくとも、CWBCO_MAX_SYS_NAME + 1 文字を保持することのできる大きさになっている必要があります。

unsigned long bufferSize - input
入力バッファーのサイズ。

unsigned long *needed - output
終了の NULL を含めて、システム名全体を保持するために必要なバイト数。

cwbSV_ErrorHandle errorhandle - input
API 呼び出しが失敗した場合、このハンドルに関連したメッセージ・オブジェクトに、エラーを説明するメッセージ・テキストが書き込まれます。このパラメーターがゼロの場合は、メッセージの利用はできません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
システム名を指すポインター、または必要なバッファー・サイズを指すポインターが NULL です。ヒストリー・ログのメッセージをチェックして、どれが NULL かを判別してください。

CWB_BUFFER_OVERFLOW
出力バッファーにシステム名全体を保持することのできる十分なスペースがありません。*needed を使用して、正しいサイズを判別してください。呼び出し側が、このエラーから回復して継続する処理を行う予定であるため、ヒストリー・ログにエラー・メッセージは記録されません。

CWBCO_DEFAULT_SYSTEM_NOT_DEFINED
活動中の環境で、デフォルト・システムの設定が定義されていません。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_API_ERROR
一般 API 障害。

使用法

なし

cwbCO_GetHostVersion:

System i Access for Windows の cwbCO_GetHostVersion コマンドを使用します。

目的

ホストのバージョンとリリース・レベルを取得します。

構文

```
unsigned int CWB_ENTRY cwbCO_GetHostVersion(
                            LPCSTR         system,
                            unsigned int * version,
                            unsigned int * release );
```

パラメーター

LPCSTR systemName - input
システム名が含まれているバッファーを指すポインター。

unsigned int * version - output
システムのバージョン・レベルが戻されるバッファーを指すポインター。

unsigned int * release - output
システムのリリース・レベルが戻されるバッファーを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBCO_SYSTEM_NOT_CONFIGURED
現在活動中である環境において、このシステムは構成されていません。

CWBCO_SYSTEM_NOT_CONNECTED
現在活動中である環境の使用中に、このシステムは一度も接続されていません。

CWB_INVALID_POINTER
渡されたポインターの 1 つが NULL です。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

システムに接続が行われるたびに、ホスト・バージョンが検索され、保管されます。この API は、ホスト・バージョンを取得するために、呼び出しのたびにホストに送信されることはありません。システムは、API が呼び出されるときとは限りませんが、前に接続されたことがある可能性があります。ホスト情報は、現在活動中の環境内で構成されているシステムについてのみ、検索することができます。

cwbCO_GetUserID:

System i Access for Windows の cwbCO_GetUserID コマンドを使用します。

目的

現在、活動中である環境で構成され、接続されている可能性のある、入力システムのサインオン・ユーザー ID、またはデフォルトのユーザー ID を取得します。この API は廃止されており、置き換えられています。

構文

```
unsigned int CWB_ENTRY cwbCO_GetUserID(
                            LPCSTR             systemName,
                            char              *userID,
                            unsigned int       userID_Type,
                            unsigned long     *bufferSize);
```

パラメーター

LPCSTR systemName - input
システム名が含まれているバッファーを指すポインター。

char *userID - output
必要なシステムのユーザー ID が戻されるバッファーを指すポインター。このバッファーは、終了の NULL 文字を含めて、少なくとも、CWBCO_MAX_USER_ID + 1 文字を保持することのできる大きさが必要です。

unsigned int userID_Type - input
接続されたシステムの現在のユーザー ID (CWBCO_CURRENT_USER_ID) か、構成されたシステムのデフォルトのユーザー ID (CWBCO_DEFAULT_USER_ID) のいずれを戻すかを指定します。

unsigned long * bufferSize - input/output
userID バッファーのサイズを示す値を指すポインター。バッファーの大きさが十分ではない場合は、必要なバッファーの値が戻ります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
1 つ、または複数の入力ポインターが無効。

CWB_INVALID_PARAMETER
userID_Type の値が無効。

CWB_BUFFER_OVERFLOW
userID バッファーに、ユーザー ID を保管するための十分なスペースがありません。*bufferSize を使用して、正しいサイズを判別してください。呼び出し側が、このエラーから回復して継続する処理を行う予定であるため、ヒストリー・ログにエラー・メッセージは記録されません。

CWBCO_SYSTEM_NOT_CONNECTED
「現行ユーザー ID」のシステムは、接続されていません。

CWBCO_SYSTEM_NOT_CONFIGURED
現在活動中である環境では、この「デフォルト・ユーザー ID」のシステムは、構成されていません。

CWBCO_INTERNAL_ERROR
内部エラー

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

デフォルトのユーザー ID が指定され、接続が構成された際に何も入力されなかった場合、CWB_OK が戻され、呼び出し側に戻されるユーザー ID は、空ストリング ″¥0″ になります。検索されたユーザー ID は、現在活動中の環境からの、指定されたシステムのユーザー ID になります。

cwbCO_IsSystemConfigured:

System i Access for Windows の cwbCO_IsSystemConfigured コマンドを使用します。

目的

入力システムが、現在使用中の環境の中で構成されているかどうかをチェックします。

構文

```
cwb_Boolean CWB_ENTRY cwbCO_IsSystemConfigured(
                                          LPCSTR    systemName);
```

パラメーター

LPCSTR systemName - input
システム名が含まれているバッファーを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_TRUE:
システムは、構成されています。

CWB_FALSE:
システムが構成されていないか、systemName が NULL であるか、または変換できない 1 つまたは複数のユニコード文字がシステム名に含まれています。

使用法

なし (None)

cwbCO_IsSystemConfiguredEnv:

System i Access for Windows の cwbCO_IsSystemConfiguredEnv コマンドを使用します。

目的

入力システムが、指定された環境で構成されているかどうかをチェックします。

構文

```
cwb_Boolean CWB_ENTRY cwbCO_IsSystemConfiguredEnv(
                            LPCSTR    systemName,
                            LPCSTR    pEnvironment);
```

パラメーター

LPCSTR systemName - input
システム名が含まれているバッファーを指すポインター。

LPCSTR pEnvironment - input
環境名が含まれているバッファーを指すポインター。 pEnvironment が NULL であるか、または空ストリングを指している場合、現在使用中の環境がチェックされます。

戻りコード

以下は、共通の戻り値です。

CWB_TRUE:
システムは、構成されています。

CWB_FALSE:
システムが構成されていないか、systemName が NULL であるか、または変換できない 1 つまたは複数のユニコード文字がシステム名に含まれています。

使用法

なし (None)

cwbCO_IsSystemConnected:

System i Access for Windows の cwbCO_IsSystemConnected コマンドを使用します。

目的

入力システムが現在接続されているかどうかをチェックします。

構文

```
cwb_Boolean CWB_ENTRY cwbCO_IsSystemConnected(
                                          LPCSTR systemName);
```

パラメーター

LPCSTR systemName - input
システム名が含まれているバッファーを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_TRUE:
システムは、接続されています。

CWB_FALSE:
システムが接続されていないか、systemName が NULL であるか、または変換できない 1 つまたは複数のユニコード文字がシステム名に含まれています。

使用法

この API は、現在のプロセス内のみの接続状況を示しています。別のプロセス内ではシステムは接続されている可能性がありますが、このことはこの API の出力には影響しません。

通信: 構成済み環境情報

これらの System i Access for Windows API を使用して、構成されている環境名を取得します。

cwbCO_GetActiveEnvironment:

System i Access for Windows の cwbCO_GetActiveEnvironment コマンドを使用します。

目的

現在活動中である環境の名前を取得します。

構文

```
unsigned int CWB_ENTRY cwbCO_GetActiveEnvironment(
                            char               *environmentName,
                            unsigned long      *bufferSize);
```

パラメーター

char *environmentName - output
渡されるバッファーがその名前を保持することのできる十分な大きさがある場合は、活動中の環境の名前がコピーされるバッファーを指すポインターです。バッファーは、終了の NULL 文字を含めて、少なくとも、CWBCO_MAX_ENV_NAME + 1 文字を保持することのできる大きさが必要です。

unsigned long * bufferSize - input/output

入力 *environmentName が指しているバッファーのサイズ。

出力 必要なバッファー・サイズ。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
1 つまたは複数のポインター・パラメーターが NULL です。

CWB_BUFFER_OVERFLOW
出力バッファーに環境名全体を保持することのできる十分なスペースがありません。*bufferSize を使用して、正しいサイズを判別してください。呼び出し側が、このエラーから回復して継続する処理を行う予定であるため、ヒストリー・ログにエラー・メッセージは記録されません。

CWBCO_NO_SUCH_ENVIRONMENT
環境が構成されていないため、活動中の環境がありません。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_API_ERROR
一般 API 障害。

使用法

なし

cwbCO_GetEnvironmentName:

System i Access for Windows の cwbCO_GetEnvironmentName コマンドを使用します。

目的

索引に対応する環境名を取得します。

構文

```
unsigned int CWB_ENTRY cwbCO_GetEnvironmentName(
                            char               *environmentName,
                            unsigned long      *bufferSize,
                            unsigned long       index);
```

パラメーター

char *environmentName - output
環境名が含まれるバッファーを指すポインター。このバッファーは、終了の NULL 文字を含めて少なくとも、CWBCO_MAX_ENV_NAME + 1 文字を保持することのできる大きさが必要です。

unsigned long * bufferSize - input/output

入力 *environmentName が指しているバッファーのサイズ。

出力 用意されたバッファーが小さすぎた場合、必要なバッファーのサイズ。

unsigned long index - input
0 は、1 番目の環境に対応します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
1 つまたは複数のポインター・パラメーターが NULL です。

CWB_BUFFER_OVERFLOW
出力バッファーに環境名全体を保持することのできる十分なスペースがありません。*bufferSize を使用して、正しいサイズを判別してください。呼び出し側が、このエラーから回復して継続する処理を行う予定であるため、ヒストリー・ログにエラー・メッセージは記録されません。

CWBCO_END_OF_LIST
環境リストの最後に到達しました。環境名は、戻されませんでした。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_API_ERROR
一般 API 障害。

使用法

なし

cwbCO_GetNumberOfEnvironments:

System i Access for Windows の cwbCO_GetNumberOfEnvironments コマンドを使用します。

目的

存在する System i Access 環境の数を取得します。活動中の環境と活動中ではない環境の両方が含まれます。

構文

```
unsigned int CWB_ENTRY cwbCO_GetNumberOfEnvironments(
                           unsigned long      *numberOfEnv);
```

パラメーター

unsigned long *numberOfEnv - output
出力時に、環境の数に設定されます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
numberOfEnv ポインター・パラメーターが NULL です。

使用法

なし

通信: 環境および接続情報

これらの System i Access for Windows API を使用して、呼び出し側のアプリケーションで環境と接続情報を変更できるかどうかを判別します。

cwbCO_CanConnectNewSystem:

System i Access for Windows の cwbCO_CanConnectNewSystem コマンドを使用します。

目的

活動中の環境内のシステム・リストで、構成されていないシステムにユーザーが接続できるかどうかを示しています。

構文

```
cwb_Boolean CWB_ENTRY cwbCO_CanConnectNewSystem();
```

パラメーター

なし (None)

戻りコード

以下は、共通の戻り値です。

CWB_TRUE
まだ構成されていないシステムに接続可能。

CWB_FALSE
まだ構成されていないシステムに接続は不可能。

使用法

この API が CWB_FALSE を戻す場合、現在構成されていないシステム名による cwbCO_CreateSystem への呼び出しは失敗します。その他の System i Access for Windows API において、システム名をパラメーターとして取るものについても、同様に失敗します。

cwbCO_CanModifyEnvironmentList:

System i Access for Windows の cwbCO_CanModifyEnvironmentList コマンドを使用します。

目的

ユーザーが環境を作成 / 除去 / 名前変更できるかどうかを示します。

構文

```
cwb_Boolean CWB_ENTRY cwbCO_CanModifyEnvironmentList();
```

パラメーター

なし (None)

戻りコード

以下は、共通の戻り値です。

CWB_TRUE
環境を作成 / 除去 / 名前変更 / 削除することができます。

CWB_FALSE
環境を作成 / 除去 / 名前変更 / 削除することはできません。

使用法

この API は、環境が操作可能であるかどうかを示しています。環境内のシステムが、操作可能であるかどうかを調べるには、cwbCO_CanModifySystemList および cwbCO_CanModifySystemListEnv の API を使用します。

cwbCO_CanModifySystemList:

System i Access for Windows の cwbCO_CanModifySystemList コマンドを使用します。

目的

ユーザーが活動中の環境内のシステムを作成 / 除去 / 削除できるかどうかを示します。ポリシーを介して管理者が「指定した」システムは、除去することはできない点に注意してください。

構文

```
cwb_Boolean CWB_ENTRY cwbCO_CanModifySystemList();
```

パラメーター

なし (None)

戻りコード

以下は、共通の戻り値です。

CWB_TRUE
システム・リストを変更可能。

CWB_FALSE
システム・リストは変更不可。

使用法

この API は、活動中の環境内のシステムが操作可能かどうかを示しています。環境が操作可能かどうかを調べる場合は、cwbCO_CanModifyEnvironmentList を参照してください。

cwbCO_CanModifySystemListEnv:

System i Access for Windows の cwbCO_CanModifySystemListEnv コマンドを使用します。

目的

ユーザーが入力環境内のシステムを作成 / 除去 / 削除できるかどうかを示します。ポリシーを介して管理者が「指定した」システムは、除去することはできない点に注意してください。

構文

```
cwb_Boolean CWB_ENTRY cwbCO_CanModifySystemListEnv(
                            char              *environmentName);
```

パラメーター

char *environmentName - input
必要な環境名が入っているストリングを指すポインターです。このポインターが NULL であるか、または空ストリングを指している場合、現在活動中の環境がチェックされます。

戻りコード

以下は、共通の戻り値です。

CWB_TRUE
システム・リストを変更可能。

CWB_FALSE
システム・リストは変更不可。あるいは、存在しない環境名が渡されたというようなエラーが起こりました。

使用法

この API は、環境内のシステムが操作可能かどうかを示しています。環境が操作可能かどうかを調べる場合は、cwbCO_CanModifyEnvironmentList を参照してください。

cwbCO_CanSetActiveEnvironment:

System i Access for Windows の cwbCO_CanSetActiveEnvironment コマンドを使用します。

目的

ユーザーが環境を活動中の環境に設定できるかどうかを示します。

構文

```
cwb_Boolean CWB_ENTRY cwbCO_CanSetActiveEnvironment();
```

パラメーター

なし (None)

戻りコード

以下は、共通の戻り値です。

CWB_TRUE
活動中の環境を設定できる。

CWB_FALSE
活動中の環境を設定できない。

使用法

なし (None)

例: System i Access for Windows の通信 API の使用

System i Access for Windows の通信 API を使用して、デフォルト (管理) システムの名前、および活動環境の中で構成されたすべてのシステムの名前を検索して表示する場合の、プログラム例です。

```
/*********************************************************************
*
* Module:
*   GETSYS.C
*
* Purpose:
*   This module is used to demonstrate how an application might use the
*   Communication API's.  In this example, these APIs are used to get
*   and display the list of all configured systems.  The user can then
*   select one, and that system's connection properties (the attributes
*   of the created system object) are displayed.  All Client Access
*   services are then checked for connectabliity, and the results displayed.
*
* Usage notes:
*
*   Include CWBCO.H, CWBCOSYS.H, and CWBSV.H
*   Link with CWBAPI.LIB
*
* IBM grants you a nonexclusive license to use this as an example
* from which you can generate similar function tailored to your own
* specific needs. This sample is  provided in the form of source
* material which you may change and use.
* If you change the source, it is recommended that you first copy the
* source to a different directory.  This will ensure that your changes
* are preserved when the tool kit contents are changed by IBM.
*
*                              DISCLAIMER
*                              ----------
*
* This sample code is provided by IBM for illustrative purposes only.
* These examples have not been thoroughly tested under all conditions.
* IBM, therefore, cannot guarantee or imply reliability,
* serviceability, or function of these programs.  All programs
* contained herein are provided to you "AS IS" without any warranties
* of any kind.  ALL WARRANTIES, INCLUDING BUT NOT LIMITED TO THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE, ARE EXPRESSLY DISCLAIMED.
*
* Your license to this sample code provides you no right or licenses to
* any IBM patents.  IBM has no obligation to defend or indemnify against
* any claim of infringement, including but not limited to: patents,
* copyright, trade secret, or intellectual property rights of any kind.
*
*
*
*
*
*                              COPYRIGHT
*                              ---------
*            5722-XE1 (C) Copyright IBM CORP. 1996, 2004
*            All rights reserved.
*            US Government Users Restricted Rights -
*            Use, duplication or disclosure restricted
*            by GSA ADP Schedule Contract with IBM Corp.
*            Licensed Material - Property of IBM
*
*
*********************************************************************/

#include windows.h
#include stdio.h

#include "cwbsv.h"       /* Service APIs for retrieving any FAILURE messages  */
#include "cwbco.h"       /* Comm APIs for enumerating systems configured    */
#include "cwbcosys.h"    /* Comm APIs for creating and using system objects */

#define SUCCESS   (0)
```

```
#define FAILURE   (1)


/*
 *  Arrays of attribute description strings, for human-readable
 *  display of these values.
 */
char* valModeStr[2] = { "CWBCO_VALIDATE_IF_NECESSARY" ,
                        "CWBCO_VALIDATE_ALWAYS" } ;

char* promptModeStr[3] = { "CWBCO_PROMPT_IF_NECESSARY" ,
                           "CWBCO_PROMPT_ALWAYS" ,
                           "CWBCO_PROMPT_NEVER" } ;

char* dfltUserModeStr[4] = { "CWBCO_DEFAULT_USER_MODE_NOT_SET" ,
                             "CWBCO_DEFAULT_USER_USE" ,
                             "CWBCO_DEFAULT_USER_IGNORE" ,
                             "CWBCO_DEFAULT_USER_USEWINLOGON",
                             "CWBCO_DEFAULT_USER_USE_KERBEROS" } ;

char* IPALModeStr[6] = { "CWBCO_IPADDR_LOOKUP_ALWAYS" ,
                         "CWBCO_IPADDR_LOOKUP_1HOUR" ,
                         "CWBCO_IPADDR_LOOKUP_1DAY" ,
                         "CWBCO_IPADDR_LOOKUP_1WEEK" ,
                         "CWBCO_IPADDR_LOOKUP_NEVER" ,
                         "CWBCO_IPADDR_LOOKUP_AFTER_STARTUP" } ;

char* portLookupModeStr[3] = { "CWBCO_PORT_LOOKUP_SERVER" ,
                               "CWBCO_PORT_LOOKUP_LOCAL" ,
                               "CWBCO_PORT_LOOKUP_STANDARD" } ;

char* cwbBoolStr[2] = { "False", "True" } ;

/* NOTE! The corresponding service CONSTANT integers start
 *       at 1, NOT at 0; that is why the dummy "FAILURE" value
 *       was added at position 0.
 */
char* serviceStr[15] = { "CWBCO_SERVICE_THISISABADSERVICE!",
                         "CWBCO_SERVICE_CENTRAL" ,
                         "CWBCO_SERVICE_NETFILE" ,
                         "CWBCO_SERVICE_NETPRINT" ,
                         "CWBCO_SERVICE_DATABASE" ,
                         "CWBCO_SERVICE_ODBC" ,
                         "CWBCO_SERVICE_DATAQUEUES" ,
                         "CWBCO_SERVICE_REMOTECMD" ,
                         "CWBCO_SERVICE_SECURITY" ,
                         "CWBCO_SERVICE_DDM" ,
                         "", /* not used */
                         "", /* not used */
                         "CWBCO_SERVICE_WEB_ADMIN" ,
                         "CWBCO_SERVICE_TELNET" ,
                         "CWBCO_SERVICE_MGMT_CENTRAL" } ;




/*
 *  Node in a singly-linked list to hold a pointer
 *  to a system name.  Note that the creator of an
 *  instance of this node must allocate the space to
 *  hold the system name himself, only a pointer is
 *  supplied here.
 */
typedef struct sysListNodeStruct SYSLISTNODE, *PSYSLISTNODE;
struct sysListNodeStruct
{
   char*              sysName;
```

```
   cwbCO_SysHandle hSys;
   PSYSLISTNODE       next;
} ;



/*****************************************************************************
 *  Add a system name to the list of configured systems we will keep around.
 *****************************************************************************/
UINT addSystemToList(
   char* sysName,
   SYSLISTNODE** ppSysList )
{
   SYSLISTNODE* pNewSys;
   char*        pNewSysName;

   pNewSys = (SYSLISTNODE*) malloc (sizeof( SYSLISTNODE ));
   if ( pNewSys == NULL )
   {
      return FAILURE;
   }

   pNewSysName = (char*) malloc (strlen( sysName ) + 1 );
   if ( pNewSysName == NULL )
   {
      free (pNewSys);
      return FAILURE;
   }

   strcpy( pNewSysName, sysName );
   pNewSys->sysName = pNewSysName;
   pNewSys->hSys = 0;           /* delay creating sys object until needed */
   pNewSys->next = *ppSysList;
   *ppSysList = pNewSys;

   return SUCCESS;
}



/*****************************************************************************
 *  Clear the list of system names and clean up used storage.
 *****************************************************************************/
void clearList( SYSLISTNODE* pSysList )
{
   PSYSLISTNODE pCur, pNext;

   pCur = pSysList;

   while ( pCur != NULL )
   {
      pNext = pCur->next;
      free (pCur->sysName);
      free (pCur);
      pCur = pNext;
   }
}


/*****************************************************************************
 *  Retrieve and display Client Access FAILURE messages.
 *****************************************************************************/
void reportCAErrors( cwbSV_ErrHandle hErrs )
{
   ULONG msgCount;
   UINT apiRC;
   UINT i;
```

```
   char msgText[ 200 ];              /* 200 is big enuf to hold most msgs */
   ULONG bufLen = sizeof( msgText );  /* holds size of msgText buffer      */
   ULONG lenNeeded;                  /* to hold length of buf needed      */

   apiRC = cwbSV_GetErrCount( hErrs, &msgCount );
   if ( CWB_OK != apiRC )
   {
      printf( "Failed to get message count, cwbSV_GetErrCount rc=%u¥n", apiRC );
      if ( ( CWB_INVALID_POINTER == apiRC )   ||
           ( CWB_INVALID_HANDLE  == apiRC ) )
      {
         printf( "  --> likely a programming FAILURE!¥n");
      }
      return;
   }

   bufLen = sizeof( msgText );
   for ( i=1; i<=msgCount; i++ )
   {
      apiRC = cwbSV_GetErrTextIndexed(hErrs, i, msgText, bufLen, &lenNeeded);
      if ( ( CWB_OK == apiRC ) ||
           ( CWB_BUFFER_OVERFLOW == apiRC ) )  /* if truncated, that's ok */
      {
         printf( "CA FAILURE #%u: %s¥n", i, msgText );
      }
      else
      {
         printf( "CA FAILURE #%u unuvailable, cwbSV_GetErrTextIndexed rc=%u¥n",
            i, apiRC );
      }
   }
}


/*****************************************************************************
 *  Build the list of systems as it is currently configured in Client
 *  Access.
 *****************************************************************************/
UINT buildSysList(
   SYSLISTNODE** ppSysList )
{
   cwbSV_ErrHandle      hErrs;
   cwbCO_SysListHandle  hList;
   char sysName[ CWBCO_MAX_SYS_NAME + 1 ];
   ULONG                bufSize = sizeof( sysName );
   ULONG                needed;
   UINT apiRC;
   UINT                 myRC = SUCCESS;
   UINT                 rc = SUCCESS;

   /*  Create a FAILURE handle so that, in case of FAILURE, we can
    *  retrieve and display the messages (if any) associated with
    *  the failure.
    */
   apiRC = cwbSV_CreateErrHandle( &hErrs );
   if ( CWB_OK != apiRC )
   {
      /*  Failed to create a FAILURE handle, use NULL instead.
       *  This means we'll not be able to get at FAILURE messages.
       */
      hErrs = 0;
   }

   apiRC = cwbCO_CreateSysListHandle( *hList, hErrs );
   if ( CWB_OK != apiRC )
   {
      printf( "Failure to get a handle to the system list.¥n" );
```

```
      reportCAErrors( hErrs );
      myRC = FAILURE;
   }

   /*  Get each successive system name and add the system to our
    *  internal list for later use.
    */
   while ( ( CWB_OK == apiRC ) && ( myRC == SUCCESS ) )
   {
      apiRC = cwbCO_GetNextSysName( hList, sysName, bufSize, &needed );

      /*  Note that since the sysName buffer is as large as it will
       *    ever need to be, we don't check specifically for the return
       *    code CWB_BUFFER_OVERFLOW.  We could instead choose to use a
       *    smaller buffer, and if CWB_BUFFER_OVERFLOW were returned,
       *    allocate one large enough and call cwbCO_GetNextSysName
       *    again.
       */
      if ( CWB_OK == apiRC )
      {
         myRC = addSystemToList( sysName, ppSysList );
         if ( myRC != SUCCESS )
         {
            printf( "Failure to add the next system name to the list.¥n");
         }
      }
      else if ( CWBCO_END_OF_LIST != apiRC )
      {
         printf( "Failed to get the next system name.¥n" );
         myRC = FAILURE;
      }
   } /* end while (to build a list of system names) */

   /*
    *  Free the FAILURE handle if one was created
    */
   if ( hErrs != 0 )   /* (non-NULL if it was successfully created) */
   {
      apiRC = cwbSV_DeleteErrHandle( hErrs );
      if ( CWB_INVALID_HANDLE == apiRC )
      {
         printf("Failure: FAILURE handle invalid, could not delete!¥n");
         myRC = FAILURE;
      }
   }

   return myRC;
}


/*****************************************************************************
 *  Get a system object given an index into our list of systems.
 *****************************************************************************/
UINT getSystemObject(
   UINT sysNum,
   SYSLISTNODE* pSysList,
   cwbCO_SysHandle* phSys )
{
   SYSLISTNODE*         pCur;
   UINT myRC, apiRC;

   pCur = pSysList;
   for ( ; sysNum > 1; sysNum-- )
   {
      /*  We have come to the end of the list without finding
       *  the system requested, break out of loop and set FAILURE rc.
       */
```

```
      if ( NULL == pCur )
      {
         myRC = FAILURE;
         break;
      }

      pCur = pCur->next;
   }

   /*  If we're at a real system node, continue
    */
   if ( NULL != pCur )
   {
      /*  We're at the node/sysname of the user's choice.  If no
       *  Client Access "system object" has yet been created for this
       *  system, create one.  Pass back the one for the selected system.
       */
      if ( 0 == pCur->hSys )
      {
         apiRC = cwbCO_CreateSystem( pCur->sysName, &(pCur->hSys) );
         if ( CWB_OK != apiRC )
         {
            printf(
               "Failed to create system object, cwbCO_CreateSystem rc = %u¥n",
               apiRC );
            myRC = FAILURE;
         }
      }
      *phSys = pCur->hSys;
   }

   return myRC;
}



/*****************************************************************************
 *  Allow the user to select a system from the list we have.
 *****************************************************************************/
UINT selectSystem(
   UINT* pNumSelected,
   SYSLISTNODE* pSysList,
   BOOL refreshList )
{
   UINT                 myRC = SUCCESS;
   SYSLISTNODE*         pCur;
   UINT                 sysNum, numSystems;
   char                 choiceStr[ 20 ];

   /*  If the user wants the list refreshed, clear any existing list
    *  so we can rebuilt it from scratch.
    */
   if ( refreshList )
   {
      clearList( pSysList );
      pSysList = NULL;
   }

   /*  If the list of system names is NULL (no list exists), build
    *  the list of systems using Client Access APIs.
    */
   if ( NULL == pSysList )
   {
      myRC = buildSysList( &pSysList );
      if ( SUCCESS != myRC )
      {
         *pNumSelected = 0;
```

```
         printf( "Failed to build sys list, cannot select a system.¥n");
      }
   }

   if ( SUCCESS == myRC )
   {
      printf( "------------------------------------------ ¥n" );
      printf( "The list of systems configured is as follows:¥n" );
      printf( "------------------------------------------ ¥n" );
      for ( sysNum = 1, pCur = pSysList;
            pCur != NULL;
            sysNum++, pCur = pCur->next )
      {
         printf( "   %u) %s¥n", sysNum, pCur->sysName );
      }
      numSystems = sysNum - 1;

      printf( "Enter the number of the system of your choice:¥n");
      gets( choiceStr );
      *pNumSelected = atoi( choiceStr );

      if ( *pNumSelected > numSystems )
      {
         printf( "Invalid selection, there are only %u systems configured.¥n");
         *pNumSelected = 0;
         myRC = FAILURE;
      }
   }

   return myRC;
}



/*****************************************************************************
 *  Display a single attribute and its value, or a failing return code
 *  if one occurred when trying to look it up.
 *****************************************************************************/
void dspAttr(
   char* label,
   char* attrVal,
   UINT  lookupRC,
   BOOL* pCanBeModified,
   UINT  canBeModifiedRC )
{
   if ( CWB_OK == lookupRC )
   {
      printf( "%25s : %-30s   ", label, attrVal );
      if ( CWB_OK == canBeModifiedRC )
      {
         if ( pCanBeModified != NULL )
         {
            printf( "%s¥n", cwbBoolStr[ *pCanBeModified ] );
         }
         else
         {
            printf( "(N/A)¥n" );
         }
      }
      else
      {
         printf( "(Error, rc=%u)¥n", canBeModifiedRC );
      }
   }
   else
   {
      printf( "%30s : (Error, rc=%u)¥n", label, lookupRC );
```

```
   }
}


/*****************************************************************************
 *
 *  Load the host/version string into the buffer specified.  The
 *  buffer passed in must be at least 7 bytes long!  A pointer to
 *  the buffer itself is passed back so that the output from this
 *  function can be used directly as a parameter.
 *
 *****************************************************************************/
char* hostVerModeDescr(
   ULONG ver,
   ULONG rel,
   char* verRelBuf )
{
   char* nextChar = verRelBuf;

   if ( verRelBuf != NULL )
   {
      *nextChar++ = 'v';
      if ( ver < 10 )
      {
         *nextChar++ = '0' + (char)ver;
      }
      else
      {
         *nextChar++ = '?';
      *nextChar++ = '?';
      }

      *nextChar++ = 'r';
      if ( rel < 10 )
      {
         *nextChar++ = '0' + (char)rel;
      }
      else
      {
         *nextChar++ = '?';
      *nextChar++ = '?';
      }

      *nextChar = '¥0';
   }

   return verRelBuf;
}


/*****************************************************************************
 *  Display all attributes of the system whose index in the passed list
 *  is passed in.
 *****************************************************************************/
void dspSysAttrs(
   SYSLISTNODE* pSysList,
   UINT sysNum )
{
   cwbCO_SysHandle hSys;
   UINT rc;
   char sysName[ CWBCO_MAX_SYS_NAME + 1 ];
   char IPAddr[ CWBCO_MAX_IP_ADDRESS + 1 ];
   ULONG bufLen, IPAddrLen;
   ULONG IPAddrBufLen;
   UINT apiRC, apiRC2;
```

```
cwbCO_ValidateMode         valMode;
cwbCO_DefaultUserMode      dfltUserMode;
cwbCO_PromptMode           promptMode;
cwbCO_PortLookupMode       portLookupMode;
cwbCO_IPAddressLookupMode  IPALMode;
ULONG ver, rel;
char verRelBuf[ 10 ];
ULONG verRelBufLen;
cwb_Boolean isSecSoc;
cwb_Boolean canModify;

IPAddrBufLen = sizeof( IPAddr );
verRelBufLen = sizeof( verRelBuf );

rc = getSystemObject( sysNum, pSysList, &hSys );
if ( rc == FAILURE )
{
   printf( "Failed to get system object for selected system.¥n");
   return;
}

printf("¥n¥n");
printf("-------------------------------------------------------------¥n");
printf("            S y s t e m   A t t r i b u t e s              ¥n");
printf("-------------------------------------------------------------¥n");
printf("¥n");
printf( "%25s : %-30s   %s¥n", "Attribute", "Value", "Modifiable" );
printf( "%25s : %-30s   %s¥n", "---------", "-----", "----------" );
printf("¥n");

apiRC = cwbCO_GetSystemName( hSys, sysName, &bufLen );
dspAttr( "System Name", sysName, apiRC, NULL, 0 );

apiRC = cwbCO_GetIPAddress( hSys, IPAddr, &IPAddrLen );
dspAttr( "IP Address", IPAddr, apiRC, NULL, 0 );

apiRC = cwbCO_GetHostVersionEx( hSys, &ver, &rel );
dspAttr( "Host Version/Release",
   hostVerModeDescr( ver, rel, verRelBuf ), apiRC, NULL, 0 );

apiRC = cwbCO_IsSecureSockets( hSys, &isSecSoc );
apiRC2 = cwbCO_CanModifyUseSecureSockets( hSys, &canModify );
dspAttr( "Secure Sockets In Use", cwbBoolStr[ isSecSoc ],
   apiRC, &canModify, apiRC2 );

apiRC = cwbCO_GetValidateMode( hSys, &valMode );
canModify = CWB_TRUE;
dspAttr( "Validate Mode", valModeStr[ valMode ], apiRC,
   &canModify, 0 );

apiRC = cwbCO_GetDefaultUserMode( hSys, &dfltUserMode );
apiRC2 = cwbCO_CanModifyDefaultUserMode( hSys, &canModify );
dspAttr( "Default User Mode", dfltUserModeStr[ dfltUserMode ], apiRC,
   &canModify, apiRC2 );

apiRC = cwbCO_GetPromptMode( hSys, &promptMode );
canModify = CWB_TRUE;
dspAttr( "Prompt Mode", promptModeStr[ promptMode ], apiRC,
   &canModify, 0 );

apiRC = cwbCO_GetPortLookupMode( hSys, &prtLookupMode );
apiRC2 = cwbCO_CanModifyPortLookupMode( hSys, &canModify );
dspAttr( "Port Lookup Mode", portLookupModeStr[ portLookupMode ], apiRC,
   &canModify, apiRC2 );

apiRC = cwbCO_GetIPAddressLookupMode( hSys, &IPALMode );
apiRC2 = cwbCO_CanModifyIPAddressLookupMode( hSys, &canModify );
```

```
   dspAttr( "IP Address Lookup Mode", IPALModeStr[ IPALMode ], apiRC,
      &canModify, apiRC2 );

   printf("¥n¥n");
}



/*****************************************************************************
 *  Display connectability to all Client Access services that are
 *  possible to connect to.
 *****************************************************************************/
void dspConnectability(
   PSYSLISTNODE pSysList,
   UINT sysNum )
{
   UINT rc;
   UINT apiRC;
   cwbCO_Service service;
   cwbCO_SysHandle hSys;

   rc = getSystemObject( sysNum, pSysList, &hSys );
   if ( rc == FAILURE )
   {
      printf( "Failed to get system object for selected system.¥n");
   }
   else
   {
      printf("¥n¥n");
      printf("-------------------------------------------------------------¥n");
      printf("       S y s t e m   S e r v i c e s   S t a t u s       ¥n");
      printf("-------------------------------------------------------------¥n");
      for ( service=(cwbCO_Service)1;
            service <= CWBCO_SERVICE_MGMT_CENTRAL;
            service++ )
      {
         apiRC = cwbCO_Verify( hSys, service, 0 );   // 0=no err handle
         printf("  Service '%s': ", serviceStr[ service ] );
         if ( apiRC == CWB_OK )
         {
            printf("CONNECTABLE¥n");
         }
         else
         {
            printf("CONNECT TEST FAILED, rc = %u¥n", apiRC );
         }
      }
   }

   printf("¥n");
}



/*****************************************************************************
 *  MAIN PROGRAM BODY
 *****************************************************************************/
void main(void)
{
   PSYSLISTNODE pSysList = NULL;
   UINT numSelected;
   UINT rc;
   char choiceStr[10];
   UINT choice;

   rc = buildSysList( &pSysList );
   if ( SUCCESS != rc )
```

```
{
   printf( "Failure to build the system list, exiting.¥n¥n");
   exit( FAILURE );
}

do
{
   printf( "Select one of the following options:¥n" );
   printf( "   (1) Display current system attributes¥n");
   printf( "   (2) Display service connectability for a system¥n");
   printf( "   (3) Refresh the list of systems¥n" );
   printf( "   (9) Quit¥n" );
   gets( choiceStr );
   choice = atoi( choiceStr );
   switch ( choice )
   {
      // ---- Display current system attributes ---------------
      case 1 :
      {
         rc = selectSystem( &numSelected, pSysList, FALSE );
         if ( SUCCESS == rc )
         {
            dspSysAttrs( pSysList, numSelected );
         }

         break;
      }

      // ---- Display service connectability for a system -----
      case 2 :
      {
         rc = selectSystem( &numSelected, pSysList, FALSE );
         if ( SUCCESS == rc )
         {
            dspConnectability( pSysList, numSelected );
         }

         break;
      }

      // ---- Refresh the list of systems ---------------------
      case 3 :
      {
         clearList( pSysList );
         pSysList = NULL;
         rc = buildSysList( &pSysList );
         break;
      }

      // ---- Quit --------------------------------------------
      case 9 :
      {
         printf("Ending the program!¥n");
         break;
      }

      default :
      {
         printf("Invalid choice.  Please make a different selection.¥n");
      }
   }
} while ( choice != 9 );

/* Cleanup the list, we're done */
clearList( pSysList );
```

```
    pSysList = NULL;

    printf( "¥nEnd of program.¥n¥n" );
}
```

System i データ待ち行列 API

System i Access for Windows のデータ待ち行列アプリケーション・プログラミング・インターフェース (API) を使用すると、System i のデータ待ち行列に簡単にアクセスできます。データ待ち行列を使用することによって、通信 API を使う必要のないクライアント/サーバー・アプリケーションを作成することができるようになります。

System i データ待ち行列 API に必要なファイル

ヘッダー・ファイル	インポート・ライブラリー	ダイナミック・リンク・ライブラリー
cwbdq.h	cwbapi.lib	cwbdq.dll

Programmer's Toolkit:

Programmer's Toolkit には、データ待ち行列資料、cwbdq.h ヘッダー・ファイルへのアクセス、およびプログラム例へのリンクが用意されています。この情報にアクセスするには、Programmer's Toolkit をオープンして、「**データ待ち行列**」 → 「**C/C++ API**」と選択します。

注: コード例を使用することで、 633 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』の条件に同意します。

関連資料

27 ページの『データ待ち行列 API の戻りコード』
System i Access for Windows のデータ待ち行列 API の戻りコードを示します。

7 ページの『接続 API 用の System i 名の形式』
パラメーターとして System i 名を取得する API では、3 つの異なる形式の名前を使用できます。

8 ページの『OEM、ANSI、およびユニコードの考慮事項』
System i Access for Windows の、ストリング・パラメーターを受け入れる C/C++ API の大部分は、OEM、ANSI、Unicode の 3 つのうち、いずれかの形式になっています。

データ待ち行列

データ待ち行列は、System i のオブジェクトです。

データ待ち行列を使用する利点

PC 開発者および System i アプリケーション開発者にとって、データ待ち行列には数多くの利点があります。

- データ待ち行列を使用すると、System i 通信を高速かつ効率的に行うことができます。
- データ待ち行列は、システム・オーバーヘッドが少なくて済み、ごくわずかのセットアップですみます。
- 1 つのバッチ・ジョブが単一のデータ待ち行列を使用して、複数の対話式ジョブをサービスすることができるので、データ待ち行列は効率的です。
- データ待ち行列メッセージの内容は不定様式であり (フィールドは不要)、他のシステム・オブジェクトにはない柔軟性を備えています。

- データ待ち行列へのアクセスは、System i API ならびに CL コマンドを介して行われます。これによって、クライアント/サーバー・アプリケーションの開発を容易に行うことができます。

データ待ち行列メッセージの配列

System i データ待ち行列に入れるメッセージの配列方法を指定するには、次の 3 つの方法があります。

LIFO 後入れ先出し法です。データ待ち行列に最後に入れられた (最新) メッセージを待ち行列から最初に取り出します。

FIFO 先入れ先出し法です。データ待ち行列に最初に入れられた (最古) メッセージを待ち行列から最初に取り出します。

KEYED
データ待ち行列に入れられたメッセージごとに、キーが関連付けられます。メッセージは、関連付けられているキーを要求することによってのみ、待ち行列から取り出すことができます。

データ待ち行列の操作

System i CL コマンド、または呼び出し可能なプログラミング・インターフェースを使用して、データ待ち行列の操作を行うことができます。アプリケーションを作成したプログラミング言語に関係なく、すべての System i アプリケーションが、データ待ち行列にアクセスできます。

次に挙げる System i インターフェースを使用して、データ待ち行列の操作を行えます。

i5/OS コマンド

CRTDTAQ
データ待ち行列を作成し、指定のライブラリーに保管します。

DLTDTAQ
指定のデータ待ち行列をシステムから削除します。

i5/OS アプリケーション・プログラミング・インターフェース

QSNDDTAQ
指定のデータ待ち行列にメッセージ (レコード) を送ります。

QRCVDTAQ
指定のデータ待ち行列のメッセージ (レコード) を読み込みます。

QCLRDTAQ
指定のデータ待ち行列からすべてのメッセージを消去します。

QMHQRDQD
データ待ち行列の記述を検索します。

QMHRDQM
項目を削除することなく、データ待ち行列の記入項目を検索します。

データ待ち行列の一般的な使用方法

データ待ち行列は、強力なプログラム間インターフェースです。System i プログラミングの経験が豊富なプログラマーであれば、待ち行列の使い方には慣れているはずです。データ待ち行列とは、単に、情報を別のプログラムに渡すために使用される手段に過ぎません。

このインターフェースには通信プログラミングが不要であるため、同期処理にも非同期 (切断) 処理にも使用することができます。

ホスト・アプリケーションと PC アプリケーションは、サポートされている言語であれば、どの言語を使用しても開発できます。例えば、ホスト・アプリケーションでは RPG を使用し、PC アプリケーションでは C++ を使用するといったことが可能です。このような場合の待ち行列の役割は、一方のプログラムから入力を取り込み、それを他方のプログラムに渡すことです。

データ待ち行列の使用例を次に示します。

- PC ユーザーが、終日電話注文を取り、注文を 1 つ 1 つプログラムにキー入力し、プログラムがそれぞれの注文を System i データ待ち行列に入れると想定します。
- パートナー・プログラム (PC プログラムまたは System i プログラムのいずれか) は、データ待ち行列をモニターし、待ち行列から情報を引き出します。このパートナー・プログラムは、同時に稼働させることもできれば、ユーザー使用のピーク時を過ぎてから開始することもできます。
- パートナー・プログラムは、開始 PC プログラムに入力を戻すこともあれば、戻さないこともあります。また、別の PC または System i のプログラムの待ち行列に何らかの情報を入れることもあります。
- 最終的には、受注が完了し、顧客に請求書が送られ、在庫レコードが更新され、PC ユーザーに、顧客に電話をして出荷予定日を知らせるための指示情報が、PC アプリケーションの「待ち行列」に入れられます。

オブジェクト

データ待ち行列機能を使用するアプリケーションでは、4 つの**オブジェクト**を利用します。これらのオブジェクトは、それぞれに、ハンドルを介してそのアプリケーションに識別されます。オブジェクトには、以下のものがあります。

待ち行列オブジェクト
このオブジェクトは、System i データ待ち行列を表します。

属性 このオブジェクトは、System i データ待ち行列を記述します。

データ これらのオブジェクトは、System i データ待ち行列との間でレコードの書き込みや読み取りを行うのに使用されます。

読み取りオブジェクト
このオブジェクトは、非同期読み取り API の場合にのみ使用されます。読み取りオブジェクトは、Sytem i データ待ち行列からレコードを読み取る要求を、固有に識別します。このハンドルは、データが戻されたかどうかを検査するのに、以降の呼び出しで使用されます。詳細については、cwbDQ_AsyncRead API を参照してください。

関連資料

141 ページの『cwbDQ_AsyncRead』
System i Access for Windows の cwbDQ_AsyncRead コマンドを使用します。

データ待ち行列: 作成、削除、およびオープンの API

これらの System i API は、cwbCO_SysHandle システム・オブジェクト・ハンドルと一緒に使用します。

cwbDQ_CreateEx:

System i Access for Windows の cwbDQ_CreateEx コマンドを使用します。

目的

System i データ待ち行列オブジェクトを作成します。オブジェクトが作成されたら、cwbDQ_OpenEx API を使用してそのオブジェクトをオープンすることができます。オブジェクトは、属性ハンドルに指定した属性を持ちます。

構文

```
unsigned int CWB_ENTRY cwbDQ_CreateEx(
                            cwbCO_SysHandle     sysHandle,
                            const char         *queue,
                            const char         *library,
                            cwbDQ_Attr          queueAttributes,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbCO_SysHandle sysHandle - input
システム・オブジェクトを指すハンドル。

const char * queue - input
ASCIIZ ストリングに入っているデータ待ち行列名を指すポインター。

const char * library - input
ASCIIZ ストリングに入っているライブラリー名を指すポインター。このポインターが NULL の場合は、現行ライブラリーが使用されます (ライブラリーを ″*CURLIB″ に設定)。

cwbDQ_Attr queueAttributes - input
データ待ち行列の属性へのハンドル。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_COMMUNICATIONS_ERROR
通信エラーが発生しました。

CWB_SERVER_PROGRAM_NOT_FOUND
System i アプリケーションが見つかりません。

CWB_HOST_NOT_FOUND
システムが非活動中であるか、または存在しません。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWB_SECURITY_ERROR
セキュリティー・エラーが発生しました。

CWB_LICENSE_ERROR
ライセンス・エラーが発生しました。

CWB_CONFIG_ERROR
構成エラーが発生しました。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

CWBDQ_BAD_QUEUE_NAME
待ち行列名が正しくありません。

CWBDQ_BAD_LIBRARY_NAME
ライブラリー名が正しくありません。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBDQ_USER_EXIT_ERROR
ユーザー出口プログラムでエラーが発生しました。

CWBDQ_USER_EXIT_ERROR
ユーザー出口プログラムでエラーが発生しました。

CWBDQ_LIBRARY_NOT_FOUND
システムにライブラリーがありません。

CWBDQ_NO_AUTHORITY
ライブラリーへの権限がありません。

CWBDQ_QUEUE_EXISTS
待ち行列が既に存在します。

CWBDQ_QUEUE_SYNTAX
待ち行列構文が正しくありません。

CWBDQ_LIBRARY_SYNTAX
ライブラリー構文が正しくありません。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

CWB_INVALID_HANDLE
システム・ハンドルが無効。

使用法

この関数を使用する場合は、あらかじめ次の API を発行する必要があります。

- cwbDQ_CreateSystem
- cwbDQ_CreateAttr
- cwbDQ_SetMaxRecLen

cwbDQ_DeleteEx:

System i Access for Windows の cwbDQ_DeleteEx コマンドを使用します。

目的

System i データ待ち行列からすべてのデータを除去し、そのデータ待ち行列オブジェクトを削除します。

構文

```
unsigned int CWB_ENTRY cwbDQ_DeleteEx(
                            cwbCO_SysHandle     sysHandle
                            const char         *queue,
                            const char         *library,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbCO_SysHandle - input
システム・オブジェクトを指すハンドル。

const char * queue - input
ASCIIZ ストリングに入っているデータ待ち行列名を指すポインター。

const char * library - input
ASCIIZ ストリングに入っているライブラリー名を指すポインター。このポインターが NULL の場合は、現行ライブラリーが使用されます (ライブラリーを ″*CURLIB″ に設定)。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_COMMUNICATIONS_ERROR
通信エラーが発生しました。

CWB_SERVER_PROGRAM_NOT_FOUND
System i アプリケーションが見つかりません。

CWB_HOST_NOT_FOUND
システムが非活動中であるか、または存在しません。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWB_SECURITY_ERROR
セキュリティー・エラーが発生しました。

CWB_LICENSE_ERROR
ライセンス・エラーが発生しました。

CWB_CONFIG_ERROR
構成エラーが発生しました。

CWBDQ_BAD_QUEUE_NAME
待ち行列名が長すぎます。

CWBDQ_BAD_LIBRARY_NAME
ライブラリー名が長すぎます。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBDQ_USER_EXIT_ERROR
ユーザー出口プログラムでエラーが発生しました。

CWBDQ_LIBRARY_NOT_FOUND
システムにライブラリーがありません。

CWBDQ_QUEUE_NOT_FOUND
システムに待ち行列がありません。

CWBDQ_NO_AUTHORITY
待ち行列への権限がありません。

CWBDQ_QUEUE_SYNTAX
待ち行列構文が正しくありません。

CWBDQ_LIBRARY_SYNTAX
ライブラリー構文が正しくありません。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

CWB_INVALID_HANDLE
システム・ハンドルが無効。

使用法

この関数を使用する場合は、あらかじめ cwbCO_CreateSystem を発行する必要があります。

cwbDQ_OpenEx:

System i Access for Windows の cwbDQ_OpenEx コマンドを使用します。

目的

指定のデータ待ち行列への接続を開始します。これによって、System i との会話が開始されます。正常に接続されなかった場合は、非ゼロ・ハンドルが戻されます。

構文

```
unsigned int CWB_ENTRY cwbDQ_OpenEx(
                            cwbCO_SysHandle     sysHandle
                            const char         *queue,
                            const char         *library,
                            cwbDQ_QueueHandle  *queueHandle,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbCO_SysHandle sysHandle - input
システム・オブジェクトを指すハンドル。

const char * queue - input
ASCIIZ ストリングに入っているデータ待ち行列名を指すポインター。

const char * library - input
ASCIIZ ストリングに入っているライブラリー名を指すポインター。このポインターが NULL の場合は、ライブラリー・リストが使用されます (ライブラリーを「*LIBL」に設定)。

cwbDQ_QueueHandle * queueHandle - output
ハンドルが戻される先の cwbDQ_QueueHandle を指すポインター。以降の呼び出しではすべて、このハンドルを使用してください。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_COMMUNICATIONS_ERROR
通信エラーが発生しました。

CWB_SERVER_PROGRAM_NOT_FOUND
System i アプリケーションが見つかりません。

CWB_HOST_NOT_FOUND
システムが非活動中であるか、または存在しません。

CWB_COMM_VERSION_ERROR
データ待ち行列は、このバージョンの通信では稼働しません。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWB_SECURITY_ERROR
セキュリティー・エラーが発生しました。

CWB_LICENSE_ERROR
ライセンス・エラーが発生しました。

CWB_CONFIG_ERROR
構成エラーが発生しました。

CWBDQ_BAD_QUEUE_NAME
待ち行列名が長すぎます。

CWBDQ_BAD_LIBRARY_NAME
ライブラリー名が長すぎます。

CWBDQ_BAD_SYSTEM_NAME
システム名が長すぎます。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBDQ_USER_EXIT_ERROR
ユーザー出口プログラムでエラーが発生しました。

CWBDQ_LIBRARY_NOT_FOUND
システムにライブラリーがありません。

CWBDQ_QUEUE_NOT_FOUND
システムに待ち行列がありません。

CWBDQ_NO_AUTHORITY
待ち行列またはライブラリーへの権限がありません。

CWBDQ_DAMAGED_QUE
待ち行列が、使えない状態になっています。

CWBDQ_CANNOT_CONVERT
データを、この待ち行列に合うように変換できません。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

CWB_INVALID_HANDLE
システム・ハンドルが無効。

使用法

この関数を使用する場合は、あらかじめ cwbCO_CreateSystem を発行する必要があります。

データ待ち行列: データ待ち行列 API へのアクセス

cwbDQ_Open API を使用して、特定の System i データ待ち行列との接続を確立したら、これらの他の API を呼び出して、その接続を利用します。その接続が不要になったら、cwbDQ_Close API を使用します。

cwbDQ_AsyncRead:

System i Access for Windows の cwbDQ_AsyncRead コマンドを使用します。

目的

指定のハンドルで識別される、System i データ待ち行列オブジェクトから、レコードを読み取ります。この AsyncRead は、即時に制御権を呼び出し側に戻します。この呼び出しは、CheckData API と一緒に使用します。レコードは、データ待ち行列から読み取られると、そのデータ待ち行列から除去されます。指定の待ち時間を過ぎてもデータ待ち行列が空の場合、読み取りは打ち切られ、CheckData API によって CWBDQ_TIMED_OUT の値が戻されます。0 から 99,999 (秒単位) か永久 (-1) の待ち時間を指定することが可能です。ゼロの待ち時間を指定すると、データ待ち行列にデータがない場合、 CheckData API は、最初の呼び出し時に CWBDQ_TIMED_OUT の値を戻します。

構文

```
unsigned int CWB_ENTRY cwbDQ_AsyncRead(
                            cwbDQ_QueueHandle   queueHandle,
                            cwbDQ_Data          data,
                            signed long         waitTime,
                            cwbDQ_ReadHandle   *readHandle,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbDQ_QueueHandle queueHandle - input
cwbDQ_Open 関数への先行の呼び出しで戻されたハンドル。これは System i データ待ち行列オブジェクトを識別します。

cwbDQ_Data data - input
System i データ待ち行列から読み取られる、データ・オブジェクト。

signed long waitTime - input
データ待ち行列が空の場合、データを待つ、秒単位の時間の長さ。待機時間が -1 の場合は、永久に待機することを示します。

cwbDQ_ReadHandle * readHandle - output
cwbDQ_ReadHandle が書き込まれる場所を指すポインター。このハンドルは、後続の cwbDQ_CheckData API の呼び出しで使用されます。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_TIME
無効な待機時間。

CWBDQ_INVALID_QUEUE_HANDLE
待ち行列ハンドルが無効。

CWBDQ_INVALID_SEARCH
無効な検索順序。

使用法

この関数を使用する場合は、あらかじめ次の API を発行する必要があります。

- cwbDQ_Open または cwbDQ_OpenEx
- cwbDQ_CreateData

関連概念

134 ページの『データ待ち行列の一般的な使用方法』
データ待ち行列は、強力なプログラム間インターフェースです。System i プログラミングの経験が豊富なプログラマーであれば、待ち行列の使い方には慣れているはずです。データ待ち行列とは、単に、情報を別のプログラムに渡すために使用される手段に過ぎません。

cwbDQ_Cancel:

System i Access for Windows の cwbDQ_Cancel コマンドを使用します。

目的

前に出された AsyncRead を取り消します。これによって、System i データ待ち行列の読み取りが終了します。

構文

```
unsigned int CWB_ENTRY cwbDQ_Cancel(
                            cwbDQ_ReadHandle    readHandle,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbDQ_ReadHandle readHandle - input
AsyncRead API が戻したハンドル。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_READ_HANDLE
無効な読み取りハンドル。

使用法

この関数を使用する場合は、あらかじめ次の API を発行する必要があります。

- cwbDQ_Open または cwbDQ_OpenEx
- cwbDQ_CreateData
- cwbDQ_AsyncRead

cwbDQ_CheckData:

System i Access for Windows の cwbDQ_CheckData コマンドを使用します。

目的

前に出された AsyncRead API からデータが戻されたかどうかを検査します。この API は、1 回の AsyncRead 呼び出しに対して何度も出すことができます。実際にデータが戻されていれば、この API は 0 を戻します。

構文

```
unsigned int CWB_ENTRY cwbDQ_CheckData(
                            cwbDQ_ReadHandle    readHandle,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbDQ_ReadHandle readHandle - input
AsyncRead API が戻したハンドル。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_READ_HANDLE
無効な読み取りハンドル。

CWBDQ_DATA_TRUNCATED
データが切り捨てられました。

CWBDQ_TIMED_OUT
待機時間の有効期限が切れ、データは戻されませんでした。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBDQ_QUEUE_DESTROYED
待ち行列が破棄されました。

CWBDQ_NO_DATA
データがありません。

CWBDQ_CANNOT_CONVERT
データを変換できません。

使用法

この関数を使用する場合は、あらかじめ次の API を発行する必要があります。

- cwbDQ_Open または cwbDQ_OpenEx
- cwbDQ_CreateData
- cwbDQ_AsyncRead

AsyncRead に時間限界が指定されている場合、この API は、データが戻されるまで (戻りコードは CWB_OK)、あるいは、時間限界が過ぎるまで (戻りコードは CWBDQ_TIMED_OUT)、CWBDQ_NO_DATA を戻します。

cwbDQ_Clear:

System i Access for Windows の cwbDQ_Clear コマンドを使用します。

目的

指定のハンドルで識別される、System i データ待ち行列オブジェクトから、すべてのメッセージを取り除きます。待ち行列にキーが関連付けられている場合は、キーおよびキーの長さを指定して、特定のキーに合ったメッセージを取り除くこともできます。待ち行列からすべてのメッセージを消去したい場合には、キーの値を NULL に設定し、キーの長さの値をゼロに設定します。

構文

```
unsigned int CWB_ENTRY cwbDQ_Clear(
                            cwbDQ_QueueHandle   queueHandle,
                            unsigned char      *key,
                            unsigned short      keyLength,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbDQ_QueueHandle queueHandle - input
cwbDQ_Open 関数への先行の呼び出しで戻されたハンドル。これは System i データ待ち行列オブジェクトを識別します。

unsigned char * key - input
キーを指すポインター。このキーには、組み込み NULL が含まれることがあります。したがって、このキーは ASCIIZ ストリングではありません。

unsigned short keyLength - input
キーの長さ (バイト数)。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_QUEUE_HANDLE
待ち行列ハンドルが無効。

CWBDQ_BAD_KEY_LENGTH
キーの長さは正しくありません。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

使用法

この関数を使うには、前もって次の API を発行する必要があります。

- cwbDQ_Open または cwbDQ_OpenEx

cwbDQ_Close:

System i Access for Windows の cwbDQ_Close コマンドを使用します。

目的

指定のハンドルで識別される System i データ待ち行列オブジェクトとの接続を終了します。これによって、System i との会話が終了します。

構文

```
unsigned int CWB_ENTRY cwbDQ_Close(
                            cwbDQ_QueueHandle   queueHandle);
```

パラメーター

cwbDQ_QueueHandle queueHandle - input
cwbDQ_Open 関数や cwbDQ_OpenEx 関数への先行の呼び出しで戻されたハンドル。これは System i データ待ち行列オブジェクトを識別します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_QUEUE_HANDLE
待ち行列ハンドルが無効。

使用法

この関数を使用する場合は、あらかじめ次の API を発行する必要があります。

- cwbDQ_Open または cwbDQ_OpenEx

cwbDQ_Create:

System i Access for Windows の cwbDQ_Create コマンドを使用します。

目的

System i データ待ち行列オブジェクトを作成します。オブジェクトが作成された後で、cwbDQ_Open API を使用してそのオブジェクトをオープンすることができます。オブジェクトは、属性ハンドルに指定した属性を持ちます。

注: この API は現在では使用されていません。 135 ページの『cwbDQ_CreateEx』を使用してください。

構文

```
unsigned int CWB_ENTRY cwbDQ_Create(
                              char               *queue,
                              char               *library,
                              char               *systemName,
                              cwbDQ_Attr          queueAttributes,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

char * queue - input
ASCIIZ ストリングに入っているデータ待ち行列名を指すポインター。

char * library - input
ASCIIZ ストリングに入っているライブラリー名を指すポインター。このポインターが NULL の場合は、現行ライブラリーが使用されます (ライブラリーを ″*CURLIB″ に設定)。

char * systemName - input
ASCIIZ ストリングに入っているシステム名を指すポインター。

cwbDQ_Attr queueAttributes - input
データ待ち行列の属性へのハンドル。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_COMMUNICATIONS_ERROR
通信エラーが発生しました。

CWB_SERVER_PROGRAM_NOT_FOUND
System i アプリケーションが見つかりません。

CWB_HOST_NOT_FOUND
System i が非活動中であるか、または存在しません。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWB_SECURITY_ERROR
セキュリティー・エラーが発生しました。

CWB_LICENSE_ERROR
ライセンス・エラーが発生しました。

CWB_CONFIG_ERROR
構成エラーが発生しました。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

CWBDQ_BAD_QUEUE_NAME
待ち行列名が正しくありません。

CWBDQ_BAD_LIBRARY_NAME
ライブラリー名が正しくありません。

CWBDQ_BAD_SYSTEM_NAME
システム名が正しくありません。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBDQ_USER_EXIT_ERROR
ユーザー出口プログラムでエラーが発生しました。

CWBDQ_LIBRARY_NOT_FOUND
システムにライブラリーがありません。

CWBDQ_NO_AUTHORITY
ライブラリーへの権限がありません。

CWBDQ_QUEUE_EXISTS
待ち行列が既に存在します。

CWBDQ_QUEUE_SYNTAX
待ち行列構文が正しくありません。

CWBDQ_LIBRARY_SYNTAX
ライブラリー構文が正しくありません。

使用法

この関数を使用する場合は、あらかじめ次の API を発行する必要があります。

- cwbDQ_CreateAttr
- cwbDQ_SetMaxRecLen

cwbDQ_Delete:

System i Access for Windows の cwbDQ_Delete コマンドを使用します。

目的

System i データ待ち行列からすべてのデータを除去し、そのデータ待ち行列オブジェクトを削除します。

注: この API は現在では使用されていません。 138 ページの『cwbDQ_DeleteEx』を使用してください。

構文

```
unsigned int CWB_ENTRY cwbDQ_Delete(
                              char               *queue,
                              char               *library,
                              char               *systemName,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

char * queue - input
ASCIIZ ストリングに入っているデータ待ち行列名を指すポインター。

char * library - input
ASCIIZ ストリングに入っているライブラリー名を指すポインター。このポインターが NULL の場合は、現行ライブラリーが使用されます (ライブラリーを ″*CURLIB″ に設定)。

char * systemName - input
ASCIIZ ストリングに入っているシステム名を指すポインター。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_COMMUNICATIONS_ERROR
通信エラーが発生しました。

CWB_SERVER_PROGRAM_NOT_FOUND
System i アプリケーションが見つかりません。

CWB_HOST_NOT_FOUND
System i が非活動中であるか、または存在しません。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWB_SECURITY_ERROR
セキュリティー・エラーが発生しました。

CWB_LICENSE_ERROR
ライセンス・エラーが発生しました。

CWB_CONFIG_ERROR
構成エラーが発生しました。

CWBDQ_QUEUE_NAME
待ち行列名が長すぎます。

CWBDQ_LIBRARY_NAME
ライブラリー名が長すぎます。

CWBDQ_SYSTEM_NAME
システム名が長すぎます。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBDQ_USER_EXIT_ERROR
ユーザー出口プログラムでエラーが発生しました。

CWBDQ_LIBRARY_NOT_FOUND
システムにライブラリーがありません。

CWBDQ_QUEUE_NOT_FOUND
システムに待ち行列がありません。

CWBDQ_NO_AUTHORITY
待ち行列への権限がありません。

CWBDQ_QUEUE_SYNTAX
待ち行列構文が正しくありません。

CWBDQ_LIBRARY_SYNTAX
ライブラリー構文が正しくありません。

使用法

なし (None)

cwbDQ_GetLibName:

System i Access for Windows の cwbDQ_GetLibName コマンドを使用します。

目的

cwbDQ_Open API で使用されるライブラリー名を検索します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetLibName(
                           cwbDQ_QueueHandle   queueHandle,
                           char               *libName);
```

パラメーター

cwbDQ_QueueHandle queueHandle - input
cwbDQ_Open 関数への先行の呼び出しで戻されたハンドル。これは System i データ待ち行列オブジェクトを識別します。

char * libName - output
ライブラリー名が書き込まれる先のバッファーを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_QUEUE_HANDLE
待ち行列ハンドルが無効。

使用法

この関数を使用する場合は、あらかじめ cwbDQ_Open を発行する必要があります。

cwbDQ_GetQueueAttr:

System i Access for Windows の cwbDQ_GetQueueAttr コマンドを使用します。

目的

指定のハンドルで識別される、System i データ待ち行列オブジェクトの属性を検索します。データ待ち行列属性へのハンドルが戻されます。そこで、属性を個々に検索することが可能になります。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetQueueAttr(
                            cwbDQ_QueueHandle   queueHandle,
                            cwbDQ_Attr          queueAttributes,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbDQ_QueueHandle queueHandle - input
cwbDQ_Open 関数への先行の呼び出しで戻されたハンドル。これは System i データ待ち行列オブジェクトを識別します。

cwbDQ_Attr queueAttributes - input/output
属性オブジェクト。これは、cwbDQ_CreateAttr 呼び出しからの出力です。属性は、この関数によって書き入れられるため、このオブジェクトから属性を検索した後で、cwbDQ_DeleteAttr 関数を呼び出してこのオブジェクトを削除する必要があります。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_QUEUE_HANDLE
待ち行列ハンドルが無効。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

使用法

この関数を使用する場合は、あらかじめ次の API を発行する必要があります。

- cwbDQ_Open または cwbDQ_OpenEx
- cwbDQ_CreateAttr

cwbDQ_GetQueueName:

System i Access for Windows の cwbDQ_GetQueueName コマンドを使用します。

目的

cwbDQ_Open API で使用される待ち行列名を検索します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetQueueName(
                            cwbDQ_QueueHandle   queueHandle,
                            char               *queueName);
```

パラメーター

cwbDQ_QueueHandle queueHandle - input
cwbDQ_Open 関数への先行の呼び出しで戻されたハンドル。これは System i データ待ち行列オブジェクトを識別します。

char * queueName - output
待ち行列名が書き込まれる先のバッファーを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_QUEUE_HANDLE
待ち行列ハンドルが無効。

使用法

この関数を使用する場合は、あらかじめ cwbDQ_Open を発行する必要があります。

cwbDQ_GetSysName:

System i Access for Windows の cwbDQ_GetSysName コマンドを使用します。

目的

cwbDQ_Open API で使用されるシステム名を検索します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetSysName(
                            cwbDQ_QueueHandle   queueHandle,
                            char               *systemName);
```

パラメーター

cwbDQ_QueueHandle queueHandle - input
cwbDQ_Open 関数への先行の呼び出しで戻されたハンドル。これは System i データ待ち行列オブジェクトを識別します。

char *systemName - output
システム名が書き込まれる先のバッファーを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_QUEUE_HANDLE
待ち行列ハンドルが無効。

使用法

この関数を使用する場合は、あらかじめ cwbDQ_Open か cwbDQ_OpenEx を発行する必要があります。

cwbDQ_Open:

System i Access for Windows の cwbDQ_Open コマンドを使用します。

目的

指定のデータ待ち行列への接続を開始します。これによって、System i との会話が開始されます。正常に接続されなかった場合は、非ゼロ・ハンドルが戻されます。

注: この API は現在では使用されていません。 139 ページの『cwbDQ_OpenEx』を使用してください。

構文

```
unsigned int CWB_ENTRY cwbDQ_Open(
                            char               *queue,
                            char               *library,
                            char               *systemName,
                            cwbDQ_QueueHandle  *queueHandle,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

char * queue - input
ASCIIZ ストリングに入っているデータ待ち行列名を指すポインター。

char * library - input
ASCIIZ ストリングに入っているライブラリー名を指すポインター。このポインターが NULL の場合は、ライブラリー・リストが使用されます (ライブラリーを「*LIBL」に設定)。

char * systemName - input
ASCIIZ ストリングに入っているシステム名を指すポインター。

cwbDQ_QueueHandle * queueHandle - output
ハンドルが戻される先の cwbDQ_QueueHandle を指すポインター。以降の呼び出しではすべて、このハンドルを使用してください。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_COMMUNICATIONS_ERROR
通信エラーが発生しました。

CWB_SERVER_PROGRAM_NOT_FOUND
System i アプリケーションが見つかりません。

CWB_HOST_NOT_FOUND
System i が非活動中であるか、または存在しません。

CWB_COMM_VERSION_ERROR
データ待ち行列は、このバージョンの通信では稼働しません。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWB_SECURITY_ERROR
セキュリティー・エラーが発生しました。

CWB_LICENSE_ERROR
ライセンス・エラーが発生しました。

CWB_CONFIG_ERROR
構成エラーが発生しました。

CWBDQ_BAD_QUEUE_NAME
待ち行列名が長すぎます。

CWBDQ_BAD_LIBRARY_NAME
ライブラリー名が長すぎます。

CWBDQ_BAD_SYSTEM_NAME
システム名が長すぎます。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBDQ_USER_EXIT_ERROR
ユーザー出口プログラムでエラーが発生しました。

CWBDQ_LIBRARY_NOT_FOUND
システムにライブラリーがありません。

CWBDQ_QUEUE_NOT_FOUND
システムに待ち行列がありません。

CWBDQ_NO_AUTHORITY
待ち行列またはライブラリーへの権限がありません。

CWBDQ_DAMAGED_QUE
待ち行列が、使えない状態になっています。

CWBDQ_CANNOT_CONVERT
データを、この待ち行列に合うように変換できません。

使用法

なし (None)

cwbDQ_Peek:

System i Access for Windows の cwbDQ_Peek コマンドを使用します。

目的

指定のハンドルで識別される、System i データ待ち行列オブジェクトから、レコードを読み取ります。レコードは、データ待ち行列から読み取られた後も、そのデータ待ち行列内に入れられたままとなります。データ待ち行列が空の場合は、待機時間に 0 から 99,999 または永久 (-1) を指定して、レコードを待機することもできます。待機時間をゼロにすると、データ待ち行列の中にデータがない場合は、直ちに制御権が戻ります。

構文

```
unsigned int CWB_ENTRY cwbDQ_Peek(
                            cwbDQ_QueueHandle   queueHandle,
                            cwbDQ_Data          data,
                            signed long         waitTime,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbDQ_QueueHandle queueHandle - input
cwbDQ_Open API への先行の呼び出しで戻されたハンドル。これは System i データ待ち行列オブジェクトを識別します。

cwbDQ_Data data - input
System i データ待ち行列から読み取られる、データ・オブジェクト。

signed long waitTime - input
データ待ち行列が空の場合、データを待つ、秒単位の時間の長さ。待機時間が -1 の場合は、永久に待機することを示します。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_TIME
無効な待機時間。

CWBDQ_INVALID_QUEUE_HANDLE
待ち行列ハンドルが無効。

CWBDQ_INVALID_SEARCH
無効な検索順序。

CWBDQ_DATA_TRUNCATED
データが切り捨てられました。

CWBDQ_TIMED_OUT
待機時間の有効期限が切れ、データは戻されませんでした。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBDQ_QUEUE_DESTROYED
待ち行列が破棄されました。

CWBDQ_CANNOT_CONVERT
データを変換できません。

使用法

この関数を使用する場合は、あらかじめ cwbDQ_Open または cwbDQ_OpenEx と cwbDQ_CreateData を発行する必要があります。

cwbDQ_Read:

System i Access for Windows の cwbDQ_Read コマンドを使用します。

目的

指定のハンドルで識別される、System i データ待ち行列オブジェクトから、レコードを読み取ります。レコードは、データ待ち行列から読み取られると、そのデータ待ち行列から除去されます。データ待ち行列が空の場合は、待機時間に 0 から 99,999 または永久 (-1) を指定して、レコードを待機することもできます。待機時間をゼロにすると、データ待ち行列の中にデータがない場合は、直ちに制御権が戻ります。

構文

```
unsigned int CWB_ENTRY cwbDQ_Read(
                            cwbDQ_QueueHandle   queueHandle,
                            cwbDQ_Data          data,
                            long                waitTime,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbDQ_QueueHandle queueHandle - input
cwbDQ_Open 関数への先行の呼び出しで戻されたハンドル。これは System i データ待ち行列オブジェクトを識別します。

cwbDQ_Data data - input
System i データ待ち行列から読み取られる、データ・オブジェクト。

long waitTime - input
データ待ち行列が空の場合、データを待つ、秒単位の時間の長さ。待機時間が -1 の場合は、永久に待機することを示します。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_TIME
無効な待機時間。

CWBDQ_INVALID_QUEUE_HANDLE
待ち行列ハンドルが無効。

CWBDQ_INVALID_SEARCH
無効な検索順序。

CWBDQ_DATA_TRUNCATED
データが切り捨てられました。

CWBDQ_TIMED_OUT
待機時間の有効期限が切れ、データは戻されませんでした。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBDQ_QUEUE_DESTROYED
待ち行列が破棄されました。

CWBDQ_CANNOT_CONVERT
データを変換できません。

使用法

この関数を使用する場合は、あらかじめ cwbDQ_Open と cwbDQ_CreateData を発行する必要があります。

cwbDQ_Write:

System i Access for Windows の cwbDQ_Write コマンドを使用します。

目的

指定のハンドルで識別される、System i データ待ち行列オブジェクトに、レコードを書き込みます。

構文

```
unsigned int CWB_ENTRY cwbDQ_Write(
                            cwbDQ_QueueHandle   queueHandle,
                            cwbDQ_Data          data,
                            cwb_Boolean         commit,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbDQ_QueueHandle queueHandle - input
cwbDQ_Open 関数または cwbDQ_OpenEx 関数への先行の呼び出しで戻されたハンドル。これは System i データ待ち行列オブジェクトを識別します。

cwbDQ_Data data - input
System i データ待ち行列に書き込まれる、データ・オブジェクト。

cwb_Boolean commit - input
このフラグは使用されなくなったため無視されます。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_BAD_DATA_LENGTH
データの長さが正しくありません。

CWBDQ_INVALID_MESSAGE_LENGTH
無効なメッセージ長。

CWBDQ_INVALID_QUEUE_HANDLE
待ち行列ハンドルが無効。

CWBDQ_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBDQ_CANNOT_CONVERT
データを変換できません。

使用法

この関数を使用する場合は、あらかじめ cwbDQ_Open か cwbDQ_OpenEx、および cwbDQ_CreateData を発行する必要があります。

データ待ち行列: 属性 API

これらの API を使用して、System i データ待ち行列の属性の宣言を行います。データ待ち行列の作成時やデータ待ち行列属性の入手時には、属性オブジェクトを使用します。

cwbDQ_CreateAttr:

System i Access for Windows の cwbDQ_CreateAttr コマンドを使用します。

目的

データ待ち行列属性オブジェクトを作成します。cwbDQ_Create API または cwbDQ_CreateEx API の入力として使用する前に、この API によって戻されたハンドルを使用して、データ待ち行列に指定したい特定の属性を設定することができます。このハンドルは、cwbDQ_GetQueueAttr API の入力として使用した後、データ待ち行列の特定の属性を調べる場合にも使用することができます。

構文

```
cwbDQ_Attr CWB_ENTRY cwbDQ_CreateAttr(void);
```

パラメーター

なし (None)

戻りコード

以下は、共通の戻り値です。

cwbDQ_Attr - cwbDQ_Attr オブジェクトのハンドル。
このハンドルは、属性の入手と設定に使用します。作成後、属性オブジェクトは、次のデフォルト値を持ちます。

- 最大レコード長 - 1000
- 順序 - FIFO (先入れ先出し法)
- 権限 - LIBCRTAUT
- 記憶装置へ強制 - FALSE
- 送信側 ID - FALSE (偽)
- キーの長さ - 0

使用法

なし (None)

cwbDQ_DeleteAttr:

System i Access for Windows の cwbDQ_DeleteAttr コマンドを使用します。

目的

データ待ち行列属性を削除します。

構文

```
unsigned int CWB_ENTRY cwbDQ_DeleteAttr(
                            cwbDQ_Attr          queueAttributes);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

使用法

なし (None)

cwbDQ_GetAuthority:

System i Access for Windows の cwbDQ_GetAuthority コマンドを使用します。

目的

他のユーザーがデータ待ち行列に対して持つ権限の属性を取得します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetAuthority(
                            cwbDQ_Attr          queueAttributes,
                            unsigned short     *authority);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

unsigned short * authority - output
権限の書き込み先である無符号短精度整数を指すポインター。この値は、次の定義済みの値のいずれかです。

- CWBDQ_ALL
- CWBDQ_EXCLUDE
- CWBDQ_CHANGE
- CWBDQ_USE
- CWBDQ_LIBCRTAUT

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

使用法

なし (None)

cwbDQ_GetDesc:

System i Access for Windows の cwbDQ_GetDesc コマンドを使用します。

目的

データ待ち行列の記述についての属性を取得します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetDesc(
                            cwbDQ_Attr          queueAttributes,
                            char               *description);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

char * description - output
記述が書き込まれる先の、51 文字バッファーを指すポインター。記述は、ASCIIZ ストリングです。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

使用法

なし (None)

cwbDQ_GetForceToStorage:

System i Access for Windows の cwbDQ_GetForceToStorage コマンドを使用します。

目的

レコードが待ち行列に入れられた時点で、それらのレコードを強制的に補助記憶装置に移すかどうかに関する属性を設定します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetForceToStorage(
                            cwbDQ_Attr          queueAttributes,
                            cwb_Boolean        *forceToStorage);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

cwb_Boolean * forceToStorage - output
強制記憶標識の書き込み先であるブールを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

使用法

なし (None)

cwbDQ_GetKeySize:

System i Access for Windows の cwbDQ_GetKeySize コマンドを使用します。

目的

バイト単位でのキー・サイズについての属性を取得します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetKeySize(
                            cwbDQ_Attr          queueAttributes,
                            unsigned short     *keySize);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

unsigned short * keySize - output
キー・サイズが書き込まれる先の無符号短精度整数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

使用法

なし (None)

cwbDQ_GetMaxRecLen:

System i Access for Windows の cwbDQ_GetMaxRecLen コマンドを使用します。

目的

データ待ち行列の最大レコード長を取得します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetMaxRecLen(
                            cwbDQ_Attr          queueAttributes,
                            unsigned long      *maxRecordLength);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

unsigned long * maxRecordLength - output
最大レコード長が書き込まれる先の無符号長精度整数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

使用法

なし (None)

cwbDQ_GetOrder:

System i Access for Windows の cwbDQ_GetOrder コマンドを使用します。

目的

待ち行列順序についての属性を取得します。順序が CWBDQ_SEQ_LIFO の場合、最後に書き込まれたレコードが最初に読み取られます (後入れ先出し法)。順序が CWBDQ_SEQ_FIFO の場合、最初に書き込まれたレコードが最初に読み取られます (先入れ先出し法)。順序が CWBDQ_SEQ_KEYED である場合、データ待ち行列からレコードを読み取る順序は、データ・オブジェクトの検索順序属性の値、ならびに、cwbDQ_SetKey API に指定されたキー値に応じて異なります。複数のレコードに検索順序の条件を満たすキーが含まれている場合は、これらのレコードの間で、FIFO (先入れ先出し法) 方式が使用されます。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetOrder(
                            cwbDQ_Attr          queueAttributes,
                            unsigned short     *order);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

unsigned short * order - output
順序が書き込まれる先の無符号短精度整数を指すポインター。指定できる値は以下のとおりです。

- CWBDQ_SEQ_LIFO
- CWBDQ_SEQ_FIFO
- CWBDQ_SEQ_KEYED

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

使用法

なし (None)

cwbDQ_GetSenderID:

System i Access for Windows の cwbDQ_GetSenderID コマンドを使用します。

目的

送信側に関する情報を待ち行列上のそれぞれのレコードと一緒に保持するかどうかに関する属性を取得します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetSenderID(
                            cwbDQ_Attr         queueAttributes,
                            cwb_Boolean       *senderID);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

cwb_Boolean * senderID - output
送信側 ID 標識が書き込まれる先のブールを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

使用法

なし (None)

cwbDQ_SetAuthority:

System i Access for Windows の cwbDQ_SetAuthority コマンドを使用します。

目的

他のユーザーが持つデータ待ち行列への権限についての属性を設定します。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetAuthority(
                            cwbDQ_Attr          queueAttributes,
                            unsigned short      authority);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

unsigned short authority - input
システム上の他のユーザーが保持する、データ待ち行列にアクセスする権限。権限には、次のいずれかの定義済みタイプを使用してください。

- CWBDQ_ALL
- CWBDQ_EXCLUDE
- CWBDQ_CHANGE

- CWBDQ_USE
- CWBDQ_LIBCRTAUT

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

CWBDQ_INVALID_AUTHORITY
待ち行列の権限が無効。

使用法

なし (None)

cwbDQ_SetDesc:

System i Access for Windows の cwbDQ_SetDesc コマンドを使用します。

目的

データ待ち行列の記述についての属性を設定します。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetDesc(
                            cwbDQ_Attr          queueAttributes,
                            char               *description);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

char * description - input
データ待ち行列についての記述が入っている ASCIIZ ストリングを指すポインター。記述の最大長は、50 文字です。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

CWBDQ_INVALID_QUEUE_TITLE
待ち行列の記述が長すぎます。

使用法

なし (None)

cwbDQ_SetForceToStorage:

System i Access for Windows の cwbDQ_SetForceToStorage コマンドを使用します。

目的

レコードが待ち行列に入れられた時に、それらのレコードを強制的に補助記憶装置に移すかどうかに関する属性を設定します。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetForceToStorage(
                            cwbDQ_Attr          queueAttributes,
                            cwb_Boolean         forceToStorage);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

cwb_Boolean forceToStorage - input
レコードが待ち行列に入れられたときに、それぞれのレコードを強制的に補助記憶装置に移すかどうかを示すブール標識。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

使用法

なし (None)

cwbDQ_SetKeySize:

System i Access for Windows の cwbDQ_SetKeySize コマンドを使用します。

目的

バイトでのキー・サイズについての属性を設定します。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetKeySize(
                            cwbDQ_Attr          queueAttributes,
                            unsigned short      keySize);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

unsigned short keySize - input
バイトでのキーのサイズ。この値は、順序が LIFO または FIFO の場合はゼロであり、キー順データ待ち行列の場合は 1 から 256 の間の値でなければなりません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_KEY_LENGTH
キーの長さが無効。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

使用法

なし (None)

cwbDQ_SetMaxRecLen:

System i Access for Windows の cwbDQ_SetMaxRecLen コマンドを使用します。

目的

データ待ち行列について最大レコード長を設定します。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetMaxRecLen(
                            cwbDQ_Attr          queueAttributes,
                            unsigned long       maxRecordLength);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

unsigned long maxLength - input
データ待ち行列レコードについての最大長。この値は、1 と 31744 の間の値でなければなりません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

CWBDQ_INVALID_QUEUE_LENGTH
無効な待ち行列レコード長。

使用法

なし (None)

cwbDQ_SetOrder:

System i Access for Windows の cwbDQ_SetOrder コマンドを使用します。

目的

待ち行列の順序についての属性を設定します。順序が CWBDQ_SEQ_LIFO の場合、最後に書き込まれたレコードが最初に読み取られます (後入れ先出し法)。順序が CWBDQ_SEQ_FIFO の場合、最初に書き込まれたレコードが最初に読み取られます (先入れ先出し法)。順序が CWBDQ_SEQ_KEYED である場合、データ待ち行列からレコードを読み取る順序は、データ・オブジェクトの検索順序属性の値、ならびに、cwbDQ_SetKey API に指定されたキー値に応じて異なります。複数のレコードに検索順序の条件を満たすキーが含まれている場合は、これらのレコードの間で、FIFO (先入れ先出し法) 方式が使用されます。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetOrder(
                            cwbDQ_Attr          queueAttributes,
                            unsigned short      order);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

unsigned short order - input
新規の入力が待ち行列に入れられる順序。順序には、次のいずれかの定義済みタイプを使用してください。

- CWBDQ_SEQ_LIFO
- CWBDQ_SEQ_FIFO
- CWBDQ_SEQ_KEYED

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

CWBDQ_INVALID_ORDER
待ち行列の順序が無効。

使用法

なし (None)

cwbDQ_SetSenderID:

System i Access for Windows の cwbDQ_SetSenderID コマンドを使用します。

目的

送信側に関する情報を待ち行列上のそれぞれのレコードと一緒に保持するかどうかに関する属性を設定します。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetSenderID(
                            cwbDQ_Attr          queueAttributes,
                            cwb_Boolean         senderID);
```

パラメーター

cwbDQ_Attr queueAttributes - input
cwbDQ_CreateAttr への先行の呼び出しで戻されたデータ待ち行列属性のハンドル。

cwb_Boolean senderID - input
送信側に関する情報を待ち行列に入れられているレコードと一緒に保持するかどうかに関するブール標識。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_ATTRIBUTE_HANDLE
属性ハンドルが無効。

使用法

なし (None)

データ待ち行列: 読み取りおよび書き込み API

これらの System i Access for Windows API を使用して、データ待ち行列との間で書き込みおよび読み取りを行います。

cwbDQ_CreateData:

System i Access for Windows の cwbDQ_CreateData コマンドを使用します。

目的

データ・オブジェクトを作成します。作成したデータ・オブジェクトは、データ待ち行列からのデータの読み取りとデータ待ち行列へのデータの書き込みの両方に使用します。

構文

```
cwbDQ_Data CWB_ENTRY cwbDQ_CreateData(void);
```

パラメーター

なし (None)

戻りコード

以下は、共通の戻り値です。

cwbDQ_Data - データ・オブジェクトのハンドル
作成後、データ・オブジェクトは、次のデフォルト値を持ちます。

- データ - NULL および長さ 0
- キー - NULL および長さ 0
- 送信側 ID 情報 - NULL
- 検索順序 - NONE (なし)
- 変換 - FALSE (偽)

使用法

なし (None)

cwbDQ_DeleteData:

System i Access for Windows の cwbDQ_DeleteData コマンドを使用します。

目的

データ・オブジェクトを削除します。

構文

```
unsigned int CWB_ENTRY cwbDQ_DeleteData(
                            cwbDQ_Data          data);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_GetConvert:

System i Access for Windows の cwbDQ_GetConvert コマンドを使用します。

目的

データ・ハンドル用の変換フラグの値を取得します。この変換フラグにより、ホストへ送信したデータおよびホストから受信したデータが、変換された (例えば、ASCII から EBCDIC に変換された) CCSID であるかどうかが判別されます。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetConvert(
                            cwbDQ_Data          data,
                            cwb_Boolean        *convert);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

cwb_Boolean * convert - output
変換フラグが書き込まれる先のブール値を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_GetData:

System i Access for Windows の cwbDQ_GetData コマンドを使用します。

目的

データ・オブジェクトのデータ属性を取得します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetData(
                            cwbDQ_Data          data,
                            unsigned char      *dataBuffer);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned char * data - output
データを指すポインター。データには、組み込み NULL が含まれている場合があります。したがって、このデータは、ASCIIZ ストリングではありません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_GetDataAddr:

System i Access for Windows の cwbDQ_GetDataAddr コマンドを使用します。

目的

データ・バッファーの位置のアドレスを取得します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetDataAddr(
                            cwbDQ_Data          data,
                            unsigned char     **dataBuffer);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned char * * data - output
バッファー・アドレスが書き込まれる場所を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

CWBDQ_ADDRESS_NOT_SET
アドレスが cwbDQ_SetDataAddr で設定されていません。

使用法

この関数は、データが保管されている場所のアドレスを検索するのに使用します。データ・アドレスは、cwbDQ_SetDataAddr API を使用して設定する必要があります。そうでない場合は、戻りコード CWBDQ_ADDRESS_NOT_SET が戻されます。

cwbDQ_GetDataLen:

System i Access for Windows の cwbDQ_GetDataLen コマンドを使用します。

目的

データ・オブジェクトのデータ長属性を取得します。この属性は、データ・オブジェクトの全長です。読み取られたデータの長さを取得するには、cwbDQ_GetRetDataLen API を使用します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetDataLen(
                            cwbDQ_Data          data,
                            unsigned long      *dataLength);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned long * dataLength - output
データの長さが書き込まれる先の無符号長精度整数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_GetKey:

System i Access for Windows の cwbDQ_GetKey コマンドを使用します。

目的

データ・オブジェクトのキー属性で、前に cwbDQ_SetKey API によって設定されたキー属性を取得します。このキーが、キー順データ待ち行列へのデータの書き込みに使用するキーです。検索順序に使う以外に、このキーは、キー順データ待ち行列からデータを読み取る場合にも使用します。検索されたレコードと関連したキーは、cwbDQ_GetRetKey API を呼び出すと取得することができます。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetKey(
                            cwbDQ_Data          data,
                            unsigned char      *key);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned char * key - output
キーを指すポインター。このキーには、組み込み NULL が含まれることがあります。したがって、このキーは、ASCIIZ ストリングではありません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_GetKeyLen:

System i Access for Windows の cwbDQ_GetKeyLen コマンドを使用します。

目的

データ・オブジェクトのキーの長さ属性を取得します。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetKeyLen(
                            cwbDQ_Data          data,
                            unsigned short     *keyLength);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned short * keyLength - output
キーの長さが書き込まれる先の無符号短精度整数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_GetRetDataLen:

System i Access for Windows の cwbDQ_GetRetDataLen コマンドを使用します。

目的

戻されたデータの長さを取得します。戻されたデータ長は、cwbDQ_Read または cwbDQ_Peek API が呼び出されるまではゼロですが、呼び出された後は、実際に戻されたデータの長さになります。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetRetDataLen(
                            cwbDQ_Data          data,
                            unsigned long      *retDataLength);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned long * retDataLength - output
戻されたデータの長さが書き込まれる先の、無符号長精度整数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_GetRetKey:

System i Access for Windows の cwbDQ_GetRetKey コマンドを使用します。

目的

データ・オブジェクトの戻されたキーを取得します。これは、キー順データ待ち行列から検索されるメッセージに関連したキーです。検索順序が CWBDQ_EQUAL 以外の値である場合、このキーは、メッセージの検索に使用されたキーとは異なることがあります。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetRetKey(
                            cwbDQ_Data          data,
                            unsigned char      *key);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned char * retKey - output
戻されたキーを指すポインター。このキーには、組み込み NULL が含まれることがあります。したがって、このキーは ASCIIZ ストリングではありません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_GetRetKeyLen:

System i Access for Windows の cwbDQ_GetRetKeyLen コマンドを使用します。

目的

データ・オブジェクトの戻されたキーの長さ属性を取得します。これは、cwbDQ_GetKey API によって戻されるキーの長さです。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetRetKeyLen(
                            cwbDQ_Data          data,
                            unsigned short     *retKeyLength);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned short * retKeyLength - output
キーの長さが書き込まれる先の無符号短精度整数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_GetSearchOrder:

System i Access for Windows の cwbDQ_GetSearchOrder コマンドを使用します。

目的

オープン属性の検索順序を取得します。検索順序は、検索するレコードのキーと cwbDQ_SetKey API 上に指定されたキー値との関係の識別に使用するためにキー順データ待ち行列の読み取り時や検査時に使用されます。データ待ち行列順序属性が CWBDQ_SEQ_KEYED 以外の場合、このプロパティーは無視されます。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetSearchOrder(
                            cwbDQ_Data          data,
                            unsigned short     *searchOrder);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned short * searchOrder - output
順序が書き込まれる先の無符号短精度整数を指すポインター。指定できる値は以下のとおりです。

- CWBDQ_NONE
- CWBDQ_EQUAL
- CWBDQ_NOT_EQUAL
- CWBDQ_GT_OR_EQUAL
- CWBDQ_GREATER
- CWBDQ_LT_OR_EQUAL
- CWBDQ_LESS

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_GetSenderInfo:

System i Access for Windows の cwbDQ_GetSenderInfo コマンドを使用します。

目的

オープン属性の送信側情報属性を取得します。データ待ち行列の送信側 ID 属性が作成時に設定された場合にのみ、この情報を使用することができます。

構文

```
unsigned int CWB_ENTRY cwbDQ_GetSenderInfo(
                            cwbDQ_Data          data,
                            unsigned char      *senderInfo);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned char * senderInfo - output
送信側情報が書き込まれる先の、36 文字バッファーを指すポインター。このバッファーには、次のものが入っています。

- ジョブ名 (10 バイト)
- ユーザー名 (10 バイト)
- ジョブ ID (6 バイト)
- ユーザー・プロファイル (10 バイト)

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_SetConvert:

System i Access for Windows の cwbDQ_SetConvert コマンドを使用します。

目的

変換フラグを設定します。フラグが設定されている場合、書き込まれるすべてのデータは、PC CCSID (例えば ASCII) からホスト CCSID (例えば EBCDIC) に変換され、読み取られるすべてのデータは、ホスト CCSID (例えば EBCDIC) から PC CCSID (例えば ASCII) に変換されます。デフォルトの設定は、データ変換なしです。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetConvert(
                            cwbDQ_Data          data,
                            cwb_Boolean         convert);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

cwb_Boolean convert - input
待ち行列へ書き込むデータと待ち行列から読み取るデータを、CCSID 変換するかしないかを指示するフラグです。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

使用法

なし (None)

cwbDQ_SetData:

System i Access for Windows の cwbDQ_SetData コマンドを使用します。

目的

データ・オブジェクトのデータとデータ長属性を設定します。デフォルトでは、長さはゼロでデータを持ちません。この関数は、データのコピーを作成します。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetData(
                            cwbDQ_Data          data,
                            unsigned char      *dataBuffer,
                            unsigned long       dataLength);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned char * dataBuffer - input
データを指すポインター。データには、組み込み NULL が含まれている場合があります。したがって、このデータは、ASCIIZ ストリングではありません。

unsigned long dataLength - input
バイトでのデータの長さ。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

CWBDQ_BAD_DATA_LENGTH
データの長さが正しくありません。

使用法

この関数は、少量のデータを書き込みたい場合、またはアプリケーションの中でデータ用のメモリー管理を行いたくない場合に使用します。データがコピーされるため、ユーザーのアプリケーションのパフォーマンスに影響を及ぼす場合があります。

cwbDQ_SetDataAddr:

System i Access for Windows の cwbDQ_SetDataAddr コマンドを使用します。

目的

データ・オブジェクトのデータとデータ長属性を設定します。デフォルトでは、長さはゼロでデータを持ちません。この関数は、データをコピーしません。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetDataAddr(
                            cwbDQ_Data          data,
                            unsigned char      *dataBuffer,
                            unsigned long       dataLength);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned char * dataBuffer - input
データを指すポインター。データには、組み込み NULL が含まれている場合があります。したがって、このデータは、ASCIIZ ストリングではありません。

unsigned long dataLength - input
バイトでのデータの長さ。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

CWBDQ_BAD_DATA_LENGTH
データの長さが正しくありません。

使用法

大量のデータを扱う場合、またはユーザーのアプリケーションの中でメモリーを管理したい場合には、この関数の方が便利です。データはコピーされないため、パフォーマンスは向上します。

cwbDQ_SetKey:

System i Access for Windows の cwbDQ_SetKey コマンドを使用します。

目的

データ属性のキーとキーの長さ属性を設定します。このキーが、キー順データ待ち行列へのデータの書き込みに使用するキーです。検索順序のほかに、このキーは、キー順データ待ち行列からデータを読み取る場合

にも使用します。デフォルトでは、長さはゼロでキーを持ちません。このデフォルト値は、非キー順 (LIFO または FIFO) データ待ち行列についての正しい値です。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetKey(
                            cwbDQ_Data          data,
                            unsigned char      *key,
                            unsigned short      keyLength);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned char * key - input
キーを指すポインター。このキーには、組み込み NULL が含まれることがあります。したがって、このキーは、ASCIIZ ストリングではありません。

unsigned short keyLength - input
キーの長さ (バイト数)。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

CWBDQ_BAD_KEY_LENGTH
キーの長さは正しくありません。

使用法

なし (None)

cwbDQ_SetSearchOrder:

System i Access for Windows の cwbDQ_SetSearchOrder コマンドを使用します。

目的

オープン属性の検索順序を設定します。デフォルトは、検索順序なしです。 cwbDQ_SetKey API が呼び出されると、検索順序はキー順に変更されます。別の検索順序に設定するのに、この API を使用します。検索順序は、検索するレコードのキーと cwbDQ_SetKey API 上に指定されたキー値との関係の識別に使用するためにキー順データ待ち行列の読み取り時や検査時に使用されます。データ待ち行列順序属性が CWBDQ_SEQ_KEYED 以外の場合、このプロパティーは無視されます。

構文

```
unsigned int CWB_ENTRY cwbDQ_SetSearchOrder(
                            cwbDQ_Data          data,
                            unsigned short      searchOrder);
```

パラメーター

cwbDQ_Data data - input
cwbDQ_CreateData への先行の呼び出しで戻されたデータ・オブジェクトのハンドル。

unsigned short searchOrder - input
キー順待ち行列から読み取る場合に使用する順序。指定できる値は以下のとおりです。

- CWBDQ_NONE
- CWBDQ_EQUAL
- CWBDQ_NOT_EQUAL
- CWBDQ_GT_OR_EQUAL
- CWBDQ_GREATER
- CWBDQ_LT_OR_EQUAL
- CWBDQ_LESS

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBDQ_INVALID_DATA_HANDLE
データ・ハンドルが無効。

CWBDQ_INVALID_SEARCH
無効な検索順序。

使用法

なし (None)

例: データ待ち行列 API の使用法

System i データ待ち行列 API の使用法について、以下の例で説明します。

```
// Sample Data Queues application

#ifdef UNICODE
   #define _UNICODE
#endif
#include <windows.h>

// Include the necessary DQ Classes
#include <stdlib.h>
#include <iostream.h>
#include "cwbdq.h"

/*********************************************************************/

void main()
{

   cwbDQ_Attr queueAttributes;
   cwbDQ_QueueHandle queueHandle;
   cwbDQ_Data queueData;

   // Create an attribute object
   if ( (queueAttributes = cwbDQ_CreateAttr()) == 0 )
```

```
      return;

   // Set the maximum record length to 100
   if ( cwbDQ_SetMaxRecLen(queueAttributes,
                           100) != 0 )
      return;

   // Set the order to First-In-First-Out
   if (cwbDQ_SetOrder(queueAttributes, CWBDQ_SEQ_FIFO) != 0 )
      return;

   // Create the data queue DTAQ in library QGPL on system SYS1
   if ( cwbDQ_Create(_TEXT("DTAQ"),
                     _TEXT("QGPL"),
                     _TEXT("SYSNAMEXXX"),
                     queueAttributes,
                     NULL) != 0 )
      return;

   // Delete the attributes
   if ( cwbDQ_DeleteAttr( queueAttributes ) != 0 )
      return;

   // Open the data queue
   if ( cwbDQ_Open(_TEXT("DTAQ"),
                   _TEXT("QGPL"),
                   _TEXT("SYSNAMEXXX"),
                   &queueHandle,
                   NULL) != 0 )

       return;

   // Create a data object
   if ( (queueData = cwbDQ_CreateData()) == 0 )
      return;

   // Set the data length and the data
   if ( cwbDQ_SetData(queueData, (unsigned char*)"Test Data!", 10) != 0 )
      return;

   // Write the data to the data queue
   if ( cwbDQ_Write(queueHandle, queueData, CWB_TRUE, NULL) != 0 )
      return;

   // Delete the data object
   if ( cwbDQ_DeleteData(queueData) != 0 )
      return;

   // Close the data queue
   if ( cwbDQ_Close(queueHandle) != 0 )
      return;

}
```

System i Access for Windows のデータ形式変更および各国語サポート (NLS) API

データ形式変更および各国語サポート (NLS) API を使用して、ご使用のアプリケーションで System i Access for Windows データの形式変更を行えるようにします。

System i Access for Windows データ形式変更 API

System i Access for Windows の**データ形式変換**アプリケーション・プログラミング・インターフェース (API) を使用すると、クライアント/サーバー・アプリケーションで System i 数値データの形式変更 (シス

テム形式と PC 形式との相互変更) を行えるようになります。 System i 数値データの送受信をシステムとの間で行う場合に、形式変更が必要になることがあります。データ形式変更 API は、数多くの数値形式の変換をサポートします。

System i Access for Windows データ形式変更 API に必要なファイル

ヘッダー・ファイル	インポート・ライブラリー	ダイナミック・リンク・ライブラリー
cwbdt.h	cwbapi.lib	cwbdt.dll

Programmer's Toolkit:

Programmer's Toolkit には、データ形式変更の資料、cwbdt.h ヘッダー・ファイルへのアクセス、およびプログラム例へのリンクが用意されています。この情報にアクセスするには、Programmer's Toolkit をオープンして、「**データ操作**」 → 「**C/C++ API**」と選択します。

注: コード例を使用することで、 633 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』の条件に同意します。

関連資料

7 ページの『接続 API 用の System i 名の形式』
パラメーターとして System i 名を取得する API では、3 つの異なる形式の名前を使用できます。

8 ページの『OEM、ANSI、およびユニコードの考慮事項』
System i Access for Windows の、ストリング・パラメーターを受け入れる C/C++ API の大部分は、OEM、ANSI、Unicode の 3 つのうち、いずれかの形式になっています。

System i Access for Windows データ形式変更 API リスト:

以下の System i Access for Windows データ形式変更 API は、アルファベット順にリストされています。

注: ストリングを受け入れる System i Access for Windows データ形式変更 API は、ユニコード・バージョンで提供されます。これらの API の場合、「ASCII」は「Wide」で置換されます (例えば、cwbDT_ASCII11ToBin4 のユニコード・バージョンは cwbDT_Wide11ToBin4 になります)。以下の表は、これらの API を示しています。ユニコード・バージョンは、それらに対応している ASCII バージョンとは異なる構文、パラメーター、および戻り値を使用します。

cwbDT_ASCII11ToBin4:

System i Access for Windows の cwbDT_ASCII11ToBin4 コマンドを使用します。

目的

11 桁の ASCII 数字を、有効桁の最高位バイトを最初に保管して、4 バイト整数に (正確に) 変換します。(ソース・ストリングはゼロで終わっていなくてもかまいません。)この関数は、ASCII 数値データを、System i 整数形式に変換するために使用されます。

ユニコード・バージョン

cwbDT_Wide11ToBin4

構文

```
unsigned int CWB_ENTRY cwbDT_ASCII11ToBin4(
                            char *target,
                            char *source);
```

パラメーター

char * target - output
ターゲット (4 バイト整数) を指すポインター。

char * source - input
ソース (11 バイトの ASCII) を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

CWB_BUFFER_OVERFLOW
オーバーフロー・エラー。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

ターゲット・データは、有効桁の最高位バイトが最初に保管されます。これはシステムが使用する System i 形式であり、Intel® x86 プロセッサーが使用する形式とは逆になっています。 ASCII ソース・データで有効な形式は以下のとおりです。

- [blankspaces][sign][blankspaces][digits] または
- [sign][blankspaces][digits][blankspaces]

例:

```
 "   +    123"
 "-    123   "
 "     +123  "
 "        123"
 "       -123"
 "+123       "
```

cwbDT_ASCII6ToBin2:

System i Access for Windows の cwbDT_ASCII6ToBin2 コマンドを使用します。

目的

6 桁の ASCII 数字を、有効桁の最高位バイトを最初に保管して、2 バイト整数に (正確に) 変換します。(ソース・ストリングはゼロで終わっていなくてもかまいません。)この関数は、ASCII 数値データを、System i 整数形式に変換するために使用されます。

ユニコード・バージョン

cwbDT_Wide6ToBin2

構文

```
unsigned int CWB_ENTRY cwbDT_ASCII6ToBin2(
                              char *target,
                              char *source);
```

パラメーター

char * target - output
ターゲット (2 バイト整数) を指すポインター。

char * source - input
ソース (6 バイト ASCII) を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

CWB_BUFFER_OVERFLOW
オーバーフロー・エラー。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

ターゲット・データは、有効桁の最高位バイトが最初に保管されます。これはシステムが使用する System i 形式であり、Intel x86 プロセッサーが使用する形式とは逆になっています。 ASCII ソース・データで有効な形式は以下のとおりです。

- [blankspaces][sign][blankspaces][digits] または
- [sign][blankspaces][digits][blankspaces]

例:

```
 " + 123"
 "- 123 "
 " +123 "
 "   123"
 "  -123"
 "+123  "
```

cwbDT_ASCIIPackedToPacked:

System i Access for Windows の cwbDT_ASCIIPackedToPacked コマンドを使用します。

目的

ASCII パック形式のデータをパック 10 進数に変換します。この関数は、ASCII ファイルのデータを、System i 形式に変換するために使用します。

ユニコード・バージョン

なし

構文

```
unsigned int CWB_ENTRY cwbDT_ASCIIPackedToPacked(
                            char         *target,
                            char         *source,
                            unsigned long length);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース・データを指すポインター。

unsigned long length - input
変換するソース・データのバイト数。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

呼び出し側は、ターゲット情報を保持するための十分なスペースを確保しておく必要があります。この関数は、パック 10 進数データの各ハーフバイトが 0 から 9 までの範囲内にあるかどうかを検査します。最後のハーフバイトは例外で、ここには符号標識 (0x3 または 0xb) が入ります。

cwbDT_ASCIIToHex:

System i Access for Windows の cwbDT_ASCIIToHex コマンドを使用します。

目的

データを ASCII (16 進表示) から 2 進数に変換します。ソース中の 2 バイトごとに、1 バイトがターゲットに保管されます。

ユニコード・バージョン

cwbDT_WideToHex

構文

```
unsigned int CWB_ENTRY cwbDT_ASCIIToHex(
                            char         *target,
                            char         *source,
                            unsigned long length);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース (ASCII 16 進数) データを指すポインター。

unsigned long length - input
変換するソース・データのバイト数 /2。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

ソース・データの 'long' バイトに対して、'long'/2 バイトのターゲット・データが保管されます。呼び出し側は、ターゲット情報を保持するための十分なスペースを確保しておく必要があります。

cwbDT_ASCIIToPacked:

System i Access for Windows の cwbDT_ASCIIToPacked コマンドを使用します。

目的

ASCII 数値データをパック 10 進数形式に変換します。この関数を使用すると、ASCII テキスト・データを変換して、System i プラットフォームで使用可能にすることができます。

ユニコード・バージョン

cwbDT_WideToPacked

構文

```
unsigned int CWB_ENTRY cwbDT_ASCIIToPacked(
                            char         *target,
                            char         *source,
                            unsigned long length,
                            unsigned long decimalPosition);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース・データを指すポインター。ゼロで終わる必要があります。

unsigned long length - input
変換するターゲット・データのバイト数。

unsigned long decimalPosition - input
小数点の位置。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

CWB_BUFFER_OVERFLOW
オーバーフロー・エラー。

CWB_NOT_ENOUGH_MEMORY
一時メモリーの割り振りができません。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

呼び出し側は、ターゲット情報を保持するための十分なスペースを確保しておく必要があります。符号用のハーフバイトには、負数を表す場合は 16 進の 0xd がセットされ、正数を表す場合は 0xc がセットされます。0 <= 小数点位置 < (長さ * 2)。 ASCII 数値データで有効な形式は以下のとおりです。

- [blankspaces][sign][blankspaces][digits] または
- [sign][blankspaces][digits][blankspaces] または
- [sign][digits][.digits][blankspaces] または
- [blankspaces][sign][digits][.digits][blankspaces]

例:

```
"   +  123¥0"
"- 123   ¥0"
"     +123  ¥0"
"    123¥0"
"       -12.3¥0"
"+1.23       ¥0"
```

cwbDT_ASCIIToZoned:

System i Access for Windows の cwbDT_ASCIIToZoned コマンドを使用します。

目的

ASCII 数値データを EBCDIC ゾーン 10 進形式に変換します。この関数を使用すると、ASCII テキスト・データを変換して、System i プラットフォームで使用可能にすることができます。

ユニコード・バージョン

cwbDT_WideToZoned

構文

```
unsigned int CWB_ENTRY cwbDT_ASCIIToZoned(
                            char         *target,
                            char         *source,
                            unsigned long length,
                            unsigned long decimalPosition);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース・データを指すポインター。ゼロで終わる必要があります。

unsigned long length - input
変換するターゲット・データのバイト数。

unsigned long decimalPosition - input
小数点の位置。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

CWB_BUFFER_OVERFLOW
オーバーフロー・エラー。

CWB_NOT_ENOUGH_MEMORY
一時メモリーの割り振りができません。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

呼び出し側は、ターゲット情報を保持するための十分のスペースを確保しておく必要があります。符号用のハーフバイトには、負数を表す場合は 16 進の 0xd がセットされ、正数を表す場合は 0xc がセットされます。0 <= 小数点位置 <= 長さです。ASCII 数値データで有効な形式は以下のとおりです。

- [blankspaces][sign][blankspaces][digits] または
- [sign][blankspaces][digits][blankspaces] または
- [sign][digits][.digits][blankspaces] または

- [blankspaces][sign][digits][.digits][blankspaces]

例:

```
"   +  123¥0"
"- 123   ¥0"
"     +123  ¥0"
"    123¥0"
"       -12.3¥0"
"+1.23       ¥0"
```

cwbDT_ASCIIZonedToZoned:

System i Access for Windows の cwbDT_ASCIIZonedToZoned コマンドを使用します。

目的

データを、ASCII ゾーン 10 進形式から EBCDIC ゾーン 10 進数に変換します。この関数を使用すると、ASCII ファイルのデータを変換して、System i プラットフォームで使用可能にすることができます。

ユニコード・バージョン

なし。

構文

```
unsigned int CWB_ENTRY cwbDT_ASCIIZonedToZoned(
                            char         *target,
                            char         *source,
                            unsigned long length);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース・データを指すポインター。

unsigned long length - input
変換するソース・データのバイト数。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

ASCII ゾーン 10 進形式中の各バイトの左半分 (0x3) は、最後のバイト (符号) を除き EBCDIC ゾーン・データの左のハーフバイト中の 0xf に変換されます。ASCII ゾーン 10 進データ中の各バイトの左半分は、最後のバイトを除き 0x3 でなければなりません。この関数はそれを検査します。最後のバイトの高位

の半分は 0x3 または 0xb でなければなりません。ASCII ゾーン 10 進データ中の各バイトの右半分は 0 から 9 の範囲内でなければなりません。

cwbDT_Bin2ToASCII6:

System i Access for Windows の cwbDT_Bin2ToASCII6 コマンドを使用します。

目的

有効桁の最高位バイトを最初に保管した 2 バイト整数を、(正確に) 6 桁の ASCII 数字に変換します。(ターゲットはゼロで終わりません。) この関数を使用すると、System i の数値データを ASCII に変換することができます。

ユニコード・バージョン

cwbDT_Bin2ToWide6

構文

```
unsigned int CWB_ENTRY cwbDT_Bin2ToASCII6(
                            char *target,
                            char *source);
```

パラメーター

char * target - output
ターゲット (6 バイト) の領域を指すポインター。

char * source - input
ソース (2 バイト整数) を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

使用法

ソース・データには、有効桁の最高位バイトが最初に保管されるものと想定します。これはシステムが使用する System i 形式であり、Intel x86 プロセッサーが使用する形式とは逆になっています。

cwbDT_Bin2ToBin2:

System i Access for Windows の cwbDT_Bin2ToBin2 コマンドを使用します。

目的

2 バイト整数のバイトの順序を入れ替えます。この関数を使用すると、2 バイトの整数と System i 形式とを相互に変換することができます。

ユニコード・バージョン

なし

構文

```
unsigned int CWB_ENTRY cwbDT_Bin2ToBin2(
                            char *target,
                            char *source);
```

パラメーター

char * target - output
ターゲット (2 バイト整数) を指すポインター。

char * source - input
ソース (2 バイト整数) を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

使用法

ソース・データとターゲット・データはオーバーラップしてはなりません。以下に、この変換の結果の例を示します。

- ソース・データ: 0x1234
- ターゲット・データ: 0x3412

cwbDT_Bin4ToASCII11:

System i Access for Windows の cwbDT_Bin4ToASCII11 コマンドを使用します。

目的

有効桁の最高位バイトを最初に保管した 4 バイト整数を、(正確に) 11 桁の ASCII 数字に変換します。(ターゲットはゼロで終わりません。) この関数を使用すると、System i の数値データを ASCII に変換することができます。

ユニコード・バージョン

cwbDT_Bin4ToWide11

構文

```
unsigned int CWB_ENTRY cwbDT_Bin4ToASCII11(
                            char *target,
                            char *source );
```

パラメーター

char * target - output
ターゲット (11 バイト) 域を指すポインター。

char * source - input
ソース (4 バイト整数) を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

使用法

ソース・データには、有効桁の最高位バイトが最初に保管されるものと想定します。これはシステムが使用する System i 形式であり、Intel x86 プロセッサーが使用する形式とは逆になっています。

cwbDT_Bin4ToBin4:

System i Access for Windows の cwbDT_Bin4ToBin4 コマンドを使用します。

目的

4 バイト整数のバイトの順序を入れ替えます。この関数を使用すると、4 バイトの整数と System i 形式とを相互に変換することができます。

ユニコード・バージョン

なし

構文

```
unsigned int CWB_ENTRY cwbDT_Bin4ToBin4(
                            char *target,
                            char *source);
```

パラメーター

char * target - output
ターゲット (4 バイト整数) を指すポインター。

char * source - input
ソース (4 バイト整数) を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

使用法

ソース・データとターゲット・データはオーバーラップしてはなりません。以下に、この変換の結果の例を示します。

- ソース・データ: 0x12345678
- ターゲット・データ: 0x78563412

cwbDT_EBCDICToEBCDIC:

System i Access for Windows の cwbDT_EBCDICToEBCDIC コマンドを使用します。

目的

EBCDIC データを EBCDIC に変換 (0x40 よりも小さい文字値の場合を除き、コピー) します。

ユニコード・バージョン

なし

構文

```
unsigned int CWB_ENTRY cwbDT_EBCDICToEBCDIC(
                            char         *target,
                            char         *source,
                            unsigned long length);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース・データを指すポインター。

unsigned long length - input
変換するターゲット・データのバイト数。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

呼び出し側は、ターゲット情報を保持するための十分なスペースを確保しておく必要があります。

cwbDT_HexToASCII:

System i Access for Windows の cwbDT_HexToASCII コマンドを使用します。

目的

2 進データを ASCII 16 進表示に変換します。ソース・データのバイトごとに、2 桁の ASCII 文字がターゲットに保管されます。

ユニコード・バージョン

cwbDT_HexToWide

構文

```
unsigned int CWB_ENTRY cwbDT_HexToASCII(
                            char         *target,
                            char         *source,
                            unsigned long length);
```

パラメーター

char * target - output
ターゲット (ASCII 16 進) データを指すポインター。

char * source - input
ソース・データを指すポインター。

unsigned long length - input
変換するソース・データのバイト数。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

使用法

ソース・データの 'length' バイトに対して、ターゲット・データの 'length'*2 バイトが保管されます。呼び出し側は、ターゲット情報を保持するための十分なスペースを確保しておく必要があります。

cwbDT_PackedToASCII:

System i Access for Windows の cwbDT_PackedToASCII コマンドを使用します。

目的

データをパック 10 進数形式から ASCII 数値データに変換します。この関数を使用すると、システムの System i データを変換して、ASCII テキスト形式で使用できるようになります。

ユニコード・バージョン

cwbDT_PackedToWide

構文

```
unsigned int CWB_ENTRY cwbDT_PackedToASCII(
                            char         *target,
                            char         *source,
                            unsigned long length,
                            unsigned long decimalPosition);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース・データを指すポインター。

unsigned long length - input
変換するソース・データのバイト数。

unsigned long decimalPosition - input
小数点の位置。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

呼び出し側は、ターゲット情報を保持するための十分なスペースを確保しておく必要があります。この関数は、パック 10 進数データの各ハーフバイトが 0 から 9 までの範囲内にあるかどうかを検査します。最後のハーフバイトは例外で、ここには符号標識が入ります。 0 <= 小数点位置 < (長さ * 2)。

cwbDT_PackedToASCIIPacked:

System i Access for Windows の cwbDT_PackedToASCIIPacked コマンドを使用します。

目的

データをパック 10 進数形式から ASCII パック形式に変換します。この関数を使用すると、システムの System i データを変換して、ASCII 形式で使用できるようになります。

ユニコード・バージョン

なし

構文

```
unsigned int CWB_ENTRY cwbDT_PackedToASCIIPacked(
                            char         *target,
                            char         *source,
                            unsigned long length);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース・データを指すポインター。

unsigned long length - input
変換するソース・データのバイト数。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

呼び出し側は、ターゲット情報を保持するための十分なスペースを確保しておく必要があります。この関数は、パック 10 進数データの各ハーフバイトが 0 から 9 までの範囲内にあるかどうかを検査します。最後のハーフバイトは例外で、ここには符号標識 (0 から 9、0xd または 0xb のいずれも可) が入ります。

cwbDT_PackedToPacked:

System i Access for Windows の cwbDT_PackedToPacked コマンドを使用します。

目的

パック 10 進データをパック 10 進数に変換します。この関数を使用すると、システムの System i データと非変換ファイルとを相互に変換できるようになります。

ユニコード・バージョン

なし

構文

```
unsigned int CWB_ENTRY cwbDT_PackedToPacked(
                            char         *target,
                            char         *source,
                            unsigned long length);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース・データを指すポインター。

unsigned long length - input
変換するソース・データのバイト数。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

呼び出し側は、ターゲット情報を保持するための十分なスペースを確保しておく必要があります。この関数は、パック 10 進数データの各ハーフバイトが 0 から 9 までの範囲内にあるかどうかを検査します。最後のハーフバイトは例外で、ここには符号標識が入ります。

cwbDT_ZonedToASCII:

System i Access for Windows の cwbDT_ZonedToASCII コマンドを使用します。

目的

EBCDIC ゾーン 10 進データを ASCII 数値形式に変換します。この関数を使用すると、システムの System i データを変換して、ASCII テキスト形式で使用できるようになります。

ユニコード・バージョン

cwbDT_ZonedToWide

構文

```
unsigned int CWB_ENTRY cwbDT_ZonedToASCII(
                            char         *target,
                            char         *source,
                            unsigned long length,
                            unsigned long decimalPosition);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース・データを指すポインター。

unsigned long length - input
変換するソース・データのバイト数。

unsigned long decimalPosition - input
小数点の位置。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

CWB_BUFFER_OVERFLOW
オーバーフロー・エラー。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

呼び出し側は、ターゲット情報を保持するための十分なスペースを確保しておく必要があります。ゾーン・データの最後のバイトの高位の半分はその数値の符号を表します。高位のハーフバイトが 0xb または 0xd の場合は負数を表します。それ以外の値の場合は正数を表します。ゾーン・データの各バイトの高位の半分は、最後のバイトを除き 0xf でなければなりません。この関数はそれを検査します。ゾーン・データの各バイトの低位の半分は 0 から 9 の範囲内でなければなりません。 0 <= 小数点位置 < 長さ。

cwbDT_ZonedToASCIIZoned:

System i Access for Windows の cwbDT_ZonedToASCIIZoned コマンドを使用します。

目的

データを、EBCDIC ゾーン 10 進形式から ASCII ゾーン 10 進数に変換します。この関数を使用すると、システムの System i データを変換して、ASCII ファイルで使用できるようになります。

ユニコード・バージョン

なし

構文

```
unsigned int CWB_ENTRY cwbDT_ZonedToASCIIZoned(
                            char         *target,
                            char         *source,
                            unsigned long length);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース・データを指すポインター。

unsigned long length - input
変換するソース・データのバイト数。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

呼び出し側は、ターゲット情報を保持するための十分なスペースを確保しておく必要があります。EBCDIC ゾーン 10 進データ中の左のハーフバイト (0xf) は、最後のバイト (符号) を除き ASCII ゾーン 10 進データの左のハーフバイト中の 0x3 に変換されます。EBCDIC ゾーン 10 進データの最後のバイトの高位の半分はその数値の符号を表します。高位のハーフバイトが 0xb または 0xd の場合は負数を表し、それ以外の値の場合は正数を表します。EBCDIC ゾーン 10 進データの各バイトの高位の半分は、最後のバイトを除き 0xf でなければなりません。この関数はそれを検査します。EBCDIC ゾーン 10 進データの各バイトの低位の半分は 0 から 9 の範囲内でなければなりません。

cwbDT_ZonedToZoned:

System i Access for Windows の cwbDT_ZonedToZoned コマンドを使用します。

目的

データを、ゾーン 10 進形式からゾーン 10 進数に変換します。この関数を使用すると、システムの System i データを変換して、非変換ファイルで使用することができ、その反対の変換を行うことも可能になります。

ユニコード・バージョン

なし

構文

```
unsigned int CWB_ENTRY cwbDT_ZonedToZoned(
                            char         *target,
                            char         *source,
                            unsigned long length);
```

パラメーター

char * target - output
ターゲット・データを指すポインター。

char * source - input
ソース・データを指すポインター。

unsigned long length - input
変換するソース・データのバイト数。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
呼び出し側から、NULL ポインターが渡されました。

other 最初の非変換文字に 1 を加えたオフセット。

使用法

呼び出し側は、ターゲット情報を保持するための十分なスペースを確保しておく必要があります。ゾーン・データの最後のバイトの高位の半分はその数値の符号を表します。高位のハーフバイトが 0xb または 0xd の場合は負数を表し、それ以外の値の場合は正数を表します。ゾーン・データの各バイトの高位の半分は、最後のバイトを除き 0xf でなければなりません。この関数はそれを検査します。ゾーン・データの各バイトの低位の半分は 0 から 9 の範囲内でなければなりません。

例: データ形式変更 API の使用法:

System i Access for Windows データ形式変更 API の使用法について、この例で説明します。

```
/*********************************************************************/
/* Sample Data Transform Program using cwbDT_Bin4ToBin4 to reverse */
/* the order of bytes in a 4-byte integer.                           */
/*********************************************************************/

#include <iostream.h>
#include "cwbdt.h"


void main()
{
   unsigned int returnCode;
   long source,
        target;

   cout << "Enter source number:¥n";

   while (cin >> source) {
     cout << "Source in Dec = " << dec << source;
     cout << "¥nSource in Hex = " << hex << source << '¥n';
     if (((returnCode = cwbDT_Bin4ToBin4((char *)&target,(char *)&source)) == CWB_OK)) {
        cout << "Target in Dec = " << dec << target;
        cout << "¥nTarget in Hex = " << hex << target << '¥n';
     } else {
        cout << "Conversion failed, Return code = " << returnCode << '¥n' ;
     }; /* endif */
     cout << "¥nEnter source number:¥n";


   }; /* endwhile */

}
```

System i Access for Windows 各国語サポート (NLS) API

各国語サポート API によって、各国の言語バージョンに関連した System i Access for Windows の設定値の取得および保存 (照会および変更) を、アプリケーションで行えるようになります。

System i Access for Windows 製品では、NLS を通じて各国語をサポートします。 NLS によって、ユーザーは、システム上で選択した言語で作業することができます。また、このサポートによって、システムとの間で送受信されるデータを、意図どおりの形式と順序で表示させることができます。数多くの異なる言語をサポートすることによって、言語的ならびに文化的観点の双方から、システムを意図したとおりに動作させます。

すべての System i 関数が、システム上でユーザーが使用する言語に関係なく、共通のプログラム・コードのセットを使用します。例えば、米国英語バージョンの System i プログラム・コードと、スペイン語バージョンの System i プログラム・コードは、同じものです。ただし、異なる言語においては、異なるセットのテキスト・データが使用されます。ここでいうテキスト・データとは、メニュー、画面、リスト、プロンプト、オプション、オンライン・ヘルプ情報、およびメッセージを一括して指す用語です。これは、次のことを意味します。すなわち、米国英語システムでは、オンライン・ヘルプ情報に対する機能キーの説明に *Help* が表示されますが、スペイン語システムでは *Ayuda* が表示されます。同じプログラム・コードを異なるテキスト・データ群と共に使用することによって、システムは、単一システム上で 複数の言語をサポートすることができます。

これらの API を使用すると、以下のような便利な機能を System i Access for Windows アプリケーションに追加することができます。

- インストール済みの各国語のリストから選択する。
- ある 1 つのコード・ページから別のコード・ページに文字データを変換する。これによって、パーソナル・コンピューターと System i オペレーティング・システムのように、異なるコード・ページを使用するコンピューターで情報を共用することが可能になります。
- ダイアログ・ボックス内の変換可能なテキスト (表題およびコントロール名) を自動的に置換する。これによって、コントロールのサイズが、それらに関連しているテキストに応じて拡張されます。ダイアログ・ボックス・フレームのサイズも自動的に調整されます。

注: プログラムを設計するに当たっては、その開始時点から各国語サポートに関する考慮事項を組み込んでおくことが必要です。プログラムを設計、あるいはコード化し終わってからでは、NLS または DBCS サポートを追加することは非常に困難です。

System i Access for Windows NLS API に必要なファイル

NLS API タイプ	ヘッダー・ファイル	インポート・ライブラリー	ダイナミック・リンク・ライブラリー
その他	cwbnl.h	cwbapi.lib	cwbnl.dll
変換	cwbnlcnv.h		cwbnl1.dll
ダイアログ・ボックス	cwbnldlg.h		cwbnldlg.dll

Programmer's Toolkit:

Programmer's Toolkit には、NLS 資料、NLS API ヘッダー・ファイルへのアクセス、およびプログラム例へのリンクが用意されています。この情報にアクセスするには、Programmer's Toolkit をオープンして、「**データ操作**」 → 「**C/C++ API**」と選択します。

関連資料

7 ページの『接続 API 用の System i 名の形式』
パラメーターとして System i 名を取得する API では、3 つの異なる形式の名前を使用できます。

8 ページの『OEM、ANSI、およびユニコードの考慮事項』
System i Access for Windows の、ストリング・パラメーターを受け入れる C/C++ API の大部分は、OEM、ANSI、Unicode の 3 つのうち、いずれかの形式になっています。

コード化文字セット:

System i Access for Windows 製品では、文字エンコード・スキームを使用します。

グラフィック文字とは、文字、数字や句読点の記号のような、印刷可能な記号または画面表示可能な記号のことです。グラフィック文字の集合はグラフィック文字セット と呼ばれ、多くの場合、省略して文字セット と呼ばれます。

各言語には、正しく印刷したり画面表示したりするための独自のグラフィック文字セットが必要です。文字は、コード・ページ に従ってエンコードされます。コード・ページとは、グラフィック文字および制御文字を、コード・ポイント と呼ばれる特定の値に割り当てるテーブルのことです。

コード・ページは、そのエンコード・スキームに従って多くのタイプに分類されます。System i Access ファミリーには、ホスト・コード・ページと PC コード・ページの、2 つの重要なエンコード・スキームがあります。ユニコードもまた、重要なエンコード・スキームになりつつあります。ユニコードは、ホストおよびパーソナル・コンピューターの両方において一般的になりつつある、16 ビットの世界的文字エンコード・スキームです。

- ホスト・コード・ページは、IBM 標準の拡張 2 進化 10 進コード (EBCDIC) に沿ってエンコードされ、通常 S/390® および System i プラットフォームで使用されます。
- PC コード・ページは ANSI X3.4、ASCII に基づいてエンコードされ、通常 IBM パーソナル・コンピューターで使用されます。

System i Access for Windows 汎用 NLS API のリスト:

System i Access for Windows の汎用 NLS API を使用します。

System i Access for Windows は、多くの言語に翻訳されています。 1 つまたは複数の言語をパーソナル・コンピューターにインストールすることができます。以下の System i Access for Windows 汎用 NLS API を使用すると、アプリケーションで次の作業ができるようになります。

- インストール済み言語のリストを取得する。
- 現行の言語設定値を取得する。
- 言語設定値を保管する。

cwbNL_FindFirstLang:

System i Access for Windows の cwbNL_FindFirstLang コマンドを使用します。

目的

使用可能な最初の言語を戻します。

構文

```
unsigned int CWB_ENTRY cwbNL_FindFirstLang(
                            char            *mriBasePath,
                            char            *resultPtr,
                            unsigned short   resultLen,
```

```
unsigned short  *requiredLen,
unsigned long   *searchHandle,
cwbSV_ErrHandle  errorHandle);
```

パラメーター

char * mriBasePath - input
mriBasePath を指すポインター (例えば C:¥Program Files¥IBM¥ClientAccess/400)。NULL の場合は、ClientAccess/400 プロダクトの mriBasePath が使用されます。

char * resultPtr - output
結果を入れるバッファーを指すポインター。

unsigned short resultLen - input
結果を入れるバッファーの長さ。推奨サイズは CWBNL_MAX_LANG_SIZE です。

unsigned short * requiredLen - output
結果の実際の長さ。requiredLen > resultLen の場合、戻り値は CWB_BUFFER_OVERFLOW になります。

unsigned long * searchHandle - output
後続の cwbNL_FindNextLang への呼び出しで渡される検索ハンドル。

cwbSV_ErrHandle errorHandle - input
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_FILE_NOT_FOUND
ファイルが見付かりませんでした。

CWB_PATH_NOT_FOUND
パスが見付かりませんでした。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

使用法

結果を入れるバッファーに言語が入ります。

cwbNL_FindNextLang:

System i Access for Windows の cwbNL_FindNextLang コマンドを使用します。

目的

使用可能な次の言語を戻します。

構文

```
unsigned int CWB_ENTRY cwbNL_FindNextLang(
                            char            *resultPtr,
                            unsigned short   resultLen,
                            unsigned short  *requiredLen,
                            unsigned long   *searchHandle,
                            cwbSV_ErrHandle  errorHandle);
```

パラメーター

char * resultPtr - output
結果を入れるバッファーを指すポインター。

unsigned short resultLen - input
結果を入れるバッファーの長さ。推奨サイズは CWBNL_MAX_LANG_SIZE です。

unsigned short * requiredLen - output
結果の実際の長さ。requiredLen > resultLen の場合、戻り値は CWB_BUFFER_OVERFLOW になります。

unsigned long * searchHandle - output
後続の cwbNL_FindNextLang への呼び出しで渡される検索ハンドル。

cwbSV_ErrHandle errorHandle - input
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NO_MORE_FILES
これ以上ファイルは見付かりません。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

使用法

結果を入れるバッファーに言語が入ります。

cwbNL_GetLang:

System i Access for Windows の cwbNL_GetLang コマンドを使用します。

目的

現行の言語設定値を取得します。

構文

```
unsigned int CWB_ENTRY cwbNL_GetLang(
                            char           *mriBasePath,
                            char           *resultPtr,
                            unsigned short  resultLen,
                            unsigned short *requiredLen,
                            cwbSV_ErrHandle errorHandle);
```

パラメーター

char * mriBasePath - input
mriBasePath を指すポインター (例えば C:¥Program Files¥IBM¥ClientAccess/400)。 NULL の場合は、ClientAccess/400 プロダクトの mriBasePath が使用されます。

char * resultPtr - output
結果を入れるバッファーを指すポインター。

unsigned short resultLen - input
結果を入れるバッファーの長さ。推奨サイズは CWBNL_MAX_LANG_SIZE です。

unsigned short * requiredLen - output
結果の実際の長さ。requiredLen > resultLen の場合、戻り値は CWB_BUFFER_OVERFLOW になります。

cwbSV_ErrHandle errorHandle - input
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎて結果を入れることができません。

使用法

結果を入れるバッファーには、言語サブディレクトリーの名前が入ります。この言語サブディレクトリーには言語特有のファイルが入っています。この言語サブディレクトリー名も、cwbNL_GetLangName に渡すことができます。

cwbNL_GetLangName:

System i Access for Windows の cwbNL_GetLangName コマンドを使用します。

目的

言語設定値の記述名を戻します。

構文

```
unsigned int CWB_ENTRY cwbNL_GetLangName(
                            char           *lang,
                            char           *resultPtr,
                            unsigned short  resultLen,
                            unsigned short *requiredLen,
                            cwbSV_ErrHandle errorHandle);
```

パラメーター

char * lang - input
言語を表す ASCIIZ ストリングのアドレス。

char * resultPtr - output
結果を入れるバッファーを指すポインター。

unsigned short resultLen - input
結果を入れるバッファーの長さ。推奨サイズは CWBNL_MAX_NAME_SIZE です。

unsigned short * requiredLen - output
結果の実際の長さ。requiredLen > resultLen の場合、戻り値は CWB_BUFFER_OVERFLOW になります。

cwbSV_ErrHandle errorHandle - input
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

使用法

言語は、次のいずれかの API から戻される値でなければなりません。

- cwbNL_GetLang
- cwbNL_FindFirstLang
- cwbNL_FindNextLang

cwbNL_GetLangPath:

System i Access for Windows の cwbNL_GetLangPath コマンドを使用します。

目的

言語ファイルについて、完全なパスを戻します。

構文

```
unsigned int CWB_ENTRY cwbNL_GetLangPath(
                            char            *mriBasePath,
                            char            *resultPtr,
                            unsigned short   resultLen,
                            unsigned short  *requiredLen,
                            cwbSV_ErrHandle  errorHandle);
```

パラメーター

char * mriBasePath - input
mriBasePath を指すポインター (例えば C:¥Program Files¥IBM¥ClientAccess/400)。 NULL の場合は、ClientAccess/400 プロダクトの mriBasePath が使用されます。

char * resultPtr - output
結果を入れるバッファーを指すポインター。

unsigned short resultLen - input
結果を入れるバッファーの長さ。推奨サイズは CWBNL_MAX_PATH_SIZE です。

unsigned short * requiredLen - output
結果の実際の長さ。requiredLen > resultLen の場合、戻り値は CWB_BUFFER_OVERFLOW になります。

cwbSV_ErrHandle errorHandle - input
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_PATH_NOT_FOUND
パスが見付かりませんでした。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

使用法

結果を入れるバッファーには言語サブディレクトリーの完全なパスが入ります。言語ファイルはこのパスからロードしてください。

cwbNL_SaveLang:

System i Access for Windows の cwbNL_SaveLang コマンドを使用します。

目的

言語設定値をプロダクト・レジストリーに保管します。

構文

```
unsigned int CWB_ENTRY cwbNL_SaveLang(
                            char            *lang,
                            cwbSV_ErrHandle  errorHandle);
```

パラメーター

char * lang - input
言語を表す ASCIIZ ストリングのアドレス。

cwbSV_ErrHandle errorHandle - input
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

使用法

言語は、次のいずれかの API から戻される値でなければなりません。

- cwbNL_GetLang
- cwbNL_FindFirstLang
- cwbNL_FindNextLang

以下の API は、この呼び出しによって影響を受けます。

- cwbNL_GetLang
- cwbNL_GetLangPath

System i Access for Windows の変換 NLS API リスト:

このトピックでは、System i Access for Windows の変換 NLS API について説明します。

以下の System i Access for Windows 変換 NLS API を使用すると、アプリケーションで次のことが行えるようになります。

- ある 1 つのコード・ページから別のコード・ページに文字データを変換する。
- 現行のコード・ページ設定値を入手する。
- 最新の CCSID 設定値を判別する。
- コード・ページ値とコード化文字セット識別コード (CCSID) との間の変換を行う。

cwbNL_CCSIDToCodePage:

System i Access for Windows の cwbNL_CCSIDToCodePage コマンドを使用します。

目的

CCSID をコード・ページにマップします。

構文

```
unsigned int CWB_ENTRY cwbNL_CCSIDToCodePage(
                            unsigned long   CCSID,
                            unsigned long  *codePage,
                            cwbSV_ErrHandle errorHandle);
```

パラメーター

unsigned long CCSID - input
コード・ページに変換する CCSID。

unsigned long * codePage - output
結果のコード・ページ。

cwbSV_ErrHandle errorHandle - output
エラー・オブジェクトのハンドル。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

使用法

なし (None)

cwbNL_CodePageToCCSID:

System i Access for Windows の cwbNL_CodePageToCCSID コマンドを使用します。

目的

コード・ページを CCSID にマップします。

構文

```
unsigned int CWB_ENTRY cwbNL_CodePageToCCSID(
                            unsigned long   codePage,
                            unsigned long  *CCSID,
                            cwbSV_ErrHandle errorHandle);
```

パラメーター

unsigned long codePage - input
CCSID に変換するコード・ページ。

unsigned long * CCSID - output
結果の CCSID。

cwbSV_ErrHandle errorHandle - output
エラー・オブジェクトのハンドル。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

使用法

なし (None)

cwbNL_Convert:

System i Access for Windows の cwbNL_Convert コマンドを使用します。

目的

前にオープンしたコンバーターを使用してストリングを変換します。

構文

```
unsigned int CWB_ENTRY cwbNL_Convert(
                            cwbNL_Converter theConverter,
                            unsigned long   sourceLength,
                            unsigned long   targetLength,
                            char           *sourceBuffer,
                            char           *targetBuffer,
                            unsigned long  *numberOfErrors,
                            unsigned long  *firstErrorIndex,
                            unsigned long  *requiredLen,
                            cwbSV_ErrHandle errorHandle);
```

パラメーター

cwbNL_Converter theConverter - output
前にオープンされたコンバーターへのハンドル。

unsigned long sourceLength - input
ソース・バッファーの長さ。

unsigned long targetLength - input
ターゲット・バッファーの長さ。DBCS 文字を含む ASCII コード・ページを変換する場合は、その結果のデータにはシフトアウト・バイトおよびシフトイン・バイトが含まれている可能性があることに留意してください。したがって、targetBuffer は sourceBuffer よりも大きくしておく必要があります。

char *sourceBuffer - input
変換すべきデータが入っているバッファー。

char *targetBuffer - output
変換されたデータが入るバッファー。

unsigned long *numberOfErrors - output
正しく変換できなかった文字の数が入ります。

unsigned long *firstErrorIndex - output
正しく変換できなかったソース・バッファー中の最初の文字のオフセットが入ります。

unsigned long *requiredLen - output
結果の実際の長さ。requiredLen > resultLen の場合、戻り値は CWB_BUFFER_OVERFLOW になります。

cwbSV_ErrHandle errorHandle - output
エラー・オブジェクトのハンドル。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

使用法

なし (None)

cwbNL_ConvertCodePages:

System i Access for Windows の cwbNL_ConvertCodePages コマンドを使用します。

コメント

cwbNL_ConvertCodePages はサポートされなくなりました。cwbNL_ConvertCodePagesEx を参照してください。

cwbNL_ConvertCodePagesEx:

System i Access for Windows の cwbNL_ConvertCodePagesEx コマンドを使用します。

目的

ストリングを 1 つのコード・ページから別のコード・ページに変換します。この API は、デフォルト変換のために次の 3 つのコンバーター API を結合します。

- cwbNL_CreateConverterEx
- cwbNL_Convert
- cwbNL_DeleteConverter

構文

```
unsigned int CWB_ENTRY cwbNL_ConvertCodePagesEx(
                            unsigned long   sourceCodePage,
                            unsigned long   targetCodePage,
```

```
unsigned long   sourceLength,
unsigned long   targetLength,
char           *sourceBuffer,
char           *targetBuffer,
unsigned long  *numberOfErrors,
unsigned long  *positionOfFirstError,
unsigned long  *requiredLen,
cwbSV_ErrHandle errorHandle);
```

パラメーター

unsigned long sourceCodePage - input
ソース・バッファー中のデータのコード・ページ。

unsigned long targetCodePage - input
データの変換先のコード・ページ。

unsigned long sourceLength - input.
ソース・バッファーの長さ。

unsigned long targetLength - input.
ターゲット・バッファーの長さ。

char *sourceBuffer - input
変換すべきデータが入っているバッファー。

char *targetBuffer - output
変換されたデータが入るバッファー。

unsigned long *numberOfErrors - output
正しく変換できなかった文字の数が入ります。

unsigned long *positionOfFirstError - output
正しく変換できなかったソース・バッファー中の最初の文字のオフセットが入ります。

unsigned long *requiredLen - output
結果の実際の長さ。requiredLen > resultLen の場合、戻り値は CWB_BUFFER_OVERFLOW になります。

cwbSV_ErrHandle errorHandle - output
エラー・オブジェクトのハンドル。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWBNL_ERR_CNV_UNSUPPORTED
文字の変換をしようとしたときにエラーが発生しました。変換は行われませんでした。最もよく見

られる理由は、変換テーブルが欠落しているということです。変換テーブルは、System i Access for Windows と一緒にインストールするか、または必要に応じてデフォルト・システムから取得します。デフォルト・システムとの通信に、何らかの障害が発生している可能性があります。

CWBNL_ERR_CNV_ERR_STATUS
この戻りコードは、要求した変換がサポートされている間とその変換が完了した時点で、一部の文字が正しく変換されなかったことを示す場合に使用されます。ソース・バッファーの中に NULL 文字が入れられたか、あるいは、ターゲット・コード・ページの中にこれらの文字が入っていないかのいずれかです。アプリケーションは、この戻りコードを無視するか、またはそれを警告として処理することができます。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

使用法

sourceCodePage および targetCodePage パラメーターで次の値を指定することができます。

値	意味
CWBNL_CP_UNICODE_F200	UCS2 バージョン 1.1 UNICODE
CWBNL_CP_UNICODE	UCS2 現行バージョン UNICODE
CWBNL_CP_AS400	i5/OS ホスト・コード・ページ
CWBNL_CP_CLIENT_OEM	OEM クライアント・コード・ページ
CWBNL_CP_CLIENT_ANSI	ANSI クライアント・コード・ページ
CWBNL_CP_CLIENT_UNICODE	UNICODE クライアント・コード・ページ
CWBNL_CP_UTF8	UCS 変換形式、8 ビット形式
CWBNL_CP_CLIENT	汎用クライアント・コード・ページ。デフォルトは CWBNL_CP_CLIENT_OEM。CWBNL_CP_CLIENT は、CWB_ANSI が定義されるときに CWBNL_CP_CLIENT_ANSI に設定され、CWB_UNICODE が定義されるときに CWBNL_CP_CLIENT_UNICODE に設定され、CWB_OEM が定義されるときに CWBNL_CP_CLIENT_OEM に設定されます。
CWBNL_CP_UTF16BE	UTF-16 (ビッグ・エンディアン)
CWBNL_CP_UTF16LE	UTF-16 (リトル・エンディアン)
CWBNL_CP_UTF16	CWBNL_CP_UTF16BE または CWBNL_CP_UTF16LE (プラットフォームによって異なる)
CWBNL_CP_UTF32BE	UTF-32 (ビッグ・エンディアン)
CWBNL_CP_UTF32LE	UTF-34 (リトル・エンディアン)
CWBNL_CP_UTF32	CWBNL_CP_UTF32BE または CWBNL_CP_UTF32LE (プラットフォームによって異なる)

cwbNL_CreateConverter:

System i Access for Windows の cwbNL_CreateConverter コマンドを使用します。

コメント

cwbNL_CreateConverter はサポートされなくなりました。cwbNL_CreateConverterEx を参照してください。

目的

後続の cwbNL_Convert() への呼び出しで使用される cwbNL_Converter を作成します。

構文

```
unsigned int CWB_ENTRY cwbNL_CreateConverter(
                            unsigned long    sourceCodePage,
                            unsigned long    targetCodePage,
                            cwbNL_Converter *theConverter,
                            cwbSV_ErrHandle  errorHandle,
                            unsigned long    shiftInShiftOutStatus,
                            unsigned long    padLength,
                            char            *pad);
```

パラメーター

unsigned long sourceCodePage - input
ソース・データのコード・ページ。

unsigned long targetCodePage - input
データの変換先のコード・ページ。

cwbNL_Converter * theConverter - output
新しく作成されたコンバーター。

cwbSV_ErrHandle errorHandle - output
エラー・オブジェクトのハンドル。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

unsigned long shiftInShiftOutStatus - input
シフトイン・バイトおよびシフトアウト・バイトが、入力データまたは出力データの一部かどうかを表示します。0 - 偽。シフトインおよびシフトアウト・バイトはデータ・ストリングの一部ではない。1 - 真。シフトインおよびシフトアウト・バイトはデータ・ストリングの一部である。

unsigned long padLength - input
埋め込み文字の長さ。0 - この変換要求には埋め込み文字はない。1 - 1 バイトの埋め込み文字。これは、ターゲット・コード・ページが SBCS コード・ページや DBCS コード・ページのいずれかの場合にのみ有効です。2 - 2 バイトの埋め込み文字。これは、コード・ページが 1 バイト・コード・ページでない場合にのみ有効です。

char * pad - input
埋め込みのための文字 (複数の場合もある)。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWBNL_ERR_CNV_UNSUPPORTED
文字の変換をしようとしたときにエラーが発生しました。変換は行われませんでした。最もよく見られる理由は、変換テーブルが欠落しているということです。変換テーブルは、System i Access

for Windows と一緒にインストールするか、または必要に応じてデフォルト・システムから取得します。デフォルト・システムとの通信に、何らかの障害が発生している可能性があります。

CWBNL_ERR_CNV_ERR_STATUS
この戻りコードは、要求した変換がサポートされている間とその変換が完了した時点で、一部の文字が正しく変換されなかったことを示す場合に使用されます。ソース・バッファーの中に NULL 文字が入れられたか、あるいは、ターゲット・コード・ページの中にこれらの文字が入っていないかのいずれかです。アプリケーションは、この戻りコードを無視するか、またはそれを警告として処理することができます。

CWBNL_ERR_CNV_INVALID_SISO_STATUS
SISO パラメーターが無効です。

CWBNL_ERR_CNV_INVALID_PAD_LENGTH
埋め込みの長さパラメーターが無効です。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

使用法

sourceCodePage および targetCodePage パラメーターで次の値を指定することができます。

値	意味
CWBNL_CP_UNICODE_F200	UCS2 バージョン 1.1 UNICODE
CWBNL_CP_UNICODE	UCS2 現行バージョン UNICODE
CWBNL_CP_AS400	AS/400 ホスト・コード・ページ
CWBNL_CP_CLIENT_OEM	OEM クライアント・コード・ページ
CWBNL_CP_CLIENT_ANSI	ANSI クライアント・コード・ページ
CWBNL_CP_CLIENT_UNICODE	UNICODE クライアント・コード・ページ
CWBNL_CP_UTF8	UCS 変換形式、8 ビット形式
CWBNL_CP_CLIENT	汎用クライアント・コード・ページ。デフォルトは CWBNL_CP_CLIENT_OEM。CWBNL_CP_CLIENT は、CWB_ANSI が定義されるときに CWBNL_CP_CLIENT_ANSI に設定され、CWB_UNICODE が定義されるときに CWBNL_CP_CLIENT_UNICODE に設定され、CWB_OEM が定義されるときに CWBNL_CP_CLIENT_OEM に設定されます。
CWBNL_CP_UTF16BE	UTF-16 (ビッグ・エンディアン)
CWBNL_CP_UTF16LE	UTF-16 (リトル・エンディアン)
CWBNL_CP_UTF16	CWBNL_CP_UTF16BE または CWBNL_CP_UTF16LE (プラットフォームによって異なる)
CWBNL_CP_UTF32BE	UTF-32 (ビッグ・エンディアン)
CWBNL_CP_UTF32LE	UTF-34 (リトル・エンディアン)
CWBNL_CP_UTF32	CWBNL_CP_UTF32BE または CWBNL_CP_UTF32LE (プラットフォームによって異なる)

同じコード・ページを使用して、次のように何度も cwbNL_ConvertCodePagesEx を呼び出す代わりに、

- cwbNL_ConvertCodePagesEx(850, 500, ...);
- cwbNL_ConvertCodePagesEx(850, 500, ...);
- cwbNL_ConvertCodePagesEx(850, 500, ...);

コンバーターを作成してそれを複数回使用するほうが、より効率的です。

- cwbNL_CreateConverter(850, 500, &conv, ...);

- cwbNL_Convert(conv, ...);
- cwbNL_Convert(conv, ...);
- cwbNL_Convert(conv, ...);
- cwbNL_DeleteConverter(conv, ...);

cwbNL_CreateConverterEx:

System i Access for Windows の cwbNL_CreateConverterEx コマンドを使用します。

目的

後続の cwbNL_Convert() への呼び出しで使用される cwbNL_Converter を作成します。

構文

```
unsigned int CWB_ENTRY cwbNL_CreateConverterEx(
                            unsigned long    sourceCodePage,
                            unsigned long    targetCodePage,
                            cwbNL_Converter *theConverter,
                            cwbSV_ErrHandle  errorHandle,
                            unsigned long    shiftInShiftOutStatus,
                            unsigned long    padLength,
                            char            *pad);
```

パラメーター

unsigned long sourceCodePage - input
ソース・データのコード・ページ。

unsigned long targetCodePage - input
データの変換先のコード・ページ。

cwbNL_Converter * theConverter - output
新しく作成されたコンバーター。

cwbSV_ErrHandle errorHandle - output
エラー・オブジェクトのハンドル。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

unsigned long shiftInShiftOutStatus - input
シフトイン・バイトおよびシフトアウト・バイトが、入力データまたは出力データの一部かどうかを表示します。0 - 偽。シフトインおよびシフトアウト・バイトはデータ・ストリングの一部ではない。1 - 真。シフトインおよびシフトアウト・バイトはデータ・ストリングの一部である。

unsigned long padLength - input
埋め込み文字の長さ。0 - この変換要求には埋め込み文字はない。1 - 1 バイトの埋め込み文字。これは、ターゲット・コード・ページが SBCS コード・ページや DBCS コード・ページのいずれかの場合にのみ有効です。2 - 2 バイトの埋め込み文字。これは、コード・ページが 1 バイト・コード・ページでない場合にのみ有効です。

char * pad - input
埋め込みのための文字 (複数の場合もある)。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWBNL_ERR_CNV_UNSUPPORTED
文字の変換をしようとしたときにエラーが発生しました。変換は行われませんでした。最もよく見られる理由は、変換テーブルが欠落しているということです。変換テーブルは、System i Access for Windows 製品と一緒にインストールするか、または必要に応じてデフォルト・システムから取得します。デフォルト・システムとの通信に、何らかの障害が発生している可能性があります。

CWBNL_ERR_CNV_ERR_STATUS
この戻りコードは、要求した変換がサポートされている間とその変換が完了した時点で、一部の文字が正しく変換されなかったことを示す場合に使用されます。ソース・バッファーの中に NULL 文字が入れられたか、あるいは、ターゲット・コード・ページの中にこれらの文字が入っていないかのいずれかです。アプリケーションは、この戻りコードを無視するか、またはそれを警告として処理することができます。

CWBNL_ERR_CNV_INVALID_SISO_STATUS
SISO パラメーターが無効です。

CWBNL_ERR_CNV_INVALID_PAD_LENGTH
埋め込みの長さパラメーターが無効です。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

使用法

sourceCodePage および targetCodePage パラメーターで次の値を指定することができます。

値	意味
CWBNL_CP_UNICODE_F200	UCS2 バージョン 1.1 UNICODE
CWBNL_CP_UNICODE	UCS2 現行バージョン UNICODE
CWBNL_CP_AS400	AS/400 ホスト・コード・ページ
CWBNL_CP_CLIENT_OEM	OEM クライアント・コード・ページ
CWBNL_CP_CLIENT_ANSI	ANSI クライアント・コード・ページ
CWBNL_CP_CLIENT_UNICODE	UNICODE クライアント・コード・ページ
CWBNL_CP_UTF8	UCS 変換形式、8 ビット形式
CWBNL_CP_CLIENT	汎用クライアント・コード・ページ。デフォルトは CWBNL_CP_CLIENT_OEM。CWBNL_CP_CLIENT は、CWB_ANSI が定義されるときに CWBNL_CP_CLIENT_ANSI に設定され、CWB_UNICODE が定義されるときに CWBNL_CP_CLIENT_UNICODE に設定され、CWB_OEM が定義されるときに CWBNL_CP_CLIENT_OEM に設定されます。
CWBNL_CP_UTF16BE	UTF-16 (ビッグ・エンディアン)
CWBNL_CP_UTF16LE	UTF-16 (リトル・エンディアン)

値	意味
CWBNL_CP_UTF16	CWBNL_CP_UTF16BE または CWBNL_CP_UTF16LE (プラットフォームによって異なる)
CWBNL_CP_UTF32BE	UTF-32 (ビッグ・エンディアン)
CWBNL_CP_UTF32LE	UTF-34 (リトル・エンディアン)
CWBNL_CP_UTF32	CWBNL_CP_UTF32BE または CWBNL_CP_UTF32LE (プラットフォームによって異なる)

同じコード・ページを使用して、次のように何度も cwbNL_ConvertCodePagesEx を呼び出す代わりに、

- cwbNL_ConvertCodePagesEx(850, 500, ...);
- cwbNL_ConvertCodePagesEx(850, 500, ...);
- cwbNL_ConvertCodePagesEx(850, 500, ...);

コンバーターを作成してそれを複数回使用するほうが、より効率的です。

- cwbNL_CreateConverterEx(850, 500, &conv, ...);
- cwbNL_Convert(conv, ...);
- cwbNL_Convert(conv, ...);
- cwbNL_Convert(conv, ...);
- cwbNL_DeleteConverter(conv, ...);

cwbNL_DeleteConverter:

System i Access for Windows の cwbNL_DeleteConverter コマンドを使用します。

目的

cwbNL_Converter を削除します。

構文

```
unsigned int CWB_ENTRY cwbNL_DeleteConverter(
                            cwbNL_Converter theConverter,
                            cwbSV_ErrHandle errorHandle);
```

パラメーター

cwbNL_Converter theConverter - input
前に作成したコンバーター。

cwbSV_ErrHandle errorHandle - output
エラー・オブジェクトのハンドル。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

なし (None)

cwbNL_GetCodePage:

System i Access for Windows の cwbNL_GetCodePage コマンドを使用します。

目的

クライアント・システムの現行コード・ページを取得します。

構文

```
unsigned int CWB_ENTRY cwbNL_GetCodePage(
                            unsigned long  *codePage,
                            cwbSV_ErrHandle errorHandle);
```

パラメーター

unsigned long * codePage - output
クライアント・システムの現行コード・ページ、または OEM コード・ページ文字変換オーバーライド値が「System i Access Familyのプロパティー」ダイアログの言語タブに指定されていれば、それを戻します。

cwbSV_ErrHandle errorHandle - output
エラー・オブジェクトのハンドル。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

使用法

なし (None)

cwbNL_GetANSICodePage:

System i Access for Windows の cwbNL_GetANSICodePage コマンドを使用します。

目的

クライアント・システムの現行 ANSI コード・ページを取得します。

構文

```
unsigned int CWB_ENTRY cwbNL_GetANSICodePage(
                            unsigned long  *codePage,
                            cwbSV_ErrHandle errorHandle);
```

パラメーター

unsigned long * codePage - output
クライアント・システムの現行 ANSI コード・ページ、または ANSI コード・ページ文字変換オーバーライド値が「System i Access Familyのプロパティー」ダイアログの言語タブに指定されていれば、それを戻します。

cwbSV_ErrHandle errorHandle - output
エラー・オブジェクトのハンドル。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

使用法

なし (None)

cwbNL_GetHostCCSID:

System i Access for Windows の cwbNL_GetHostCCSID コマンドを使用します。

目的

所定のホスト・システムまたは管理システムに関連する CCSID を戻します。また、System i Access Windows の「**プロパティー**」ダイアログの「**言語**」 タブに、EBCDIC コード・ページの文字変換オーバーライド値が指定されている場合は、その値も戻します。

構文

```
unsigned long CWB_ENTRY cwbNL_GetHostCCSID(
                              char * system,
                              unsigned long * CCSID );
```

パラメーター

char * system - input
ホスト・システムの名前。NULL の場合は、管理システムが使用されます。

unsigned * CCSID - output
結果を入れるバッファーの長さ。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWBNL_DEFAULT_HOST_CCSID_USED
ホスト CCSID 500 が戻されます。

使用法

この API は、関連する CCSID 値を検索するのに、ホスト・システムへの活動接続を行わず、またそれが必要でもありません。それはホスト・システムへの以前の正常な接続に依存します。ホスト・システムに対して正常な接続が前に行われていない場合は、API は、内部マッピング・テーブルを使用して最も適切な関連するホスト CCSID を判別します。

System i Access for Windows ダイアログ・ボックス NLS API のリスト:

System i Access for Windows ダイアログ・ボックス NLS API は、ダイアログ・ボックス内の変換可能テキストを操作するために使用されるインターフェースです。

以下の System i Access for Windows ダイアログ・ボックス NLS API を使用すると、アプリケーションで次の作業が行えるようになります。

- ダイアログ・ボックス内の変換可能テキストを置き換える
- テキストに従ってダイアログ・ボックス・コントロールを拡張する

使用上の注意

このモジュールは、次のような種類のダイアログ・ボックス・コントロールの場合にのみ動作します。

- 静的テキスト
- ボタン

- グループ・ボックス
- 編集ボックス
- チェック・ボックス
- ラジオ・ボタン

組み合わせボックスなどの複合コントロールの場合は、動作しません。

cwbNL_CalcControlGrowthXY:

System i Access for Windows の cwbNL_CalcControlGrowthXY コマンドを使用します。

目的

ダイアログ・ボックス中の個々のコントロールの拡大係数を計算するためのルーチン。

構文

```
unsigned int CWB_ENTRY cwbNL_CalcControlGrowthXY(
                       HWND windowHandle,
                       HDC  hDC,
                       float* growthFactorX,
                  float* growthFactorY);
```

パラメーター

HWND windowHandle - input
拡大係数を計算する対象のコントロールのウィンドウ・ハンドル。

HDC hDC - input
装置コンテキスト。コントロール内の変換済みストリングに必要な範囲を決めるのに GetTextExtentPoint32 によって使用されます。

float* growthFactorX - output
コントロールのストリングを入れるために必要な幅に対する +/- 拡大。

float* growthFactorY - output
コントロールのストリングを入れるために必要な高さに対する +/- 拡大。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

使用法

この関数を呼び出す前に、変換済みテキストがコントロール内にロードされているものと想定します。テキストを含まないコントロールは、1.00 の拡大係数を戻します。これは、サイズを変更する必要がないことを意味します。

cwbNL_CalcDialogGrowthXY:

System i Access for Windows の cwbNL_CalcDialogGrowthXY コマンドを使用します。

目的

ダイアログ・ボックスの拡大係数を計算するためのルーチン。ダイアログ・ボックスのサイズをどれだけ調整する必要があるかを判別するために、ダイアログ・ボックス内のすべてのコントロールが調査されます。

構文

```
unsigned int CWB_ENTRY cwbNL_CalcDialogGrowthXY(
                       HWND windowHandle,
                       float* growthFactorX,
                  float* growthFactorY);
```

パラメーター

HWND windowHandle - input
拡大係数を計算する対象のダイアログ・ボックスのウィンドウ・ハンドル。

float* growthFactorX - output
ダイアログ・ボックス内のすべてのコントロール用のストリングを入れるために必要な幅に対する +/- 拡大。

float* growthFactorY - output
ダイアログ・ボックス内のすべてのコントロール用のストリングを入れるために必要な高さに対する +/- 拡大。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

使用法

この関数を呼び出す前に、変換済みテキストがコントロール内にロードされているものと想定します。

cwbNL_GrowControlXY:

System i Access for Windows の cwbNL_GrowControlXY コマンドを使用します。

目的

ダイアログ・ボックス内の個々のコントロールを拡大するためのルーチン。

構文

```
unsigned int CWB_ENTRY cwbNL_GrowControlXY(
                            HWND        windowHandle,
                            HWND        parentWindowHandle,
                            float       growthFactorX,
                            float       growthFactorY,
                            cwb_Boolean growAllControls);
```

パラメーター

HWND windowHandle - input
サイズ変更されるコントロールのウィンドウ・ハンドル。

HWND parentWindowHandle - input
コントロールを含むダイアログ・ボックスのウィンドウ・ハンドル。

float growthFactorX - input
コントロールの幅の拡大に使用される乗算係数。1.00 = 同じサイズのまま。1.50 = 元のサイズの 1.5 倍。

float growthFactorY - input
コントロールの高さの拡大に使用される乗算係数。1.00 = 同じサイズのまま。1.50 = 元のサイズの 1.5 倍。

cwb_Boolean growAllControls - input
CWB_TRUE = すべてのコントロールは growthFactor によってサイズ変更される。CWB_FALSE = テキストを持つコントロールのみがサイズ変更される。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

使用法

コントロールが実際の表示装置に合わない拡大係数を渡さないよう注意が必要です。

cwbNL_GrowDialogXY:

System i Access for Windows の cwbNL_GrowDialogXY コマンドを使用します。

目的

ダイアログ・ボックスおよびそのコントロールを、入力される拡大係数に比例して拡大する内部ルーチン。

構文

```
unsigned int CWB_ENTRY cwbNL_GrowDialogXY(
                            HWND        windowHandle,
                            float       growthFactorX,
                            float       growthFactorY,
                            cwb_Boolean growAllControls);
```

パラメーター

HWND windowHandle - input
コントロールを所有するウィンドウのウィンドウ・ハンドル。

float growthFactorX - input
ダイアログ・ボックスを拡大する乗算係数。1.00 = 同じサイズのまま。1.50 = 元のサイズの 1.5 倍。

float growthFactorY - input
ダイアログ・ボックスを拡大する乗算係数。1.00 = 同じサイズのまま。1.50 = 元のサイズの 1.5 倍。

cwb_Boolean growAllControls - input
CWB_TRUE = すべてのコントロールは growthFactor によってサイズ変更される。CWB_FALSE = テキストをもつコントロールのみがサイズ変更される。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

使用法

この関数を呼び出す前に、変換済みテキストがコントロール内にロードされているものと想定します。ダイアログ・ボックス・フレームは、デスクトップ・ウィンドウ・サイズよりも大きく拡大することはできません。

cwbNL_LoadDialogStrings:

System i Access for Windows の cwbNL_LoadDialogStrings コマンドを使用します。

目的

このルーチンは、ダイアログ・ボックス内の変換可能テキストの置き換えを制御します。これには、ダイアログ・ボックスの表題だけでなくダイアログ・コントロールのテキストも含まれます。

構文

```
unsigned int CWB_ENTRY cwbNL_LoadDialogStrings(
                              HINSTANCE   MRIHandle,
                              HWND        windowHandle,
                        int       nCaptionID,
                              USHORT      menuID,
                              HINSTANCE   menuLibHandle,
                              cwb_Boolean growAllControls);
```

パラメーター

HINSTANCE MRIHandle - input
ダイアログ用のストリングが入っているモジュールのハンドル。

HWND windowHandle - input
ダイアログ・ボックスのウィンドウ・ハンドル。

int nCaptionID - input
ダイアログ・ボックス用の表題ストリングの ID。

USHORT menuID - input
ダイアログ・ボックス用メニューの ID。

HINSTANCE menuLibHandle - input
ダイアログ・メニューが入っているモジュールのハンドル。

cwb_Boolean growAllControls - input
CWB_TRUE = すべてのコントロールは growthFactor によってサイズ変更される。CWB_FALSE = テキストをもつコントロールのみがサイズ変更される。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBNL_DLG_MENU_LOAD_ERROR
メニューをロードすることができません。

CWBNL_DLG_INVALID_HANDLE
MRIHandle が誤り。

使用法

このプロセスは、ダイアログ・ボックス内のすべてのダイアログ・コントロールを列挙し、そのテキストを置換し、横方向に調整することで始まり、最後にそこで調整されたコントロールを基準にして、ダイアログ・ボックス自体を右寄せします。これらの調整は、現行ウィンドウの範囲が、テキストまたはすべてのコントロールに必要な拡張スペースを十分に確保していない場合にのみ行われます。すべてのテキスト置換が完了したあと、メニュー ID が渡されている場合は、その ID がロードされ、ダイアログ・ボックスに付加されます。それぞれのダイアログ・ボックス・プロシージャーごとに、このルーチンを INITDLG メッセージ処理中に行われる最初のものとして呼び出すことをお勧めします。

cwbNL_LoadMenu:

System i Access for Windows の cwbNL_LoadMenu コマンドを使用します。

目的

このルーチンは、モジュールからの所定のメニューのロードと、メニュー内の変換可能テキストの置き換えを制御します。

構文

```
HWND CWB_ENTRY cwbNL_LoadMenu(
                    HWND      windowHandle,
                    HINSTANCE menuResourceHandle,
                    USHORT    menuID,
                    HINSTANCE MRIHandle);
```

パラメーター

HWND windowHandle - input
メニューが入っているダイアログ・ボックスのウィンドウ・ハンドル。

HINSTANCE menuResourceHandle - input
メニューが入っている資源 DLL のハンドル。

USHORT menuID - input
ダイアログ・ボックス用メニューの ID。

HINSTANCE MRIHandle - input
メニュー用ストリングが入っている資源 DLL のハンドル。

戻りコード

以下は、共通の戻り値です。

HINSTANCE
メニューのハンドル。

使用法

なし (None)

cwbNL_LoadMenuStrings:

System i Access for Windows の cwbNL_LoadMenuStrings コマンドを使用します。

目的

このルーチンは、メニュー内の変換可能テキストの置き換えを制御します。

構文

```
unsigned int CWB_ENTRY cwbNL_LoadMenuStrings(
                            HWND      WindowHandle,
                            HINSTANCE menuHandle,
                            HINSTANCE MRIHandle);
```

パラメーター

HWND windowHandle - input
メニューが入っているダイアログ・ボックスのウィンドウ・ハンドル。

HMODULE menuHandle - input
ダイアログ用メニューのハンドル。

HMODULE MRIHandle - input
メニュー用ストリングが入っている資源 DLL のハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

使用法

なし (None)

cwbNL_SizeDialog:

System i Access for Windows の cwbNL_SizeDialog コマンドを使用します。

目的

このルーチンは、ダイアログ・ボックスとその子コントロールのサイズを制御します。拡張量は、テキストの範囲の長さと各コントロールの長さに基づきます。ダイアログ・ボックスとそのコントロールの拡張と縮小は比例します。growAllControls を FALSE に設定すると、テキストを使用するコントロールだけが拡張または縮小されます。これにより、プログラマーに対して、変換不可フィールドを元のサイズのまま残しておくことができるという柔軟性が与えられます。このことは、ドロップダウン・リスト、組み合わせボックス、またはスピン・ボタンを使用するダイアログに適していると考えられます。

構文

```
unsigned int CWB_ENTRY cwbNL_SizeDialog(
                            HWND        windowHandle,
                            cwb_Boolean growAllControls);
```

パラメーター

HWND windowHandle - input
コントロールを所有するウィンドウのウィンドウ・ハンドル。

cwb_Boolean growAllControls - input
CWB_TRUE = すべてのコントロールは growthFactor によってサイズ変更される。CWB_FALSE = テキストをもつコントロールのみがサイズ変更される。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

使用法

このルーチンは、変換済みテキストが既にダイアログ・ボックス・コントロールにロードされていることを想定しています。テキストがコントロール内にロードされていない場合は、cwbNL_LoadDialog を使用してください。

例: System i Access for Windows NLS API:

System i Access for Windows NLS API の使用法について、次の例で説明します。

```
/* National Language Support Code Snippet                       */
/* Used to demonstrate how the APIs would be run.               */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "CWBNL.H"
#include "CWBNLCNV.H"
#include "CWBSV.H"

cwbSV_ErrHandle errhandle;

/*  Return the message text associated with the top-level       */
/*  error identified by the error handle provided.   Since      */
/*  all APIs that fail use the error handle, this was moved     */
/*  into a separate routine.                                    */
void resolveErr(cwbSV_ErrHandle errhandle)
{
    static unsigned char buf[  BUFSIZ  ];
    unsigned long retlen;
    unsigned int rc;

    if ((rc = cwbSV_GetErrText(errhandle, buf, (unsigned long) BUFSIZ, &retlen)) != CWB_OK)
        printf("cwbSV_GetErrText() Service API failed with return code 0x%x.¥n", rc);
    else
        printf("%s¥n", (char *) buf);
}

void main(void){
```

```
/* define some variables
    ------------------- */
int SVrc = 0;
int NLrc = 0;
char *myloadpath = "";
char *resultPtr;
char *mylang;
unsigned short resultlen;
unsigned short reqlen;
unsigned long searchhandle;
unsigned long codepage;
unsigned long trgtpage;
char *srcbuf = "Change this string";
char *trgtbuf;
unsigned long srclen;
unsigned long trgtlen;
unsigned long nmbrerrs;
unsigned long posoferr;
unsigned long rqdlen;
unsigned long ccsid;


/*  Create an error message object and return a handle to   */
/*  it.  This error handle can be passed to APIs that       */
/*  support it.  If an error occurs, the error handle can   */
/*  be used to retrieve the message text associated with    */
/*  the API error.                                          */
SVrc = cwbSV_CreateErrHandle(&errhandle);
if (SVrc != CWB_OK) {
    printf("cwbSV_CreateErrHandle failed with return code %d.¥n", SVrc);
}

/* Retreive the current language setting.                   */
resultlen = CWBNL_MAX_LANG_SIZE+1;
resultPtr = (char *) malloc(resultlen * sizeof(char));
NLrc = cwbNL_GetLang(myloadpath, resultPtr, resultlen, &reqlen, errhandle);
if (NLrc != CWB_NO_ERR) {
    if (NLrc == CWB_BUFFER_TOO_SMALL)
        printf("GetLang buffer too small, recommended size %d.¥n", reqlen);
    resolveErr(errhandle);
}
printf("GetLang API returned %s.¥n", resultPtr);
mylang = (char *) malloc(resultlen * sizeof(char));
strcpy(mylang, resultPtr);

/* Retrieve the descriptive name of a language setting.     */
resultlen = CWBNL_MAX_NAME_SIZE+1;
resultPtr = (char *) realloc(resultPtr, resultlen * sizeof(char));
NLrc = cwbNL_GetLangName(mylang, resultPtr, resultlen, &reqlen, errhandle);
if (NLrc != CWB_NO_ERR) {
    if (NLrc == CWB_BUFFER_TOO_SMALL)
        printf("GetLangName buffer too small, recommended size %d.¥n", reqlen);
    resolveErr(errhandle);
}
printf("GetLangName API returned %s.¥n", resultPtr);

/*  Return the complete path for language files.            */
resultlen = CWBNL_MAX_PATH_SIZE+1;
resultPtr = (char *) realloc(resultPtr, resultlen * sizeof(char));
NLrc = cwbNL_GetLangPath(myloadpath, resultPtr, resultlen, &reqlen, errhandle);
if (NLrc != CWB_NO_ERR) {
    if (NLrc == CWB_BUFFER_TOO_SMALL)
        printf("GetLangPath buffer too small, recommended size %d.¥n", reqlen);
    resolveErr(errhandle);
}
printf("GetLangPath API returned %s.¥n", resultPtr);
```

```
    /*  Get the code page of the current process.               */
    NLrc = cwbNL_GetCodePage(&codepage, errhandle);
    if (NLrc != CWB_NO_ERR) {
        resolveErr(errhandle);
    }
    printf("GetCodePage API returned %u.¥n", codepage);

    /*  Convert strings from one code page to another. This     */
    /*  API combines three converter APIs for the default       */
    /*  conversion.  The three converter APIs it combines are:  */
    /*      cwbNL_CreateConverterEx                               */
    /*      cwbNL_Convert                                       */
    /*      cwbNL_DeleteConverter                               */
    srclen = strlen(srcbuf) + 1;
    trgtlen = srclen;
    trgtpage = 437;
    trgtbuf = (char *) malloc(trgtlen * sizeof(char));
    printf("String to convert is %s.¥n",srcbuf);
    NLrc = cwbNL_ConvertCodePagesEx(codepage, trgtpage, srclen,
            trgtlen, srcbuf, trgtbuf, &nmbrerrs, &posoferr, &rqdlen,
            errhandle);
    if (NLrc != CWB_NO_ERR) {
        resolveErr(errhandle);
        printf("number of errors detected is %u.¥n", nmbrerrs);
        printf("location of first error is %u.¥n", posoferr);
    }
    printf("ConvertCodePagesEx API returned %s.¥n", trgtbuf);

    /*  Map a code page to the corresponding CCSID.             */
    NLrc = cwbNL_CodePageToCCSID(codepage, &ccsid, errhandle);
    if (NLrc != CWB_NO_ERR) {
        resolveErr(errhandle);
    }
    printf("CodePageToCCSID returned %u.¥n", ccsid);

    cwbSV_DeleteErrHandle(errhandle);

}
```

System i Access for Windows ディレクトリー更新 API

System i Access for Windows ディレクトリー更新機能を使用する PC ディレクトリー更新を指定します。

System i Access for Windows ディレクトリー更新 C/C++ API

System i Access for Windows のディレクトリー更新 C/C++ アプリケーション・プログラミング・インターフェース (API) を使用することにより、ソフトウェア開発者は、System i Access for Windows のディレクトリー更新機能で使用する更新項目の追加、変更、および削除を行うことができます。

注: System i Access for Windows ディレクトリー更新 API では、実際の更新は行いません。これらの API は、構成の目的でだけ使用されます。ファイルを更新するというタスクは、すべてディレクトリー更新アプリケーションのみで扱われます。

System i Access for Windows のディレクトリー更新 API を使用すると、次の作業が可能になります。

- 更新項目の作成。
- 更新項目の削除。
- 更新項目の変更。
- 更新項目からの情報の検索。

- 戻りコードなどの情報の検索。例えば、一度に 1 つのアプリケーションしか更新項目にアクセスすることができません。**ロック状態**を示す戻りコードを受け取った場合には、この情報によって項目がオープンになっているアプリケーション名を見付けることができます。

重要: System i Access for Windows クライアントには、ネットワーク・ドライブや汎用命名規則に関するサポートは組み込まれていません。現在、これは、**System i NetServer™** 機能によって提供されています。 System i Access でマップしたネットワーク・ドライブは、System i NetServer サポートを使用してマップする必要があります。 System i ファイル・サービス提供を実行するには、i5/OS に付属の System i NetServer をセットアップしてください。

NetServer 情報の資源

- i5/OS Information Center にある System i NetServer のトピック
- IBM System i NetServer ホーム・ページ (英語)

System i Access for Windows ディレクトリー更新 API に必要なファイル

ヘッダー・ファイル	インポート・ライブラリー	ダイナミック・リンク・ライブラリー
cwbup.h	cwbapi.lib	cwbup.dll

Programmer's Toolkit:

Programmer's Toolkit には、ディレクトリー更新の資料、cwbup.h ヘッダー・ファイルへのアクセス、およびプログラム例へのリンクが用意されています。この情報にアクセスするには、Programmer's Toolkit をオープンして、「**ディレクトリー更新**」 → 「**C/C++ API**」と選択します。

注: コード例を使用することで、 633 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』の条件に同意します。

関連資料

29 ページの『ディレクトリー更新 API の戻りコード』
System i Access for Windows のディレクトリー更新 API の戻りコードを示します。

7 ページの『接続 API 用の System i 名の形式』
パラメーターとして System i 名を取得する API では、3 つの異なる形式の名前を使用できます。

8 ページの『OEM、ANSI、およびユニコードの考慮事項』
System i Access for Windows の、ストリング・パラメーターを受け入れる C/C++ API の大部分は、OEM、ANSI、Unicode の 3 つのうち、いずれかの形式になっています。

System i Access for Windows ディレクトリー更新 API の一般的な使用法

System i Access for Windows のディレクトリー更新 API は、マップされたネットワーク・ドライブからファイルを更新する際に使用される更新項目について、作成および構成を行うために使用されるのが一般的です。更新 API が実際にファイルを更新するのではなく、ディレクトリー更新の実行可能ファイルがこれを行います。

例えば、System i ファイルに顧客の名前と住所が収められているとします。この System i ファイルは、顧客の新規追加や削除、または顧客の名前や住所の変更が行われた時点で更新される、マスター・ファイルになります。ネットワーク化されたパーソナル・コンピューター上にも同じファイルがあり、(郵便番号、県、年齢、子供の数などによって) 選別してダイレクト・メールを送るために使用されます。この System i ファイルは、ユーザーのマスター・ファイルであるため、これを保護することも必要ですが、データを業務用に提供する必要も生じます。

ディレクトリー更新 API を使用するプログラムを作成し、更新項目を作成および構成することができます。これによって、ネットワーク化されたパーソナル・コンピューター上にあるファイルが更新されます。

ディレクトリー更新項目の必須情報

System i Access for Windows ディレクトリー更新項目には、以下の項目が必須となります。

記述　ディレクトリー更新アプリケーションが、更新される内容をユーザーに示すために表示する記述。

ソース・パス
ソースまたは「マスター」ファイルのパス。例えば、次のようにします。

```
E:¥MYSOURCE
```

または、

```
¥¥myserver¥mysource
```

ターゲット・パス
マスター・ファイルとの同期を保つ必要のあるファイルのパス。例えば、次のようにします。

```
C:¥mytarget
```

ディレクトリー更新項目のオプション

次に挙げるものは、System i Access for Windows ディレクトリー更新項目のオプションです。

パッケージ・ファイル
更新される他のファイルに関する情報が収められている PC ファイル。詳しくは、238 ページの『ディレクトリー更新パッケージ・ファイルの構文と形式』を参照してください。パッケージ・ファイルは、項目を更新するために、245 ページの『cwbUP_AddPackageFile』API を使用して追加します。

コールバック DLL
アプリケーション・プログラマーによって提供される DLL。ディレクトリー更新では、更新プロセスのさまざまな段階で呼び込みます。これによって、プログラマーは、更新のさまざまな段階でアプリケーション固有の処理を実行させることができます。コールバック DLL は、247 ページの『cwbUP_SetCallbackDLL』API を使用して更新項目に追加されます。

更新時に、ディレクトリー更新がコールバック DLL を呼び込む可能性のあるさまざまな段階を次に示します。

事前更新
これは、ディレクトリー更新が更新項目の処理を開始しようとする段階です。**unsigned long _declspec(dllexport) cwbUP_PreUpdateCallback();** というエントリー・ポイント・プロトタイプをコールバック DLL 内に指定する必要があります。

事後更新
これは、ディレクトリー更新がファイルの移動を完了した段階です。**unsigned long _declspec(dllexport) cwbUP_PostUpdateCallback();** というエントリー・ポイント・プロトタイプをコールバック DLL 内に指定する必要があります。

事前移行
これは、ディレクトリー更新が更新項目のバージョンからバージョンへの移行を開始しようとする段階です。バージョンからバージョンへの移行は、QPTFIDX ファイルによって起動されます。**unsigned long _declspec(dllexport) cwbUP_PreMigrationCallback();** というエントリー・ポイント・プロトタイプをコールバック DLL 内に指定する必要があります。

事後移行

これは、ディレクトリー更新が更新項目のバージョンからバージョンへの移行の処理を完了した段階です。**unsigned long _declspec(dllexport) cwbUP_PostMigrationCallback();** というエントリー・ポイント・プロトタイプをコールバック DLL 内に指定する必要があります。

属性 実行しようとしている更新のタイプまたはモードを設定します。属性は、組み合わせることが可能です。属性には、次のものがあります。

ファイル主導型更新

ターゲット・ディレクトリー内のファイルは、ソース・ディレクトリー内のファイルと比較されます。ソース・ファイルの日付よりも古い日付のターゲット・ファイルが更新されます。ターゲット内では、新規ファイルは作成されません。

パッケージ主導型更新

更新項目にリストされているパッケージ・ファイルが、更新されるファイルに関してスキャンされます。パッケージ・ファイルにリストされているファイルの日付は、ソース・ディレクトリーとターゲット・ディレクトリーとの間で比較されます。ターゲット・ディレクトリーよりも日付が新しいソース・ファイルが更新されるか、あるいは、ターゲット・ディレクトリーの中に「移され」ます。パッケージ・ファイル内にリストされているファイルがターゲット内には存在せず、ソース内には存在している場合、そのファイルは、ターゲット・ディレクトリー内に作成されます。

サブディレクトリー更新

ターゲット・ディレクトリーに属するサブディレクトリーは、更新に組み込まれます。

1 段階更新

更新は、ソースからターゲットへ直接に行われます。これを指定しなかった場合、更新は 2 つの段階で行われます。更新の最初の段階は、更新しようとしているファイルを一時ディレクトリーの中にコピーします。その後で PC が再始動されます。再始動時に、これらのファイルがターゲット・ディレクトリーにコピーされます。これは、ロック・ファイルの場合に役に立ちます。

バックレベル更新

ソース・ファイルがターゲット・ファイルよりも古い場合に、更新を行うかどうかを制御します。

ディレクトリー更新パッケージ・ファイルの構文と形式

System i Access for Windows 製品で使用される**パッケージ・ファイル**には、ソース・ファイルについてユーザーが現行ファイルにしておきたいターゲット・ファイルの指定および記述を行う情報が収められています。

パッケージ・ファイルの構文

```
PKGF 記述 テキスト
MBRF PROG1.EXE
MBRF INFO.TXT
MBRF SUBDIR¥SHEET.XLS
DLTF PROG2.EXE
```

注: テキストは、ファイルの 1 行目の 1 列目から開始する必要があります。それぞれのパッケージ・ファイルの先頭には、PKGF キーワードを指定してください。

パッケージ・ファイルの形式

パッケージ・ファイルは次の要素で構成されています。

PKGF 記述 (オプション)
この識別コードでは、指定ファイルがパッケージ・ファイルであることを示します。そのファイルの先頭の 4 文字にこのタグがない場合、ディレクトリー更新は、更新するファイルを検索している間にそのファイルを処理することはありません。記述は任意指定です。

MBRF ファイル名
これは、更新しようとしているパッケージの一部としてファイルを識別します。パス名も指定することができます。これによって、このファイルがソース・ディレクトリーのサブディレクトリー内に収められていることを示します。

このパスには、ドライブ文字を使用することはできません。また、このパスの先頭に円記号 (¥) を使用することもできません。更新機能を開始する場合は、ターゲット・ディレクトリーを指定します。パッケージ・ファイルに指定されているパスは、このターゲット・ディレクトリーのサブディレクトリーであると見なされます。

DLTF ファイル名
これは、ターゲット・ディレクトリーから削除するファイルを識別します。パス名も指定することができます。これによって、このファイルがターゲット・ディレクトリーのサブディレクトリー内に収められていることを示します。MBRF 識別コードの場合と同様に、このパスには、ドライブ文字を使用することも、このパスの先頭に円記号 (¥) を使用することもできません。

関連トピック
ディレクトリー更新 API の例、ならびに、それらの API の属性に関する詳細については、『ディレクトリー更新サンプル・プログラム』を参照してください。

ディレクトリー更新サンプル・プログラム

ディレクトリー更新用の C/C++ サンプル・プログラムを入手するには、System i Access for Windows Programmer's Toolkit - ディレクトリー更新についての Web ページにアクセスしてください。

注: コード例を使用することで、 633 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』の条件に同意します。

Programmer's Toolkit - ディレクトリー更新の Web ページ (英語) にアクセスしてください。 **dirupdat.exe** を選択して、サンプルの記述を入手し、サンプル・プログラムをダウンロードしてください。

サンプル・プログラムでは、ディレクトリー更新項目の作成、構成、および削除を参照することができます。

詳しくは、System i Access for Windows の「ユーザーズ・ガイド」を参照してください。

ディレクトリー更新: 作成および削除の API

以下の System i Access for Windows ディレクトリー更新を使用して、更新項目の作成および削除を行います。API はアルファベット順にリストされます。

注: 基本的には、アプリケーションで更新項目をこれ以上アクセスすることがなくなった時点で、を呼び出します。cwbUP_FreeLock が呼び出されないと、他のアプリケーションは、それらの更新項目にアクセスすることも変更することもできません。

cwbUP_CreateUpdateEntry:

System i Access for Windows の cwbUP_CreateUpdateEntry コマンドを使用します。

目的

新規の更新項目を作成し、ハンドルをその項目に返します。

構文

```
unsigned int CWB_ENTRY cwbUP_CreateUpdateEntry(
                             char * entryDescription,
                             char * entrySource,
                             char * entryTarget,
                             cwbUP_EntryHandle *entryHandle);
```

パラメーター

char * entryDescription - input
更新項目を識別するための記述が含まれている、NULL 文字で終わるストリングを指します。

char * entrySource - input
更新項目のソースが含まれている、NULL 文字で終わるストリングを指します。これは、ドライブおよびパス、または UNC 名とすることが可能です。

char * entryTarget - input
更新項目のターゲットが含まれている、NULL 文字で終わるストリングを指します。これは、ドライブおよびパス、または UNC 名とすることが可能です。

cwbUP_EntryHandle * entryHandle - input/output
ハンドルが戻される先である cwbUP_EntryHandle を指すポインター。このハンドルは、以降の更新項目 API 呼び出しで使用する必要があります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL がアドレスとして渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリーが十分にないため、ハンドルを作成できません。

CWBUP_TOO_MANY_ENTRIES
既に存在する更新項目の最大数。これ以上の項目は作成できません。

CWBUP_STRING_TOO_LONG
入力ストリングが、最大長 CWBUP_MAX_LENGTH よりも長くなっています。

CWBUP_ENTRY_IS_LOCKED
現在、別のアプリケーションが更新項目リストを変更中です。現時点では変更できません。

使用法

この呼び出しを使用し、更新項目の処理を完了した後で、cwbUP_FreeEntryHandle を呼び出す必要があります。この呼び出しは項目をアンロックし、その項目に関連する資源を解放します。

cwbUP_DeleteEntry:

System i Access for Windows の cwbUP_DeleteEntry コマンドを使用します。

目的

更新項目を更新項目リストから削除します。

構文

```
unsigned int CWB_ENTRY cwbUP_DeleteEntry(
                            cwbUP_EntryHandle entryHandle);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWBUP_ENTRY_IS_LOCKED
現在、別のアプリケーションが更新項目リストを変更中です。現時点では変更できません。

使用法

この呼び出しの後は、cwbUP_FreeEntryHandle を呼び出す必要はありません。項目が正常に削除された後、その項目は解放されます。cwbUP_GetEntryHandle API を使用して最初の更新項目を検索してから、この API を呼び出してその項目を削除した場合、更新項目は、すべて、その削除によって残されたスロットを充てんするために、1 桁分のみ桁移動されます。したがって、次の更新項目を取得するには、先行の cwbUP_GetEntryHandle API 呼び出しで行った場合と同じ索引を渡します。

ディレクトリー更新: アクセス API

以下の System i Access for Windows ディレクトリー更新を使用して、更新項目へのアクセスを取得します。API はアルファベット順にリストされます。

注: 基本的には、アプリケーションで更新項目をこれ以上アクセスすることがなくなった時点で、を呼び出します。cwbUP_FreeLock が呼び出されないと、他のアプリケーションは、それらの更新項目にアクセスすることも変更することもできません。

cwbUP_FindEntry:

System i Access for Windows の cwbUP_FindEntry コマンドを使用します。

目的

entrySource と entryTarget を検索パラメーターとして使用して、既存の更新項目へのハンドルを獲得します。

構文

```
unsigned int CWB_ENTRY cwbUP_FindEntry(
                            char * entrySource,
                            char * entryTarget,
                            unsigned long *searchStart,
                            cwbUP_EntryHandle *entryHandle);
```

パラメーター

char * entrySource - input
更新項目のソースが含まれている、NULL 文字で終わるストリングを指します。これは、ドライブおよびパス、または UNC 名とすることが可能です。このストリングは、一致する更新項目を検索するために使用されます。

char * entryTarget - input
更新項目のターゲットが含まれている、NULL 文字で終わるストリングを指します。これは、ドライブおよびパス、または UNC 名とすることが可能です。このストリングは、一致する更新項目を検索するのに使用されます。

unsigned long * searchStart - input/output
検索を始めるための更新項目のリストにある索引を指すポインター。これは、複数の更新項目が、一致するソースおよびターゲットをもつ場合に使用されます。このパラメーターは、検索の項目をスキップし、リスト上の searchStart の後にある、一致する更新項目の検索を続行するために使用します。正常に終了した場合、searchStart は、更新項目が見付かったリスト内の位置に設定されます。すべての更新項目を検索したい場合には、searchStart を CWBUP_SEARCH_FROM_BEGINNING に設定してください。

cwbUP_EntryHandle * entryHandle - input/output
ハンドルが戻される先である cwbUP_EntryHandle を指すポインター。このハンドルは、以降の更新項目 API 呼び出しで使用する必要があります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL がアドレスとして渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリーが十分にないため、ハンドルを作成できません。

CWBUP_SEARCH_POSITION_ERROR
検索開始位置が無効。

CWBUP_ENTRY_NOT_FOUND
検索値に一致する更新項目がありません。

CWBUP_STRING_TOO_LONG
入力ストリングが、最大長 CWBUP_MAX_LENGTH よりも長くなっています。

使用法

この呼び出しから戻されたハンドルは、他の更新 API で更新項目にアクセスするために使用されます。この呼び出しを使用し、更新項目の処理を完了した後で、cwbUP_FreeEntryHandle を呼び出す必要があります。この呼び出しはその項目を「アンロック」し、その項目に関連する資源を解放します。

cwbUP_FreeLock:

System i Access for Windows の cwbUP_FreeLock コマンドを使用します。

目的

更新項目に対するロックを解放します。この API は、アプリケーションが更新項目のアクセスを終えた際に呼び出す必要があります。この API が呼び出されない場合、他のアプリケーションはその更新項目にアクセスできなくなります。

構文

```
unsigned int CWB_ENTRY cwbUP_FreeLock();
```

パラメーター

なし (None)

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBUP_UNLOCK_WARNING
アプリケーションは、更新項目をロックしていません。

使用法

アプリケーションが更新項目にアクセスしたり変更したりすると、更新項目に対するロックを獲得します。アプリケーションは、更新項目にアクセスする必要がなくなった際に、この API を呼び出す必要があります。

cwbUP_GetEntryHandle:

System i Access for Windows の cwbUP_GetEntryHandle コマンドを使用します。

目的

リスト内の所定の位置にある既存の更新項目へのハンドルを獲得します。

構文

```
unsigned int CWB_ENTRY cwbUP_GetEntryHandle(
                           unsigned long entryPosition,
                           cwbUP_EntryHandle *entryHandle);
```

パラメーター

unsigned long entryPosition - input
ハンドルの検索を必要とする項目の更新項目リストへの索引。(最初の更新項目を検索したい場合は 1 を渡します)

cwbUP_EntryHandle * entryHandle - input/output
ハンドルが戻される先である cwbUP_EntryHandle を指すポインター。このハンドルは、以降の更新項目 API 呼び出しで使用する必要があります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL がアドレスとして渡されました。

CWBUP_ENTRY_NOT_FOUND
所定の位置に更新項目がありません。

CWBUP_POSITION_INVALID
与えられた位置が範囲内ではありません。

使用法

この呼び出しから戻されたハンドルは、他の更新 API で更新項目にアクセスするために使用されます。この呼び出しを使用し、更新項目の処理を完了した後で、cwbUP_FreeEntryHandle を呼び出す必要があります。この呼び出しは項目をアンロックし、その項目に関連する資源を解放します。項目ハンドルを戻す API を呼び出すごとに、1 回ずつ cwbUP_FreeEntryHandle を呼び出す必要があります。

ディレクトリー更新: 資源を解放する API

以下の System i Access for Windows ディレクトリー更新 API を使用して、項目ハンドルに関連する資源を解放します。 API はアルファベット順にリストされます。

注: 基本的には、アプリケーションで更新項目をこれ以上アクセスすることがなくなった時点で、を呼び出します。cwbUP_FreeLock が呼び出されないと、他のアプリケーションは、それらの更新項目にアクセスすることも変更することもできません。

cwbUP_FreeEntryHandle:

System i Access for Windows の cwbUP_FreeEntryHandle コマンドを使用します。

目的

項目ハンドルおよびそれに関連するすべての資源を解放します。

構文

```
unsigned int CWB_ENTRY cwbUP_FreeEntryHandle(
                            cwbUP_EntryHandle entryHandle);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
解放される項目ハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
ハンドルが無効か、あるいは既に解放されています。

使用法

この呼び出しの後は、更新項目へのアクセスはできません。この更新項目あるいは別の更新項目にアクセスするには、新規の項目ハンドルを取得する必要があります。

ディレクトリー更新: 変更 API

以下の System i Access for Windows ディレクトリー更新 API を使用して、更新項目を変更します。 API はアルファベット順にリストされます。

注: 基本的には、アプリケーションで更新項目をこれ以上アクセスすることがなくなった時点で、を呼び出します。cwbUP_FreeLock が呼び出されないと、他のアプリケーションは、それらの更新項目にアクセスすることも変更することもできません。

cwbUP_AddPackageFile:

System i Access for Windows の cwbUP_AddPackageFile コマンドを使用します。

目的

パッケージ・ファイルを更新項目のパッケージ・ファイル・リストに追加します。

構文

```
unsigned int CWB_ENTRY cwbUP_AddPackageFile(
                            cwbUP_EntryHandle entryHandle,
                            char *entryPackage);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

char * entryPackage - input
更新項目に追加されるパッケージ・ファイルの名前が含まれている、ヌル終了ストリングを指すポインター。このファイルのパスは組み込まないでください。パッケージ・ファイルはソース・パスおよびターゲット・パスに存在する必要があります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWB_INVALID_POINTER
NULL がアドレスとして渡されました。

CWBUP_TOO_MANY_PACKAGES
この項目に既に存在するパッケージ・ファイルの最大数。

CWBUP_STRING_TOO_LONG
パッケージ・ファイル名が CWBUP_MAX_LENGTH よりも長くなっています。

CWBUP_ENTRY_IS_LOCKED
現在、別のアプリケーションが更新項目リストを変更中です。現時点では変更できません。

使用法

なし (None)

cwbUP_RemovePackageFile:

System i Access for Windows の cwbUP_RemovePackageFile コマンドを使用します。

目的

更新項目に属するパッケージ・ファイルのリストからパッケージ・ファイルを除去します。

構文

```
unsigned int CWB_ENTRY cwbUP_RemovePackageFile(
                            cwbUP_EntryHandle entryHandle,
                            char *entryPackage);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

char * entryPackage - input
パッケージ・ファイル・リストから除去されるパッケージ・ファイル名が含まれている、NULL 文字で終わるストリングを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWB_INVALID_POINTER
NULL がアドレス・パラメーターとして渡されました。

CWBUP_PACKAGE_NOT_FOUND
パッケージ・ファイルが見付かりません。

CWBUP_STRING_TOO_LONG
パッケージ・ファイル・ストリングが、最大長 CWBUP_MAX_LENGTH よりも長くなっています。

CWBUP_ENTRY_IS_LOCKED
現在、別のアプリケーションが更新項目リストを変更中です。現時点では変更できません。

使用法

なし (None)

cwbUP_SetCallbackDLL:

System i Access for Windows の cwbUP_SetCallbackDLL コマンドを使用します。

目的

更新項目のコールバック DLL の完全修飾名を設定します。

構文

```
unsigned int CWB_ENTRY  cwbUP_SetCallbackDLL(
                            cwbUP_EntryHandle entryHandle,
                            char *dllPath);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

char * dllPath - input
更新の個々の段階で呼び出される DLL の完全修飾名が含まれている、NULL 文字で終わるストリングを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWB_INVALID_POINTER
NULL がアドレス・パラメーターとして渡されました。

CWBUP_STRING_TOO_LONG
コールバック DLL ストリングが、最大長 CWBUP_MAX_LENGTH よりも長くなっています。

CWBUP_ENTRY_IS_LOCKED
現在、別のアプリケーションが更新項目リストを変更中です。現時点では変更できません。

使用法

なし (None)

cwbUP_SetDescription:

System i Access for Windows の cwbUP_SetDescription コマンドを使用します。

目的

更新項目の記述を設定します。

構文

```
unsigned int CWB_ENTRY cwbUP_SetDescription(
                            cwbUP_EntryHandle entryHandle,
                            char *entryDescription);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

char * entryDescription - input
更新項目に関連する完全な記述が含まれている、NULL 文字で終わるストリングを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWB_INVALID_POINTER
NULL がアドレス・パラメーターとして渡されました。

CWBUP_STRING_TOO_LONG
記述ストリングが、最大長 CWBUP_MAX_LENGTH よりも長くなっています。

CWBUP_ENTRY_IS_LOCKED
現在、別のアプリケーションが更新項目リストを変更中です。現時点では変更できません。

使用法

なし (None)

cwbUP_SetEntryAttributes:

System i Access for Windows の cwbUP_SetEntryAttributes コマンドを使用します。

目的

更新項目の次のいずれかの属性値を設定します。

CWBUP_FILE_DRIVEN
更新は、ターゲット・ファイルとソース・ファイルとの間でのファイル日付の比較に基づいて行われます。

CWBUP_PACKAGE_DRIVEN
更新は、パッケージ・ファイル (複数の場合あり)、ならびに、ターゲットとソースとの間でのファイルの日付の比較に基づいて行われます。

CWBUP_SUBDIRECTORY
更新によって、所定のパスに属しているディレクトリーの比較および更新が行われます。

CWBUP_ONEPASS
更新は、1 つの段階内で直接行われます。この値を指定しなかった場合、更新は 2 つの段階で行われます。最初の段階で、更新されるファイルを一時ディレクトリーにコピーしてから、PC がリブートされた時点で、それらのファイルをターゲット・ディレクトリーにコピーします。

CWBUP_BACKLEVEL_OK
この値を設定すると、更新は、ソースとターゲット上のファイルの日付が一致していない場合に行われます。この値を設定しなかった場合には、ソース・ファイルの方がターゲット・ファイルよりも日付が新しい場合にのみ更新が行われます。

これらのいずれの組み合わせも有効です。

構文

```
unsigned int CWB_ENTRY cwbUP_SetEntryAttributes(
                            cwbUP_EntryHandle entryHandle,
                            unsigned long entryAttributes);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

unsigned long entryAttributes - input
属性値の組み合わせ。(値については定義セクションを参照してください。)

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWBUP_ENTRY_IS_LOCKED
現在、別のアプリケーションが更新項目リストを変更中です。現時点では変更できません。

使用法

次にこの呼び出しの例を挙げます。

```
rc = cwbUP_SetEntryAttributes(entryHandle, CWBUP_FILEDRIVEN | CWBUP_ONEPASS );
```

この呼び出しによって更新項目がファイル主導型になり、更新が 1 段階で行われます。

cwbUP_SetSourcePath:

System i Access for Windows の cwbUP_SetSourcePath コマンドを使用します。

目的

更新項目のソース・パスを設定します。

構文

```
unsigned int CWB_ENTRY cwbUP_SetSourcePath(
                            cwbUP_EntryHandle entryHandle,
                            char *entrySource);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

char * entrySource - input
更新項目に関連する完全ソース・パスが含まれている、NULL 文字で終わるストリングを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWB_INVALID_POINTER
NULL がアドレス・パラメーターとして渡されました。

CWBUP_STRING_TOO_LONG
ソース・パス・ストリングが、最大長 CWBUP_MAX_LENGTH よりも長くなっています。

CWBUP_ENTRY_IS_LOCKED
現在、別のアプリケーションが更新項目リストを変更中です。現時点では変更できません。

使用法

なし (None)

cwbUP_SetTargetPath:

System i Access for Windows の cwbUP_SetTargetPath コマンドを使用します。

目的

更新項目のターゲット・パスを設定します。

構文

```
unsigned int CWB_ENTRY cwbUP_SetTargetPath(
                            cwbUP_EntryHandle entryHandle,
                            char *entryTarget);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

char * entryTarget - input
更新項目の完全ターゲット・パスが含まれている、NULL 文字で終わるストリングを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWB_INVALID_POINTER
NULL がアドレス・パラメーターとして渡されました。

CWBUP_STRING_TOO_LONG
ターゲット・パス・ストリングが、最大長 CWBUP_MAX_LENGTH よりも長くなっています。

CWBUP_ENTRY_IS_LOCKED
現在、別のアプリケーションが更新項目リストを変更中です。現時点では変更できません。

使用法

なし (None)

ディレクトリー更新: 情報 API

以下の System i Access for Windows ディレクトリー更新 API を使用して、更新項目からの情報を取得し、一般的なディレクトリー更新情報を検索します。 API はアルファベット順にリストされます。

注: 基本的には、アプリケーションで更新項目をこれ以上アクセスすることがなくなった時点で、を呼び出します。cwbUP_FreeLock が呼び出されないと、他のアプリケーションは、それらの更新項目にアクセスすることも変更することもできません。

cwbUP_GetCallbackDLL:

System i Access for Windows の cwbUP_GetCallbackDLL コマンドを使用します。

目的

更新項目用のコールバック DLL の完全修飾名を獲得します。

構文

```
unsigned int CWB_ENTRY  cwbUP_GetCallbackDLL(
                             cwbUP_EntryHandle entryHandle,
                             char *dllPath,
                             unsigned long bufferLength,
                             unsigned long *actualLength);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

char * dllPath - input/output
更新の個々の段階で呼び出される、DLL の完全修飾名を受け取るバッファーを指すポインター。

unsigned long bufferLength - input
dllPath バッファーの長さ。NULL 終了文字の 1 バイトを追加する必要があります。バッファーが DLL 名全体を保管することのできる十分な大きさでない場合は、エラーが戻され、actualLength パラメーターには dllPath バッファーがデータを入れるために必要とするバイト数が設定されます。

unsigned long * actualLength - input/output
完全修飾 DLL 名を入れるために必要なバッファー・サイズが設定される、長さ変数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWB_INVALID_POINTER
NULL がアドレス・パラメーターとして渡されました。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎて戻りデータが保管できません。

使用法

なし (None)

cwbUP_GetDescription:

System i Access for Windows の cwbUP_GetDescription コマンドを使用します。

目的

更新項目の記述を獲得します。

構文

```
unsigned int CWB_ENTRY cwbUP_GetDescription(
                            cwbUP_EntryHandle entryHandle,
```

```
                              char *entryDescription,
                              unsigned long bufferLength,
                              unsigned long *actualLength);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

char * entryDescription - input/output
更新項目の記述を受け取るバッファーを指すポインター。

unsigned long bufferLength - input
バッファーの長さ。NULL 終了文字の 1 バイトを追加する必要があります。バッファーが記述全体を保管することのできる十分な大きさでない場合、エラーが戻され、actualLength パラメーターには entryDescription バッファーがデータを入れるために必要とするバイト数が設定されます。

unsigned long * actualLength - input/output
記述を入れるために必要なバッファー・サイズが設定される、長さ変数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWB_INVALID_POINTER
NULL がアドレス・パラメーターとして渡されました。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎて戻りデータが保管できません。

使用法

なし (None)

cwbUP_GetEntryAttributes:

System i Access for Windows の cwbUP_GetEntryAttributes コマンドを使用します。

目的

更新項目の属性を獲得します。これらの属性には、1 段階更新、ファイル主導型の更新、パッケージ主導型の更新および更新サブディレクトリーが含まれます。これらのいずれの組み合わせも有効です。

構文

```
unsigned int CWB_ENTRY cwbUP_GetEntryAttributes(
                              cwbUP_EntryHandle entryHandle,
                              unsigned long *entryAttributes);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

unsigned long * entryAttributes - input/output
属性値を受け取る区域を指すポインター。 (値については定義セクションを参照してください。)

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWB_INVALID_POINTER
NULL がアドレス・パラメーターとして渡されました。

使用法

この呼び出しが行われた後で entryAttributes に含まれる値は、このファイルの上部付近にリストされた属性フラグの組み合わせになることがあります。

cwbUP_GetLockHolderName:

System i Access for Windows の cwbUP_GetLockHolderName コマンドを使用します。

目的

現在ロック状態の更新項目をもつプログラムの名前を獲得します。

構文

```
unsigned int CWB_ENTRY cwbUP_GetLockHolderName(char *lockHolder,
                                               unsigned long bufferLength,
                                               unsigned long *actualLength);
```

パラメーター

char * lockHolder - input/output
現在更新項目をロック状態にしているアプリケーション名を受け取るバッファーを指すポインター。

unsigned long bufferLength - input
バッファーの長さ。NULL 終了文字の 1 バイトを追加する必要があります。バッファーが名前全体を保管することのできる十分な大きさでない場合、エラーが戻され、actualLength パラメーターには lockHolder バッファーがデータを入れるために必要とするバイト数が設定されます。

unsigned long * actualLength - input/output
アプリケーション名を入れるために必要なバッファー・サイズが設定される長さ変数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL がアドレス・パラメーターとして渡されました。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎて戻りデータが保管できません。

使用法

なし (None)

cwbUP_GetSourcePath:

System i Access for Windows の cwbUP_GetSourcePath コマンドを使用します。

目的

更新項目のソース・パスを獲得します。

構文

```
unsigned int CWB_ENTRY cwbUP_GetSourcePath(
                            cwbUP_EntryHandle entryHandle,
                            char *entrySource,
                            unsigned long bufferLength,
                            unsigned long *actualLength);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

char * entrySource - input/output
更新項目のソース・パスを受け取るバッファーを指すポインター。

unsigned long bufferLength - input
バッファーの長さ。NULL 終了文字の 1 バイトを追加する必要があります。バッファーがソース・パス全体を保管することのできる十分な大きさでない場合、エラーが戻され、actualLength パラメーターには entrySource バッファーがデータを入れるために必要とするバイト数が設定されます。

unsigned long * actualLength - input/output
ソース・パスを入れるために必要なバッファー・サイズが設定される長さ変数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWB_INVALID_POINTER
NULL がアドレス・パラメーターとして渡されました。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎて戻りデータが保管できません。

使用法

なし (None)

cwbUP_GetTargetPath:

System i Access for Windows の cwbUP_GetTargetPath コマンドを使用します。

目的

更新項目のターゲット・パスを獲得します。

構文

```
unsigned int CWB_ENTRY cwbUP_GetTargetPath(
                            cwbUP_EntryHandle entryHandle,
                            char *entryTarget,
                            unsigned long bufferLength,
                            unsigned long *actualLength);
```

パラメーター

cwbUP_EntryHandle entryHandle - input
cwbUP_CreateUpdateEntryHandle、cwbUP_GetUpdateEntryHandle、または cwbUP_FindEntry への以前の呼び出しで戻されたハンドル。

char * entryTarget - input/output
更新項目のターゲット・パスを受け取るバッファーを指すポインター。

unsigned long bufferLength - input
バッファーの長さ。NULL 終了文字の 1 バイトを追加する必要があります。バッファーがターゲット・パス全体を保管することのできる十分な大きさでない場合、エラーが戻され、actualLength パラメーターには entryTarget バッファーがデータを入れるために必要とするバイト数が設定されます。

unsigned long * actualLength - input/output
ターゲット・パスを入れるために必要なバッファー・サイズが設定される長さ変数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
更新項目ハンドルが無効。

CWB_INVALID_POINTER
NULL がアドレス・パラメーターとして渡されました。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎて戻りデータが保管できません。

使用法

なし (None)

System i Access for Windows PC5250 エミュレーション API

System i Access for Windows の PC5250 エミュレーターは、既存のシステム・アプリケーションに使用できるグラフィカル・ユーザー・インターフェースを、デスクトップ・ユーザーに提供します。 PC5250 を使用することで、ユーザーは、System i に保管されているデータおよびアプリケーションと、簡単かつ明快に対話することができます。

PC5250 には、C/C++ アプリケーション・プログラミング・インターフェース (API) が用意されており、これによってワークステーション・プログラムで i5/OS ホスト・システムと対話することができます。

System i Access for Windows PC5250 C/C++ API:

エミュレーター高水準言語 API (EHLLAPI)
単純な単一のエントリー・ポイント・インターフェースであり、エミュレーター画面の解釈を行います。

パーソナル・コミュニケーションズ・セッション API (PCSAPI)
このインターフェースは、エミュレーター・セッションの開始、停止、および制御を行うために使用します。

ホスト・アクセス・クラス・ライブラリー (HACL)
このインターフェースは、アプリケーション開発に使用できる一連のクラスとメソッドを提供します。これによって、ホスト情報をデータ・ストリーム・レベルでアクセスすることができます。

System i Access for Windows エミュレーション API に必要なファイル

エミュレーション・インターフェース	ヘッダー・ファイル	インポート・ライブラリー	ダイナミック・リンク・ライブラリー
標準 HLLAPI	hapi_c.h	pscal32.lib	pcshll.dll pcshll32.dll
拡張 HLLAPI	ehlapi32.h	ehlapi32.lib	ehlapi32.dll
Windows EHLLAPI	whllapi.h	whllapi.lib whlapi32.lib	whllapi.dll whllapi32.dll
HACL インターフェース	eclall.hpp	pcseclva.lib pcseclvc.lib	pcseclva.dll pcseclvc.dll
PCSAPI インターフェース	pcsapi.h	pcscal32.lib	pcsapi.dll pcsapi32.dll

Programmer's Toolkit:

Programmer's Toolkit には、エミュレーター・インターフェースの資料、ヘッダー・ファイルへのアクセス、およびプログラム例へのリンクが用意されています。この情報にアクセスするには、Programmer's Toolkit をオープンして、「**エミュレーション**」->「**C/C++ API**」と選択します。

注: コード例を使用することで、 633 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』の条件に同意します。

System i Access for Windows 用システム・オブジェクト API

System i Access for Windows 用システム・オブジェクトのアプリケーション・プログラミング・インターフェース (API) を使用して、システム上にある印刷関連のオブジェクトを処理できます。これらの API は、System i スプール・ファイル、書き出しプログラム・ジョブ、出力待ち行列、プリンターなどを使った作業を可能にします。

システム・オブジェクト API を使用すると、ご使用の環境に応じてカスタマイズしたワークステーション・アプリケーションを作成できます。例えば、単一ユーザー、あるいは、System i オペレーティング・システムのネットワークを介して結ばれているすべてのユーザーのスプール・ファイルを管理するアプリケーションを作成することができます。これには、スプール・ファイルの保留、解放、属性の変更、削除、送信、検索およびスプール・ファイルに関するメッセージの応答が含まれます。

System i Access for Windows 用システム・オブジェクト API に必要なファイル

ヘッダー・ファイル	インポート・ライブラリー	ダイナミック・リンク・ライブラリー
cwbobj.h	cwbapi.lib	cwbobj.dll

Programmer's Toolkit:

Programmer's Toolkit には、システム・オブジェクト資料、cwbobj.h ヘッダー・ファイルへのアクセス、およびプログラム例へのリンクが用意されています。この情報にアクセスするには、Programmer's Toolkit をオープンして「**System i オペレーション (System i Operations)**」 → 「**C/C++ API**」と選択します。

注: コード例を使用することで、 633 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』の条件に同意します。

関連資料

30 ページの『システム・オブジェクト API の戻りコード』
System i Access for Windows システム・オブジェクト API の戻りコードを示します。

7 ページの『接続 API 用の System i 名の形式』
パラメーターとして System i 名を取得する API では、3 つの異なる形式の名前を使用できます。

8 ページの『OEM、ANSI、およびユニコードの考慮事項』
System i Access for Windows の、ストリング・パラメーターを受け入れる C/C++ API の大部分は、OEM、ANSI、Unicode の 3 つのうち、いずれかの形式になっています。

システム・オブジェクトの属性

ネットワーク印刷サーバーのオブジェクトには属性があります。ネットワーク印刷サーバーは、以下の属性をサポートします。特定の組み合わせに対してサポートされている属性を判別するには、それぞれのオブジェクト / アクションのデータ・ストリームに関する説明を参照してください。

Advanced Function Printing:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_AFP

ID 0x000A

タイプ char[11]

説明 このスプール・ファイルが、このファイルの外部にある AFP 資源を使用するかどうかを示します。有効な値は *YES または *NO です。

ページの位置合わせ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_ALIGN

ID 0x000B

タイプ char[11]

説明 このスプール・ファイルを印刷する前に、用紙位置決めメッセージを送るかどうかを示します。有効な値は *YES または *NO です。

直接印刷可能:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_ALWDRTPRT

ID 0x000C

タイプ char[11]

説明 プリントに直接印刷するジョブに対して、印刷装置書き出しプログラムにプリンターを割り振りさせるかどうかを示します。有効な値は *YES または *NO です。

権限:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_AUT

ID 0x000D

タイプ char[11]

説明 出力待ち行列に対する特定の権限を持たないユーザーに対して、与える権限を指定します。有効な値は、*USE、*ALL、*CHANGE、*EXCLUDE、*LIBCRTAUT です。

検査権限:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_AUTCHK

ID 0x000E

タイプ char[11]

説明 出力待ち行列に対するどんなタイプの権限によって、ユーザーが出力待ち行列の全ファイルを制御できるようにするかを示します。有効な値は *OWNER または *DTAAUT です。

書き出しプログラムの自動終了:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_AUTOEND

ID 0x0010

タイプ char[11]

説明 書き出しプログラムが自動的に終了するかどうかを示します。有効な値は *YES または *NO です。

バック・マージン・オフセット (横方向):

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_BACKMGN_ACR

ID 0x0011

タイプ float

説明 用紙の裏側について、ページの左端からどれだけ離れた位置で印刷が開始されるかを指定します。特殊値 *FRONTMGN は -1 としてエンコードされます。

バック・マージン・オフセット (下方向):

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_BACKMGN_DWN

ID 0x0012

タイプ float

説明 用紙の裏側について、ページの上端からどれだけ離れた位置で印刷が開始されるかを指定します。特殊値 *FRONTMGN は -1 としてエンコードされます。

背面オーバーレイ・ライブラリー名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_BKOVRLLIB

ID 0x0013

タイプ char[11]

説明 背面オーバーレイが入っているライブラリー名。背面オーバーレイの名前フィールドに特殊値が入っている場合は、このライブラリー・フィールドはブランクです。

背面オーバーレイの名前:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_BKOVRLAY

ID 0x0014

タイプ char[11]

説明 背面オーバーレイの名前。有効な特殊値には *FRONTMGN が含まれます。

背面オーバーレイ・オフセット (横方向):

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_BKOVL_ACR

ID 0x0016

タイプ float

説明 オーバーレイが印刷される起点から横方向へのオフセット。

背面オーバーレイ・オフセット (下方向):

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_BKOVL_DWN

ID 0x0015

タイプ float

説明 オーバーレイが印刷される起点から下方へのオフセット。

1 インチ当たり文字数:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_CPI

ID 0x0017

タイプ float

説明 横方向 1 インチ当たりの文字数。

コード・ページ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_CODEPAGE

ID 0x0019

タイプ char[11]

説明 このスプール・ファイルについて、グラフィック文字のコード・ポイントへのマッピング。グラフィック文字セット・フィールドに特殊値が入っていると、このフィールドにはゼロ (0) が入ります。

コード化フォント名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_CODEDFNT

ID 0x001A

タイプ char[11]

説明 コード化フォントの名前。コード化フォントとは、文字セットとコード・ページで構成される AFP 資源のことです。特殊値には *FNTCHRSET が含まれます。

コード化フォント・ライブラリー名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_CODEDFNTLIB

ID 0x0018

タイプ char[11]

説明 コード化フォントが入っているライブラリーの名前。コード化フォントの名前フィールドが特殊値をもつ場合は、このフィールドにはブランクが入ります。

コピー:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_COPIES

ID 0x001C

タイプ long

説明 このスプール・ファイル用に作成されるコピーの合計数。

作成されていない残りのコピー:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_COPIESLEFT

ID 0x001D

タイプ long

説明 このスプール・ファイル用に作成されるコピーの残りの数。

現行ページ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_CURPAGE

ID 0x001E

タイプ long

説明 書き出しプログラム・ジョブが書き出し中の現行ページ。

データ形式:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DATAFORMAT

ID 0x001F

タイプ char[11]

説明 データ形式。有効な値は *RCDDATA または *ALLDATA です。

データ待ち行列ライブラリー名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DATAQUELIB

ID 0x0020

タイプ char[11]

説明 データ待ち行列が入っているライブラリーの名前。

データ待ち行列名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DATAQUE

ID 0x0021

タイプ char[11]

説明 出力待ち行列に関連するデータ待ち行列の名前。

ファイルがオープンされた日付:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DATE

ID 0x0022

タイプ char[8]

説明 スプール・ファイルがオープンされた日付。日付は、C YY MM DD の形式で文字ストリングにエンコードされています。

ユーザー指定の DBCS データ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DBCSDATA

ID 0x0099

タイプ char[11]

説明 スプール・ファイルに 2 バイト文字セット (DBCS) データが入っているかどうかを示します。有効な値は *NO または *YES です。

DBCS 拡張文字:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DBCSEXTENSN

ID 0x009A

タイプ char[11]

説明 システムが DBCS 拡張文字を処理するかどうかを示します。有効な値は *NO または *YES です。

DBCS 文字回転:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DBCAROTATE

ID 0x009B

タイプ char[11]

説明 印刷前に DBCS 文字を反時計方向に 90 度回転させるかどうかを示します。有効な値は *NO または *YES です。

1 インチ当たりの DBCS 文字数:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DBCSCPI

ID 0x009C

タイプ long

説明 1 インチ当たり印刷される 2 バイト文字の数。有効な値は、-1、-2、5、6、および 10 です。値 *CPI は -1 としてエンコードされます。値 *CONDENSED は -2 としてエンコードされます。

DBCS SO/SI スペース:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DBCSSISO

ID 0x009D

タイプ char[11]

説明 印刷時にシフトアウト文字とシフトイン文字を表示するかどうかを決めます。有効な値は、*NO、*YES、および *RIGHT です。

書き出し据え置き:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DFR_WRITE

ID 0x0023

タイプ char[11]

説明 印刷データが印刷前にシステム・バッファーに保持されるかどうかを示します。

ページ回転の角度:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PAGRTT

ID 0x0024

タイプ long

説明 用紙がプリンターにロードされる向きに対する、ページ上のテキストの回転角度。有効な値は、-1、-2、-3、0、90、180、および 270 です。値 *AUTO は -1 に、*DEVD は -2 に、*COR は -3 にそれぞれエンコードされます。

送信後にファイルを削除:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DELETESPLF

ID 0x0097

タイプ char[11]

説明 スプール・ファイルを送信後削除します。有効な値は *NO または *YES です。

宛先オプション:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DESTOPTION

ID 0x0098

タイプ char[129]

説明 宛先オプション。ユーザーが受信システムにオプションを渡すことができるようにするテキスト・ストリングです。

宛先タイプ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DESTINATION

ID 0x0025

タイプ char[11]

説明 宛先タイプ。有効な値は *OTHER、*AS400、および *PSF2 です。

装置クラス:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DEVCLASS

ID 0x0026

タイプ char[11]

説明 装置クラス。

装置型式:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DEVMODEL

ID 0x0027

タイプ char[11]

説明 装置の型式番号。

装置タイプ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DEVTYPE

ID 0x0028

タイプ char[11]

説明 装置タイプ。

ファイルの表示:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DISPLAYANY

ID 0x0029

タイプ char[11]

説明 この出力待ち行列を読み取る権限をもつユーザーが、この待ち行列のいずれの出力ファイルの出力データも表示できるか、ユーザー自身のファイルのデータしか表示できないかを示します。有効な値は、*YES、*NO、および *OWNER です。

区切りページの用紙入れ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DRWRSEP

ID 0x002A

タイプ long

説明 ジョブおよびファイルの区切りページが取り出される用紙入れを識別します。有効な値は、-1、-2、1、2、および 3 です。値 *FILE は -1、値 *DEVD は -2 としてエンコードされます。

終了ページ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_ENDPAGE

ID 0x002B

タイプ long

説明 スプール・ファイルの印刷を終了するときのページ番号。有効な値は 0 か、または終了ページ番号です。値 *END は 0 としてエンコードされます。

ファイル区切り:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FILESEP

ID 0x002C

タイプ long

説明 スプール・ファイルの各コピーの最初に置かれる、ファイル区切りページ数。有効な値は、-1 または区切りページの数です。値 *FILE は -1 としてエンコードされます。

レコードの折り返し:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FOLDREC

ID 0x002D

タイプ char[11]

説明 印刷用紙幅を超えるレコードが次行に折り返されるかどうかを示します。有効な値は *YES または *NO です。

フォント識別コード:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FONTID

ID 0x002E

タイプ char[11]

説明 使用される印刷フォント。有効な特殊値には *CPI と *DEVD が含まれます。

用紙送り:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FORMFEED

ID 0x002F

タイプ char[11]

説明 プリンターでの用紙送りの方法を示します。有効な値は、*CONT、*CUT、*AUTOCUT、および *DEVD です。

用紙タイプ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FORMTYPE

ID 0x0030

タイプ char[11]

説明 このスプール・ファイルを印刷するためにプリンターにロードされる用紙のタイプ。

用紙タイプ・メッセージ・オプション:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FORMTYPEMSG

ID 0x0043

タイプ char[11]

説明 この現行用紙タイプが終了したときに、メッセージを書き出しプログラムのメッセージ待ち行列に送信するメッセージ・オプション。有効な値は、*MSG、*NOMSG、*INFOMSG、および *INQMSG です。

フロント・マージン・オフセット (横方向):

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FTMGN_ACR

ID 0x0031

タイプ float

説明 用紙の表側について、ページの左端からどの程度離れた位置で印刷を開始するかを指定します。特殊値 *DEVD は -2 としてエンコードされます。

フロント・マージン・オフセット (下方向):

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FTMGN_DWN

ID 0x0032

タイプ float

説明 用紙の表側について、ページの上端からどの程度離れた位置で印刷を開始するかを指定します。特殊値 *DEVD は -2 としてエンコードされます。

前面オーバーレイ・ライブラリー名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FTOVRLLIB

ID 0x0033

タイプ char[11]

説明 前面オーバーレイが入っているライブラリー名。前面オーバーレイ名前フィールドに特殊値が入っている場合は、このフィールドはブランクです。

前面オーバーレイ名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FTOVRLAY

ID 0x0034

タイプ char[11]

説明 前面オーバーレイの名前。有効な特殊値には *NONE が含まれます。

前面オーバーレイ・オフセット (横方向):

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FTOVL_ACR

ID 0x0036

タイプ float

説明 オーバーレイが印刷される起点から横方向へのオフセット。

前面オーバーレイ・オフセット (下方向):

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FTOVL_DWN

ID 0x0035

タイプ float

説明 オーバーレイが印刷される起点から下方へのオフセット。

グラフィック文字セット:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_CHAR_ID

ID 0x0037

タイプ char[11]

説明 このファイルを印刷するときに使用されるグラフィック文字セット。有効な特殊値には、*DEVD、*SYSVAL、および *JOBCCSID が含まれます。

ハードウェア位置合わせ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_JUSTIFY

ID 0x0038

タイプ long

説明 出力が右寄せされるその割合。有効な値は、0、50、および 100 です。

スプール・ファイルの保留:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_HOLD

ID 0x0039

タイプ char[11]

説明 スプール・ファイルが保留されるかどうかを示します。有効な値は *YES または *NO です。

書き出しプログラムの初期設定:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_WTRINIT

ID 0x00AC

タイプ char[11]

説明 プリンターを初期設定する時期をユーザーが指定することができます。有効な値は、*WTR、*FIRST、*ALL です。

IP アドレス:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_INTERNETADDR

ID 0x0094

タイプ char[16]

説明 受信システムの IP アドレス。

ジョブ名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_JOBNAME

ID 0x003B

タイプ char[11]

説明 スプール・ファイルを作成したジョブの名前。

ジョブ番号:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_JOBNUMBER

ID 0x003C

タイプ char[7]

説明 スプール・ファイルを作成したジョブの番号。

ジョブ区切り:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_JOBSEPRATR

ID 0x003D

タイプ long

説明 この出力待ち行列にスプール・ファイルをもつ各ジョブの出力の最初に置かれるジョブ区切りの数。有効な値は、-2 および 0 から 9 です。値 *MSG は -2 としてエンコードされます。ジョブ区切りは、出力待ち行列が作成されるときに指定されます。

ジョブ・ユーザー:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USER

ID 0x003E

タイプ char[11]

説明 スプール・ファイルを作成したユーザーの名前。

印刷された最終ページ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_LASTPAGE

ID 0x003F

タイプ long

説明 ジョブが処理を完了する前に印刷が終了した場合、ファイルの、最後に印刷されたページ番号。

ページの長さ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PAGELEN

ID 0x004E

タイプ float

説明 ページの長さ。測定単位は、測定方法属性に指定されます。

ライブラリー名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_LIBRARY

ID 0x000F

タイプ char[11]

説明 ライブラリーの名前。

1 インチ当たりの行数:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_LPI

ID 0x0040

タイプ float

説明 スプール・ファイルの縦方向 1 インチ当たりの行数。

メーカー、機種型式:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MFGTYPE

ID 0x0041

タイプ char[21]

説明 印刷データを SCS から ASCII へ変換するときに、メーカー、機種、および型式を指定します。

スプール出力レコードの最大数:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MAXRECORDS

ID 0x0042

タイプ long

説明 このファイルがオープンされたときの、このファイルの最大許容レコード数。値 *NOMAX は 0 としてエンコードされます。

測定方法:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MEASMETHOD

ID 0x004F

タイプ char[11]

説明 ページ長属性およびページ幅属性で使用される測定方法。有効な値は *ROWCOL または *UOM です。

メッセージ・ヘルプ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MSGHELP

ID 0x0081

タイプ char(*)

説明 2 次レベル・テキストとしても知られているメッセージ・ヘルプで、メッセージ検索要求がこのメッセージ・ヘルプを戻すことができます。長さはシステムによって 3000 文字に制限されています (英語バージョンでは、翻訳される場合を考慮してこれよりも 30 % 少なくなければなりません)。

メッセージ ID:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MESSAGEID

ID 0x0093

タイプ char[8]

説明 メッセージ ID。

メッセージ待ち行列ライブラリー名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MSGQUELIB

ID 0x0044

タイプ char[11]

説明 メッセージ待ち行列が入っているライブラリーの名前。

メッセージ待ち行列:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MSGQUE

ID 0x005E

タイプ char[11]

説明 書き出しプログラムが操作メッセージ用に使用する、メッセージ待ち行列の名前。

メッセージ応答:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MSGREPLY

ID 0x0082

タイプ char[133]

説明 メッセージ応答。クライアントが出すテキスト・ストリングで、「照会」タイプのメッセージに応

答します。検索されるメッセージの場合は、属性値がサーバーによって戻され、これにはクライアントが使用できるデフォルト応答が含まれます。長さはシステムによって 132 文字に制限されています。可変長のため、NULL 文字で終わるようにしてください。

メッセージ・テキスト:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MSGTEXT

ID 0x0080

タイプ char[133]

説明 第 1 レベル・テキストとしても知られているメッセージ・テキストで、メッセージ検索要求がこのメッセージ・ヘルプを戻すことができます。長さはシステムによって 132 文字に制限されています。

メッセージ・タイプ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MSGTYPE

ID 0x008E

タイプ char[3]

説明 メッセージ・タイプで、2 つの数字の EBCDIC エンコードされたものです。2 つのタイプのメッセージによって、検索されたメッセージに応答できるかどうかを示します。すなわち、通知メッセージ '04' は応答を要求しないで情報を伝送し (代わりに訂正アクションが必要な場合があります)、照会メッセージ '05' は情報を伝送して応答を要求します。

メッセージ重大度:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MSGSEV

ID 0x009F

タイプ long

説明 メッセージ重大度。値の範囲は 00 から 99 までです。値が高いほど、状況はより重大、もしくはより重要です。

読み取り/書き込みバイト数:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_NUMBYTES

ID 0x007D

タイプ long

説明 読み取り操作において読み取るバイト数、または書き込み操作において書き込むバイト数。オブジェクト・アクションがこの属性の解釈方法を決めます。

ファイル数:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_NUMFILES

ID 0x0045

タイプ long

説明 出力待ち行列に存在するスプール・ファイルの数。

待ち行列に対して開始された書き出しプログラムの数:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_NUMWRITERS

ID 0x0091

タイプ long

説明 出力待ち行列に対して開始された書き出しプログラム・ジョブの数。

オブジェクト拡張属性:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_OBJEXTATTR

ID 0x000B1

タイプ char[11]

説明 フォント資源のような、いくつかのオブジェクトによって使用される拡張属性。この値は、System i コマンドの WRKOBJ および DSPOBJD を通じて表示されます。 System i 画面上の表題は、「属性」のみを示します。例えば、フォント資源のオブジェクト・タイプの場合、共通の値は、CDEPAG、CDEFNT、および FNTCHRSET になります。

オープン時のコマンド:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_OPENCMDS

ID 0x00A0

タイプ char[11]

説明 スプール・ファイル・データに先立って、ユーザーが、SCS オープン時のコマンドをデータ・ストリームに挿入するかどうかを指定します。有効な値は *YES または *NO です。

オペレーター制御:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_OPCNTRL

ID 0x0046

タイプ char[11]

説明 ジョブ制御権限をもつユーザーが、この待ち行列上のスプール・ファイルの管理または制御を許可されているかどうかを示します。有効な値は *YES または *NO です。

待ち行列上のファイルの順序:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_ORDER

ID 0x0047

タイプ char[11]

説明 この出力待ち行列上のスプール・ファイルの順序。有効な値は *FIFO または *JOBNBR です。

出力優先順位:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_OUTPTY

ID 0x0048

タイプ char[11]

説明 スプール・ファイルの優先順位。優先順位は 1 (最高) から 9 (最低) までです。有効な値は 0 から 9 で、0 は *JOB を表します。

出力待ち行列ライブラリー名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_OUTQUELIB

ID 0x0049

タイプ char[11]

説明 出力待ち行列が入っているライブラリーの名前。

出力待ち行列名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_OUTQUE

ID 0x004A

タイプ char[11]

説明 出力待ち行列の名前。

出力待ち行列の状況:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_OUTQUESTS

ID 0x004B

タイプ char[11]

説明 出力待ち行列の状況。有効な値は RELEASED または HELD です。

オーバーフロー行番号:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_OVERFLOW

ID 0x004C

タイプ long

説明 印刷中のデータが、次のページへオーバーフローする前に印刷される最後の行。

面当たりページ数:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_MULTIUP

ID 0x0052

タイプ long

説明 ファイルの印刷時に、各物理ページの各面に印刷する論理ページの数。有効な値は、1、2、および 4 です。

ペル密度:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PELDENSITY

ID 0x00B2

タイプ char[2]

説明 フォント資源についてのみ、この値は、ペル数をエンコードしたものになります (″1″ は、ペル・サイズ 240 を表し、″2″ はペル・サイズ 320 を表します)。追加した値は、システムによる定義が行われると有効になります。

ポイント・サイズ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_POINTSIZE

ID 0x0053

タイプ float

説明 このスプール・ファイルのテキストが印刷されるポイント・サイズ。特殊値 *NONE は 0 としてエンコードされます。

印刷精度:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_FIDELITY

ID 0x0054

タイプ char[11]

説明 印刷時に実行されるエラー処理の種類。有効な値は *ABSOLUTE または *CONTENT です。

両面印刷:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DUPLEX

ID 0x0055

タイプ char[11]

説明 情報が印刷される方法を示します。有効な値は、*FORMDF、*NO、*YES、および *TUMBLE です。

印刷品質:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PRTQUALITY

ID 0x0056

タイプ char[11]

説明 このスプール・ファイルを印刷するときに使用される印刷品質。有効な値は、*STD、*DRAFT、*NLQ、および *FASTDRAFT です。

印刷順序:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PRTSEQUENCE

ID 0x0057

タイプ char[11]

説明 印刷順序。有効な値は *NEXT です。

印刷テキスト:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PRTTEXT

ID 0x0058

タイプ char[31]

説明 印刷出力の各ページの下部および区切りページ上に印刷されるテキスト。有効な特殊値には *BLANK と *JOB が含まれます。

プリンター:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PRINTER

ID 0x0059

タイプ char[11]

説明 プリンターの名前。

プリンター・タイプ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PRTDEVTYPE

ID 0x005A

タイプ char[11]

説明 プリンター・データ・ストリーム・タイプ。有効な値は、*SCS、*IPDS(*)、*USERASCII、および *AFPDS です。

プリンター・ファイル・ライブラリー名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PRTRFILELIB

ID 0x005B

タイプ char[11]

説明 プリンター・ファイルが入っているライブラリーの名前。

プリンター・ファイル名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PRTRFILE

ID 0x005C

タイプ char[11]

説明 プリンター・ファイルの名前。

プリンター待ち行列:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_RMTPRTQ

ID 0x005D

タイプ char[129]

説明 SNDTCPSPLF (LPR) によりスプール・ファイルを送信するときの宛先プリンター待ち行列の名前。

レコードの長さ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_RECLENGTH

ID 0x005F

タイプ long

説明 レコードの長さ

リモート・システム:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_RMTSYSTEM

ID 0x0060

タイプ char[256]

説明 リモート・システムの名前。有効な特殊値には *INTNETADR が含まれます。

印刷不能文字の置き換え:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_RPLUNPRT

ID 0x0061

タイプ char[11]

説明 印刷できない文字が別の文字に置き換えられるかどうかを示します。有効な値は *YES または *NO です。

置換文字:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_RPLCHAR

ID 0x0062

タイプ char[2]

説明 印刷不能文字を置き換える文字。

資源ライブラリー名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_RSCLIB

ID 0x00AE

タイプ char[11]

説明 外部 AFP (高機能印刷) 資源が入っているライブラリーの名前。

資源名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_RSCNAME

ID 0x00AF

タイプ char[11]

説明 外部 AFP 資源の名前。

資源オブジェクト・タイプ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_RSCTYPE

ID 0x00B0

タイプ Long

説明 外部 AFP 資源オブジェクト・タイプの数値的、ビット・エンコード方式。値は、*FNTRSC、*FORMDF、*OVL、*PAGSEG、*PAGDFN にそれぞれ対応して、0x0001、0x0002、0x0004、0x0008、0x0010 になります。

印刷の再始動:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_RESTART

ID 0x0063

タイプ long

説明 印刷の再始動。有効な値は、-1、-2、-3、または再始動する場所のページ番号です。値 *STRPAGE は -1 として、*ENDPAGE は -2 として、*NEXT は -3 としてそれぞれエンコードされます。

スプール・ファイルの保管:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SAVESPLF

ID 0x0064

タイプ char[11]

説明 スプール・ファイルが書き込まれた後、保管されるかどうかを示します。有効な値は *YES または *NO です。

シーク・オフセット:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SEEKOFF

ID 0x007E

タイプ long

説明 シーク・オフセット。シーク起点に対応して正の値と負の値の両方が可能です。

起点のシーク:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SEEKORG

ID 0x007F

タイプ long

説明 有効な値には 1 (最初または上部)、2 (現行)、3 (終わりまたは下部) が含まれます。

送信優先順位:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SENDPTY

ID 0x0065

タイプ char[11]

説明 送信優先順位。有効な値は *NORMAL または *HIGH です。

区切りページ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SEPPAGE

ID 0x00A1

タイプ char[11]

説明 バナー・ページの印刷のオプションの使用許可をユーザーに与えます。有効な値は *YES または *NO です。

ソース・ドロワー:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SRCDRWR

ID 0x0066

タイプ long

説明 カット用紙自動送りオプションが選択されたときに使用される用紙入れ。有効な値は、-1、-2、および 1 - 255 です。値 *E1 は -1、値 *FORMDF は -2 としてそれぞれエンコードされます。

スプール SCS:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SPLSCS

ID 0x00AD

タイプ Long

説明 スプール・ファイルの作成中、どのようにして SCS データを使用するかを指示します。有効な値は -1、0、1、またはページ番号です。値 *ENDPAGE は -1 としてエンコードされます。値 0 では、印刷はページ 1 から開始されます。値 1 では、ファイル全体が印刷されます。

データのスプール:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SPOOL

ID 0x0067

タイプ char[11]

説明 プリンターの出力データがスプールされるかどうかを示します。有効な値は *YES または *NO です。

スプール・ファイル名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SPOOLFILE

ID 0x0068

タイプ char[11]

説明 スプール・ファイルの名前。

スプール・ファイル番号:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SPLFNUM

ID 0x0069

タイプ long

説明 スプール・ファイルの番号

スプール・ファイルの状況:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SPLFSTATUS

ID 0x006A

タイプ char[11]

説明 スプール・ファイルの状況。有効な値は、
*CLOSED、*HELD、*MESSAGE、*OPEN、*PENDING、 *PRINTER、*READY、*SAVED、および *WRITING です。

スプール出力のスケジュール:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SCHEDULE

ID 0x006B

タイプ char[11]

説明 スプール・ファイルが書き出しプログラムで使用可能になったときに、スプール・ファイルについてだけ指定します。有効な値は、*IMMED、*FILEEND、および *JOBEND です。

開始ページ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_STARTPAGE

ID 0x006C

タイプ long

説明 スプール・ファイルの印刷を開始するページの番号。有効な値は -1、0、1、またはページ番号です。値 *ENDPAGE は -1 としてエンコードされます。値 0 では、印刷はページ 1 から開始されます。値 1 では、ファイル全体が印刷されます。

テキスト記述:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_DESCRIPTION

ID 0x006D

タイプ [51]

説明 System i オブジェクトのインスタンスを記述するテキスト。

ファイルがオープンされた時刻:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_TIMEOPEN

ID 0x006E

タイプ char[7]

説明 このスプール・ファイルがオープンされた時刻。時刻は HH MM SS 形式で、文字 0x0005 にエンコードされます。

合計ページ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PAGES

ID 0x006F

タイプ long

説明 スプール・ファイル中に含まれるページ数。

SCS から ASCII への変換:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_SCS2ASCII

ID 0x0071

タイプ char[11]

説明 印刷データが SCS から ASCII に変換されるかどうかを示します。有効な値は *YES または *NO です。

計測単位:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_UNITOFMEAS

ID 0x0072

タイプ char[11]

説明 距離を指定するために使用する測定単位。有効な値は *CM または *INCH です。

ユーザー・コメント:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USERCMT

ID 0x0073

タイプ char[101]

説明 スプール・ファイルを説明するユーザー指定の 100 文字の注釈。

ユーザー・データ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USERDATA

ID 0x0074

タイプ char[11]

説明 スプール・ファイルを説明するユーザー指定の 10 文字のデータ。有効な特殊値には *SOURCE が含まれます。

ユーザー定義データ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USRDFNDTA

ID 0x00A2

タイプ char[]

説明 スプール・ファイルを処理する、ユーザー・アプリケーションまたはユーザー指定プログラムによって利用されるユーザー定義データ。すべての文字が受け入れられます。最大値は 255 です。

ユーザー定義オブジェクト・ライブラリー:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USRDFNOBJLIB

ID 0x00A4

タイプ char[11]

説明 スプール・ファイルを処理するユーザー・アプリケーションによって検索するためのユーザー定義オブジェクト・ライブラリー。

ユーザー定義オブジェクト名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USRDFNOBJ

ID 0x00A5

タイプ char[11]

説明 スプール・ファイルを処理するユーザー・アプリケーションによって利用される、ユーザー定義オブジェクト名。

ユーザー定義オブジェクト・タイプ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USRDFNOBJTYP

ID 0x00A6

タイプ char[11]

説明 ユーザー定義オブジェクトに関係するユーザー定義オブジェクト・タイプ。

ユーザー定義オプション:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USEDFNOPTS

ID 0x00A3

タイプ char[*]

説明 スプール・ファイルを処理するユーザー・アプリケーションによって利用されるユーザー定義オプション。最大 4 オプションまで指定することができ、それぞれの値の長さは、char(10) です。すべての文字が受け入れられます。

ユーザー・ドライバー・プログラム:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USRDRVPGMDTA

ID 0x00A9

タイプ char[11]

説明 ユーザー・ドライバー・プログラムで使用されるユーザー・データ。すべての文字が受け入れられます。最大サイズは 5000 文字です。

ユーザー・ドライバー・プログラム・ライブラリー:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USRDRVPGMLIB

ID 0x00AA

タイプ char[11]

説明 スプール・ファイルを処理するドライバー・プログラムを検索するための、ユーザー定義ライブラリー。

ユーザー・ドライバー・プログラム名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USRDRVPGM

ID 0x00AB

タイプ char[11]

説明 スプール・ファイルを処理するユーザー定義プログラム名。

ユーザー ID:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_TOUSERID

ID 0x0075

タイプ char[9]

説明 スプール・ファイルが送信される先のユーザー ID。

ユーザー ID アドレス:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_TOADDRESS

ID 0x0076

タイプ char[9]

説明 スプール・ファイルが送信される先のユーザーのアドレス。

ユーザー変換プログラム・ライブラリー:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USRTFMPGMLIB

ID 0x00A7

タイプ char[11]

説明 変換プログラムを検索するユーザー定義ライブラリー。

ユーザー変換プログラム名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_USETFMPGM

ID 0x00A8

タイプ char[11]

説明 スプール・ファイル・データを、それがドライバー・プログラムによって処理される前に変換するユーザー定義変換プログラム名。

VM/MVS クラス:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_VMMVSCLASS

ID 0x0077

タイプ char[2]

説明 VM/MVS クラス。有効な値は、A から Z および 0 から 9 です。

書き出しプログラムの自動終了時点:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_WTRAUTOEND

ID 0x0078

タイプ char[11]

説明 書き出しプログラムを自動的に終了する場合に、いつ終了させるかを指定します。有効な値は *NORDYF または *FILEEND です。書き出しプログラムの自動終了の属性を *YES に設定しておかなければなりません。

書き出しプログラムの終了時点:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_WTREND

ID 0x0090

タイプ char[11]

説明 書き出しプログラムをいつ終了させるかを指定します。有効な値は、*CNTRLD、*IMMED、および *PAGEEND です。これは「書き出しプログラムの自動終了時点」とは異なります。

ファイルの保留時点:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_HOLDTYPE

ID 0x009E

タイプ char[11]

説明 スプール・ファイルをいつ保留するかを指定します。有効な値は *IMMED および *PAGEEND です。

ページ幅:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_PAGEWIDTH

ID 0x0051

タイプ float

説明 ページの幅。測定単位は、測定方法属性に指定されます。

ワークステーション・カスタマイズ・オブジェクト名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_WSCUSTMOBJ

ID 0x0095

タイプ char[11]

説明 ワークステーション・カスタマイズ・オブジェクトの名前。

ワークステーション・カスタマイズ・オブジェクト・ライブラリー:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_WSCUSTMOBJL

ID 0x0096

タイプ char[11]

説明 ワークステーション・カスタマイズ・オブジェクトが入っているライブラリーの名前。

書き出しプログラム・ジョブ名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_WRITER

ID 0x0079

タイプ char[11]

説明 書き出しプログラム・ジョブの名前。

書き出しプログラム・ジョブ番号:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_WTRJOBNUM

ID 0x007A

タイプ char[7]

説明 書き出しプログラム・ジョブの番号。

書き出しプログラム・ジョブ状況:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_WTRJOBSTS

ID 0x007B

タイプ char[11]

説明 書き出しプログラム・ジョブの状況。有効な値は、STR、END、JOBQ、HLD、および MSGW です。

書き出しプログラム・ジョブ・ユーザー名:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_WTRJOBUSER

ID 0x007C

タイプ char[11]

説明 書き出しプログラム・ジョブを開始したユーザーの名前。

書き出しプログラム開始ページ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_WTRSTRPAGE

ID 0x008F

タイプ long

説明 書き出しプログラム・ジョブを開始したときに、最初のスプール・ファイルから印刷する最初のページのページ番号を指定します。これは、書き出しプログラムを開始したときにスプール・ファイル名もまた指定されている場合にのみ有効です。

ネットワーク印刷サーバー・オブジェクトの属性:

System i Access for Windows 製品使用時の、ネットワーク印刷サーバーのオブジェクト属性のリストを、以下に示します。

NPS 属性のデフォルト値:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_ATTRDEFAULT

ID 0x0083

タイプ dynamic

説明 属性のデフォルト値。

NPS 属性の高限界:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_ATTRMAX

ID 0x0084

タイプ dynamic

説明 属性値の高限界。

NPS 属性 ID:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_ATTRID

ID 0x0085

タイプ long

説明 属性の ID。

NPS 属性の低限界:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_ATTRMIN

ID 0x0086

タイプ dynamic

説明 属性値の低限界。

NPS 属性の可能値:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_ATTRPOSSIBL

ID 0x0087

タイプ dynamic

説明 属性の可能値。複数の NPS 可能値インスタンスがコード・ポイントに存在する場合があります。

NPS 属性テキスト記述:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_ATTRDESCRIPT

ID 0x0088

タイプ char(*)

説明 属性の名前を与えるテキスト記述。

NPS 属性タイプ:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_ATTRTYPE

ID 0x0089

タイプ long

説明 属性のタイプ。有効な値は、ネットワーク印刷サーバーが定義するタイプです。

NPS CCSID:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_NPSCCSID

ID 0x008A

タイプ long

説明 すべてのストリングがこれによってエンコードされているものとネットワーク印刷サーバーが予測している CCSID。

NPS オブジェクト:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_NPSOBJECT

ID 0x008B

タイプ long

説明 オブジェクト ID。有効な値は、ネットワーク印刷サーバーが定義するオブジェクトです。

NPS オブジェクト・アクション:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_NPSACTION

ID 0x008C

タイプ long

説明 アクション ID。有効な値は、ネットワーク印刷サーバーが定義するアクションです。

NPS レベル:

これは、System i Access for Windows 製品で使用する API です。

キー CWBOBJ_KEY_NPSLEVEL

ID 0x008D

タイプ char[7]

説明 ネットワーク印刷サーバーのバージョン・レベル、リリース・レベル、およびモディフィケーション・レベルです。この属性は VXRYMY としてエンコードされた文字ストリング (例えば、「V3R1M0」など) です。この場合の X と Y は次のようになります。

```
X is in (0..9)
Y is in (0..9,A..Z)
```

System i Access for Windows 用のリスト API

以下の System i Access for Windows API は、リスト・オブジェクトに関するものです。 API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_CloseList:

これは、System i Access for Windows 製品で使用する API です。

目的

オープンされているリストをクローズします。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_CloseList(
                              cwbOBJ_ListHandle  listHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ListHandle listHandle - input
クローズされるリストのハンドル。このリストはオープンされていなければなりません。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたリスト・ハンドルではありません。

CWBOBJ_RC_LIST_NOT_OPEN
リストがオープンされていません。

使用法

リストをクローズすると、その項目を保持するためにリストが使用した記憶域が解放されます。cwbOBJ_GetObjHandle() API で得られたいずれのオブジェクト・ハンドルも、資源を解放するため、リストをクローズする前に解放してください。これらのハンドルは、もはや有効ではありません。

cwbOBJ_CreateListHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトのリスト用のハンドルを割り振ります。このリスト・ハンドルが割り振られると、cwbOBJ_SetListFilter() API によるリストのフィルター基準の設定、cwbOBJ_OpenList() API によるリストの作成などが可能になります。このリスト・ハンドルを割り振り解除し、これによって使用されていたすべての資源を解放するには、cwbOBJ_DeleteListHandle() を呼び出してください。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_CreateListHandle(
                              const char         *systemName,
                              cwbOBJ_ListType     type,
                              cwbOBJ_ListHandle  *listHandle,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

const char *systemName - input
ASCIIZ ストリングに入っているシステム名を指すポインター。

cwbOBJ_ListType type - input
割り振りを行うリストのタイプ (例えば、スプール・ファイル・リスト、出力待ち行列リストなど)。

cwbOBJ_ListHandle *listHandle - output
出力の際に返されるリスト・ハンドルを指すポインター。このハンドルは、他の呼び出しでリストを使用する時に必要になります。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

このリスト・ハンドルの使用が終わった後に、呼び出し側は cwbOBJ_DeleteListHandle を呼び出す必要があります。オブジェクトのリストを検索するための一般的な呼び出し手順を以下に示します。

1. cwbOBJ_CreateListHandle()
2. cwbOBJ_SetListFilter() { 必要に応じて繰り返す }

3. cwbOBJ_OpenList()
4. cwbOBJ_GetListSize() リストのサイズを取得する
5. n=0 から リスト・サイズ -1 の 位置 n にあるリスト項目に対する cwbOBJ_GetObjHandle を cwbOBJ_DeleteObjHandle() によって何らかの処理を行う。
6. cwbOBJ_CloseList() - ここからステップ 2 へ戻ることができる
7. cwbOBJ_DeleteListHandle()

cwbOBJ_DeleteListHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

cwbOBJ_CreateListHandle() API で以前割り振られていたリスト・ハンドルを割り振り解除します。これにより、リストに関連する資源はすべて解放されます。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_DeleteListHandle(
                              cwbOBJ_ListHandle  listHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ListHandle listHandle - input
削除されるリスト・ハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_INVALID_HANDLE
リスト・ハンドルが見付かりません。

使用法

このハンドルに関連するリストがオープンされている場合は、この呼び出しがリストをクローズします。このリスト内にオープンされたオブジェクトのハンドルがあっても、それらはもはや有効ではありません。この呼び出しが正常に戻ったあとでは、リスト・ハンドルはもはや有効ではありません。

cwbOBJ_GetListSize:

これは、System i Access for Windows 製品で使用する API です。

目的

オープンされたリストのサイズを取得します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_GetListSize(
                              cwbOBJ_ListHandle   listHandle,
                              unsigned long      *size,
                              cwbOBJ_List_Status *listStatus,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbOBJ_ListHandle listHandle - input
サイズを取得するリストのハンドル。このリストはオープンされていなければなりません。

unsigned long *size - output
出力の際に、リストの現行サイズに設定されます。

cwbOBJ_List_Status *listStatus - output
オプションであり、NULL でも構いません。同時にオープンされたリストでは常に、CWBOBJ_LISTSTS_COMPLETED です。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたリスト・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_LIST_NOT_OPEN
リストがオープンされていません。

使用法

なし (None)

cwbOBJ_OpenList:

これは、System i Access for Windows 製品で使用する API です。

目的

リストをオープンします。これは実際にリストを作成します。このリストの使用を終了したときは、資源を解放するために、呼び出し側は cwbOBJ_ClostList() API を呼び出さなければなりません。リストがオープンされたあと、呼び出し側は、リスト・サイズの取得、あるいはリスト中の項目のオブジェクト・ハンドルの取得などのような処理を行うためにリスト上の他の API を使用することができます。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_OpenList(
                              cwbOBJ_ListHandle     listHandle,
                              cwbOBJ_List_OpenType  openType,
                              cwbSV_ErrHandle       errorHandle);
```

パラメーター

cwbOBJ_ListHandle listHandle - input
オープンするリストのハンドル。

cwbOBJ_List_OpenType openHandle - input
リストをオープンする方法。CWBOBJ_LIST_OPEN_SYNCH に設定されなければなりません。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたリスト・ハンドルではありません。

CWBOBJ_RC_LIST_OPEN
リストは既にオープンされています。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_NOHOSTSUPPORT
ホストでは、このタイプのリストはサポートしていません。

使用法

なし (None)

cwbOBJ_ResetListAttrsToRetrieve:

これは、System i Access for Windows 製品で使用する API です。

目的

情報を検索するリストの属性を、デフォルトのリストのものにリセットします。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_ResetListAttrsToRetrieve(
                              cwbOBJ_ListHandle   listHandle,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbOBJ_ListHandle listHandle - input
リセットするリスト・ハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
ハンドルが、割り振られたリスト・ハンドルではありません。

使用法

cwbOBJ_SetListAttrsToRetrieve() を呼び出した後、この呼び出しを使用して、検索するリスト・ハンドルの属性のリストをリセットしてください。

cwbOBJ_ResetListFilter:

これは、System i Access for Windows 製品で使用する API です。

目的

リスト上のフィルターを、そのリストが最初に割り振られたときのフィルター (デフォルトのフィルター) にリセットします。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_ResetListFilter(
                              cwbOBJ_ListHandle  listHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ListHandle listHandle - input
そのフィルターがリセットされるリストのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたリスト・ハンドルではありません。

使用法

変更を反映するためには、リストをクローズしてから、再度オープンする必要があります。

cwbOBJ_SetListAttrsToRetrieve:

これは、System i Access for Windows 製品で使用する API です。

目的

リストがオープンされる前に、リスト・ハンドルに適用できるオプションの機能。これを行う目的は、cwbOBJ_OpenList() API が、アプリケーションで使用される各オブジェクトの属性のみを検索できるようにして効率を改善することです。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_SetListAttrsToRetrieve(
                              cwbOBJ_ListHandle    listHandle,
                              unsigned long        numKeys,
                              const cwbOBJ_KeyID  *keys,
                              cwbSV_ErrHandle      errorHandle);
```

パラメーター

cwbOBJ_ListHandle listHandle - input
属性キーのリストを適用するリスト・ハンドル。

unsigned long numKeys - input
キー・パラメーターが指すキーの数。0 でも構いません。この場合、リスト中のオブジェクトには属性が必要でないことを意味します。

const cwbOBJ_KeyID *keys - input
リストがオープンされたときに、リスト中の各オブジェクトごとに検索される属性の ID である numKeys キーの配列。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれま

す。このオブジェクトは cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたリスト・ハンドルではありません。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

使用法

この呼び出しは、リストされているオブジェクトのどの属性にアプリケーションが関心をもっているかについて、cwbOBJ_OpenList() API への手掛かりを与えるために使用されます。この情報を使用すると、cwbOBJ_OpenList() API をより効率的にすることができます。キー・リスト中の属性キーが有効かどうかは、リストされたオブジェクトのタイプによって決まります (cwbOBJ_CreateListHandle() に設定されます)。リストをキーの、デフォルトのリストにリセットするには、cwbOBJ_ResetListAttrsToRetrieve() を呼び出してください。

cwbOBJ_SetListFilter:

これは、System i Access for Windows 製品で使用する API です。

目的

リストのフィルターを設定します。このフィルターは、cwbOBJ_OpenList() が次に呼び出されるときに適用されます。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_SetListFilter(
                              cwbOBJ_ListHandle  listHandle,
                              cwbOBJ_KeyID       key,
                              const char        *value,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ListHandle listHandle - input
このフィルターが適用されるリスト・ハンドル。

cwbOBJ_KeyID key - input
設定されるフィルター・フィールドの ID。

const void *value - input
このフィールドに設定すべき値。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_INVALID_HANDLE
リスト・ハンドルが見付かりません。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

キーの値によって、値が指すタイプが決まります。値の長さは、そのタイプによって決まります。次のフィルターは、これらのリスト・タイプのスプール・ファイル・リストに対して設定することができます。

- CWBOBJ_LIST_SPLF:
  - CWBOBJ_KEY_USER

    どのユーザーのスプール・ファイルをリストするかを指定。特定のユーザー ID または次の特殊値: *ALL - 全ユーザー。 *CURRENT - 現行ユーザーのリスト・スプール・ファイルのみ。 *CURRENT がデフォルト。
  - CWBOBJ_KEY_OUTQUELIB

    どのライブラリーで出力待ち行列を検索するかを指定。特定の名前または次のいずれかの特殊値: "" - OUTQUEUE キーワードが *ALL の場合は、この組み合わせはシステム上のすべての出力待ち行列を検索する。 *CURLIB - 現行ライブラリー *LIBL - ライブラリー・リスト OUTQUE フィルターが *ALL でない場合 *LIBL がデフォルト。OUTQUE フィルターが *ALL に設定されている場合、"" がデフォルト。
  - CWBOBJ_KEY_OUTQUE

    どの出力待ち行列でスプール・ファイルを検索するかを指定。特定の名前または特殊値 *ALL が可。*ALL がデフォルト。
  - CWBOBJ_KEY_FORMTYPE

    持っている用紙タイプ属性によって、どのスプール・ファイルがリストされるかを指定。特定の名前または次のいずれかの特殊値: *ALL - どの用紙タイプを持つスプール・ファイルもリストされる。 *STD - 用紙タイプが *STD のスプール・ファイルがリストされる。 *ALL がデフォルト。
  - CWBOBJ_KEY_USERDATA

持っているユーザー・データによって、どのスプール・ファイルがリストされるかを指定。特定の値または次のいずれかの特殊値: *ALL - どのユーザー・データ値を持つスプール・ファイルもリストされる。*ALL がデフォルト。

出力待ち行列リスト

- CWBOBJ_LIST_OUTQ:
  - CWBOBJ_KEY_OUTQUELIB

    どのライブラリーで出力待ち行列を検索するかを指定。特定の名前、総称名、または次のいずれかの特殊値: *ALL - すべてのライブラリー *ALLUSER - すべてのユーザー定義ライブラリーに加えて、ユーザー・データが入っていて Q で始まる名前を持つライブラリー。 *CURLIB - 現行ライブラリー。 *LIBL - ライブラリー・リスト。 *USRLIBL - ライブラリー・リストのユーザー部分。*LIBL がデフォルト。

  -
    - CWBOBJ_KEY_OUTQUE

      どの出力待ち行列をリストするかを指定。特定の名前、総称名、または *ALL。*ALL がデフォルト。

プリンター記述リスト

- CWBOBJ_LIST_PRTD:
  - CWBOBJ_KEY_PRINTER

    どのプリンターをリストするかを指定。特定の名前、総称名、または *ALL。*ALL がデフォルト。

プリンター・ファイル・リスト

- CWBOBJ_LIST_PRTF:
  - CWBOBJ_KEY_PRTRFILELIB

    プリンター・ファイルを検索するライブラリーを指定。特定の名前、総称名、または次のいずれかの特殊値。

    - *ALL - すべてのライブラリー
    - *ALLUSER - すべてのユーザー定義のライブラリーに加えて、ユーザー・データが入っており Q で始まる名前を持つライブラリー
    - *CURLIB - 現行ライブラリー
    - *LIBL - ライブラリー・リスト
    - *USRLIBL - ライブラリー・リストのユーザー部分
    - *ALL がデフォルト。
  - CWBOBJ_KEY_PRTRFILE

    どのプリンター・ファイルをリストするかを指定。特定の名前、総称名、または *ALL。*ALL がデフォルト。

書き出しプログラム・ジョブ・リスト

- CWBOBJ_LIST_WTR:
  - CWBOBJ_KEY_WRITER

どの書き出しプログラム・ジョブをリストするかを指定。特定の名前、総称名、または *ALL。*ALL がデフォルト。

– CWBOBJ_KEY_OUTQUELIB および CWBOBJ_KEY_OUTQUE

これらのフィルターは、特定の出力待ち行列に対して活動中の書き出しプログラムのリストを取得するために共に使用される。OUTQUE キーが指定されると WRITER キーは無視される (指定された出力待ち行列のすべての書き出しプログラムがリストされる)。OUTQUE キーが指定されていて、OUTQUELIB が指定されていない場合は、OUTQUEULIB はデフォルトで *LIBL、つまりシステム・ライブラリー・リストになる。デフォルトは、これらのいずれにも指定されない。

ライブラリー・リスト

- CWBOBJ_LIST_LIB:
  – CWBOBJ_KEY_LIBRARY

    どのライブラリーをリストするかを指定。特定の名前、総称名、または次のいずれかの特殊値。
    - *ALL - すべてのライブラリー
    - *CURLIB - 現行ライブラリー
    - *LIBL - ライブラリー・リスト
    - *USRLIBL - ライブラリー・リストのユーザー部分
    - *USRLIBL がデフォルト。
- CWBOBJ_LIST_RSC:
  – 資源は、スプール・ファイル内のリスト (この場合は、そのスプール・ファイルで使用するすべての外部 AFP 資源のリスト)、ライブラリー内のリスト、またはライブラリー・セット内のリストの場合があります。スプール・ファイルの資源をリストする場合は、RSCTYPE 属性と RSCNAME 属性用の SetListFilter API と一緒に cwbOBJ_SetListFilterWithSplF API を使用してください。
    - CWBOBJ_KEY_RSCLIB

      資源を検索するライブラリーを指定します。リストがスプール・ファイルによってフィルターに掛けられる (たとえば、SetListFilterWithSplF を使用する) 場合、このフィルターは無視されます。特定の名前、総称名、または次のいずれかの特殊値。
      - *ALL - すべてのライブラリー
      - *ALLUSR - すべてのユーザー定義のライブラリーに加えて、ユーザー・データが収められており Q で始まる名前を持つライブラリー
      - *CURLIB - 現行ライブラリー
      - *LIBL - ライブラリー・リスト
      - *USRLIBL - ライブラリー・リストのユーザー部分
      - *LIBL がデフォルト。
    - CWBOBJ_KEY_RSCNAME

      リストする資源の名前を指定します。特定の名前、総称名、または *ALL。

      *ALL がデフォルト。
    - CWBOBJ_KEY_RESCTYPE

      リストする資源のタイプを指定します。論理和演算が行われた次のビットのどのような組み合わせを指定することもできます。

- CWBOBJ_AFPRSC_FONT
- CWBOBJ_AFPRSC_FORMDEF
- CWBOBJ_AFPRSC_OVERLAY
- CWBOBJ_AFPRSC_PAGESEG
- CWBOBJ_AFPRSC_PAGEDEF

cwbOBJ_SetListFilterWithSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

スプール・ファイルに対してリスト用のフィルターを設定します。資源のリスト表示に関して、この呼び出しは、openList によって戻される資源をスプール・ファイルで使用されるものに限定します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_SetListFilterWithSplF(
                              cwbOBJ_ListHandle  listHandle,
                              cwbOBJ_ObjHandle   splFHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ListHandle listHandle - input
このフィルターが適用されるリスト・ハンドル。

cwbOBJ_ObjHandle splFHandle - input
フィルターを行うスプール・ファイルのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWBOBJ_RC_INVALID_TYPE
リストの誤ったタイプ。

CWB_INVALID_HANDLE
リスト・ハンドルが見付からないか、またはスプール・ファイル・ハンドルが正しくない。

使用法

AFP 資源をリスト表示するとき、スプール・ファイルによるフィルター操作が使用されるため、リスト・タイプは CWBOBJ_LIST_RSC である必要があります。スプール・ファイルに基づいて資源をフィルターに掛ける場合も、1 つまたは複数のライブラリーに基づいて資源をフィルターに掛けることはできません。両方が指定された場合は、資源ライブラリー・フィルターが無視されます。リスト・フィルターをリセット

すると、スプール・ファイル・フィルターもまた、何もない状態にリセットされます。

System i Access for Windows オブジェクト API

以下の System i Access for Windows API は、オブジェクトに関するものです。 API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_CopyObjHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトに重複ハンドルを作成します。この API を使用して、同じ System i オブジェクトの別のハンドルを取得します。この新規ハンドルは、それを解放するための cwbOBJ_DeleteObjHandle() API が呼び出されるまで有効です。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_CopyObjHandle(
                              cwbOBJ_ObjHandle   objectHandle,
                              cwbOBJ_ObjHandle  *newObjectHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ObjHandle objectHandle - input
コピーするオブジェクトのハンドル。

cwbOBJ_ObjHandle *newObjectHandle - output
この呼び出しが正常に完了すると、このハンドルには新規のオブジェクト・ハンドルが入ります。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

使用法

あるリスト上のオブジェクトへのハンドルを持っていて、このリストがクローズされた後もそのオブジェクトのハンドルを保持したい場合、この API を使用してハンドルを保持することができます。このハンドル

用の資源を解放するには、cwbOBJ_DeleteObjHandle() を呼び出す必要があります。

cwbOBJ_DeleteObjHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトへのハンドルを解放します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_DeleteObjHandle(
                              cwbOBJ_ObjHandle  objectHandle,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle objectHandle - input
解放するオブジェクトのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

使用法

なし (None)

cwbOBJ_GetObjAttr:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトの属性を取得します。

構文

```
unsigned int CWB_ENTRY cwbOBJ_GetObjAttr(
                              cwbOBJ_ObjHandle  objectHandle,
                              cwbOBJ_KeyID      key,
```

```
void             *buffer,
unsigned long     bufLen,
unsigned long    *bytesNeeded,
cwbOBJ_DataType  *keyType,
cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle objectHandle - input
属性を取得するオブジェクトのハンドル。

cwbOBJ_KeyID key - input
検索する属性の識別キー。CWBOBJ_KEY_XXX 定数がキー ID を定義します。objectHandle が指すオブジェクトのタイプによって、どのキーが有効かが決まります。

void *buffer - output
この呼び出しが正常に戻った場合は、属性値を保持するバッファー。*Buffer に置かれるデータ・タイプは何かをキー値が決定します。タイプが与えられた場合、このタイプも *keyType パラメーターへ戻されます。

unsigned long bufLen - input
*Buffer が指すバッファーの長さ。

unsigned long *bytesNeeded - output
出力の際には、結果を保持するために必要なバイト数が入ります。

cwbOBJ_DataType *keyType - output
オプションであり、NULL でも構いません。出力の際には、この属性と、*buffer に何が保管されるかを表すために使用されるデータ・タイプが含まれます。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎます。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_KEY
キーが有効ではありません。

CWB_API_ERROR
一般 API 障害。

使用法

次に挙げるオブジェクト・タイプでは、以下の属性を検索することができます。

- CWBOBJ_LIST_SPLF:

```
CWBOBJ_KEY_AFP          - 使用された AFP 資源
CWBOBJ_KEY_ALIGN        - ページの位置合わせ
CWBOBJ_KEY_BKMGN_ACR    - バック・マージン (横方向)
CWBOBJ_KEY_BKMGN_DWN    - バック・マージン (下方向)
CWBOBJ_KEY_BKOVRLLIB    - 背面オーバーレイ・ライブラリー名
CWBOBJ_KEY_BKOVRLAY     - 背面オーバーレイ名
CWBOBJ_KEY_BKOVL_ACR    - 背面オーバーレイ・オフセット (横方向)
CWBOBJ_KEY_BKOVL_DWN    - 背面オーバーレイ・オフセット (下方向)
CWBOBJ_KEY_CPI          - 1 インチ当たりの文字数
CWBOBJ_KEY_CODEDFNTLIB  - コード化フォント・ライブラリー名
CWBOBJ_KEY_CODEDFNT     - コード化フォント
CWBOBJ_KEY_COPIES       - 合計コピー数
CWBOBJ_KEY_COPIESLEFT   - 作成されていない残りのコピー
CWBOBJ_KEY_CURPAGE      - 現行ページ
CWBOBJ_KEY_DATE         - ファイルがオープンされた日付
CWBOBJ_KEY_PAGRTT       - ページの回転角度
CWBOBJ_KEY_ENDPAGE      - 終了ページ
CWBOBJ_KEY_FILESEP      - ファイル区切り
CWBOBJ_KEY_FOLDREC      - レコードの折り返し
CWBOBJ_KEY_FONTID       - 使用するフォント識別コード (デフォルト)
CWBOBJ_KEY_FORMFEED     - 用紙送り
CWBOBJ_KEY_FORMTYPE     - 用紙タイプ
CWBOBJ_KEY_FTMGN_ACR    - フロント・マージン (横方向)
CWBOBJ_KEY_FTMGN_DWN    - フロント・マージン (下方向)
CWBOBJ_KEY_FTOVRLLIB    - 前面オーバーレイ・ライブラリー名
CWBOBJ_KEY_FTOVRLAY     - 前面オーバーレイ
CWBOBJ_KEY_FTOVL_ACR    - 前面オーバーレイ・オフセット (横方向)
CWBOBJ_KEY_FTOVL_DWN    - 前面オーバーレイ・オフセット (下方向)
CWBOBJ_KEY_CHAR_ID      - グラフィック文字セット
CWBOBJ_KEY_JUSTIFY      - ハードウェアの位置合わせ
CWBOBJ_KEY_HOLD         - スプール・ファイルの保留
CWBOBJ_KEY_JOBNAME      - ファイルを作成したジョブの名前
CWBOBJ_KEY_JOBNUMBER    - ファイルを作成したジョブの番号
CWBOBJ_KEY_USER         - ファイルを作成したユーザーの名前
CWBOBJ_KEY_LASTPAGE     - 印刷された最終のページ
CWBOBJ_KEY_LPI          - 1 インチ当たりの行数
CWBOBJ_KEY_MAXRECORDS   - 許容最大レコード数
CWBOBJ_KEY_OUTPTY       - 出力優先順位
CWBOBJ_KEY_OUTQUELIB    - 出力待ち行列ライブラリー名
CWBOBJ_KEY_OUTQUE       - 出力待ち行列
CWBOBJ_KEY_OVERFLOW     - オーバーフロー行番号
CWBOBJ_KEY_PAGELEN      - ページ長
CWBOBJ_KEY_MEASMETHOD   - 測定方法
CWBOBJ_KEY_PAGEWIDTH    - ページ幅
CWBOBJ_KEY_MULTIUP      - 面当たりの論理ページ数
CWBOBJ_KEY_POINTSIZE    - デフォルトのフォントのポイント・サイズ
CWBOBJ_KEY_FIDELITY     - 印刷精度
CWBOBJ_KEY_DUPLEX       - 両面印刷
CWBOBJ_KEY_PRTQUALITY   - 印刷品質
CWBOBJ_KEY_PRTTEXT      - 各ページの下部に印刷されたテキスト
CWBOBJ_KEY_PRTDEVTYPE   - プリンター・タイプ (データ・ストリーム・タイプ)
CWBOBJ_KEY_PRTRFILELIB  - プリンター・ファイル・ライブラリー
CWBOBJ_KEY_PRTRFILE     - プリンター・ファイル
CWBOBJ_KEY_RECLENGTH    - レコード長
CWBOBJ_KEY_RPLUNPRT     - 印刷不能文字の置き換え
CWBOBJ_KEY_RPLCHAR      - 印刷不能文字の置き換え文字
CWBOBJ_KEY_RESTART      - 印刷再始動位置
CWBOBJ_KEY_SAVESPLF     - 印刷後のファイルの保管
```

```
CWBOBJ_KEY_SRCDRWR      - 用紙入れ
CWBOBJ_KEY_SPOOLFILE    - スプール・ファイル名
CWBOBJ_KEY_SPLFNUM      - スプール・ファイル番号
CWBOBJ_KEY_SPLFSTATUS   - スプール・ファイル状況
CWBOBJ_KEY_STARTPAGE    - 印刷開始ページ
CWBOBJ_KEY_TIME         - スプール・ファイルがオープンされた時刻
CWBOBJ_KEY_PAGES        - スプール・ファイル中のページ数
CWBOBJ_KEY_UNITOFMEAS   - 測定単位
CWBOBJ_KEY_USERCMT      - ユーザーの注釈
CWBOBJ_KEY_USERDATA     - ユーザー・データ
CWBOBJ_KEY_USRDFNDTA    - ユーザー定義データ
CWBOBJ_KEY_USRDFNOPTS   - ユーザー定義オプション
CWBOBJ_KEY_USRDFNOBJ    - ユーザー定義オブジェクト
CWBOBJ_KEY_USRDFNOBJLIB - ユーザー定義オブジェクト・ライブラリー
CWBOBJ_KEY_USRDFNOBJTYP - ユーザー定義オブジェクト・タイプ
```

- CWBOBJ_LIST_OUTQ:

```
CWBOBJ_KEY_AUTHCHCK     - 検査権限
CWBOBJ_KEY_DATAQUELIB   - データ待ち行列ライブラリー
CWBOBJ_KEY_DATAQUE      - データ待ち行列
CWBOBJ_KEY_DESCRIPTION  - テキスト記述
CWBOBJ_KEY_DISPLAYANY   - ユーザーは待ち行列のいずれのファイルも表示可能
CWBOBJ_KEY_JOBSEPRATR   - ジョブ区切りの数
CWBOBJ_KEY_NUMFILES     - 出力待ち行列のスプール・ファイルの合計
CWBOBJ_KEY_NUMWRITERS   - 待ち行列が開始された書き出しプログラムの数
CWBOBJ_KEY_OPCNTRL      - オペレーター制御
CWBOBJ_KEY_ORDER        - 待ち行列上の配列 (順序)
CWBOBJ_KEY_OUTQUELIB    - 出力待ち行列ライブラリー名
CWBOBJ_KEY_OUTQUE       - 出力待ち行列
CWBOBJ_KEY_OUTQUESTS    - 出力待ち行列状況
CWBOBJ_KEY_PRINTER      - プリンター
CWBOBJ_KEY_SEPPAGE      - バナー・ページの印刷
CWBOBJ_KEY_USRDFNDTA    - ユーザー定義データ
CWBOBJ_KEY_USRDFNOBJ    - ユーザー定義オブジェクト
CWBOBJ_KEY_USRDFNOBJLIB - ユーザー定義オブジェクト・ライブラリー
CWBOBJ_KEY_USRDFNOBJTYP - ユーザー定義オブジェクト・タイプ
CWBOBJ_KEY_USRDFNOPTS   - ユーザー定義オプション
CWBOBJ_KEY_USRDRVPGM    - ユーザー・ドライバー・プログラム
CWBOBJ_KEY_USRDRVPGMLIB - ユーザー・ドライバー・プログラム・ライブラリー
CWBOBJ_KEY_USRDRVPGMDTA - ユーザー・ドライバー・プログラム・データ
CWBOBJ_KEY_USRTFMPGM    - ユーザー・データ変換プログラム
CWBOBJ_KEY_USRTFMPGMLIB - ユーザー・データ変換プログラム・ライブラリー
CWBOBJ_KEY_WRITER       - 書き出しプログラム・ジョブ名
CWBOBJ_KEY_WTRJOBNUM    - 書き出しプログラム・ジョブ番号
CWBOBJ_KEY_WTRJOBSTS    - 書き出しプログラム・ジョブ状況
CWBOBJ_KEY_WTRJOBUSER   - 書き出しプログラム・ジョブ・ユーザー
```

- CWBOBJ_LIST_PRTD:

```
CWBOBJ_KEY_AFP          - 使用された AFP 資源
CWBOBJ_KEY_CODEPAGE     - コード・ページ
CWBOBJ_KEY_DEVCLASS     - 装置クラス
CWBOBJ_KEY_DEVMODEL     - 装置モデル
CWBOBJ_KEY_DEVTYPE      - 装置タイプ
CWBOBJ_KEY_DRWRSEP      - 区切り用紙入れ
CWBOBJ_KEY_FONTID       - フォント識別コード
CWBOBJ_KEY_FORMFEED     - 用紙送り
CWBOBJ_KEY_CHAR_ID      - グラフィック文字セット
CWBOBJ_KEY_MFGTYPE      - メーカーの機種とモデル
CWBOBJ_KEY_MSGQUELIB    - メッセージ待ち行列ライブラリー
CWBOBJ_KEY_MSGQUE       - メッセージ待ち行列
CWBOBJ_KEY_POINTSIZE    - デフォルトのフォントのポイント・サイズ
CWBOBJ_KEY_PRINTER      - プリンター
CWBOBJ_KEY_PRTQUALITY   - 印刷品質
CWBOBJ_KEY_DESCRIPTION  - テキスト記述
CWBOBJ_KEY_SCS2ASCII    - SCS から ASCII への変換
CWBOBJ_KEY_USRDFNDTA    - ユーザー定義データ
CWBOBJ_KEY_USRDFNOPTS   - ユーザー定義オプション
```

CWBOBJ_KEY_USRDFNOBJLIB - ユーザー定義オブジェクト・ライブラリー
CWBOBJ_KEY_USRDFNOBJ - ユーザー定義オブジェクト
CWBOBJ_KEY_USRDFNOBJTYP - ユーザー定義オブジェクト・タイプ
CWBOBJ_KEY_USRTFMPGMLIB - ユーザー・データ変換プログラム・ライブラリー
CWBOBJ_KEY_USRTFMPGM - ユーザー・データ変換プログラム
CWBOBJ_KEY_USRDRVPGMDTA - ユーザー・ドライバー・プログラム・データ
CWBOBJ_KEY_USRDRVPGMLIB - ユーザー・ドライバー・プログラム・ライブラリー
CWBOBJ_KEY_USRDRVPGM - ユーザー・ドライバー・プログラム

- CWBOBJ_LIST_PRTF:

CWBOBJ_KEY_ALIGN - ページの位置合わせ
CWBOBJ_KEY_BKMGN_ACR - バック・マージン (横方向)
CWBOBJ_KEY_BKMGN_DWN - バック・マージン (下方向)
CWBOBJ_KEY_BKOVRLLIB - 背面オーバーレイ・ライブラリー
CWBOBJ_KEY_BKOVRLAY - 背面オーバーレイ名
CWBOBJ_KEY_BKOVL_DWN - 背面オーバーレイ・オフセット (下方向)
CWBOBJ_KEY_BKOVL_ACR - 背面オーバーレイ・オフセット (横方向)
CWBOBJ_KEY_CPI - 1 インチ当たりの文字数
CWBOBJ_KEY_CODEDFNTLIB - コード化フォント・ライブラリー名
CWBOBJ_KEY_CODEPAGE - コード・ページ
CWBOBJ_KEY_CODEDFNT - コード化フォント
CWBOBJ_KEY_COPIES - 合計コピー数
CWBOBJ_KEY_DBCSDATA - DBCS 文字セット・データを含む
CWBOBJ_KEY_DBCSEXTENSN - DBCS 拡張文字の処理
CWBOBJ_KEY_DBCSROTATE - DBCS 文字の回転
CWBOBJ_KEY_DBCSCPI - DBCS CPI
CWBOBJ_KEY_DBCSSISO - DBCS SI/SO 位置決め
CWBOBJ_KEY_DFR_WRITE - 書き出し据え置き
CWBOBJ_KEY_PAGRTT - ページの回転角度
CWBOBJ_KEY_ENDPAGE - 印刷終了ページ
CWBOBJ_KEY_FILESEP - ファイル区切りの数
CWBOBJ_KEY_FOLDREC - レコードの折り返し
CWBOBJ_KEY_FONTID - 使用するフォント識別コード (デフォルト)
CWBOBJ_KEY_FORMFEED - 使用する用紙送り
CWBOBJ_KEY_FORMTYPE - 使用する用紙タイプ
CWBOBJ_KEY_FTMGN_ACR - フロント・マージン (横方向)
CWBOBJ_KEY_FTMGN_DWN - フロント・マージン (下方向)
CWBOBJ_KEY_FTOVRLLIB - 前面オーバーレイ・ライブラリー
CWBOBJ_KEY_FTOVRLAY - 前面オーバーレイ名
CWBOBJ_KEY_FTOVL_ACR - 前面オーバーレイ・オフセット (横方向)
CWBOBJ_KEY_FTOVL_DWN - 前面オーバーレイ・オフセット (下方向)
CWBOBJ_KEY_CHAR_ID - このファイルに対するグラフィック文字セット
CWBOBJ_KEY_JUSTIFY - ハードウェアの位置合わせ
CWBOBJ_KEY_HOLD - スプール・ファイルの保留
CWBOBJ_KEY_LPI - 1 インチ当たりの行数
CWBOBJ_KEY_MAXRCDS - 許容最大レコード数
CWBOBJ_KEY_OUTPTY - 出力優先順位
CWBOBJ_KEY_OUTQUELIB - 出力待ち行列ライブラリー名
CWBOBJ_KEY_OUTQUE - 出力待ち行列
CWBOBJ_KEY_OVERFLOW - オーバーフロー行番号
CWBOBJ_KEY_LINES_PAGE - ページ当たり行数でのページ長
CWBOBJ_KEY_PAGELEN - 測定単位でのページ長
CWBOBJ_KEY_MEASMETHOD - 測定方法 (*ROWCOL または *UOM)
CWBOBJ_KEY_CHAR_LINE - 行当たり文字数でのページ幅
CWBOBJ_KEY_PAGEWIDTH - 測定単位でのページ幅
CWBOBJ_KEY_MULTIUP - 面当たりの論理ページ数
CWBOBJ_KEY_POINTSIZE - デフォルトのフォントのポイント・サイズ
CWBOBJ_KEY_FIDELITY - 印刷精度
CWBOBJ_KEY_DUPLEX - 両面印刷
CWBOBJ_KEY_PRTQUALITY - 印刷品質
CWBOBJ_KEY_PRTTEXT - 各ページの下部に印刷されたテキスト
CWBOBJ_KEY_PRINTER - プリンター名
CWBOBJ_KEY_PRTDEVTYPE - プリンター・タイプ (データ・ストリーム・タイプ)
CWBOBJ_KEY_PRTRFILELIB - プリンター・ファイル・ライブラリー
CWBOBJ_KEY_PRTRFILE - プリンター・ファイル
CWBOBJ_KEY_RPLUNPRT - 印刷不能文字の置き換え
CWBOBJ_KEY_RPLCHAR - 印刷不能文字の置き換え文字

```
CWBOBJ_KEY_SAVE         - 印刷後のスプール・ファイルの保管
CWBOBJ_KEY_SRCDRWR      - 用紙入れ
CWBOBJ_KEY_SPOOL        - データのスプール
CWBOBJ_KEY_SCHEDULE     - スプール・ファイルのスケジュール
CWBOBJ_KEY_STARTPAGE    - 印刷開始ページ
CWBOBJ_KEY_DESCRIPTION  - テキスト記述
CWBOBJ_KEY_UNITOFMEAS   - 測定単位
CWBOBJ_KEY_USERDATA     - ユーザー・データ
CWBOBJ_KEY_USRDFNDTA    - ユーザー定義データ
CWBOBJ_KEY_USRDFNOPTS   - ユーザー定義オプション
CWBOBJ_KEY_USRDFNOBJLIB - ユーザー定義オブジェクト・ライブラリー
CWBOBJ_KEY_USRDFNOBJ    - ユーザー定義オブジェクト
CWBOBJ_KEY_USRDFNOBJTYP - ユーザー定義オブジェクト・タイプ
```

- CWBOBJ_LIST_WTR:

```
CWBOBJ_KEY_WRITER       - 書き出しプログラム・ジョブ名
CWBOBJ_KEY_WTRJOBNUM    - 書き出しプログラム・ジョブ番号
CWBOBJ_KEY_WTRJOBSTS    - 書き出しプログラム・ジョブ状況
CWBOBJ_KEY_WTRJOBUSER   - 書き出しプログラム・ジョブ・ユーザー
```

- CWBOBJ_LIST_LIB:

```
CWBOBJ_KEY_LIBRARY      - ライブラリー名
CWBOBJ_KEY_DESCRIPTION  - ライブラリーの記述
```

- CWBOBJ_LIST_RSC:

```
CWBOBJ_KEY_RSCNAME      - 資源名
CWBOBJ_KEY_RSCLIB       - 資源ライブラリー
CWBOBJ_KEY_RSCTYPE      - 資源オブジェクト・タイプ
CWBOBJ_KEY_OBJEXTATTR   - オブジェクトの拡張属性
CWBOBJ_KEY_DESCRIPTION  - 資源の記述
CWBOBJ_KEY_DATE         - オブジェクトの最終変更日付
CWBOBJ_KEY_TIME         - オブジェクトの最終変更時刻
```

cwbOBJ_GetObjAttrs:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトのいくつかの属性を取得します。

構文

```
unsigned int CWB_ENTRY cwbOBJ_GetObjAttrs(
                            cwbOBJ_ObjHandle        objectHandle,
                            unsigned long           numAttrs,
                            cwbOBJ_GetObjAttrParms *getAttrParms,
                            cwbSV_ErrHandle         errorHandle);
```

パラメーター

cwbOBJ_ObjHandle objectHandle - input
属性を取得するオブジェクトのハンドル。

unsigned long numAttrs - input
検索する属性の数。

cwbOBJ_GetObjAttrParms *getAttrParms - input
検索する属性ごとに、属性キー (id)、その属性の値を保管するバッファー、およびそのバッファーのサイズを与える、numAttrs の要素からなる配列。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎます。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_KEY
キーが有効ではありません。

CWB_API_ERROR
一般 API 障害。

使用法

各種オブジェクト・タイプに対して有効な属性を調べるには、cwbOBJ_GetObjAttr の使用上の注意を参照してください。

cwbOBJ_GetObjHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

リスト・オブジェクトを取得します。この呼び出しは、オープンされたリスト中のオブジェクトのハンドルを取得します。資源を解放するために呼び出し側が cwbOBJ_DeleteObjHandle を使用したときは、戻されたハンドルもこれによって解放されなければなりません。戻されたハンドルは、リストがオープンされている間だけ有効です。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_GetObjHandle(
                              cwbOBJ_ListHandle  listHandle,
                              unsigned long      ulPosition,
                              cwbOBJ_ObjHandle  *objectHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ListHandle listHandle - input
オブジェクト・ハンドルを取得するその元のリストのハンドル。このリストはオープンされていなければなりません。

unsigned long ulPosition - input
ハンドルを取得するオブジェクトのリスト内の位置。0 が基準になります。0 から「リストのオブジェクト数 - 1」までが有効な値です。cwbOBJ_GetListSize() を使用して、リストのサイズを取得することができます。

cwbOBJ_ObjHandle *objectHandle - output
出力の際に、オブジェクトのハンドルが入ります。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたリスト・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_LIST_NOT_OPEN
リストがオープンされていません。

CWBOBJ_RC_INVALID_INDEX
ulPosition が範囲外です。

使用法

なし (None)

cwbOBJ_GetObjHandleFromID:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクト・ハンドルを、その 2 進数 ID およびタイプから再生成します。このオブジェクト・ハンドルの使用を終了したときは、資源を解放するため cwbOBJ_DeleteObjHandle() を呼び出す必要があります。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_GetObjHandleFromID(
                                void              *idBuffer,
                                unsigned long      bufLen,
                                cwbOBJ_ObjType     objectType,
                                cwbOBJ_ObjHandle  *objectHandle,
                                cwbSV_ErrHandle    errorHandle);
```

パラメーター

void *idBuffer - input
このオブジェクトの ID を保持するバッファー。

unsigned long bufLen - input
*IDBuffer が指すデータの長さ。

cwbOBJ_ObjType type - input
この ID 用のオブジェクトのタイプ。これは、この ID を提供したオブジェクトのタイプと一致する必要があります。

cwbOBJ_ObjHandle *objectHandle - output
この呼び出しが正常に戻った場合は、これがオブジェクトのハンドルとなります。このオブジェクト・ハンドルの使用を終了したときは、cwbOBJ_DeleteObjHandle() API を使用してこのハンドルを解放してください。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWBOBJ_RC_INVALID_TYPE
objectType が正しくありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

なし (None)

cwbOBJ_GetObjID:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトの ID を取得します。これは、サーバー上でオブジェクトを固有に識別するデータです。取得されるデータは読み取り不能の 2 進数です。このデータは、ハンドルをそのオブジェクトに取り戻すために、cwbOBJ_GetObjHandleFromID() API で返すことができます。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_GetObjID(
                              cwbOBJ_ObjHandle  objectHandle,
                              void             *idBuffer,
                              unsigned long     bufLen,
                              unsigned long    *bytesNeeded,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle objectHandle - input
ID の取得元のオブジェクトのハンドル。

void *idBuffer - output
このオブジェクトの ID を保持するバッファー。

unsigned long bufLen - input
*IDBuffer が指すバッファーの長さ。

unsigned long *bytesNeeded - output
出力の際には、ID を保持するために必要なバイト数が入ります。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎます。

使用法

なし (None)

cwbOBJ_RefreshObj:

これは、System i Access for Windows 製品で使用する API です。

目的

最新の System i 情報でオブジェクトをリフレッシュします。これにより、オブジェクトの戻された属性を最新のものにします。

構文

```
unsigned int CWB_ENTRY cwbOBJ_RefreshObj(
                            cwbOBJ_ObjHandle  objectHandle,
                            cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle objectHandle - input
リフレッシュされるオブジェクトのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

以下のオブジェクト・タイプをリフレッシュすることができます。

- CWBOBJ_LIST_SPLF (スプール・ファイル)
- CWBOBJ_LIST_PRTF (プリンター・ファイル)
- CWBOBJ_LIST_OUTQ (出力待ち行列)
- CWBOBJ_LIST_PRTD (プリンター)
- CWBOBJ_LIST_WTR (書き出しプログラム)

例: リストの中に少なくとも 1 つの項目があるスプール・ファイル・リストを listHandle が指すものと想定します。

```
cwbOBJ_ObjHandle splFileHandle;
ulRC = cwbOBJ_GetObjHandle(listHandle,
0,
&splFileHandle,
NULL);
if (ulRC == CWB_NO_ERROR)
{
  ulRC = cwbOBJ_RefreshObj(splFileHandle);
  .....
  get attributes for object
  .....
  ulRC = cwbOBJ_DeleteObjHandle(splFileHandle);
}
```

cwbOBJ_SetObjAttrs:

これは、System i Access for Windows 製品で使用する API です。

目的

サーバー上のオブジェクトの属性を変更します。

構文

```
unsigned int CWB_ENTRY cwbOBJ_SetObjAttrs(
                              cwbOBJ_ObjHandle   objectHandle,
                              cwbOBJ_ParmHandle  parmListHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ObjHandle objectHandle - input
変更されるオブジェクトへのハンドル。

cwbOBJ_ParmHandle parmListHandle - input
そのオブジェクト用に変更される属性が入っている、パラメーター・オブジェクトへのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

以下のオブジェクトによって、これらの属性を変更できるようになります。

- CWBOBJ_LIST_SPLF (スプール・ファイル):

```
CWBOBJ_KEY_ALIGN        - ページの位置合わせ
CWBOBJ_KEY_BKOVRLLIB    - 背面オーバーレイ・ライブラリー名
CWBOBJ_KEY_BKOVRLAY     - 背面オーバーレイ
CWBOBJ_KEY_BKOVL_ACR    - 背面オーバーレイ・オフセット (横方向)
CWBOBJ_KEY_BKOVL_DWN    - 背面オーバーレイ・オフセット (下方向)
CWBOBJ_KEY_COPIES       - コピー枚数
CWBOBJ_KEY_ENDPAGE      - 終了ページ
CWBOBJ_KEY_FILESEP      - ファイル区切り
CWBOBJ_KEY_FORMFEED     - 用紙送り
CWBOBJ_KEY_FORMTYPE     - 用紙タイプ
CWBOBJ_KEY_FTOVRLLIB    - 前面オーバーレイ・ライブラリー名
CWBOBJ_KEY_FTOVRLAY     - 前面オーバーレイ
CWBOBJ_KEY_FTOVL_ACR    - 前面オーバーレイ・オフセット (横方向)
CWBOBJ_KEY_FTOVL_DWN    - 前面オーバーレイ・オフセット (下方向)
CWBOBJ_KEY_OUTPTY       - 出力優先順位
CWBOBJ_KEY_OUTQUELIB    - 出力待ち行列ライブラリー名
CWBOBJ_KEY_OUTQUE       - 出力待ち行列
CWBOBJ_KEY_MULTIUP      - 片面当たりの論理ページ数
CWBOBJ_KEY_FIDELITY     - 印刷精度
CWBOBJ_KEY_DUPLEX       - 両面印刷
CWBOBJ_KEY_PRTQUALITY   - 印刷品質
CWBOBJ_KEY_PRTSEQUENCE  - P 印刷順序
CWBOBJ_KEY_PRINTER      - プリンター
CWBOBJ_KEY_RESTART      - 印刷再始動位置
CWBOBJ_KEY_SAVESPLF     - 印刷後のスプール・ファイルの保管
CWBOBJ_KEY_SCHEDULE     - スプール・ファイルのスケジュール
CWBOBJ_KEY_STARTPAGE    - 開始ページ
CWBOBJ_KEY_USERDATA     - ユーザー・データ
CWBOBJ_KEY_USRDFNDTA    - ユーザー定義データ
CWBOBJ_KEY_USRDFNOPTS   - ユーザー定義オプション
CWBOBJ_KEY_USRDFNOBJLIB - ユーザー定義オブジェクト・ライブラリー
CWBOBJ_KEY_USRDFNOBJ    - ユーザー定義オブジェクト
CWBOBJ_KEY_USRDFNOBJTYP - ユーザー定義オブジェクト・タイプ
```

- CWBOBJ_LIST_PRTF (プリンター・ファイル):

```
CWBOBJ_KEY_ALIGN        - ページの位置合わせ
CWBOBJ_KEY_BKMGN_ACR    - バック・マージン・オフセット (横方向)
CWBOBJ_KEY_BKMGN_DWN    - バック・マージン・オフセット (下方向)
CWBOBJ_KEY_BKOVRLLIB    - 背面オーバーレイ・ライブラリー名
CWBOBJ_KEY_BKOVRLAY     - 背面オーバーレイ
CWBOBJ_KEY_BKOVL_ACR    - 背面オーバーレイ・オフセット (横方向)
CWBOBJ_KEY_BKOVL_DWN    - 背面オーバーレイ・オフセット (下方向)
CWBOBJ_KEY_CPI          - 1 インチ当たりの文字数
CWBOBJ_KEY_CODEPAGE     - コード・ページ
CWBOBJ_KEY_CODEDFNTLIB  - コード化フォント・ライブラリー名
CWBOBJ_KEY_CODEDFNT     - コード化フォント名
CWBOBJ_KEY_COPIES       - コピー枚数
CWBOBJ_KEY_DBCSDATA     - DBCS データを含む
CWBOBJ_KEY_DBCSEXTENSN  - DBCS 拡張文字の処理
CWBOBJ_KEY_DBCSROTATE   - DBCS 文字回転
CWBOBJ_KEY_DBCSCPI      - DBCS CPI
CWBOBJ_KEY_DBCSSISO     - DBCS SO/SI のスペース
CWBOBJ_KEY_DFR_WRITE    - 書き出し据え置き
CWBOBJ_KEY_ENDPAGE      - 終了ページ
CWBOBJ_KEY_FILESEP      - ファイル区切り (*FILE は使用不可)
CWBOBJ_KEY_FOLDREC      - レコードの折り返し
CWBOBJ_KEY_FONTID       - フォント識別コード
```

CWBOBJ_KEY_FORMFEED - 用紙送り
CWBOBJ_KEY_FORMTYPE - 用紙タイプ
CWBOBJ_KEY_FTMGN_ACR - フロント・マージン・オフセット (横方向)
CWBOBJ_KEY_FTMGN_DWN - フロント・マージン・オフセット (下方向)
CWBOBJ_KEY_FTOVRLLIB - 前面オーバーレイ・ライブラリー名
CWBOBJ_KEY_FTOVRLAY - 前面オーバーレイ
CWBOBJ_KEY_FTOVL_ACR - 前面オーバーレイ・オフセット (横方向)
CWBOBJ_KEY_FTOVL_DWN - 前面オーバーレイ・オフセット (下方向)
CWBOBJ_KEY_CHAR_ID - グラフィック文字セット ID
CWBOBJ_KEY_JUSTIFY - ハードウェアの位置合わせ
CWBOBJ_KEY_HOLD - スプール・ファイルの保留
CWBOBJ_KEY_LPI - 1 インチ当たりの行数
CWBOBJ_KEY_MAXRECORDS - スプール出力レコードの最大数
CWBOBJ_KEY_OUTPTY - 出力優先順位
CWBOBJ_KEY_OUTQUELIB - 出力待ち行列ライブラリー名
CWBOBJ_KEY_OUTQUE - 出力待ち行列
CWBOBJ_KEY_OVERFLOW - オーバーフロー行番号
CWBOBJ_KEY_PAGELEN - ページ長
CWBOBJ_KEY_MEASMETHOD - 測定方法
CWBOBJ_KEY_PAGEWIDTH - ページ幅
CWBOBJ_KEY_MULTIUP - 片面当たりの論理ページ数
CWBOBJ_KEY_POINTSIZE - デフォルトのフォントのポイント・サイズ
CWBOBJ_KEY_FIDELITY - 印刷精度
CWBOBJ_KEY_DUPLEX - 両面印刷
CWBOBJ_KEY_PRTQUALITY - 印刷品質
CWBOBJ_KEY_PRTTEXT - 印刷テキスト
CWBOBJ_KEY_PRINTER - プリンター
CWBOBJ_KEY_PRTDEVTYPE - プリンター・タイプ
CWBOBJ_KEY_RPLUNPRT - 印刷不能文字の置き換え
CWBOBJ_KEY_RPLCHAR - 置き換え文字
CWBOBJ_KEY_SAVESPLF - 印刷後のスプール・ファイルの保管
CWBOBJ_KEY_SRCDRWR - 用紙入れ
CWBOBJ_KEY_SPOOL - データのスプール
CWBOBJ_KEY_SCHEDULE - スプール・ファイルのスケジュール
CWBOBJ_KEY_STARTPAGE - 開始ページ
CWBOBJ_KEY_DESCRIPTION - テキスト記述
CWBOBJ_KEY_UNITOFMEAS - 測定単位
CWBOBJ_KEY_USERDATA - ユーザー・データ
CWBOBJ_KEY_USRDFNDTA - ユーザー定義データ
CWBOBJ_KEY_USRDFNOPTS - ユーザー定義オプション
CWBOBJ_KEY_USRDFNOBJLIB - ユーザー定義オブジェクト・ライブラリー
CWBOBJ_KEY_USRDFNOBJ - ユーザー定義オブジェクト
CWBOBJ_KEY_USRDFNOBJTYP - ユーザー定義オブジェクト・タイプ

- CWBOBJ_LIST_OUTQ (出力待ち行列):
- CWBOBJ_LIST_PRTD (プリンター):
- CWBOBJ_LIST_WTR (書き出しプログラム):
- CWBOBJ_LIST_LIB (ライブラリー):
  - なし

System i Access for Windows パラメーター・オブジェクト API

以下の System i Access for Windows API は、パラメーター・オブジェクトに関するものです。 API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_CopyParmObjHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

重複パラメーター・リスト・オブジェクトを作成します。パラメーター・リスト・オブジェクト中のすべての属性キーと属性値は、新しいパラメーター・リスト・オブジェクトにコピーされます。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_CopyParmObjHandle(
                              cwbOBJ_ParmHandle   parmListHandle,
                              cwbOBJ_ParmHandle  *newParmListHandle,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbOBJ_ParmHandle parmListHandle - input
コピーするパラメーター・リスト・オブジェクトのハンドル

cwbOBJ_ParmHandle *newParmListHandle - output
この呼び出しが正常に完了すると、このハンドルには新規のパラメーター・リスト・オブジェクト・ハンドルが入ります。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

使用法

この呼び出しで割り振られた資源を解放するため、cwbOBJ_DeleteParmObjectHandle API を呼び出す必要があります。

cwbOBJ_CreateParmObjHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

パラメーター・リスト・オブジェクト・ハンドルを割り振ります。パラメーター・リスト・オブジェクトは、他の API 上で渡すことのできるパラメーターのリストを保持するために使用できます。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_CreateParmObjHandle(
                              cwbOBJ_ParmHandle  *parmListHandle,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbOBJ_ParmHandle *parmListHandle - output
パラメーター・オブジェクトのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

使用法

この呼び出しで割り振られた資源を解放するため、cwbOBJ_DeleteParmObjectHandle API を呼び出す必要があります。

cwbOBJ_DeleteParmObjHandle:
目的

パラメーター・リスト・オブジェクト・ハンドルを割り振り解除し、このハンドルによって使用された資源を解放します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_DeleteParmObjHandle(
                              cwbOBJ_ParmHandle  parmListHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ParmHandle parmListHandle - input
パラメーター・オブジェクトのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、パラメーター・オブジェクト・ハンドルではありません。

使用法

この呼び出しが正常に戻った後は、parmListHandle は有効ではなくなります。

cwbOBJ_GetParameter:

これは、System i Access for Windows 製品で使用する API です。

目的

パラメーター・リスト・オブジェクトのパラメーターの値を取得します。

構文

```
unsigned int CWB_ENTRY cwbOBJ_GetParameter(
                            cwbOBJ_ParmHandle parmListHandle,
                            cwbOBJ_KeyID      key,
                            void             *buffer,
                            unsigned long     bufLen,
                            unsigned long    *bytesNeeded,
                            cwbOBJ_DataType  *keyType,
                            cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ParmHandle parmListHandle - input
パラメーター・オブジェクトのハンドル。

cwbOBJ_KeyID key - input
設定するパラメーターの ID。

void *buffer - output
属性値を保持するバッファー (この呼び出しが正常に戻った場合)。*Buffer に置かれるデータ・タイプは何かをキー値が決定します。タイプが与えられた場合、このタイプも *keyType パラメーターへ戻されます。

unsigned long bufLen - input
バッファーが指すバッファーの長さ。

unsigned long *bytesNeeded - output
出力の際には、結果を保持するために必要なバイト数が入ります。

cwbOBJ_DataType *keyType - output
オプションであり、NULL でも構いません。出力の際には、この属性と、*buffer に何が保管されるかを表すために使用されるデータ・タイプが含まれます。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎます。

CWBOBJ_RC_KEY_NOT_FOUND
キーがパラメーター・リストに指定されていません。

CWB_API_ERROR
一般 API 障害。

使用法

なし (None)

cwbOBJ_SetParameter:

これは、System i Access for Windows 製品で使用する API です。

目的

パラメーターの値をパラメーター・リスト・オブジェクトに設定します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_SetParameter(
                              cwbOBJ_ParmHandle  parmListHandle,
                              cwbOBJ_KeyID       key,
                              const void        *value,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ParmHandle parmListHandle - input
パラメーター・オブジェクトのハンドル。

cwbOBJ_KeyID key - input
設定するパラメーターの ID。

void *value - input
パラメーターに設定するその値。値が指すタイプは、キーの値によって決まります。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、パラメーター・オブジェクト・ハンドルではありません。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

なし (None)

System i Access for Windows 書き出しプログラム・ジョブ API

以下の System i Access for Windows API は、書き出しプログラム・ジョブに関するものです。 API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_EndWriter:

これは、System i Access for Windows 製品で使用する API です。

目的

System i 書き出しプログラム・ジョブを終了します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_EndWriter(
                              cwbOBJ_ObjHandle   writerHandle,
                              cwbOBJ_ParmHandle *parmListHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ObjHandle writerHandle - input
停止される書き出しプログラム・ジョブのハンドル。このハンドルは、書き出しプログラムをリストし

てそのリストから書き出しプログラム・ハンドルを取得するか、あるいは書き出しプログラムを開始して、書き出しプログラム・ハンドルが戻されるよう要求するかのいずれかによって獲得することができます。

cwbOBJ_ParmHandle *parmListHandle - input
オプションです。書き出しプログラムを終了させるためのパラメーターが入っている、有効なパラメーター・リスト・オブジェクト・ハンドルを指すポインター。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

この呼び出しが正常に戻った後、writerHandle を解放するため cwbOBJ_DeleteObjHandle() を呼び出してください。以下のパラメーター・キーを pParmListHandl オブジェクトに設定することができます。

- CWBOBJ_KEY_WTREND - 書き出しプログラムが終了する時間以下の特殊値のうちの 1 つ。
  - *CNTRLD - 現行ファイルの印刷後書き出しプログラムを終了させる
  - *IMMED - 書き出しプログラムを即刻終了させる
  - *PAGEEND - 現行ページの終わりで書き出しプログラムを終了させる

cwbOBJ_StartWriter:

これは、System i Access for Windows 製品で使用する API です。

目的

System i 書き出しプログラム・ジョブを開始します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_StartWriter(
                              cwbOBJ_ObjHandle   *printerHandle,
                              cwbOBJ_ObjHandle   *outputQueueHandle,
```

```
cwbOBJ_ParmHandle  *parmListHandle,
cwbOBJ_ObjHandle   *writerHandle,
cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbOBJ_ObjHandle *printerHandle - input
必須です。どのプリンターに対してこの書き出しプログラムを開始させるかを識別する有効なプリンター・オブジェクト・ハンドルを指すポインター。

cwbOBJ_ObjHandle *outputQueueHandle - input
オプションです。どの出力待ち行列からこの書き出しプログラムを開始させるかを識別する有効な出力待ち行列オブジェクト・ハンドルを指すポインター。parmListHandle もまた指定され、CWBOBJ_KEY_OUTQUE パラメーター・キーが入っている場合は、当パラメーターは無視されます。

cwbOBJ_ParmHandle *parmListHandle - input
オプションです。書き出しプログラムを開始させるためのパラメーターが入っている有効なパラメーター・リスト・オブジェクト・ハンドルを指すポインター。

cwbOBJ_ObjHandle *writerHandle - output
オプションです。この API から正常に戻ったときに埋められる書き出しプログラム・オブジェクト・ハンドルを指すポインター。このパラメーターが指定された場合は、この書き出しプログラム・ハンドル用に割り振られた資源を解放するために、呼び出し側は cwbOBJ_DeleteObjHandle() を呼び出さなければなりません。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは cwbSV_CreateErrHandle() API で作成されます。メッセージは cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

この API を呼び出すと、実行される書き出しプログラム・ジョブが投入されますが、この API が正常に戻っても書き出しプログラム・ジョブは開始できない場合があります (ジョブの投入は完了しましたが、ジョブを開始できません)。これは System i STRPRTWTR コマンドの動作です。次のパラメーター・キーを parmListHandle オブジェクト内に設定することができます。

```
CWBOBJ_KEY_ALIGN        - ページの位置合わせ
CWBOBJ_KEY_ALWDRTPRT    - 直接印刷可能
CWBOBJ_KEY_AUTOEND      - 書き出しプログラムの自動終了
CWBOBJ_KEY_DRWRSEP      - 区切り用紙入れ
CWBOBJ_KEY_FILESEP      - ファイル区切りの数
CWBOBJ_KEY_FORMTYPE     - 使用する用紙タイプ
CWBOBJ_KEY_JOBNAME      - ファイルを作成したジョブの名前
CWBOBJ_KEY_JOBNUMBER    - ファイルを作成したジョブの番号
CWBOBJ_KEY_USER         - ファイルを作成したユーザーの名前
CWBOBJ_KEY_FORMTYPEMSG  - 用紙タイプ・メッセージ・オプション
CWBOBJ_KEY_MSGQUELIB    - メッセージ待ち行列ライブラリー
CWBOBJ_KEY_MSGQUE       - メッセージ待ち行列
CWBOBJ_KEY_OUTQUELIB    - 出力待ち行列ライブラリー名
CWBOBJ_KEY_OUTQUE       - 出力待ち行列
CWBOBJ_KEY_SPOOLFILE    - スプール・ファイル名
CWBOBJ_KEY_SPLFNUM      - スプール・ファイル番号
CWBOBJ_KEY_WTRSTRPAGE   - 書き出しプログラムの開始ページ
CWBOBJ_KEY_WTREND       - 書き出しプログラムが終了する時間
CWBOBJ_KEY_WRITER       - 書き出しプログラム・ジョブ名
CWBOBJ_KEY_WTRINIT      - プリンターの初期設定
```

System i Access for Windows 出力待ち行列 API

以下の System i Access for Windows API は、出力待ち行列に関するものです。 API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_HoldOutputQueue:

これは、System i Access for Windows 製品で使用する API です。

目的

System i 出力待ち行列を保留にします。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_HoldOutputQueue(
                              cwbOBJ_ObjHandle  queueHandle,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle queueHandle - input
保留される出力待ち行列のハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効な待ち行列ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

なし (None)

cwbOBJ_PurgeOutputQueue:

これは、System i Access for Windows 製品で使用する API です。

目的

System i 出力待ち行列にあるスプール・ファイルを除去します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_PurgeOutputQueue(
                              cwbOBJ_ObjHandle    queueHandle,
                              cwbOBJ_ParmHandle  *parmListHandle,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbOBJ_ObjHandle queueHandle - input
除去される出力待ち行列のハンドル。

cwbOBJ_ParmHandle * parmListHandle - input
オプションです。出力待ち行列を除去するためのパラメーターが入っている、有効なパラメーター・リスト・オブジェクト・ハンドルを指すポインター。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

parmListHandle に指定されたパラメーターが与えられると、そのパラメーターはどのスプール・ファイルが除去されるかを指定します。parmListHandle が NULL の場合、現行ユーザーのスプール・ファイルはすべて除去されます。以下のパラメーター・キーを parmListHandle オブジェクトに設定できます。

- CWBOBJ_KEY_USER

  どのユーザーのスプール・ファイルを除去するかを指定します。特定のユーザー ID、″*ALL″ または ″*CURRENT″。″*CURRENT″ がデフォルト。

- CWBOBJ_KEY_FORMTYPE

  指定されている用紙タイプに基づいて、どのスプール・ファイルを除去するかを指定します。特定の用紙タイプ、″*ALL″ または ″*STD″。″*ALL″ がデフォルト。

- CWBOBJ_KEY_USERDATA

  指定されているユーザー・データに基づいて、どのスプール・ファイルを除去するかを指定します。特定の値、または ″*ALL″。″*ALL″ がデフォルト。

cwbOBJ_ReleaseOutputQueue:

これは、System i Access for Windows 製品で使用する API です。

目的

System i 出力待ち行列を解放します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_ReleaseOutputQueue(
                              cwbOBJ_ObjHandle  queueHandle,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle queueHandle - input
解放される出力待ち行列のハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効な待ち行列ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

なし (None)

System i Access for Windows AFP 資源 API

以下の System i access for Windows API は、AFP 資源に関するものです。 API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_CloseResource:

これは、System i Access for Windows 製品で使用する API です。

目的

読み取りのために以前オープンした AFP 資源オブジェクトをクローズします。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_CloseResource(
                              cwbOBJ_ObjHandle   resourceHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ObjHandle resourceHandle - input
クローズされる資源のハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが有効な資源ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_RSCNOTOPEN
資源がオープンされていません。

CWBOBJ_RC_SPLFNOTOPEN
スプール・ファイルがオープンされていません。

使用法

資源に対するハンドルが cwbOBJ_OpenResourceForSplF() API への呼び出しを経由して取得された場合、この API は、ユーザーに対するハンドルを削除します (ユーザーが資源をオープンしたとき、そのユーザーに対してハンドルが動的に割り振られ、この呼び出しはそのハンドルを割り振り解除します)。

cwbOBJ_CreateResourceHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

指定されたシステム上の特定の AFP 資源の資源ハンドルを作成します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_CreateResourceHandle(
                              const char             *systemName,
                              const char             *resourceName,
                              const char             *resourceLibrary,
                              cwbOBJ_AFPResourceType  resourceType,
                              cwbOBJ_ObjHandle       *objectHandle,
                              cwbSV_ErrHandle         errorHandle);
```

パラメーター

const char *systemName - input
ASCIIZ ストリングに入っているシステム名を指すポインター。

const char *resourceName - input
AFP 資源名を指すポインター。

const char *resourceLibrary - input
資源が入っている System i ライブラリーの名前を指すポインター。

cwbOBJ_AFPResourceType resourceType - input
どのタイプの資源であるかを指定します。下記のうちのいずれかでなければなりません。

- CWBOBJ_AFPRSC_FONT
- CWBOBJ_AFPRSC_FORMDEF
- CWBOBJ_AFPRSC_OVERLAY
- CWBOBJ_AFPRSC_PAGESEG
- CWBOBJ_AFPRSC_PAGEDEF

cwbOBJ_ObjHandle *objectHandle - output
出力の際は、資源ハンドルがこれに含まれます。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべて、このオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

名前ライブラリーと資源のタイプを知っている場合は、資源へのハンドルを取得するために、この API を使用してください。そのいずれも分からない場合、または、リストから選択したい場合は、代わりにリスト API を使用して AFP 資源をリストしてください。この API は、ホスト上の AFP 資源を検査しません。このハンドルが最初に資源についてのデータの検索に使用されるときに、資源ファイルが存在しないとホスト・エラーが起こります。

cwbOBJ_DisplayResource:

これは、System i Access for Windows 製品で使用する API です。

目的

指定された AFP 資源をユーザーに表示します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_DisplayResource(
                              cwbOBJ_ObjHandle     resourceHandle,
                              const char          *view,
                              const unsigned long  flags,
                              cwbSV_ErrHandle      errorHandle);
```

パラメーター

cwbOBJ_ObjHandle resourceHandle - input
AFP 資源オブジェクトのハンドル。このハンドルは、オーバーレイまたはページ・セグメントの資源のタイプである必要があります。

const char *view - input
オプションであり、NULL でも構いません。指定された場合、AFP ビューアーを呼び出す際に使用するビューを指定する、ASCIIZ ストリングを指すポインターです。ビューアーとともに出荷される 2 つの事前定義されたビューがあります。それらは、LETTER (8.5″ x 11″) と SFLVIEW (132 桁) です。ユーザー自身のビューを追加することもできます。

const unsigned long flags - input
以下のビットのいずれかが設定されることがあります。CWBOBJ_DSPSPLF_WAIT は、この呼び出しに対して、戻る前に、ビューアーのプロセスが資源を正常にオープンするまで待機するように伝えます。このビットが 0 の場合、この API は、ビューアーのプロセスを開始した後で戻ります。このビットが 1 の場合、この API は、戻る前に、ビューアーが資源をオープンするまで待機します。他のすべてのビットは、0 に設定される必要があります。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべて、このオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

CWB_NO_VIEWER
Client Access/400 のビューアー・サポートがインストールされていません。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

CWBOBJ_RC_INVALID_TYPE
resourceHandle 用に与えられたハンドルは、オーバーレイまたはページ・セグメントの資源へのハンドルではありません。

使用法

この API は、指定された AFP 資源で AFP ビューアーを呼び出すために使用してください。資源のタイプは、オーバーレイかページ・セグメントを指定する必要があります。戻りコード CWB_NO_VIEWER は、ビューアー構成要素がワークステーションにインストールされていなかったことを意味します。

cwbOBJ_OpenResource:

これは、System i Access for Windows 製品で使用する API です。

目的

読み取りのために AFP 資源オブジェクトをオープンします。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_OpenResource(
                              cwbOBJ_ObjHandle  resourceHandle,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle resourceHandle - input
読み取りのためオープンされる AFP 資源ファイルのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが有効な資源ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_NOHOSTSUPPORT
ホストは、資源を対象とする作業をサポートしません。

使用法

資源は、そこからの読み取りが終了した時点で、cwbOBJ_CloseResource() API を使用してクローズする必要があります。

cwbOBJ_OpenResourceForSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

読み取り用に既にオープンされたスプール・ファイルのために、読み取り用に AFP 資源オブジェクトをオープンします。AFP スプール・ファイルを読み取り中に読み取る必要のある外部 AFP 資源に接触した場合、この API は役立ちます。この API を使用すれば、最初に資源をリストすることなしに、その資源を読み取りのためにオープンすることができます。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_OpenResourceForSplF(
                              cwbOBJ_ObjHandle    splFHandle,
                              const char         *resourceName,
                              const char         *resourceLibrary,
                              unsigned long       resourceType,
                              const char         *reserved,
                              cwbOBJ_ObjHandle   *resourceHandle,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
読み取り用に既にオープンされており、そのための資源がオープンされるスプール・ファイルのハンドル。資源およびスプール・ファイルを読み取るために、同じシステム会話 (およびネットワーク印刷サーバー・プログラムの同じシステム・インスタンス) が使用されます。

const char *resourceName - input
ASCIIZ ストリングの AFP 資源名を指すポインター。

const char *resourceLibrary - input
オプションであり、NULL でも構いません。ASCIIZ ストリング内にある、AFP 資源の System i ライブラリーを指すポインター。ライブラリーが指定されていない場合は、資源を検索するためにスプール・ファイルのライブラリー・リストが使用されます。

unsigned long resourceType - input
以下のいずれかのビットがオンである無符号長精度整数。

- CWBOBJ_AFPRSC_FONT
- CWBOBJ_AFPRSC_FORMDEF
- CWBOBJ_AFPRSC_OVERLAY
- CWBOBJ_AFPRSC_PAGESEG
- CWBOBJ_AFPRSC_PAGEDEF

オープンする資源のタイプを指定します。

const char *reserved -
予約されています。NULL である必要があります。

cwbOBJ_OBJHandle *resourceHandle - output
資源の読み取り、シーク、および最後にクローズをするために使用できる、動的に割り振られた資源ハンドルが入る、正常に戻ってきた場合の OBJHandle を指すポインター。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_FILE_NOT_FOUND
資源が見付かりませんでした。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_INVALID_HANDLE
ハンドルが有効な資源ハンドルではありません。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_SPLFNOTOPEN
スプール・ファイルがオープンされていません。

CWBOBJ_RC_NOHOSTSUPPORT
ホストは、資源を対象とする作業をサポートしません。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

この呼び出しは、正常終了した場合は、一時的資源ハンドルを生成し、それを resourceHandle パラメーターに戻します。呼び出し側がこのハンドルを指定して cwbOBJ_CloseResource() API を呼び出した場合、このハンドルは自動的に削除されます。

資源は、そこからの読み取りが終了した時点で、cwbOBJ_CloseResource() API を使用してクローズする必要があります。

cwbOBJ_ReadResource:

これは、System i Access for Windows 製品で使用する API です。

目的

現行の読み取り位置からバイトを読み取ります。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_ReadResource(
                              cwbOBJ_ObjHandle  resourceHandle,
                              char             *bBuffer,
                              unsigned long     bytesToRead,
                              unsigned long    *bytesRead,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle resourceHandle - input
読み取られる AFP 資源オブジェクトのハンドル。

char *buffer - input
資源から読み取られるバイトを保持するバッファーを指すポインター。

unsigned long bytesToRead - input
読み取るバイトの最大数。読み取られる数はこれより少なくなります。

unsigned long *bytesRead - output
実際に読み取られたバイト数。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_RSCNOTOPEN
資源ファイルがまだオープンされていません。

CWBOBJ_RC_ENDOFFILE
ファイルの終わりが読み取られました。

使用法

この API を呼び出す前に、この資源ハンドルを使用して cwbOBJ_OpenResource() API を呼び出すか、または、cwbOBJ_OpenResourceForSplF() API への呼び出しを使用してハンドルを検索する必要があります。読み取り時にファイルの終わりに到達した場合、その戻りコードは CWBOBJ_RC_ENDOFFILE で、bytesRead には読み取られた実際のバイト数が入ります。

cwbOBJ_SeekResource:

これは、System i Access for Windows 製品で使用する API です。

目的

読み取りのためにオープンされている資源上の現行の読み取り位置を移動させます。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_SeekResource(
                              cwbOBJ_ObjHandle   resourceHandle,
```

```
cwbOBJ_SeekOrigin  seekOrigin,
signed long        seekOffset,
cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ObjHandle resourceHandle - input
シークされる AFP 資源オブジェクトのハンドル。

cwbOBJ_SeekOrigin seekOrigin - input
シークの際の開始位置。有効な値は以下のとおりです。

- CWBOBJ_SEEK_BEGINNING - ファイルの始めからシーク
- CWBOBJ_SEEK_CURRENT - 現行の読み取り位置からシーク
- CWBOBJ_SEEK_ENDING - ファイルの終わりからシーク

signed long seekOffset - input
現行の読み取りポインターを移動させるための、バイト表示によるシーク起点からのオフセット (負または正)。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_RSCNOTOPEN
資源がまだオープンされていません。

CWBOBJ_RC_SEEKOUTOFRANGE
シーク・オフセットが範囲外にあります。

使用法

この API を呼び出す前に、この資源ハンドルを使用して cwbOBJ_OpenResource() API を呼び出すか、または、cwbOBJ_OpenResourceForSplF() API への呼び出しを使用してハンドルを検索する必要があります。

System i Access for Windows 新規スプール・ファイル用 API

以下の System i Access for Windows API は、新規スプール・ファイルの処理に関するものです。API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_CloseNewSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

新しく作成されたスプール・ファイルをクローズします。

構文

```
unsigned int CWB_ENTRY cwbOBJ_CloseNewSplF(
                            cwbOBJ_ObjHandle   newSplFHandle,
                            cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ObjHandle newSplFHandle - input
新しいスプール・ファイルのハンドル。これは cwbOBJ_CreateNewSplF() API で返されるハンドルです。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

スプール・ファイルがクローズされると、それ以降スプール・ファイルに書き込むことはできません。

cwbOBJ_CloseNewSplFAndGetHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

新しく作成されたスプール・ファイルをクローズし、そのハンドルを戻します。

構文

```
unsigned int CWB_ENTRY cwbOBJ_CloseNewSplFAndGetHandle(
                              cwbOBJ_ObjHandle      newSplFHandle,
                              cwbOBJ_ObjHandle     *splFHandle,
                              cwbSV_ErrHandle       errorHandle);
```

パラメーター

cwbOBJ_ObjHandle newSplFHandle - input
新しいスプール・ファイルのハンドル。これは cwbOBJ_CreateNewSplF() API で返されるハンドルです。

cwbOBJ_ObjHandle *splFHandle - output
この呼び出しが正常に完了したときに、スプール・ファイル・ハンドルを保持するオブジェクト・ハンドルを指すポインター。このハンドルは、スプール・ファイル・ハンドルを入力として用いる他の API で使用することができます。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

splFHandle に戻されたハンドルは、資源を解放するために、cwbOBJ_DeleteObjHandle() API を使用して解放してください。

cwbOBJ_CreateNewSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

新規の System i スプール・ファイルを作成します。

構文

```
unsigned int CWB_ENTRY cwbOBJ_CreateNewSplF(
                              const char         *systemName,
                              cwbOBJ_ParmHandle  *parmListHandle,
                              cwbOBJ_ObjHandle   *printerFileHandle,
                              cwbOBJ_ObjHandle   *outputQueueHandle,
                              cwbOBJ_ObjHandle   *newSplFHandle,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

const char *systemName - input
ASCIIZ ストリングに入っているシステム名を指すポインター。

cwbOBJ_ParmHandle *parmListHandle - input
オプションです。スプール・ファイル作成のためのパラメーターを入れる、有効なパラメーター・リスト・オブジェクト・ハンドルを指すポインター。このリスト中のパラメーター群は、プリンター・ファイルおよび *outputQueueHandle パラメーターの中にあるものを変更します。

cwbOBJ_ObjHandle *printerFileHandle - input
オプションです。このスプール・ファイルの作成時に使用されるプリンター・ファイルを参照する、有効なプリンター・ファイル・オブジェクト・ハンドルを指すポインター。プリンター・ファイルは、スプール・ファイルを作成中の同じシステム上になければなりません。

cwbOBJ_ObjHandle *outputQueueHandle - input
オプションです。このスプール・ファイルが作成されるはずの出力待ち行列を参照する、有効な出力待ち行列オブジェクト・ハンドルを指すポインター。出力待ち行列は、このスプール・ファイルを作成中の同じシステム上になければなりません。出力待ち行列が *parmListHandle パラメーター (CWBOBJ_KEY_OUTQUELIB と CWBOBJ_KEY_OUTQUE) で設定されている場合、この出力待ち行列は、この出力待ち行列ハンドルによって指定される出力待ち行列を変更します。

cwbOBJ_ObjHandle *newSplFHandle - output
この呼び出しが正常に完了したときに、新しく作成されたスプール・ファイル・ハンドルで埋められるオブジェクト・ハンドルを指すポインター。新規のスプール・ファイルにデータを書き込み、それをクローズするには、このハンドルが必要です。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが無効です。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

parmListHandle が NULL の場合、または属性を指定しない場合、その属性は使用されるプリンター・ファイルから取られます。出力待ち行列を *parmListHandle とともに指定すると、この出力待ち行列は *outputQueueHandle パラメーターに指定されているものを変更します。出力待ち行列を指定しない (*parmListHandle になくて、outputQueueHandle が NULL) 場合、使用される出力待ち行列はプリンター・ファイルから取られます。プリンター・ファイルを指定しない (printerFileHandle が NULL) 場合、サーバーはデフォルトのネットワーク印刷のプリンター・ファイル、*LIBL/QNPSPRTF を使用します。次のパラメーター・キーを pParmListHandl オブジェクトに設定することができます。

```
CWBOBJ_KEY_ALIGN        - ページの位置合わせ
CWBOBJ_KEY_BKOVRLLIB    - 背面オーバーレイ・ライブラリー名
CWBOBJ_KEY_BKOVRLAY     - 背面オーバーレイ
CWBOBJ_KEY_BKOVL_ACR    - 背面オーバーレイ・オフセット (横方向)
CWBOBJ_KEY_BKOVL_DWN    - 背面オーバーレイ・オフセット (下方向)
CWBOBJ_KEY_CPI          - 1 インチ当たりの文字数
(1)CWBOBJ_KEY_CODEPAGE  - コード・ページ
CWBOBJ_KEY_COPIES       - コピー枚数
CWBOBJ_KEY_DBCSDATA     - DBCS データを含む
CWBOBJ_KEY_DBCSEXTENSN  - DBCS 拡張文字の処理
CWBOBJ_KEY_DBCSROTATE   - DBCS 文字回転
CWBOBJ_KEY_DBCSCPI      - DBCS CPI
CWBOBJ_KEY_DBCSSISO     - DBCS SO/SI のスペース
CWBOBJ_KEY_DFR_WRITE    - 書き出し据え置き
CWBOBJ_KEY_ENDPAGE      - 終了ページ
(2)CWBOBJ_KEY_FILESEP   - ファイル区切り
CWBOBJ_KEY_FOLDREC      - レコードの折り返し
CWBOBJ_KEY_FONTID       - フォント識別コード
CWBOBJ_KEY_FORMFEED     - 用紙送り
CWBOBJ_KEY_FORMTYPE     - 用紙タイプ
CWBOBJ_KEY_FTOVRLLIB    - 前面オーバーレイ・ライブラリー名
CWBOBJ_KEY_FTOVRLAY     - 前面オーバーレイ
CWBOBJ_KEY_FTOVL_ACR    - 前面オーバーレイ・オフセット (横方向)
CWBOBJ_KEY_FTOVL_DWN    - 前面オーバーレイ・オフセット (下方向)
(1)CWBOBJ_KEY_CHAR_ID   - グラフィック文字セット ID
CWBOBJ_KEY_JUSTIFY      - ハードウェアの位置合わせ
CWBOBJ_KEY_HOLD         - スプール・ファイルの保留
CWBOBJ_KEY_LPI          - 1 インチ当たりの行数
CWBOBJ_KEY_MAXRECORDS   - スプール出力レコードの最大数
CWBOBJ_KEY_OUTPTY       - 出力優先順位
CWBOBJ_KEY_OUTQUELIB    - 出力待ち行列ライブラリー名
CWBOBJ_KEY_OUTQUE       - 出力待ち行列
CWBOBJ_KEY_OVERFLOW     - オーバーフロー行番号
CWBOBJ_KEY_PAGELEN      - ページ長
CWBOBJ_KEY_MEASMETHOD   - 測定方法
CWBOBJ_KEY_PAGEWIDTH    - ページ幅
CWBOBJ_KEY_MULTIUP      - 片面当たりの論理ページ数
CWBOBJ_KEY_POINTSIZE    - デフォルトのフォントのポイント・サイズ
CWBOBJ_KEY_FIDELITY     - 印刷精度
CWBOBJ_KEY_DUPLEX       - 両面印刷
CWBOBJ_KEY_PRTQUALITY   - 印刷品質
CWBOBJ_KEY_PRTTEXT      - 印刷テキスト
CWBOBJ_KEY_PRINTER      - プリンター名
```

```
CWBOBJ_KEY_PRTDEVTYPE   - プリンター・タイプ
CWBOBJ_KEY_RPLUNPRT     - 印刷不能文字の置き換え
CWBOBJ_KEY_RPLCHAR      - 置き換え文字
CWBOBJ_KEY_SAVESPLF     - 印刷後のスプール・ファイルの保管
CWBOBJ_KEY_SRCDRWR      - 用紙入れ
CWBOBJ_KEY_SPOOL        - データのスプール
CWBOBJ_KEY_SPOOLFILE    - スプール・ファイル名
CWBOBJ_KEY_SCHEDULE     - スプール・ファイルのスケジュール
CWBOBJ_KEY_STARTPAGE    - 開始ページ
CWBOBJ_KEY_UNITOFMEAS   - 測定単位
CWBOBJ_KEY_USERCMT      - ユーザーの注釈 (100 文字)
CWBOBJ_KEY_USERDATA     - ユーザー・データ (10 文字)
CWBOBJ_KEY_SPLSCS       - スプール SCS データ
CWBOBJ_KEY_USRDFNDTA    - ユーザー定義データ
(3)CWBOBJ_KEY_USRDFNOPTS- ユーザー定義オプション
CWBOBJ_KEY_USRDFNOBJLIB - ユーザー定義オブジェクト・ライブラリー
CWBOBJ_KEY_USRDFNOBJ    - ユーザー定義オブジェクト
CWBOBJ_KEY_USRDFNOBJTYP - ユーザー定義オブジェクト・タイプ
```

注:

1. コード・ページとグラフィック文字セットは相互に依存しています。これらのうちの一方を指定すると、他方も指定する必要があります。
2. この属性を使用して新規のスプール・ファイルを作成する場合は、特殊値 *FILE は使用できません。
3. 最大 4 つまでのユーザー定義オプションを指定することができます。

cwbOBJ_GetSplFHandleFromNewSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

新規のスプール・ファイル・ハンドルを使用して、スプール・ファイル・ハンドルを生成します。自動データ・タイプ付けを使用して作成された新規スプール・ファイル上でこの API を使用する方法については、以下の注意を参照してください。

構文

```
unsigned int CWB_ENTRY cwbOBJ_GetSplFHandleFromNewSplF(
                            cwbOBJ_ObjHandle    newSplFHandle,
                            cwbOBJ_ObjHandle   *splFHandle,
                            cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbOBJ_ObjHandle newSplFHandle - input
新しいスプール・ファイルのハンドル。これは cwbOBJ_CreateNewSplF() API で返されるハンドルです。

cwbOBJ_ObjHandle *splFHandle - output
この呼び出しが正常に完了したときに、スプール・ファイル・ハンドルを保持するオブジェクト・ハンドルを指すポインター。このハンドルは、スプール・ファイル・ハンドルを入力として用いる他の API で使用することができます。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれま

す。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_SPLFNOTOPEN
まだホスト上にスプール・ファイルが作成されていません。

使用法

splFHandle に戻されたハンドルは、資源を解放するために、cwbOBJ_DeleteObjHandle() API を使用して解放してください。

スプール・ファイルに自動データ・タイプ付け (CWBOBJ_KEY_PRTDEVTYPE の属性が *AUTO に設定されているか、cwbOBJ_CreateNewSplF() API 上に設定されていない) を使用している場合、データのタイプ (*SCS、*AFPDS または *USERASCII) を判別するため、十分なデータがスプール・ファイルに書き込まれるまでスプール・ファイルの作成が遅らされます。この API を呼び出す際に、新規スプール・ファイルがこの状態にある場合、戻りコードは CWBOBJ_RC_SPLFNOTOPEN になります。

cwbOBJ_WriteNewSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

データを新規作成のスプール・ファイルに書き込みます。

構文

```
unsigned int CWB_ENTRY cwbOBJ_WriteNewSplF(
                            cwbOBJ_ObjHandle  newSplFHandle,
                            const char       *data,
                            unsigned long     dataLen,
                            cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle newSplFHandle - input
新しいスプール・ファイルのハンドル。これは cwbOBJ_CreateNewSplF() API で返されるハンドルです。

const char *data - input
スプール・ファイルに書き込まれるデータ・バッファーを指すポインター。

unsigned long ulDataLen - input
書き込まれるデータの長さ。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

なし (None)

System i Access for Windows 用スプール・ファイルの読み取り API

以下の System i Access for Windows API は、スプール・ファイルの読み取りに関するものです。 API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_CloseSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

読み取り用にオープンされている System i スプール・ファイルをクローズします。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_CloseSplF(
                              cwbOBJ_ObjHandle  splFHandle,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
シークされるスプール・ファイルのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

なし (None)

cwbOBJ_OpenSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

System i スプール・ファイルを読み取り用にオープンします。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_OpenSplF(
                              cwbOBJ_ObjHandle  splFHandle,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
読み取りのためオープンされるスプール・ファイルのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

スプール・ファイルからの読み取りを終了した際には、cwbOBJ_CloseSplF() API を使用してスプール・ファイルをクローズしてください。

cwbOBJ_ReadSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

現行の読み取り位置からバイトを読み取ります。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_ReadSplF(
                              cwbOBJ_ObjHandle  splFHandle,
                              char             *bBuffer,
                              unsigned long     bytesToRead,
                              unsigned long    *bytesRead,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
読み取られるスプール・ファイルのハンドル。

char *buffer - input
スプール・ファイルから読み取られるバイトを保持するバッファーを指すポインター。

unsigned long bytesToRead - input
読み取るバイトの最大数。読み取られる数はこれより少なくなります。

unsigned long *bytesRead - output
実際に読み取られたバイト数。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_SPLFNOTOPEN
スプール・ファイルがまだオープンされていません。

CWBOBJ_RC_SPLFENDOFFILE
ファイルの終わりが読み取られました。

使用法

この API を呼び出す前に、このスプール・ファイル・ハンドルを使用して cwbOBJ_OpenSplF() API を呼び出す必要があります。読み取り時にファイルの終わりに到達した場合、その戻りコードは CWBOBJ_SPLF_ENDOFFILE で、bytesRead には読み取られた実際のバイト数が入ります。

cwbOBJ_SeekSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

読み取りのためにオープンされているスプール・ファイル上の現行の読み取り位置を移動させます。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_SeekSplF(
                              cwbOBJ_ObjHandle   splFHandle,
                              cwbOBJ_SeekOrigin  seekOrigin,
                              signed long        seekOffset,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
シークされるスプール・ファイルのハンドル。

cwbOBJ_SeekOrigin seekOrigin - input
シークの際の開始位置。有効な値は以下のとおりです。

- CWBOBJ_SEEK_BEGINNING - ファイルの始めからシーク
- CWBOBJ_SEEK_CURRENT - 現行の読み取り位置からシーク
- CWBOBJ_SEEK_ENDING - ファイルの終わりからシーク

signed long seekOffset - input
現行の読み取りポインターを移動させるための、バイト表示によるシーク起点からのオフセット (負または正)。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれま

す。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_SPLFNOTOPEN
スプール・ファイルがまだオープンされていません。

CWBOBJ_RC_SEEKOUTOFRANGE
シーク・オフセットが範囲外にあります。

使用法

この API を呼び出す前に、このスプール・ファイル・ハンドルを使用して cwbOBJ_OpenSplF() API を呼び出す必要があります。

System i Access for Windows 用スプール・ファイルの操作 API

以下の System i Access for Windows API は、スプール・ファイルの操作に関するものです。API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_CallExitPgmForSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

System i Access のネットワーク印刷サーバー・プログラム QNPSERVR に対して、このスプール・ファイルの ID とアプリケーションが指定したデータとを、パラメーターとして渡して、出口プログラムの連鎖を呼び出すように指示します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_CallExitPgmForSplF(
                              cwbOBJ_ObjHandle    splFHandle,
                              void               *data,
                              unsigned long       dataLen,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
出口プログラムにパラメーターとして渡されるスプール・ファイルのハンドル。

void *data - input
出口プログラムに渡されるデータのブロックを指すポインター。このデータの形式は出口プログラム特有のものです。

unsigned long dataLen - input
pData が指すデータの長さ。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_TYPE
ハンドルが、スプール・ファイル・ハンドルではありません。

CWBOBJ_RC_NO_EXIT_PGM
出口プログラムがネットワーク印刷サーバーに登録されていません。

使用法

これは、クライアント・プログラムが、スプール・ファイルの処理を実行するためにそのサーバー部分と通信するための 1 つの手段です。QNPSERVR プログラムによって登録された System i 出口プログラムが、すべて呼び出されます。そのため、出口プログラムが認識できるように、*data 内のデータ形式の構成を行うのは、クライアント・プログラムと出口プログラムの役割になります。 QNPSERVR サーバー・プログラムと出口プログラムの間のインターフェースに関する情報については、System i の印刷に関するプログラミングの手引きを参照してください。

cwbOBJ_CreateSplFHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

指定されたシステム上の特定のスプール・ファイルについて、スプール・ファイル・ハンドルを作成します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_CreateSplFHandle(
                              const char          *systemName,
                              const char          *jobName,
                              const char          *jobNumber,
                              const char          *jobUser,
                              const char          *splFName,
                              const unsigned long  splFNumber,
                              cwbOBJ_ObjHandle    *objectHandle,
                              cwbSV_ErrHandle      errorHandle);
```

パラメーター

const char *systemName - input
ASCIIZ ストリングに入っているシステム名を指すポインター。

const char *jobName - input
ASCIIZ ストリング内の、スプール・ファイルを作成した System i ジョブの名前を指すポインター。

const char *jobNumber - input
ASCIIZ ストリング内の、スプール・ファイルを作成した System i ジョブの番号を指すポインター。

const char *jobNumber - input
ASCIIZ ストリング内の、スプール・ファイルを作成した System i ジョブのユーザーを指すポインター。

const char *splFName - input
ASCIIZ ストリング内の、スプール・ファイルの名前を指すポインター。

const unsigned long splFNumber - input
スプール・ファイルの番号

cwbOBJ_ObjHandle *objectHandle - output
出力の際は、スプール・ファイル・ハンドルがこれに含まれます。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

この API は、ホスト上のスプール・ファイルを検査しません。このハンドルが最初にスプール・ファイルのデータ検索に使用されるときに、スプール・ファイルが存在しないとホスト・エラーが起こります。

cwbOBJ_CreateSplFHandleEx:

これは、System i Access for Windows 製品で使用する API です。

目的

指定されたシステム上の特定のスプール・ファイルについて、スプール・ファイル・ハンドルを作成します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_CreateSplFHandleEx(
                              const char          *systemName,
                              const char          *jobName,
                              const char          *jobNumber,
                              const char          *jobUser,
                              const char          *splFName,
                              const unsigned long  splFNumber,
                              const char          *createdSystem,
                              const char          *createdDate,
                              const char          *createdTime,
                              cwbOBJ_ObjHandle    *objectHandle,
                              cwbSV_ErrHandle      errorHandle);
```

パラメーター

const char *systemName - input
ASCIIZ ストリングに入っているシステム名を指すポインター。

const char *jobName - input
ASCIIZ ストリング内の、スプール・ファイルを作成した System i ジョブの名前を指すポインター。

const char *jobNumber - input
ASCIIZ ストリング内の、スプール・ファイルを作成した System i ジョブの番号を指すポインター。

const char *jobNumber - input
ASCIIZ ストリング内の、スプール・ファイルを作成した System i ジョブのユーザーを指すポインター。

const char *splFName - input
ASCIIZ ストリング内の、スプール・ファイルの名前を指すポインター。

const unsigned long splFNumber - input
スプール・ファイルの番号

const char *createdSystem - input
ASCIIZ ストリング内の、スプール・ファイルが作成されたシステムの名前を指すポインター。

const char *createdDate - input
ASCIIZ ストリング内の、スプール・ファイルが作成された日付を指すポインター。

const char *createdTime - input
ASCIIZ ストリング内の、スプール・ファイルが作成された時刻を指すポインター。

cwbOBJ_ObjHandle *objectHandle - output
出力の際は、スプール・ファイル・ハンドルがこれに含まれます。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

この API は、ホスト上のスプール・ファイルを検査しません。このハンドルが最初にスプール・ファイルのデータ検索に使用されるときに、スプール・ファイルが存在しないとホスト・エラーが起こります。

cwbOBJ_DeleteSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

System i スプール・ファイルを削除します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_DeleteSplF(
                              cwbOBJ_ObjHandle  splFHandle,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
削除されるスプール・ファイルのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
無効なハンドル。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_TYPE
ハンドルが、スプール・ファイル・ハンドルではありません。

使用法

この呼び出しが正常に戻った後、splFHandle を解放するため cwbOBJ_DeleteObjHandle() を呼び出してください。

cwbOBJ_DisplaySplF:

これは、System i Access for Windows 製品で使用する API です。

目的

指定されたスプール・ファイルをユーザーに表示します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_DisplaySplF(
                              cwbOBJ_ObjHandle     splFHandle,
                              const char          *view,
                              const unsigned long  flags,
                              cwbSV_ErrHandle      errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
パラメーター・オブジェクトのハンドル。

const char *view - input
オプションであり、NULL でも構いません。指定された場合、スプール・ファイル・ビューアーを呼び出す際に使用するビューを指定する、ASCIIZ ストリングを指すポインターです。ビューアーとともに出荷される 2 つの事前定義されたビューがあります。

1. LETTER (8.5″ x 11″)

2. SFLVIEW (132 桁)

ユーザー自身のビューを追加することもできます。

const unsigned long flags - input
以下のビットのいずれかが設定されることがあります。CWBOBJ_DSPSPLF_WAIT - この呼び出しに対して、戻る前に、ビューアーのプロセスがスプール・ファイルを正常にオープンするまで待機するように伝えます。このビットが 0 の場合、この API は、ビューアーのプロセスを開始した後で戻ります。このビットが 1 の場合、この API は、戻る前に、ビューアーがスプール・ファイルをオープンするまで待機します。他のすべてのビットは、0 に設定される必要があります。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

CWB_NO_VIEWER
Client Access/400 のビューアー・サポートがインストールされていません。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

この API は、指定されたスプール・ファイルに AFP ビューアーを呼び出すために使用してください。AFP ビューアーは、AFP データ、SCS データ、および ASCII のプレーン・テキスト・データを表示することができます。戻りコード CWB_NO_VIEWER は、ビューアー構成要素がワークステーションにインストールされていなかったことを意味します。

cwbOBJ_HoldSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

スプール・ファイルを保留します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_HoldSplF(
                              cwbOBJ_ObjHandle    splFHandle,
                              cwbOBJ_ParmHandle  *parmListHandle,
                              cwbSV_ErrHandle     errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
保留されるスプール・ファイルのハンドル。

cwbOBJ_ParmHandle *parmListHandle - input
オプションです。スプール・ファイルを保留するためのパラメーターを含む、有効なパラメーター・リスト・オブジェクト・ハンドルを指すポインター。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_TYPE
ハンドルが、スプール・ファイル・ハンドルではありません。

使用法

以下のパラメーター・キーが parmListHandle オブジェクトに設定できます。

- CWBOBJ_KEY_HOLDTYPE

  実行する保留のタイプを指定します。″*IMMED″ または ″*PAGEEND″ で、″*IMMED″ がデフォルト。

cwbOBJ_IsViewerAvailable:

これは、System i Access for Windows 製品で使用する API です。

目的

スプール・ファイル・ビューアーが使用可能かどうかを検査します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_IsViewerAvailable(
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了 (ビューアーはインストールされています)。

CWB_NO_VIEWER
ビューアーがインストールされていません。

使用法

ワークステーションにビューアーが存在するかどうかをテストするには、この関数を使用してください。ビューアーがインストールされている場合、この関数は CWB_OK を戻します。ビューアーが使用不可の場合、この関数は CWB_NO_VIEWER を戻し、errorHandle パラメーター (与えられている場合) には適切なエラー・メッセージが入ります。この関数を使用すると、アプリケーションは cwbOBJ_DisplaySplF() API を呼び出すことなしに、ビューアー・サポートについて検査することができます。

cwbOBJ_MoveSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

System i スプール・ファイルを、別の出力待ち行列、または同じ出力待ち行列の別の位置に移動します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_MoveSplF(
                              cwbOBJ_ObjHandle   splFHandle,
                              cwbOBJ_ObjHandle  *targetSplFHandle,
                              cwbOBJ_ObjHandle  *outputQueueHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
移動されるスプール・ファイルのハンドル。

cwbOBJ_ObjHandle *targetSplFHandle - input
オプションです。同じシステム上の別のスプール・ファイルのハンドルであり、このスプール・ファイルをそのあとに移動させるスプール・ファイルを指定します。これが指定されていると、*outputQueueHandle は使用されません。

cwbOBJ_ObjHandle *outputQueueHandle - input
オプションです。どの出力待ち行列にスプール・ファイルを移動させるかを指定する、同じシステム上の出力待ち行列のハンドル。スプール・ファイルは、この待ち行列の最初の位置に移動されます。このパラメーターは、targetSplFHandle が指定されると無視されます。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
無効なハンドル。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_TYPE
ハンドルが、スプール・ファイル・ハンドルではありません。

使用法

targetSplFHandle と outputQueueHandle の両方が NULL の場合は、スプール・ファイルは、現行の出力待ち行列の最初の位置に移動されます。

cwbOBJ_ReleaseSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

スプール・ファイルを解放します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_ReleaseSplF(
                              cwbOBJ_ObjHandle  splFHandle,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
解放されるスプール・ファイルのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれま

す。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
無効なハンドル。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_TYPE
ハンドルが、スプール・ファイル・ハンドルではありません。

使用法

なし (None)

cwbOBJ_SendNetSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

スプール・ファイルを、同じシステム上の別のユーザーまたはネットワーク上のリモート・システムに送信します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_SendNetSplF(
                              cwbOBJ_ObjHandle   splFHandle,
                              cwbOBJ_ParmHandle  parmListHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
送信されるスプール・ファイルのハンドル。

cwbOBJ_ParmHandle parmListHandle - input
必須です。スプール・ファイルを送信するためのパラメーターが入っているパラメーター・リスト・オブジェクトのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_PARAMETER
指定のパラメーターが無効です。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_TYPE
ハンドルが、スプール・ファイル・ハンドルではありません。

使用法

ネット・スプール・ファイルの送信 (SNDNETSPLF) コマンドに相当するコマンドがスプール・ファイルに対して出されます。以下のパラメーター・キーを parmListHandl オブジェクトに設定する必要があります。

- CWBOBJ_KEY_TOUSERID

  スプール・ファイルを送る先のユーザー ID を指定。
- CWBOBJ_KEY_TOADDRESS

  スプール・ファイルが送られるリモート・システムを指定。″*NORMAL″ がデフォルト。

以下のパラメーター・キーを parmListHandle オブジェクトに設定できます。

- CWBOBJ_KEY_DATAFORMAT

  スプール・ファイルを伝送するデータ形式を指定。″*RCDDATA″ または ″*ALLDATA″。 ″*RCDDATA″ がデフォルト。
- CWBOBJ_KEY_VMMVSCLASS

  VM ホスト・システムまたは MVS™ ホスト・システムへ配布するための VM/MVS SYSOUT クラスを指定。″A″ から ″Z″ または ″0″ から ″9″。″A″ がデフォルト。
- CWBOBJ_KEY_SENDPTY

  SNADS ネットワークを介しての経路指定中に、このスプール・ファイル用に使用される待ち行列優先順位を指定。″*NORMAL″ または ″*HIGH″。″*NORMAL″ がデフォルト。

cwbOBJ_SendTCPSplF:

これは、System i Access for Windows 製品で使用する API です。

目的

リモート・システムで印刷されるスプール・ファイルを送信します。これは TCP/IP LPR コマンドの System i バージョンになります。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_SendTCPSplF(
                              cwbOBJ_ObjHandle   splFHandle,
                              cwbOBJ_ParmHandle  parmListHandle,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
送信されるスプール・ファイルのハンドル。

cwbOBJ_ParmHandle parmListHandle - input
必須です。スプール・ファイルを送信するためのパラメーターが入っているパラメーター・リスト・オブジェクトのハンドル。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
無効なハンドル。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_TYPE
ハンドルが、スプール・ファイル・ハンドルではありません。

CWBOBJ_KEY_SEPPAGE
区切りページを印刷するかどうかを指定します。

CWBOBJ_KEY_USRDTATFMLIB
ユーザー・データ変換ライブラリーの名前を指定します。

CWBOBJ_KEY_USRDTATFM
ユーザー・データ変換プログラムの名前を指定します。

使用法

System i の TCP/IP スプール・ファイル送信 (SNDTCPSPLF) コマンドに相当するコマンドが、スプール・ファイルに対して出されます。以下のパラメーター・キーを parmListHandl オブジェクトに設定する必要があります。

- CWBOBJ_KEY_RMTSYSTEM

  印刷要求が送られるリモート・システムを指定。リモート・システム名または ″*INTNETADR″。
- CWBOBJ_KEY_RMTPRTQ

  宛先印刷待ち行列の名前を指定。

以下のパラメーター・キーを parmListHandle オブジェクトに設定できます。

- CWBOBJ_KEY_DELETESPLF

  スプール・ファイルが正常に送信された後にそのスプール・ファイルを削除するかどうかを指定します。″*NO″ または ″*YES″。″*NO″ がデフォルト。
- CWBOBJ_KEY_DESTOPTION

  宛先によって決まるオプションを指定。これらのオプションは、スプール・ファイルとともにリモート・システムへ送られる。
- CWBOBJ_KEY_DESTINATION

  スプール・ファイルが送られているシステムのタイプを指定。別のタイプの System i に送る場合は、この値を ″*AS/400″ にします。 ″*OTHER″ または ″*PSF/2″ の場合もありえます。 ″*OTHER″ がデフォルト。
- CWBOBJ_KEY_INTERNETADDR

  受信システムの IP アドレスを指定。
- CWBOBJ_KEY_MFGTYPE

  印刷データを SCS から ASCII へ変換する際に、メーカー、機種、および型式を指定。
- CWBOBJ_KEY_SCS2ASCII

  印刷データを SCS から ASCII へ変換するかどうかを指定。″*NO″ または ″*YES″。″*NO″ がデフォルト。
- CWBOBJ_KEY_WSCUSTMOBJ

  ワークステーション・カスタマイズ・オブジェクトの名前を指定。
- CWBOBJ_KEY_WSCUSTMOBJL

  ワークステーション・カスタマイズ・オブジェクト・ライブラリーの名前を指定。

System i Access for Windows のスプール・ファイル・メッセージを処理する API

以下の System i Access for Windows API は、スプール・ファイル・メッセージの処理に関するものです。 API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_AnswerSplFMsg:

これは、System i Access for Windows 製品で使用する API です。

目的

スプール・ファイルが待機しているメッセージに応答します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_AnswerSplFMsg(
                              cwbOBJ_ObjHandle   splFHandle,
                              char              *msgAnswer,
                              cwbSV_ErrHandle    errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
メッセージを応答するスプール・ファイルのハンドル。

const char *msgAnswer - input
メッセージ応答が入っている ASCIIZ ストリングを指すポインター。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、有効なスプール・ファイル・ハンドルではありません。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_TYPE
ハンドルが、スプール・ファイル・ハンドルではありません。

CWBOBJ_RC_SPLFNOMESSAGE
スプール・ファイルがメッセージを待機していません。

使用法

なし (None)

cwbOBJ_GetSplFMsgAttr:

これは、System i Access for Windows 製品で使用する API です。

目的

スプール・ファイルに関連するメッセージの属性を検索します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_GetSplFMsgAttr(
                              cwbOBJ_ObjHandle  splFHandle,
                              cwbOBJ_KeyID      key,
                              void             *buffer,
                              unsigned long     bufLen,
                              unsigned long    *bytesNeeded,
                              cwbOBJ_DataType  *keyType,
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbOBJ_ObjHandle splFHandle - input
スプール・ファイルのハンドル。

cwbOBJ_KeyID key - input
検索する属性の識別キー。CWBOBJ_KEY_XXX 定数がキー ID を定義します。

void *buffer - output
この呼び出しが正常に戻った場合は、属性値を保持するバッファー。*Buffer に置かれるデータ・タイプは何かをキー値が決定します。タイプが与えられた場合、このタイプも *keyType パラメーターへ戻されます。

unsigned long bufLen - input
*Buffer が指すバッファーの長さ。

unsigned long *bytesNeeded - output
出力の際には、結果を保持するために必要なバイト数が入ります。

cwbOBJ_DataType *keyType - output
オプションであり、NULL でも構いません。出力の際には、この属性と、*buffer に何が保管されるかを表すために使用されるデータ・タイプが含まれます。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_INVALID_HANDLE
ハンドルが、割り振られたオブジェクト・ハンドルではありません。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎます。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_KEY
キーが有効ではありません。

CWBOBJ_RC_SPLFNOMESSAGE
スプール・ファイルがメッセージを待機していません。

CWB_API_ERROR
一般 API 障害。

使用法

以下のキーが有効です。

```
CWBOBJ_KEY_MSGTEXT      -   メッセージ・テキスト
CWBOBJ_KEY_MSGHELP      -   メッセージ・ヘルプ・テキスト
CWBOBJ_KEY_MSGREPLY     -   メッセージ応答
CWBOBJ_KEY_MSGTYPE      -   メッセージ・タイプ
CWBOBJ_KEY_MSGID        -   メッセージ ID
CWBOBJ_KEY_MSGSEV       -   メッセージ重大度
CWBOBJ_KEY_DATE         -   メッセージ日付
CWBOBJ_KEY_TIME         -   メッセージ時刻
```

メッセージ様式化文字がメッセージ・テキスト内に示されますが、この文字は、次のように使用してください。

&N 強制的にテキストを 2 桁字下げして改行します。そのテキストが 1 行を超える場合は、テキストの終わりや別の様式制御文字に達するまで、4 桁字下げして改行する必要があります。

&P 強制的にテキストを 6 桁字下げして改行します。そのテキストが 1 行を超える場合は、テキストの終わりや別の様式制御文字に達するまで、4 桁字下げして改行する必要があります。

&B 強制的にテキストを 4 桁字下げして改行します。そのテキストが 1 行を超える場合は、テキストの終わりや別の様式制御文字に達するまで、6 桁字下げして改行する必要があります。

System i Access for Windows のスプール・ファイル・データを分析する API

以下の System i Access for Windows API は、スプール・ファイル・データの分析に関するものです。API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_AnalyzeSplFData:

これは、System i Access for Windows 製品で使用する API です。

目的

スプール・ファイルのデータを分析し、データ・タイプが何であるかについての最善の予測を与えます。

構文

```
unsigned int CWB_ENTRY cwbOBJ_AnalyzeSplFData(
                            const char           *data,
                            unsigned long         bufLen,
                            cwbOBJ_SplFDataType  *dataType,
                            cwbSV_ErrHandle       errorHandle);
```

パラメーター

const char *data - input
分析されるデータを指すポインター。

unsigned long bufLen - input
データが指すバッファーの長さ。

cwbOBJ_SplFDataType *dataType - output
出力の際は、データ・タイプがこれに含まれます。データ・タイプが決められない場合は、デフォルトは CWBOBJ_DT_USERASCII になります。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

使用法

これは、データ・タイプが指定されていないか、もしくは *AUTO が指定されているスプール・ファイルの作成時に使用されるものと同じルーチンが使用されます。それが決められない場合は、その結果は *USERASCII がデフォルトになります。

System i Access for Windows 用サーバー・プログラム API

以下の System i Access for Windows API は、サーバー・プログラムに関するものです。 API はアルファベット順にリストされます。

注: 以下の API でハンドルを処理する場合に、有効なハンドルとして 0 が戻されることはありません。

cwbOBJ_DropConnections:

これは、System i Access for Windows 製品で使用する API です。

目的

このプロセスに使用するネットワーク印刷サーバーの、すべてのシステムとの未使用会話をすべて除去します。

構文

```
unsigned int CWB_ENTRY  cwbOBJ_DropConnections(
                              cwbSV_ErrHandle   errorHandle);
```

パラメーター

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべて、このオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_NO_ERROR
正常終了。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

使用法

CWBOBJ.DLL は、API 上で使用するためのネットワーク印刷サーバーで使用できる会話のプールを維持管理します。通常、これらの会話は、10 から 20 分の未使用時間が経過した後にタイムアウトになった上で除去されます。この API を使用すれば、アプリケーションは、タイムアウトを待たずに、即時に会話のプールを終結処理できるようになります。また、この API をプロセスの終了時に使用して、すべての会話が終了していることを保証することもできます。この API は、このプロセスに使用するすべてのサーバーとの「使用中」でない接続をすべて除去します。使用中の接続には、スプール・ファイルが (作成または読み取りのために) オープンされている接続も含まれます。

cwbOBJ_GetNPServerAttr:

これは、System i Access for Windows 製品で使用する API です。

目的

指定されたシステム上の QNPSERVR プログラムの属性を取得します。

構文

```
unsigned int CWB_ENTRY cwbOBJ_GetNPServerAttr(
                            const char      *systemName,
                            cwbOBJ_KeyID     key,
                            void            *buffer,
                            unsigned long    bufLen,
                            unsigned long   *bytesNeeded,
                            cwbOBJ_DataType *keyType,
                            cwbSV_ErrHandle  errorHandle);
```

パラメーター

const char *systemName - input
ASCIIZ ストリングに入っているシステム名を指すポインター。

cwbOBJ_KeyID key - input
検索する属性の識別キー。

void *buffer - output
属性値を保持するバッファー (この呼び出しが正常に戻った場合)。*Buffer に置かれるデータ・タイプは何かをキー値が決定します。タイプが与えられた場合、このタイプも *keyType パラメーターへ戻されます。

unsigned long bufLen - input
*Buffer が指すバッファーの長さ。

unsigned long *bytesNeeded - output
出力の際には、結果を保持するために必要なバイト数が入ります。

cwbOBJ_DataType *keyType - output
オプションであり、NULL でも構いません。出力の際には、この属性と、*buffer に何が保管されるかを表すために使用されるデータ・タイプが含まれます。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。

CWB_BUFFER_OVERFLOW
バッファーが小さすぎます。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

CWBOBJ_RC_HOST_ERROR
ホストでエラーが発生しました。テキストは errorHandle にあります。

CWBOBJ_RC_INVALID_KEY
キーが有効ではありません。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

以下の属性を QNPSERVR プログラムから検索することができます。

- CWBOBJ_KEY_NPSCCSID - サーバー CCSID
- CWBOBJ_KEY_NPSLEVEL - サーバー・コード・レベル

cwbOBJ_SetConnectionsToKeep:

これは、System i Access for Windows 製品で使用する API です。

目的

特定システムに対して活動状態のままにしておく必要のある接続の数を設定します。通常、ある未使用時間が経過した後に cwbobj.dll はタイムアウトになり、接続を除去します。この API を使用すれば、システムに対して、ある一定数の接続を強制的にオープンしたままにさせることが可能になります。

構文

```
unsigned int CWB_ENTRY cwbOBJ_SetConnectionsToKeep(
                              const char *systemName
                              unsigned int connections
                              cwbSV_ErrHandle  errorHandle);
```

パラメーター

const char *systemName - input
ASCIIZ ストリングに入っているシステム名を指すポインター。

unsigned int connections - input
オープンの状態を保留させる接続の数。

cwbSV_ErrHandle errorHandle - output
オプションです。0 でも構いません。戻されたメッセージはすべて、このオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle() API で作成されます。メッセージは、cwbSV_GetErrText() API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索できません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_PARAMETER
指定のパラメーターが無効。

使用法

1 つのシステムにつき、オープンの状態を保持させる接続のデフォルト数は 0 です。接続はプロセスごとに行われるため、この API は、それを呼び出したプロセスに属している接続のみに有効です。オープン状態を保持させる接続の数を設定しても、いかなる新規接続もオープンされません。

例: System i Access for Windows 用システム・オブジェクト API の使用法

次の例は、スプール・ファイルのリストの検索とそれらの表示を行うための通常の呼び出し手順を示しています。

```
/*********************************************************/
/* List all spooled files for the current user and       */
/*  display them to the user.                            */
/*********************************************************/

 #ifdef UNICODE
    #define _UNICODE
 #endif
 #include <windows.h>
```

```
 #include <stdio.h>
 #include "CWBOBJ.H"
main(int argc, char *argv[ ], char *envp[ ])
{
 cwbOBJ_ListHandle listHandle;
 cwbOBJ_ObjHandle  splFHandle;
 unsigned int ulRC;
 unsigned long ulListSize, ulObjPosition, ulBytesNeeded;
 cwbOBJ_KeyID keysWanted[] = { CWBOBJ_KEY_SPOOLFILE,
                                              CWBOBJ_KEY_USER };
 unsigned long ulNumKeysWanted = sizeof(keysWanted)/sizeof(*keysWanted);
 char szSplFName[11];
 char szUser[11];

 ulRC = cwbOBJ_CreateListHandle(_TEXT("ANYAS400"),
                               CWBOBJ_LIST_SPLF,
                               &listHandle,
                               0);
 if (ulRC == CWB_OK)
 {

     /* Set up the filter for the list to be opened with   */
     /* NOTE: this is just for example, the user defaults  */
     /*       to *CURRENT, so this isn't really needed.    */

     cwbOBJ_SetListFilter(listHandle, CWBOBJ_KEY_USER,
                          _TEXT("*CURRENT"), 0);

     /* Optionally call to cwbOBJ_SetListAttrsToRetrieve to*/
     /*  make walking the list faster                      */
     ulRC = cwbOBJ_SetListAttrsToRetrieve(listHandle,
                                          ulNumKeysWanted,
                                          keysWanted,
                                          0);

     /* open the list - this will build the list of spooled*/
     /* files.                                             */
     ulRC = cwbOBJ_OpenList(listHandle,
                            CWBOBJ_LIST_OPEN_SYNCH,
                            0);
     if (ulRC == CWB_OK)
     {
         /* Get the number of items that are in the list   */
         ulRC = cwbOBJ_GetListSize(listHandle,
                                   &ulListSize,
                                   (cwbOBJ_List_Status *)0,
                                   0);
         if (ulRC == CWB_OK)
         {

             /* walk through the list of items, displaying */
             /* each item to the user                      */

             ulObjPosition = 0;
             while (ulObjPosition < ulListSize)
             {
                /*****************************************/
                /* Get a handle to the next spooled file in*/
                /* the list.  This handle is valid while   */
                /* the list is open.  If you want to       */
                /* maintain a handle to the spooled file   */
                /* after the list is closed, you could call*/
                /* cwbOBJ_CopyObjHandle() after this call. */
                /*****************************************/
                ulRC = cwbOBJ_GetObjHandle(listHandle,
                                           ulObjPosition,
```

```
                                &splFHandle,
                                0);
            if (ulRC == CWB_OK)
            {

               /*************************************/
               /* call cwbOBJ_GetObjAttr() to get info */
               /* about this spooled file.  May also   */
               /* call spooled file specific APIs      */
               /* with this handle, such as            */
               /* cwbOBJ_HoldSplF().                   */
               /*************************************/

               ulRC = cwbOBJ_GetObjAttr(splFHandle,
                                CWBOBJ_KEY_SPOOLFILE,
                                (void *)szSplFName,
                                sizeof(szSplFName),
                                &ulBytesNeeded,
                                NULL,
                                0);
               if (ulRC == CWB_OK)
               {
                  ulRC = cwbOBJ_GetObjAttr(splFHandle,
                                   CWBOBJ_KEY_USER,
                                   (void *)szUser,
                                   sizeof(szUser),
                                   &ulBytesNeeded,
                                   NULL,
                                   0);
                  if (ulRC == CWB_OK)
                  {
                     printf("%3u: %11s %s¥n",
                            ulObjPosition, szSplFName, szUser);
                  } else {
                     /* ERROR on GetObjAttr! */
               }
               } else {
                  /* ERROR on GetObjAttr! */
               }
               /* free this object handle              */
               cwbOBJ_DeleteObjHandle(splFHandle, 0);
            } else {
              /* ERROR on GetObjHandle! */
            }
            ulObjPosition++;
          }
        } else {
          /* ERROR on GetListSize! */
        }
        cwbOBJ_CloseList(listHandle, 0);
    } else {
       /* ERROR on OpenList! */
    }

    cwbOBJ_DeleteListHandle(listHandle, 0);
}
```

System i Access for Windows のリモート・コマンド/分散プログラム呼び出し API

PC アプリケーション・プログラマーは、System i Access for Windows リモート・コマンド/分散プログラム呼び出し API を使用することで、System i の機能にアクセスできます。ユーザー・プログラムとシステ

ム・コマンドを、エミュレーション・セッションなしに呼び出します。コマンドとプログラムは単一の System i プログラムによって扱われるため、コマンドとプログラムの両方に対して、1 つのシステム・ジョブのみが開始されます。

System i Access for Windows のリモート・コマンド API

System i Access for Windows のリモート・コマンド・アプリケーション・プログラミング・インターフェース (API) を使用すると、PC アプリケーションで非対話式の System i コマンドを開始して、これらのコマンドから完了メッセージを受け取ることができるようになります。 System i コマンドは、最高 10 個までの応答メッセージを送信することができます。

System i Access for Windows の分散プログラム呼び出し API

System i Access for Windows の分散プログラム呼び出し API を使用すると、PC アプリケーションで任意の System i プログラムやコマンドを呼び出すことができるようになります。入力、出力、および入出力のパラメーターは、この関数を介して扱われます。プログラムが正しく実行されると、出力パラメーターと入出力パラメーターに、呼び出された System i プログラムが戻したデータが入ります。プログラムがシステムで正しく実行されなかった場合は、そのプログラムは、最高 10 個までの応答メッセージを送信することができます。

System i Access for Windows のリモート・コマンド/分散プログラム呼び出し API に必要なファイル

ヘッダー・ファイル	インポート・ライブラリー	ダイナミック・リンク・ライブラリー
cwbrc.h	cwbapi.lib	cwbrc.dll

Programmer's Toolkit:

Programmer's Toolkit には、リモート・コマンドおよび分散プログラム呼び出しの資料、cwbrc.h ヘッダー・ファイルへのアクセス、およびプログラム例へのリンクが用意されています。この情報にアクセスするには、Programmer's Toolkit をオープンして、「**リモート・コマンド**」または「**分散プログラム**」 → 「**C/C++ API**」と選択します。

注: コード例を使用することで、 633 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』の条件に同意します。

関連資料

30 ページの『リモート・コマンド/分散プログラム呼び出し API 戻りコード』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し API の戻りコードを示します。

7 ページの『接続 API 用の System i 名の形式』
パラメーターとして System i 名を取得する API では、3 つの異なる形式の名前を使用できます。

8 ページの『OEM、ANSI、およびユニコードの考慮事項』
System i Access for Windows の、ストリング・パラメーターを受け入れる C/C++ API の大部分は、OEM、ANSI、Unicode の 3 つのうち、いずれかの形式になっています。

System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法

System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

これらの各オブジェクトは、ハンドルによってアプリケーションに識別されます。

システム・オブジェクト
これは System i の ID です。このシステム・オブジェクトを指すハンドルは、コマンドや API が実行されるシステムを識別するために、StartSysEx 関数に与えられます。

コマンド要求オブジェクト
これは System i の要求を表します。このオブジェクト上で、コマンドを実行させ、プログラムを呼び出すことができます。

注: 以前の System i Access for Windows 製品では、コマンド要求オブジェクトは「システム・オブジェクト」と呼ばれていました。

プログラム・オブジェクト
System i プログラムを表します。パラメーターを追加し、プログラム情報をシステムへ送って、プログラムを実行することができます。

コマンド用の別個のオブジェクトはありません。コマンド・ストリングは、コマンド要求へ直接送られます。

リモート・コマンド/分散プログラム呼び出し API を使用するアプリケーションでは、最初に、cwbCO_CreateSystem 関数を呼び出してシステム・オブジェクトを作成します。この関数は、そのシステム・オブジェクトを指すハンドルを戻します。次に、このハンドルを cwbRC_StartSysEx 関数で使用して、System i の会話を開始します。cwbRC_StartSysEx 関数は、コマンド要求を指すハンドルを戻します。このコマンド要求ハンドルを使用して、プログラムを呼び出したり、あるいは、コマンドを実行することができます。コマンド要求オブジェクトに関連した API には、次のものがあります。

- cwbRC_StartSysEx
- cwbRC_CallPgm
- cwbRC_RunCmd
- cwbRC_StopSys

コマンドは、System i プラットフォーム上で実行される文字ストリングです。これは、単純なオブジェクト (文字ストリング) であるため、コマンドを実行するために追加のオブジェクトを作成する必要はありません。コマンド・ストリングは、単に、cwbRC_RunCmd API でのパラメーターです。

プログラムは、cwbRC_CreatePgm API によって作成される複合オブジェクトです。この API には、プログラム名とライブラリー名をパラメーターとして指定する必要があります。この関数によって戻されるハンドルには、0 から 35 のパラメーターを関連付けることができます。パラメーターは、cwbRC_AddParm 関数を使って追加されます。パラメーター・タイプには、入力、出力、または入出力があります。これらのパラメーターは、System i プログラムが処理できる形式 (つまり、データの変形や変換は行われないもの) で指定する必要があります。パラメーターがすべて追加されたら、プログラム・ハンドルがコマンド要求オブジェクトの cwbRC_CallPgm API で使用されます。プログラム・オブジェクトに関連する API には、次のものがあります。

- cwbRC_AddParm
- cwbRC_CreatePgm
- cwbRC_DeletePgm
- cwbRC_GetLibName
- cwbRC_GetParm
- cwbRC_GetParmCount

- cwbRC_GetPgmName
- cwbRC_SetLibName
- cwbRC_SetParm
- cwbRC_SetPgmName

関連資料

51 ページの『cwbCO_CreateSystem』
System i Access for Windows の cwbCO_CreateSystem コマンドを使用します。

374 ページの『cwbRC_StartSysEx』
これは、System i Access for Windows 製品で使用する API です。

379 ページの『cwbRC_CallPgm』
これは、System i Access for Windows 製品で使用する API です。

376 ページの『cwbRC_RunCmd』
これは、System i Access for Windows 製品で使用する API です。

375 ページの『cwbRC_StopSys』
これは、System i Access for Windows 製品で使用する API です。

380 ページの『cwbRC_CreatePgm』
これは、System i Access for Windows 製品で使用する API です。

377 ページの『cwbRC_AddParm』
これは、System i Access for Windows 製品で使用する API です。

384 ページの『cwbRC_GetParmCount』
これは、System i Access for Windows 製品で使用する API です。

383 ページの『cwbRC_GetParm』
これは、System i Access for Windows 製品で使用する API です。

385 ページの『cwbRC_GetPgmName』
これは、System i Access for Windows 製品で使用する API です。

382 ページの『cwbRC_GetLibName』
これは、System i Access for Windows 製品で使用する API です。

387 ページの『cwbRC_SetParm』
これは、System i Access for Windows 製品で使用する API です。

389 ページの『cwbRC_SetPgmName』
これは、System i Access for Windows 製品で使用する API です。

386 ページの『cwbRC_SetLibName』
これは、System i Access for Windows 製品で使用する API です。

381 ページの『cwbRC_DeletePgm』
これは、System i Access for Windows 製品で使用する API です。

リモート・コマンド/分散プログラム呼び出し: System i Access for Windows のリモート・コマンド API リストへのアクセス

System i リモート・コマンドのサーバー・プログラムにアクセスします。コマンドの実行およびプログラムの呼び出しには、要求ハンドルが使用されます。 API はアルファベット順にリストされます。

cwbRC_GetClientCCSID:

これは、System i Access for Windows 製品で使用する API です。

目的

現行のプロセスと関連した、コード化文字セット識別コード (CCSID) を取得します。この CCSID をホストの CCSID と一緒に使用すると、何らかの System i プログラムから戻される EBCDIC データを、クライアント・アプリケーションで使用可能な ASCII データに変換できます。

構文

```
unsigned int CWB_ENTRY cwbRC_GetClientCCSID(
                            cwbRC_SysHandle     system,
                            unsigned long      *clientCCSID);
```

パラメーター

cwbRC_SysHandle system - input
以前の cwbRC_StartSysEx 関数への呼び出しによって戻されたハンドル。これは System i の ID です。

unsigned long * clientCCSID - output
クライアント CCSID が書き込まれる先の、無符号長精度整数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBRC_INVALID_SYSTEM_HANDLE
システム・ハンドルが無効。

使用法

CWBNLCNV.H ファイル中の関連する API を参照してください。

cwbRC_GetHostCCSID:

これは、System i Access for Windows 製品で使用する API です。

目的

System i ジョブに関連付けられたコード化文字セット ID (CCSID) を取得します。この CCSID をクライアントの CCSID と一緒に使用すると、何らかの System i プログラムから戻される EBCDIC データを、クライアント・アプリケーションで使用可能な ASCII データに変換できます。

構文

```
unsigned int CWB_ENTRY cwbRC_GetHostCCSID(
                            cwbRC_SysHandle     system,
                            unsigned long      *hostCCSID);
```

パラメーター

cwbRC_SysHandle system - input
以前の cwbRC_StartSysEx 関数への呼び出しによって戻されたハンドル。これは System i の ID です。

unsigned long * hostCCSID - output
ホスト CCSID が書き込まれる先の、無符号長精度整数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBRC_INVALID_SYSTEM_HANDLE
システム・ハンドルが無効。

使用法

CWBNLCNV.H ファイル中の関連する API を参照してください。

cwbRC_StartSysEx:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、指定されたシステムとの会話を開始させます。会話が正常に開始されると、ハンドルが戻されます。このハンドルは、これ以降のすべてのコマンドの発行またはプログラムの呼び出しに使用します。この会話が不要になった時点で、会話を終了させるために、このハンドルを cwbRC_StopSys API で使用してください。cwbRC_StartSysEx API は、1 つのアプリケーション内で何度も呼び出すことができます。StartSysEx 呼び出しで同じシステム・オブジェクト・ハンドルを使用した場合、開始される System i の会話は 1 つだけです。複数の会話を活動状態にするには、別々のシステム・オブジェクト・ハンドルを指定して、StartSysEx を何度も呼び出す必要があります。

構文

```
unsigned int CWB_ENTRY cwbRC_StartSysEx(
                            const cwbCO_SysHandle systemObj,
                            cwbRC_SysHandle    *request);
```

パラメーター

const cwbCO_SysHandle systemObj - input
プログラムとコマンドの実行元にするシステムの既存のシステム・オブジェクトを指すハンドル。

cwbRC_SysHandle *request - output
コマンド要求のハンドルが戻される cwbRC_SysHandle を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_COMMUNICATIONS_ERROR
通信エラーが発生しました。

CWB_SERVER_PROGRAM_NOT_FOUND
System i アプリケーションが見つかりません。

CWB_HOST_NOT_FOUND
システムが非活動中であるか、または存在しません。

CWB_SECURITY_ERROR
セキュリティー・エラーが発生しました。

CWB_LICENSE_ERROR
ライセンス・エラーが発生しました。

CWB_CONFIG_ERROR
構成エラーが発生しました。

CWBRC_SYSTEM_NAME
システム名が長すぎます。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

なし

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbRC_StopSys:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、ハンドルで指定されたシステムとの会話を停止させます。これ以降このハンドルは、プログラム呼び出しまたはコマンドの発行には使用できなくなります。

構文

```
unsigned int CWB_ENTRY cwbRC_StopSys(
                            cwbRC_SysHandle     system);
```

パラメーター

cwbRC_SysHandle system - input
以前の cwbRC_StartSysEx 関数への呼び出しによって戻されたハンドル。これは System i の ID です。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBRC_INVALID_SYSTEM_HANDLE
システム・ハンドルが無効。

使用法

なし (None)

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

リモート・コマンド/分散プログラム呼び出し: System i Access for Windows の API リストの実行

これらの API は、System i コマンドの実行に使用します。 API はアルファベット順にリストされます。

cwbRC_RunCmd:

これは、System i Access for Windows 製品で使用する API です。

目的

ハンドルによって識別されたシステム上でコマンドを出します。戻りコードはコマンドが成功か失敗かを示します。戻されたメッセージ・ハンドルを使用して、追加のメッセージを戻すことができます。

構文

```
unsigned int CWB_ENTRY cwbRC_RunCmd(
                            cwbRC_SysHandle     system,
                            const char         *commandString,
                            cwbSV_ErrHandle     msgHandle);
```

パラメーター

cwbRC_SysHandle system - input
以前の cwbRC_StartSysEx 関数への呼び出しによって戻されたハンドル。これは System i の ID です。

const char *commandString - input
実行するコマンドを含んだストリングを指すポインター。これは ASCIIZ ストリングです。

cwbSV_ErrHandle msgHandle - output
戻された System i メッセージは、すべてこのオブジェクトに書き込まれます。このオブジェクトは、

cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrTextIndexed API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBRC_INVALID_SYSTEM_HANDLE
システム・ハンドルが無効。

CWBRC_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBRC_USR_EXIT_ERROR
ユーザー出口プログラムでエラーが発生しました。

CWBRC_COMMAND_FAILED
コマンドは失敗しました。

CWBRC_COMMAND_TOO_LONG
コマンド・ストリングが長すぎます。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

なし (None)

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

リモート・コマンド/分散プログラム呼び出し: System i Access for Windows のプログラム API リストへのアクセス

これらの System i Access for Windows API を使用して、プログラムおよびそのパラメーターへアクセスします。

cwbRC_AddParm:

これは、System i Access for Windows 製品で使用する API です。

目的

ハンドルで識別されるプログラムにパラメーターを追加します。プログラムに追加されるパラメーターごとに、この関数を呼び出す必要があります。プログラムが呼び出されるときには、パラメーターは、この関数を使用して追加した順序に並べられています。

構文

```
unsigned int CWB_ENTRY cwbRC_AddParm(
                            cwbRC_PgmHandle       program,
                            unsigned short        type,
                            unsigned long         length,
                            const unsigned char  *parameter);
```

パラメーター

cwbRC_PgmHandle program - input
以前の cwbRC_CreatePgm API の呼び出しによって戻されたハンドル。プログラム・オブジェクトを識別します。

unsigned short type - input
パラメーターのタイプ。定義済みパラメーター・タイプである、 CWBRC_INPUT、CWBRC_OUTPUT、CWBRC_INOUT のいずれかを使用します。ローカル CCSID とホスト CCSID との間で自動的に変換を実行させたい場合は、ビット単位 OR を指定して適切な変換フラグをこのフィールドに追加します。次のいずれかの定義済みパラメーター・タイプを使用してください。

- CWBRC_TEXT_CONVERT
- CWBRC_TEXT_CONVERT_INPUT
- CWBRC_TEXT_CONVERT_OUTPUT

後の 2 つのタイプは、変換が一方向だけに必要な場合に CWBRC_INOUT で使用するよう意図されています。

unsigned long length - input
パラメーターの長さ。CWBRC_OUTPUT パラメーターの場合、この長さは、戻されたパラメーターが書き込まれるバッファーの長さである必要があります。

const unsigned char * parameter - input
次のものが入っているバッファーを指すポインター。タイプが CWBRC_INPUT または CWBRC_INOUT の場合は値、タイプが CWBRC_OUTPUT または CWBRC_INOUT の場合は戻されたパラメーターが書き込まれる場所。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBRC_INVALID_PROGRAM
プログラム・ハンドルが無効。

CWBRC_INVALID_TYPE
無効なタイプが指定されました。

CWBRC_INVALID_PARM_LENGTH
パラメーターの長さが無効。

CWBRC_INVALID_PARM
無効なパラメーター。

使用法

パラメーター・データは 2 進数であると想定されます。変換フラグがいずれか設定されない限り、パラメーター・データの変換は行われません。例えば、以下のとおりです。

```
cwbRC_AddParm( hPgm,
CWBRC_INOUT | CWBRC_TEXT_CONVERT_OUTPUT,
bufferSize,
buffer );
```

これによって、ホストに送信される現状のままバッファーが使用され、結果がそのバッファーに入れられる前に、出力が (例えば、ASCII などに) 変換されます。

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbRC_CallPgm:

これは、System i Access for Windows 製品で使用する API です。

目的

ハンドルで識別されるプログラムを呼び出します。戻りコードはプログラムが正常か失敗かを示します。戻されたメッセージ・ハンドルを使用して、追加のメッセージを戻すことができます。

構文

```
unsigned int CWB_ENTRY cwbRC_CallPgm(
                            cwbRC_SysHandle     system,
                            cwbRC_PgmHandle     program,
                            cwbSV_ErrHandle     msgHandle);
```

パラメーター

cwbRC_SysHandle system - input
以前の cwbRC_StartSysEx 関数への呼び出しによって戻されたハンドル。これは System i の ID です。

cwbRC_PgmHandle program - input
以前の cwbRC_CreatePgm API の呼び出しによって戻されたハンドル。プログラム・オブジェクトを識別します。

cwbSV_ErrHandle msgHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。 このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrTextIndexed API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_COMMUNICATIONS_ERROR
通信エラーが発生しました。

CWBRC_INVALID_SYSTEM_HANDLE
システム・ハンドルが無効。

CWBRC_INVALID_PROGRAM
プログラム・ハンドルが無効。

CWBRC_REJECTED_USER_EXIT
ユーザー出口プログラムによりコマンドが拒否されました。

CWBRC_USER_EXIT_ERROR
ユーザー出口プログラムでエラーが発生しました。

CWBRC_PROGRAM_NOT_FOUND
プログラムが見付かりませんでした。

CWBRC_PROGRAM_ERROR
プログラムの呼び出し時のエラー。

使用法

なし (None)

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbRC_CreatePgm:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、プログラム名およびライブラリー名が与えられた、プログラム・オブジェクトを作成します。戻されるハンドルを使って、プログラムにパラメーターを追加し、そのプログラムを呼び出すことができます。

構文

```
unsigned int CWB_ENTRY cwbRC_CreatePgm(
                            const char         *programName,
                            const char         *libraryName,
                            cwbRC_PgmHandle    *program);
```

パラメーター

const char *programName - input
呼び出すプログラムの名前が含まれている、ASCIIZ ストリングを指すポインター。名前は、二重引用符で囲む場合を除き大文字です。

const char *libraryName - input
プログラムが置かれているライブラリーの名前が含まれている ASCIIZ ストリングを指すポインター。名前は、二重引用符で囲む場合を除き大文字です。

cwbRC_PgmHandle * program - output
プログラムのハンドルが戻される cwbRC_PgmHandle を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBRC_PROGRAM_NAME
プログラム名が長すぎます。

CWBRC_LIBRARY_NAME
ライブラリー名が長すぎます。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

システムで呼び出すプログラムごとに、System i プログラム・オブジェクトを個別に作成する必要があります。このファイルに記述された関数を使用して、プログラムに送られるパラメーターの値を変更することはできますが、送られるパラメーターの数を変更することはできません。

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbRC_DeletePgm:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、与えられたハンドルで識別されるプログラム・オブジェクトを削除します。

構文

```
unsigned int CWB_ENTRY cwbRC_DeletePgm(
                            cwbRC_PgmHandle     program);
```

パラメーター

cwbRC_PgmHandle program - input
以前の cwbRC_CreatePgm API の呼び出しによって戻されたハンドル。プログラム・オブジェクトを識別します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBRC_INVALID_PROGRAM
プログラム・ハンドルが無効。

使用法

なし

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbRC_GetLibName:

これは、System i Access for Windows 製品で使用する API です。

目的

このプログラム・オブジェクトを作成する際に使用されたライブラリーの名前を取得します。

構文

```
unsigned int CWB_ENTRY cwbRC_GetLibName(
                            cwbRC_PgmHandle     program,
                            char               *libraryName);
```

パラメーター

cwbRC_PgmHandle program - input
以前の cwbRC_CreatePgm API の呼び出しによって戻されたハンドル。プログラム・オブジェクトを識別します。

char * libraryName - output
ライブラリーの名前が書き込まれる先の 10 文字のバッファーを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBRC_INVALID_PROGRAM
プログラム・ハンドルが無効。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_API_ERROR
一般 API 障害。

使用法

なし (None)

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbRC_GetParm:

これは、System i Access for Windows 製品で使用する API です。

目的

指標で識別されるパラメーターを検索します。この指標の範囲は、0 から「パラメーターの合計数 - 1」です。この数値は、cwbRC_GetParmCount API を呼び出して入手することができます。

構文

```
unsigned int CWB_ENTRY cwbRC_GetParm(
                            cwbRC_PgmHandle     program,
                            unsigned short      index,
                            unsigned short     *type,
                            unsigned long      *length,
                            unsigned char     **parameter);
```

パラメーター

cwbRC_PgmHandle handle - input
以前の cwbRC_CreatePgm API の呼び出しによって戻されたハンドル。プログラム・オブジェクトを識別します。

unsigned short index - input
検索される、このプログラム内の特定のパラメーターの番号。この指標はゼロを基準とします。

unsigned short * type - output
このパラメーターのタイプを指すポインター。この値には、次のいずれかの定義済みパラメーター・タイプを指定します。

- CWBRC_INPUT
- CWBRC_OUTPUT
- CWBRC_INOUT

unsigned long * length - input
パラメーターの長さを指すポインター。

unsigned char * * parameter - output
実パラメーターのアドレスが入るバッファーを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBRC_INVALID_PROGRAM
プログラム・ハンドルが無効。

CWBRC_INDEX_RANGE_ERROR
指標が範囲外です。

使用法

なし (None)

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbRC_GetParmCount:

これは、System i Access for Windows 製品で使用する API です。

目的

このプログラム・オブジェクトについて、パラメーターの数を取得します。

構文

```
unsigned int CWB_ENTRY cwbRC_GetParmCount(
                            cwbRC_PgmHandle     program,
                            unsigned short     *count);
```

パラメーター

cwbRC_PgmHandle handle - input
以前の cwbRC_CreatePgm API の呼び出しによって戻されたハンドル。プログラム・オブジェクトを識別します。

unsigned short * count - output
パラメーター・カウントが書き込まれる先の、無符号短精度整数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBRC_INVALID_PROGRAM
プログラム・ハンドルが無効。

使用法

なし (None)

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbRC_GetPgmName:

これは、System i Access for Windows 製品で使用する API です。

目的

このプログラムを作成するときに使用されたプログラムの名前を取得します。

構文

```
unsigned int CWB_ENTRY cwbRC_GetPgmName(
                           cwbRC_PgmHandle     program,
                           char               *programName);
```

パラメーター

cwbRC_PgmHandle program - input
以前の cwbRC_CreatePgm API の呼び出しによって戻されたハンドル。プログラム・オブジェクトを識別します。

char * programName - output
プログラムの名前が書き込まれる先の 10 文字のバッファーを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ポインターが、不良または NULL ポインターです。

CWBRC_INVALID_PROGRAM
プログラム・ハンドルが無効。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_API_ERROR
一般 API 障害。

使用法

なし (None)

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbRC_SetLibName:

これは、System i Access for Windows 製品で使用する API です。

目的

このプログラム・オブジェクトについて、ライブラリーの名前を設定します。

構文

```
unsigned int CWB_ENTRY cwbRC_SetLibName(
                            cwbRC_PgmHandle     program,
                            const char         *libraryName);
```

パラメーター

cwbRC_PgmHandle program - input
以前の cwbRC_CreatePgm API の呼び出しによって戻されたハンドル。プログラム・オブジェクトを識別します。

const char *libraryName - input
プログラムが置かれているライブラリーの名前が含まれている ASCIIZ ストリングを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBRC_INVALID_PROGRAM
プログラム・ハンドルが無効。

CWBRC_LIBRARY_NAME
ライブラリー名が長すぎます。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

この関数は、呼び出す必要のあるプログラムが入っているライブラリーの名前を変更するために使用します。異なるパラメーターで異なるプログラムを呼び出す場合は、この関数を使用しないでください。

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』
System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbRC_SetParm:

これは、System i Access for Windows 製品で使用する API です。

目的

指標で識別されるパラメーター値を設定します。この指標の範囲は、0 から「パラメーターの合計数 - 1」です。この数値は、cwbRC_GetParmCount API を呼び出して入手することができます。この関数はパラメーターを変更するために使用される点に注意してください。パラメーターを作成する場合は cwbRC_AddParm を使用してください。

構文

```
unsigned int CWB_ENTRY cwbRC_SetParm(
                           cwbRC_PgmHandle       program,
                           unsigned short        index,
                           unsigned short        type,
                           unsigned long         length,
                           const unsigned char  *parameter);
```

パラメーター

cwbRC_PgmHandle handle - input
以前の cwbRC_CreatePgm API の呼び出しによって戻されたハンドル。プログラム・オブジェクトを識別します。

unsigned short index - input
変更する必要のある、このプログラム内の特定のパラメーターの番号。この指標はゼロを基準とします。

unsigned short type - input
パラメーターのタイプ。次のいずれかの定義済みパラメーター・タイプを使用します。

- CWBRC_INPUT
- CWBRC_OUTPUT
- CWBRC_INOUT

ローカル CCSID とホスト CCSID との間で自動的に変換を行いたい場合は、ビット単位 OR を指定して適切な変換フラグをこのフィールドに追加します。次のいずれかの定義済みパラメーター・タイプを使用します。

- CWBRC_TEXT_CONVERT
- CWBRC_TEXT_CONVERT_INPUT
- CWBRC_TEXT_CONVERT_OUTPUT

後の 2 つは、変換が一方向にのみ必要な場合に CWBRC_INOUT で使用するよう意図されています。

unsigned long length - input
パラメーターの長さ。CWBRC_OUT パラメーターの場合、この長さは、戻されたパラメーターが書き込まれるバッファーの長さである必要があります。

const unsigned char * parameter - input
タイプが CWBRC_INPUT または CWBRC_INOUT の場合は、該当の値が含まれているバッファーを指し、タイプが CWBRC_OUTPUT または CWBRC_INOUT の場合は、戻されたパラメーターの書き込み先である場所を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBRC_INVALID_PROGRAM
プログラム・ハンドルが無効。

CWBRC_INVALID_TYPE
無効なタイプが指定されました。

CWBRC_INVALID_PARM_LENGTH
パラメーターの長さが無効。

CWBRC_INVALID_PARM
無効なパラメーター。

使用法

パラメーター・データは 2 進数であると想定されます。変換フラグがいずれか設定されない限り、パラメーター・データの変換は行われません。例えば、以下のとおりです。

```
cwbRC_SetParm( hPgm,
               CWBRC_INOUT | CWBRC_TEXT_CONVERT_OUTPUT,
               bufferSize,
               buffer );
```

これによって、ホストに送信される現状のままバッファーが使用され、結果がそのバッファーに入れられる前に、出力が (例えば、ASCII に) 変換されます。

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』

System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

cwbRC_SetPgmName:

これは、System i Access for Windows 製品で使用する API です。

目的

このプログラム・オブジェクトにプログラムの名前を設定します。

構文

```
unsigned int CWB_ENTRY cwbRC_SetPgmName(
                            cwbRC_PgmHandle     program,
                            const char         *programName);
```

パラメーター

cwbRC_PgmHandle program - input
以前の cwbRC_CreatePgm API の呼び出しによって戻されたハンドル。プログラム・オブジェクトを識別します。

const char *programName - input
呼び出すプログラムの名前が含まれている、ASCIIZ ストリングを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWBRC_INVALID_PROGRAM
プログラム・ハンドルが無効。

CWBRC_PROGRAM_NAME
プログラム名が長すぎます。

CWB_NOT_ENOUGH_MEMORY
メモリー不足です。一時バッファーの割り振りに失敗した可能性があります。

CWB_NON_REPRESENTABLE_UNICODE_CHAR
入力された 1 つまたは複数のユニコード文字が、使用されているコード・ページで表示されていません。

CWB_API_ERROR
一般 API 障害。

使用法

呼び出したいプログラムの名前を変更する場合は、この関数を使用します。異なるパラメーターで異なるプログラムを呼び出すためにプログラム・オブジェクトを変更する場合は、この関数を使用しないでください。

関連資料

370 ページの『System i Access for Windows リモート・コマンド/分散プログラム呼び出し API の一般的な使用法』

System i Access for Windows のリモート・コマンド/分散プログラム呼び出し機能を使用するアプリケーションは、オブジェクトを利用します。

例: リモート System i Access for Windows コマンド/分散プログラム呼び出し API の使用法

リモート System i Access for Windows コマンド/分散プログラム呼び出し API の使用法を、この例で説明します。

```
#ifdef UNICODE
   #define _UNICODE
#endif
#include <windows.h>

// Include the necessary RC/DPC Classes
#include <stdlib.h>
#include <iostream.h>
#include <TCHAR.H>
#include "cwbrc.h"
#include "cwbcosys.h"
/*********************************************************************/

void main()
{
   cwbCO_SysHandle system;
   cwbRC_SysHandle request;
   cwbRC_PgmHandle program;

   // Create the system object
   if ( (cwbCO_CreateSystem("AS/400SystemName",&system)) != CWB_OK )
      return;

   // Start the system
   if ( (cwbRC_StartSysEx(system,&request)) != CWB_OK )
      return;

   // Call the command to create a library
   char* cmd1 = "CRTLIB LIB(RCTESTLIB) TEXT('RC TEST LIBRARY')";
   if ( (cwbRC_RunCmd(request, cmd1, 0)) != CWB_OK )
      return;

   cout << "Created Library" << endl;

   // Call the command to delete a library
   char* cmd2 = "DLTLIB LIB(RCTESTLIB)";
   if ( (cwbRC_RunCmd(request, cmd2, 0)) != CWB_OK )
      return;

   cout << "Deleted Library" << endl;

   // Create a program object to create a user space
   if ( cwbRC_CreatePgm(_TEXT("QUSCRTUS"),
                        _TEXT("QSYS"),
                        &program) != CWB_OK )
```

```
   return;

// Add the parameters
   // name is DPCTESTSPC/QGPL
unsigned char name[20] = {0xC4,0xD7,0xC3,0xE3,0xC5,0xE2,0xE3,0xE2,0xD7,0xC3,
                          0xD8,0xC7,0xD7,0xD3,0x40,0x40,0x40,0x40,0x40,0x40};

   // extended attribute is not needed
unsigned char attr[10] = {0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40};

   // initial size is 100 bytes
unsigned long size = 0x64000000;

   // initial value is blank
unsigned char init = 0x40;

   // public authority is CHANGE
unsigned char auth[10] = {0x5C,0xC3,0xC8,0xC1,0xD5,0xC7,0xC5,0x40,0x40,0x40};

   // description is DPC TEMP SPACE
unsigned char desc[50] = {0xC4,0xD7,0xC3,0x40,0xE3,0xC5,0xD4,0xD7,0x40,0xE2,
                          0xD7,0xC1,0xC3,0xC5,0x40,0x40,0x40,0x40,0x40,0x40,
                          0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,
                          0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,
                          0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40};

if ( cwbRC_AddParm(program, CWBRC_INPUT, 20, name) != CWB_OK)
   return;

if ( cwbRC_AddParm(program, CWBRC_INPUT, 10, attr) != CWB_OK)
   return;

if ( cwbRC_AddParm(program, CWBRC_INPUT, 4, (unsigned char*)&size) != CWB_OK)
   return;

if ( cwbRC_AddParm(program, CWBRC_INPUT, 1, &init) != CWB_OK)
   return;

if ( cwbRC_AddParm(program, CWBRC_INPUT, 10, auth) != CWB_OK)
   return;

if ( cwbRC_AddParm(program, CWBRC_INPUT, 50, desc) != CWB_OK)
   return;


// Call the program
if ( cwbRC_CallPgm(request, program, 0) != CWB_OK )
   return;

cout << "Created User Space" << endl;

// Delete the program
if ( cwbRC_DeletePgm(program) != CWB_OK )
   return;

// Create a program object to delete a user space
if ( cwbRC_CreatePgm(_TEXT("QUSDLTUS"),
                     _TEXT("QSYS"),
                     &program) != CWB_OK )
   return;

// Add the parameters
   // error code structure will not be used
   unsigned long err = 0x00000000;

if ( cwbRC_AddParm(program, CWBRC_INPUT, 20, name) != CWB_OK)
   return;
```

```
   if ( cwbRC_AddParm(program, CWBRC_INOUT, 4, (unsigned char*)&err) != CWB_OK)
      return;

   // Call the program
   if ( cwbRC_CallPgm(request, program, 0) != CWB_OK )
      return;

   // Delete the program
   if ( cwbRC_DeletePgm(program) != CWB_OK )
      return;

   cout << "Deleted User Space" << endl;

   // Stop the system
   if ( cwbRC_StopSys(request) != CWB_OK )
      return;

   // Delete the system object
   if ( cwbCO_DeleteSystem(system) != CWB_OK )
     return;

}
```

System i Access for Windows の保守容易性 API

System i Access for Windows の保守容易性アプリケーション・プログラミング・インターフェース (API) を使用すると、プログラム内のメッセージおよびイベントを保守ファイルに記録することができます。

作成された保守ファイルからレコードを読み取るために使用できる API のセットも用意されています。これらの API を使用して、カスタマイズされた保守ファイル・ブラウザーを作成することができます。

System i Access for Windows の保守容易性 API 機能の一般的なカテゴリーは、以下のとおりです。

- メッセージ・テキストをヒストリー・ログに書き込むための API
- トレース項目をトレース・ファイルに書き込むための API
- 保守ファイルの読み取り
- エラー処理に関連するメッセージ・テキストを取り出すための API

System i Access for Windows の保守容易性 API を使用する理由

System i Access for Windows の保守容易性 API は、効率的な方法を使用して、メッセージ・ロギングおよびトレース・ポイントをユーザーのコードに追加します。これらの関数は、ユーザーのプロダクトの一部として出荷されるプログラムに組み込むほかに、開発中のプログラムのデバッグを行う助けとして使用することができます。ファイル構造は、複数プログラム (固有のプロダクトおよび構成要素のストリングによって識別される) による、同じファイルへの同時ログをサポートします。これにより、クライアント・ワークステーション上のロギング・アクティビティーの全体像が得られます。

System i Access for Windows の保守容易性 API に必要なファイル

ヘッダー・ファイル	インポート・ライブラリー	ダイナミック・リンク・ライブラリー
cwbsv.h	cwbapi.lib	cwbsv.dll

Programmer's Toolkit:

Programmer's Toolkit には、保守容易性の資料、cwbsv.h ヘッダー・ファイルへのアクセス、およびプログラム例へのリンクが用意されています。この情報にアクセスするには、Programmer's Toolkit をオープンし

て、「**エラー処理**」 → 「**C/C++ API**」と選択します。

注: コード例を使用することで、 633 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』の条件に同意します。

関連資料

33 ページの『保守容易性 API の戻りコード』
System i Access for Windows の保守容易性 API の戻りコードを示します。

ヒストリー・ログとトレース・ファイル

ヒストリー・ログとトレース・ファイルを使用して、System i Access for Windows プログラムに関する情報をログに記録することができます。

ヒストリー・ログ

ログ機能を使用することにより、System i Access for Windows のヒストリー・ログに、メッセージ・テキストを書き込むことができます。メッセージ・テキストは、表示可能な ASCII 文字データである必要があります。

System i Access for Windows のすべてのプログラムのメッセージが、System i Access for Windows のヒストリー・ログに記録されます。メッセージは、プロダクトが提供する DLL によっても記録されます。

ヒストリー・ログは、cwbSV_LogMessageText API を介してメッセージ・テキスト・ストリングが記録されるファイルです。このログは、クライアント・ワークステーションで実行されたヒストリーを提供します。

トレース・ファイル

トレース機能によって、ユーザー・プログラムの実行中に起こる低レベルのイベントを記録することができます。例えば、他の関数呼び出しから受け取った種々の戻りコードをトレースすることができます。ユーザー・プログラムがデータを送受信する場合、データの特別に意味のあるフィールド (例えば、関数バイトやデータ長など) を記録して、うまくいかない場合のデバッグに役立てることができます。これを行うには、**詳細データ・トレース**関数 (cwbSV_LogTraceData) を使用します。

トレース機能の別の形態である**エントリー・ポイント・トレース**関数は、ユーザー・ルーチンへ入る活動、およびユーザー・ルーチンから出る活動をトレースできるようにします。System i Access for Windows は、異なる 2 つのタイプのエントリー・ポイントのトレース・ポイントを定義します。

API トレース・ポイント
API (アプリケーション・プログラミング・インターフェース) トレース・ポイントは、他のプログラムに対して外部化されたルーチンへ入る活動、またそこから出る活動をトレースするために使用するものです。

SPI トレース・ポイント
SPI (システム・プログラミング・インターフェース) トレース・ポイントは、トレースしたいユーザー・プログラムの重要な内部ルーチンへの、またそこからの、入りと出をトレースするために使用するものです。

API 上に用意されている、1 バイトの eventID というキー情報があります。これによって、どの API または SPI への出入りが行われているかを識別することができます。入力値のようなデータは、入るときにトレースすることができます。これは、出力値がルーチンから出るときにトレースされるのと同様です。これらのトレース関数は、これらを利用するルーチンにおいて、ペアで (例えば、cwbSV_LogAPIEntry と

cwbSV_LogAPIExit のペアで) 使用するよう意図されています。これらのタイプのトレース・ポイントにより、コードを使用して制御のフローを記録できるようにします。

System i Access for Windows は、エントリー/エグジット API トレース・ポイントを使用して、このトピックで述べるプロシージャー型 API を備えています。トレース機能が活動中の場合、これらのうちのいずれかのプロシージャーの API が呼び出されると、入り口および出口のトレース・ポイントはエントリー・ポイント・トレース・ファイルに記録されます。エントリー/エグジット SPI トレースには、内部呼び出しの順序が記録されます。詳細データのトレース機能を使用すると、問題のデバッグにおいて役立つデータを記録することができます。

System i Access for Windows は、次のタイプのトレースをサポートします。

詳細 (データ)
このタイプを使用すると、cwbSV_LogTraceData API を介し、コード内のある 1 つのポイントで情報のバッファーをトレースすることが可能になります。このバッファーは、ASCII 値または 2 進値、あるいはその両方の混合を使用することが可能ですが (例えば、C-struct)、データは 2 進数形式で記録されます。

エントリー/エグジット (API)
特殊化された形式のトレースで、これを使用すると、cwbSV_LogAPIEntry と cwbSV_LogAPIExit API を介し、外部化されたルーチンに対する出入りをトレースすることが可能になります。

エントリー/エグジット (SPI):
特殊化された形式のトレースで、これを使用すると、cwbSV_LogSPIEntry と cwbSV_LogSPIExit API を介し、外部化されたルーチンに対する出入りをトレースすることが可能になります。

エラー・ハンドル

System i Access for Windows のエラー処理関数を使用すると、エラー・ハンドル (cwbSV_CreateErrHandle) を作成して、この関数をサポートする System i Access for Windows API で使用することができます。

System i Access for Windows API 呼び出し時にエラー (ゼロ以外の戻りコード) が起こった場合は、他のエラー処理関数を呼び出して、以下の情報を検索することができます。

- 戻りコードに関連するエラー・メッセージの数 (cwbSV_GetErrCount)
- 各エラー・メッセージのメッセージ・テキスト (cwbSV_GetErrTextIndexed)

保守容易性 API の一般的な使用法

System i Access for Windows の保守容易性 API の一般的な使用法には、ヒストリー・ログやエラー・ハンドルなどがあります。

ヒストリー・ログ

保守容易性 API は、クライアント・ワークステーションで実行されるアクティビティーに関するトレース・メカニズムを提供しています。メッセージ・ロギング API を使用することで、System i Access for Windows のヒストリー・ログにメッセージを記録することができます。ログ・メッセージには、アプリケーションが開始したことやその他の重要なイベントを示す記録が含まれています。例えば、ログ・メッセージは、ファイルが正常にシステムへ転送されたこと、何らかの理由でデータベース照会が失敗したこと、または、ジョブが印刷のため投入されたことなどを示すことができます。

保守容易性 API を使用する際に提供されるプロダクト・ストリングと構成要素ストリングによって、メッセージとイベントを保守ファイル内の他の項目と区別することができます。階層については、あるプロダクト ID を定義して、その下に 1 つまたは複数の構成要素 ID を定義することをお勧めします。

エラー・ハンドル

System i Access for Windows の C/C++ API でエラー・ハンドル・パラメーターを使用して、障害戻りコードに関連するメッセージ・テキストを検索します。これにより、アプリケーションでは、System i Access 戻りコードのセット用に独自のテキストを用意しなくても、メッセージ・テキストを表示することができます。

保守容易性 API のリスト: ヒストリー・ログへの書き込み

これらの System i Access for Windows API を使用して、メッセージ・テキストをヒストリー・ログに書き込みます。

cwbSV_CreateMessageTextHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数はメッセージ・テキスト・オブジェクトを作成し、そのオブジェクトへのハンドルを戻します。このメッセージ・ハンドルをユーザー・プログラムで使用すると、メッセージ・テキストを現在活動状態のヒストリー・ログに書き込むことができます。メッセージ・テキストは、cwbSV_LogMessageText() 呼び出しで渡されたバッファーに与えられます。

構文

```
unsigned int CWB_ENTRY cwbSV_CreateMessageTextHandle(
                            char                    *productID,
                            char                    *componentID,
                            cwbSV_MessageTextHandle *messageTextHandle);
```

パラメーター

char * productID - input
このメッセージ記入項目で使用されるプロダクト ID が含まれている、NULL 文字で終わるストリングを指します。このパラメーターはオプションで、NULL の場合は productID は設定されません。注: プロダクト ID について、最大で CWBSV_MAX_PRODUCT_ID 文字が記録されます。これより長いストリングは切り捨てられます。

char * componentID - input
このメッセージ記入項目で使用される構成要素 ID が含まれている、NULL 文字で終わるストリングを指します。このパラメーターはオプションで、NULL の場合は componentID は設定されません。注: 構成要素 ID について、最大で CWBSV_MAX_COMP_ID 文字が記録されます。これより長いストリングは切り捨てられます。

cwbSV_MessageTextHandle * messageTextHandle - input/output
ハンドルが戻される先の cwbSV_MessageTextHandle を指すポインター。このハンドルは、これ以降のメッセージ・テキスト関数呼び出しで使用する必要があります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリーが十分にないため、ハンドルを作成できません。

使用法

メッセージ・ハンドルを使用してメッセージ・テキストを記録する前に、そのメッセージ・ハンドルに固有のプロダクト ID と構成要素 ID を設定することをお勧めします。これらの ID によって、ユーザー・メッセージはヒストリー・ログの他のメッセージと区別されます。

cwbSV_DeleteMessageTextHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、与えられたハンドルで識別されるメッセージ・テキスト・オブジェクトを削除します。

構文

```
unsigned int CWB_ENTRY cwbSV_DeleteMessageTextHandle(
                            cwbSV_MessageTextHandle messageTextHandle);
```

パラメーター

cwbSV_MessageTextHandle messageTextHandle - input
以前の cwbSV_CreateMessageTextHandle() 関数の呼び出しによって戻されたハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
要求時に、使用できないハンドルが渡されました。

使用法

この呼び出しは、ハンドルが必要でなくなった際に行ってください。

cwbSV_LogMessageText:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、与えられたメッセージ・テキストを現在活動状態のヒストリー・ログへ記録します。記入項目に設定されたプロダクト ID と構成要素 ID は、テキストが記録された日時と共に書き込まれます。

構文

```
unsigned int CWB_ENTRY cwbSV_LogMessageText(
                            cwbSV_MessageTextHandle  messageTextHandle,
                            char                    *messageText,
                            unsigned long            messageTextLength);
```

パラメーター

cwbSV_MessageTextHandle messageTextHandle - input
以前の cwbSV_CreateMessageTextHandle() の呼び出しによって戻されたハンドル。

char * messageText - input
記録したいメッセージ・テキストが含まれているバッファーを指します。

unsigned long messageTextLength - input
このメッセージ記入項目について記録するメッセージ・テキスト・バッファーのバイト数を指定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
要求時に、使用できないハンドルが渡されました。

使用法

なし (None)

cwbSV_SetMessageClass:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数を使用すれば、ヒストリー・ログに書き込まれるメッセージに関連付けるメッセージ・クラス (重大度) の設定が可能になります。

構文

```
unsigned int CWB_ENTRY cwbSV_SetMessageClass(
                            cwbSV_MessageTextHandle  messageTextHandle,
                            cwbSV_MessageClass       messageClass);
```

パラメーター

cwbSV_MessageTextHandle messageTextHandle - input
以前の cwbSV_CreateMessageTextHandle() の呼び出しによって戻されたハンドル。

cwbSV_MessageClass messageClass - input
次のいずれかを指定します。

- CWBSV_CLASS_INFORMATIONAL
- CWBSV_CLASS_WARNING
- CWBSV_CLASS_ERROR

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
要求時に、使用できないハンドルが渡されました。

CWBSV_INVALID_MSG_CLASS
無効なメッセージ・クラスが渡されました。

使用法

この値は、対応するログ関数 cwbSV_LogMessageText() を呼び出す前に設定してください。

cwbSV_SetMessageComponent:

この API を使用して、System i Access for Windows メッセージ・ハンドルを設定します。

目的

この関数によって、与えられたメッセージ・ハンドルに、固有な構成要素 ID を設定できます。ユーザーのメッセージ記入項目をヒストリー・ログ中の他のプロダクト記入項目と区別するためにも、プロダクト ID の設定 (cwbSV_SetMessageProduct を参照) と共にこの呼び出しを使用してください。

構文

```
unsigned int CWB_ENTRY cwbSV_SetMessageComponent(
                            cwbSV_MessageTextHandle  messageTextHandle,
                            char                    *componentID);
```

パラメーター

cwbSV_MessageTextHandle messageTextHandle - input
以前の cwbSV_CreateMessageTextHandle() の呼び出しによって戻されたハンドル。

char * componentID - input
このメッセージ記入項目で使用される構成要素 ID が含まれている、NULL 文字で終わるストリングを指します。注: 構成要素 ID について、最大で CWBSV_MAX_COMP_ID 文字が記録されます。これより長いストリングは切り捨てられます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
要求時に、使用できないハンドルが渡されました。

使用法

この値は、対応するログ関数 cwbSV_LogMessageData() を呼び出す前に設定してください。階層には、あるプロダクト ID を定義し、その下に 1 つまたは複数の構成要素を定義することをお勧めします。

cwbSV_SetMessageProduct:

この API を使用して、System i Access for Windows のプロダクト ID を設定します。

目的

この関数によって、与えられたメッセージ・ハンドルに、固有なプロダクト ID を設定できます。ユーザーのメッセージ記入項目をヒストリー・ログ中の他のプロダクト記入項目と区別するためにも、構成要素 ID の設定 (cwbSV_SetMessageComponent を参照) と共にこの呼び出しを使用してください。

構文

```
unsigned int CWB_ENTRY cwbSV_SetMessageProduct(
                           cwbSV_MessageTextHandle  messageTextHandle,
                           char                    *productID);
```

パラメーター

cwbSV_MessageTextHandle messageTextHandle - input
以前の cwbSV_CreateMessageTextHandle() の呼び出しによって戻されたハンドル。

char * productID - input
このメッセージ記入項目で使用されるプロダクト ID が含まれている、NULL 文字で終わるストリングを指します。注: プロダクト ID について、最大で CWBSV_MAX_PRODUCT_ID 文字が記録されます。これより長いストリングは切り捨てられます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
要求時に、使用できないハンドルが渡されました。

使用法

この値は、対応するログ関数 cwbSV_LogMessageData() を呼び出す前に設定してください。階層には、あるプロダクト ID を定義し、その下に 1 つまたは複数の構成要素を定義することをお勧めします。

保守容易性 API のリスト: トレース・データの書き込み

これらの System i Access for Windows API を使用して、トレース・データを詳細トレース・ファイルに書き込みます。

cwbSV_CreateTraceDataHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数はトレース・データ・オブジェクトを作成し、そのオブジェクトへのハンドルを戻します。このトレース・ハンドルをユーザー・プログラムで使用すると、トレース情報をトレース・ファイルに記録することができます。トレース情報は、cwbSV_LogTraceData() 呼び出しで渡されるバッファーに与えられます。

構文

```
unsigned int CWB_ENTRY cwbSV_CreateTraceDataHandle(
                            char                  *productID,
                            char                  *componentID,
                            cwbSV_TraceDataHandle *traceDataHandle);
```

パラメーター

char * productID - input
このメッセージ記入項目で使用されるプロダクト ID が含まれている、NULL 文字で終わるストリングを指します。このパラメーターはオプションで、NULL の場合は productID は設定されません。注: プロダクト ID について、最大で CWBSV_MAX_PRODUCT_ID 文字が記録されます。これより長いストリングは切り捨てられます。

char * componentID - input
このメッセージ記入項目で使用される構成要素 ID が含まれている、NULL 文字で終わるストリングを指します。このパラメーターはオプションで、NULL の場合は componentID は設定されません。注: 構成要素 ID について、最大で CWBSV_MAX_COMP_ID 文字が記録されます。これより長いストリングは切り捨てられます。

cwbSV_TraceDataHandle * traceDataHandle - input/output
ハンドルが戻される先の cwbSV_TraceDataHandle を指すポインター。このハンドルは、これ以降のトレース・データ関数呼び出しで使用する必要があります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリーが十分にないため、ハンドルを作成できません。

使用法

トレース・データ・ハンドルを使用してトレース記入項目を記録する前に、そのトレース・データ・ハンドルに固有のプロダクト ID と構成要素 ID を設定することをお勧めします。これらの ID によって、ユーザーのトレース記入項目はトレース・ファイルの他の記入項目と区別されます。

cwbSV_DeleteTraceDataHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、与えられたハンドルで識別されるトレース・データ・オブジェクトを削除します。

構文

```
unsigned int CWB_ENTRY cwbSV_DeleteTraceDataHandle(
                            cwbSV_TraceDataHandle traceDataHandle);
```

パラメーター

cwbSV_TraceDataHandle traceDataHandle - input
以前の cwbSV_CreateTraceDataHandle() 関数の呼び出しによって戻されたハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この呼び出しは、ハンドルが必要でなくなった際に行ってください。

cwbSV_LogTraceData:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、与えられたトレース・データを現在活動状態のトレース・ファイルへ記録します。記入項目に設定されたプロダクト ID と構成要素 ID は、データが記録された日時と共に書き込まれます。

構文

```
unsigned int CWB_ENTRY cwbSV_LogTraceData(
                           cwbSV_TraceDataHandle  traceDataHandle,
                           char                  *traceData,
                           unsigned long          traceDataLength);
```

パラメーター

cwbSV_TraceDataHandle traceDataHandle - input
以前の cwbSV_CreateTraceDataHandle() の呼び出しによって戻されたハンドル。

char * traceData - input
記録したいトレース・データが含まれているバッファーを指します。トレース量の決定には長さパラメーターが使用されるため、バッファーには 2 進データを入れることができます。

unsigned long traceDataLength - input
このトレース記入項目について記録するトレース・データ・バッファーのバイト数を指定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

なし (None)

cwbSV_SetTraceComponent:

これは、System i Access for Windows のトレース記入項目で使用する API です。

目的

この関数によって、与えられた保守記入項目に、固有な構成要素 ID を設定できます。ユーザーのトレース記入項目をトレース・ファイル中の他のプロダクト記入項目と区別するためにも、プロダクト ID の設定 (cwbSV_SetTraceProduct を参照) と共にこの呼び出しを使用してください。

構文

```
unsigned int CWB_ENTRY cwbSV_SetTraceComponent(
                            cwbSV_TraceDataHandle  traceDataHandle,
                            char                  *componentID);
```

パラメーター

cwbSV_TraceDataHandle traceDataHandle - input
以前の cwbSV_CreateTraceDataHandle() の呼び出しによって戻されたハンドル。

char * componentID - input
このトレース記入項目で使用される構成要素 ID が含まれている、NULL 文字で終わるストリングを指します。注: 構成要素 ID について、最大で CWBSV_MAX_COMP_ID 文字が記録されます。これより長いストリングは切り捨てられます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この値は、対応するログ関数 cwbSV_LogTraceData() を呼び出す前に設定してください。階層には、あるプロダクト ID を定義し、その下に 1 つまたは複数の構成要素を定義することをお勧めします。

cwbSV_SetTraceProduct:

これは、System i Access for Windows のトレース記入項目で使用する API です。

目的

この関数によって、与えられたトレース・ハンドルに、固有なプロダクト ID を設定できます。ユーザーのトレース記入項目をトレース・ファイル中の他のプロダクト記入項目と区別するためにも、構成要素 ID の設定 (cwbSV_SetTraceComponent を参照) と共にこの呼び出しを使用してください。

構文

```
unsigned int CWB_ENTRY cwbSV_SetTraceProduct(
                            cwbSV_TraceDataHandle  traceDataHandle,
                            char                  *productID);
```

パラメーター

cwbSV_TraceDataHandle traceDataHandle - input
以前の cwbSV_CreateTraceDataHandle() の呼び出しによって戻されたハンドル。

char * productID - input
このトレース記入項目で使用されるプロダクト ID が含まれている、NULL 文字で終わるストリングを指します。注: プロダクト ID について、最大で CWBSV_MAX_PRODUCT_ID 文字が記録されます。これより長いストリングは切り捨てられます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この値は、対応するログ関数 cwbSV_LogTraceData() を呼び出す前に設定してください。階層には、あるプロダクト ID を定義し、その下に 1 つまたは複数の構成要素を定義することをお勧めします。

保守容易性 API のリスト: トレース・ポイントの書き込み

これらの System i Access for Windows API を使用して、トレース・ポイントをエントリー/エグジット・トレース・ファイルに書き込みます。

cwbSV_CreateTraceAPIHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は API トレース・オブジェクトを作成し、そのオブジェクトへのハンドルを戻します。この API トレース・ハンドルをユーザー・プログラムで使用すると、ユーザーの API エントリー・ポイントにおける出入りを記録することができます。

構文

```
unsigned int CWB_ENTRY cwbSV_CreateTraceAPIHandle(
                            char                *productID,
                            char                *componentID,
                            cwbSV_TraceAPIHandle *traceAPIHandle);
```

パラメーター

char * productID - input
このメッセージ記入項目で使用されるプロダクト ID が含まれている、NULL 文字で終わるストリン

グを指します。このパラメーターはオプションで、NULL の場合は productID は設定されません。注: プロダクト ID について、最大で CWBSV_MAX_PRODUCT_ID 文字が記録されます。これより長いストリングは切り捨てられます。

char * componentID - input
このメッセージ記入項目で使用される構成要素 ID が含まれている、NULL 文字で終わるストリングを指します。このパラメーターはオプションで、NULL の場合は componentID は設定されません。注: 構成要素 ID について、最大で CWBSV_MAX_COMP_ID 文字が記録されます。これより長いストリングは切り捨てられます。

cwbSV_TraceAPIHandle * traceAPIHandle - input/output
ハンドルが戻される先の cwbSV_TraceAPIHandle を指すポインター。このハンドルは、これ以降の API トレース関数呼び出しで使用する必要があります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリーが十分にないため、ハンドルを作成できません。

使用法

トレース・データ・ハンドルを使用してトレース記入項目を記録する前に、そのトレース・データ・ハンドルに固有のプロダクト ID と構成要素 ID を設定することをお勧めします。これらの ID によって、ユーザーのトレース記入項目はトレース・ファイルの他の記入項目と区別されます。

cwbSV_CreateTraceSPIHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は SPI トレース・オブジェクトを作成し、そのオブジェクトへのハンドルを戻します。この SPI トレース・ハンドルをユーザー・プログラムの中で使用すると、ユーザーの SPI エントリー・ポイントの出入りを記録することができます。

構文

```
unsigned int CWB_ENTRY cwbSV_CreateTraceSPIHandle(
                            char                *productID,
                            char                *componentID,
                            cwbSV_TraceSPIHandle *traceSPIHandle);
```

パラメーター

char * productID - input
このメッセージ記入項目で使用されるプロダクト ID が含まれている、NULL 文字で終わるストリン

グを指します。このパラメーターはオプションで、NULL の場合は productID は設定されません。注: プロダクト ID について、最大で CWBSV_MAX_PRODUCT_ID 文字が記録されます。これより長いストリングは切り捨てられます。

char * componentID - input
このメッセージ記入項目で使用される構成要素 ID が含まれている、NULL 文字で終わるストリングを指します。このパラメーターはオプションで、NULL の場合は componentID は設定されません。注: 構成要素 ID について、最大で CWBSV_MAX_COMP_ID 文字が記録されます。これより長いストリングは切り捨てられます。

cwbSV_TraceSPIHandle * traceSPIHandle - input/output
ハンドルが戻される先の cwbSV_TraceSPIHandle を指すポインター。このハンドルは、これ以降の SPI トレース関数呼び出しで使用する必要があります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリーが十分にないため、ハンドルを作成できません。

使用法

トレース・データ・ハンドルを使用してトレース記入項目を記録する前に、そのトレース・データ・ハンドルに固有のプロダクト ID と構成要素 ID を設定することをお勧めします。これらの ID によって、ユーザーのトレース記入項目はトレース・ファイルの他の記入項目と区別されます。

cwbSV_DeleteTraceAPIHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、与えられたハンドルで識別される API トレース・オブジェクトを削除します。

構文

```
unsigned int CWB_ENTRY cwbSV_DeleteTraceAPIHandle(
                           cwbSV_TraceAPIHandle traceAPIHandle);
```

パラメーター

cwbSV_TraceAPIHandle traceAPIHandle - input
以前の cwbSV_CreateTraceAPIHandle() 関数の呼び出しによって戻されたハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この呼び出しは、ハンドルが必要でなくなった際に行ってください。

cwbSV_DeleteTraceSPIHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、与えられたハンドルで識別される SPI トレース・オブジェクトを削除します。

構文

```
unsigned int CWB_ENTRY cwbSV_DeleteTraceSPIHandle(
                            cwbSV_TraceSPIHandle traceSPIHandle);
```

パラメーター

cwbSV_TraceSPIHandle traceSPIHandle - input
以前の cwbSV_CreateTraceSPIHandle() 関数の呼び出しによって戻されたハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この呼び出しは、ハンドルが必要でなくなった際に行ってください。

cwbSV_LogAPIEntry:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、API エントリー・ポイントを現在活動状態のエントリー/エグジット・トレース・ファイルへ記録します。記入項目に設定されたプロダクト ID と構成要素 ID は、データが記録された日時と共に書き込まれます。要求時に渡されるすべてのオプションのデータと共に、apiID も記録されます。

構文

```
unsigned int CWB_ENTRY cwbSV_LogAPIEntry(
                            cwbSV_TraceAPIHandle  traceAPIHandle,
```

```
                          unsigned char         apiID,
                          char                 *apiData,
                          unsigned long         apiDataLength);
```

パラメーター

cwbSV_TraceAPIHandle traceAPIHandle - input
以前の cwbSV_CreateTraceAPIHandle() の呼び出しによって戻されたハンドル。

unsigned char apiID - input
この API トレース・ポイントを、ユーザー・プログラムで記録された他の API トレース・ポイントと区別する固有の 1 バイト・コード。これらのコードの定義は、この API の呼び出し側に任されます。プロダクトの固有な構成要素ごとに、定義された範囲 (0x00 から 0xFF) を使用するというアプローチをお勧めします (例えば、構成要素ごとに 0x00 から開始する)。

char * apiData - input
このエントリー・ポイントとともに記録する追加のデータ (例えば、呼び出し側からの入力パラメーター値) が入っているバッファーを指します。このパラメーターはオプションで、アドレスが NULL またはデータ長がゼロの場合は無視されます。トレース量の決定には長さパラメーターが使用されるため、このバッファーには 2 進データを入れることができます。

unsigned long apiDataLength - input
このトレースの記入項目について記録する API データ・バッファーのバイト数を指定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この呼び出しは、対応する cwbSV_LogAPIExit() と共に使用する必要のあるものです。これらの呼び出しは、ユーザーがコーディングした API ルーチンの始めと終わりに入れることをお勧めします。他の方法としては、他のユーザーがコーディングした外部ルーチン呼び出しの際に、これらのログ関数を使用するという方法が考えられます。

cwbSV_LogAPIExit:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、API エグジット・ポイントを現在活動状態のエントリー/エグジット・トレース・ファイルへ記録します。記入項目に設定されたプロダクト ID と構成要素 ID は、データが記録された日時と共に書き込まれます。要求時に渡されるすべてのオプションのデータと共に、API ID も記録されます。

構文

```
unsigned int CWB_ENTRY cwbSV_LogAPIExit(
                          cwbSV_TraceAPIHandle  traceAPIHandle,
```

```
                          unsigned char         apiID,
                          char                 *apiData,
                          unsigned long         apiDataLength);
```

パラメーター

cwbSV_TraceAPIHandle traceAPIHandle - input
以前の cwbSV_CreateTraceAPIHandle() の呼び出しによって戻されたハンドル。

unsigned char apiID - input
この API トレース・ポイントを、ユーザー・プログラムで記録された他の API トレース・ポイントと区別する固有の 1 バイト・コード。これらのコードの定義は、この API の呼び出し側に任されます。プロダクトの固有な構成要素ごとに、定義された範囲 (0x00 から 0xFF) を使用するというアプローチをお勧めします (例えば、構成要素ごとに 0x00 から開始する)。

char * apiData - input
このエグジット・ポイントとともに記録する追加のデータ (例えば、呼び出し側に返される出力パラメーター値) が入っているバッファーを指します。このパラメーターはオプションで、アドレスが NULL またはデータ長がゼロの場合は無視されます。トレース量の決定には長さパラメーターが使用されるため、このバッファーには 2 進データを入れることができます。

unsigned long apiDataLength - input
このトレースの記入項目について記録する API データ・バッファーのバイト数を指定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この呼び出しは、対応する cwbSV_LogAPIEntry() と共に使用する必要のあるものです。これらの呼び出しは、ユーザーがコーディングした API ルーチンの始めと終わりに入れることをお勧めします。他の方法としては、他のユーザーがコーディングした外部ルーチン呼び出しの際に、これらのログ関数を使用するという方法が考えられます。

cwbSV_LogSPIEntry:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、SPI エントリー・ポイントを現在活動状態のエントリー/エグジット・トレース・ファイルへ記録します。記入項目に設定されたプロダクト ID と構成要素 ID は、データが記録された日時と共に書き込まれます。要求時に渡されるすべてのオプションのデータと共に、spiID も記録されます。

構文

```
unsigned int CWB_ENTRY cwbSV_LogSPIEntry(
                          cwbSV_TraceSPIHandle  traceSPIHandle,
```

```
                        unsigned char         spiID,
                        char                 *spiData,
                        unsigned long         spiDataLength);
```

パラメーター

cwbSV_TraceSPIHandle traceSPIHandle - input
以前の cwbSV_CreateTraceSPIHandle() の呼び出しによって戻されたハンドル。

unsigned char spiID - input
この SPI トレース・ポイントを、ユーザー・プログラムで記録された他の SPI トレース・ポイントと区別する固有の 1 バイト・コード。これらのコードの定義は、この API の呼び出し側に任されます。プロダクトの固有な構成要素ごとに、定義された範囲 (0x00 から 0xFF) を使用するというアプローチをお勧めします (例えば、構成要素ごとに 0x00 から開始する)。

char * spiData - input
このエントリー・ポイントとともに記録する追加のデータ (例えば、呼び出し側からの入力パラメーター値) が入っているバッファーを指します。このパラメーターはオプションで、アドレスが NULL またはデータ長がゼロの場合は無視されます。トレース量の決定には長さパラメーターが使用されるため、このバッファーには 2 進データを入れることができます。

unsigned long spiDataLength - input
このトレース記入項目について記録する SPI データ・バッファーのバイト数を指定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この呼び出しは、対応する cwbSV_LogSPIExit() とともに使用する必要のあるものです。これらの呼び出しは、ユーザーがコーディングした API ルーチンの始めと終わりに入れることをお勧めします。他の方法としては、他のユーザーがコーディングした外部ルーチン呼び出しの際に、これらのログ関数を使用するという方法が考えられます。

cwbSV_LogSPIExit:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、SPI エグジット・ポイントを現在活動状態のエントリー/エグジット・トレース・ファイルへ記録します。記入項目に設定されたプロダクト ID と構成要素 ID は、データが記録された日時と共に書き込まれます。要求時に渡されるすべてのオプションのデータと共に、spiID も記録されます。

構文

```
unsigned int CWB_ENTRY cwbSV_LogSPIExit(
                       cwbSV_TraceSPIHandle  traceSPIHandle,
```

```
                    unsigned char         spiID,
                    char                 *spiData,
                    unsigned long         spiDataLength);
```

パラメーター

cwbSV_TraceSPIHandle traceSPIHandle - input
以前の cwbSV_CreateTraceSPIHandle() の呼び出しによって戻されたハンドル。

unsigned char spiID - input
この SPI トレース・ポイントを、ユーザー・プログラムで記録された他の SPI トレース・ポイントと区別する固有の 1 バイト・コード。これらのコードの定義は、この API の呼び出し側に任されます。プロダクトの固有な構成要素ごとに、定義された範囲 (0x00 から 0xFF) を使用するというアプローチをお勧めします (例えば、構成要素ごとに 0x00 から開始する)。

char * spiData - input
このエグジット・ポイントとともに記録する追加のデータ (例えば、呼び出し側に返される出力パラメーター値) が入っているバッファーを指します。このパラメーターはオプションで、アドレスが NULL またはデータ長がゼロの場合は無視されます。トレース量の決定には長さパラメーターが使用されるため、このバッファーには 2 進データを入れることができます。

unsigned long spiDataLength - input
このトレース記入項目について記録する SPI データ・バッファーのバイト数を指定します。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この呼び出しは、対応する cwbSV_LogSPIEntry() と共に使用する必要のあるものです。これらの呼び出しは、ユーザーがコーディングした API ルーチンの始めと終わりに入れることをお勧めします。他の方法としては、他のユーザーがコーディングした外部ルーチン呼び出しの際に、これらのログ関数を使用するという方法が考えられます。

cwbSV_SetAPIComponent:

これは、System i Access for Windows のトレース記入項目で使用する API です。

目的

この関数によって、与えられたトレース記入項目に、固有な構成要素 ID を設定できます。ユーザーのトレース記入項目をトレース・ファイル中の他のプロダクト記入項目と区別するためにも、プロダクト ID の設定 (cwbSV_SetAPIProduct を参照) と共にこの呼び出しを使用してください。

構文

```
unsigned int CWB_ENTRY cwbSV_SetAPIComponent(
                           cwbSV_TraceAPIHandle  traceAPIHandle,
                           char                 *componentID);
```

パラメーター

cwbSV_TraceAPIHandle traceAPIHandle - input
以前の cwbSV_CreateTraceAPIHandle() の呼び出しによって戻されたハンドル。

char * componentID - input
このトレース記入項目で使用される構成要素 ID が含まれている、NULL 文字で終わるストリングを指します。注: 構成要素 ID について、最大で CWBSV_MAX_COMP_ID 文字が記録されます。これより長いストリングは切り捨てられます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この値は、対応するログ関数 cwbSV_LogAPIEntry() および cwbSV_LogAPIExit() を呼び出す前に設定してください。プロダクト ID を定義して、その下に単一または複数の機能を定義する、という階層にすることをお勧めします。

cwbSV_SetAPIProduct:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数によって、与えられたトレース・ハンドルに、固有なプロダクト ID を設定できます。ユーザーのトレース記入項目をトレース・ファイル中の他のプロダクト記入項目と区別するため、この呼び出しを構成要素 ID の設定 (cwbSV_SetAPIComponent を参照) と共に使用してください。

構文

```
unsigned int CWB_ENTRY cwbSV_SetAPIProduct(
                           cwbSV_TraceAPIHandle  traceAPIHandle,
                           char                 *productID);
```

パラメーター

cwbSV_TraceAPIHandle traceAPIHandle - input
以前の cwbSV_CreateTraceAPIHandle() の呼び出しによって戻されたハンドル。

char * productID - input
このトレース記入項目で使用されるプロダクト ID が含まれている、NULL 文字で終わるストリングを指します。注: プロダクト ID について、最大で CWBSV_MAX_PRODUCT_ID 文字が記録されます。これより長いストリングは切り捨てられます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この値は、対応するログ関数 cwbSV_LogAPIEntry() および cwbSV_LogAPIExit() を呼び出す前に設定してください。階層には、あるプロダクト ID を定義し、その下に 1 つまたは複数の構成要素を定義することをお勧めします。

cwbSV_SetSPIComponent:

これは、System i Access for Windows トレース記入項目の設定時に使用する API です。

目的

この関数によって、与えられたトレース記入項目に、固有な構成要素 ID を設定できます。ユーザーのトレース記入項目をトレース・ファイル中の他のプロダクト記入項目と区別するためにも、プロダクト ID の設定 (cwbSV_SetSPIProduct を参照) と共にこの呼び出しを使用してください。

構文

```
unsigned int CWB_ENTRY cwbSV_SetSPIComponent(
                           cwbSV_TraceSPIHandle  traceSPIHandle,
                           char                 *componentID);
```

パラメーター

cwbSV_TraceSPIHandle traceSPIHandle - input
以前の cwbSV_CreateTraceSPIHandle() の呼び出しによって戻されたハンドル。

char * componentID - input
このトレース記入項目で使用される構成要素 ID が含まれている、NULL 文字で終わるストリングを指します。注: 構成要素 ID について、最大で CWBSV_MAX_COMP_ID 文字が記録されます。これより長いストリングは切り捨てられます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この値は、対応するログ関数 cwbSV_LogAPIEntry() および cwbSV_LogAPIExit() を呼び出す前に設定してください。階層には、あるプロダクト ID を定義し、その下に 1 つまたは複数の構成要素を定義することをお勧めします。

cwbSV_SetSPIProduct:

これは、System i Access for Windows のトレース記入項目で使用する API です。

目的

この関数によって、与えられたトレース・ハンドルに、固有なプロダクト ID を設定できます。ユーザーのトレース記入項目をトレース・ファイル中の他のプロダクト記入項目と区別するためにも、構成要素 ID の設定 (cwbSV_SetSPIComponent を参照) と共にこの呼び出しを使用してください。

構文

```
unsigned int CWB_ENTRY cwbSV_SetSPIProduct(
                            cwbSV_TraceSPIHandle  traceSPIHandle,
                            char                 *productID);
```

パラメーター

cwbSV_TraceSPIHandle traceSPIHandle - input
以前の cwbSV_CreateTraceSPIHandle() の呼び出しによって戻されたハンドル。

char * productID - input
このトレース記入項目で使用されるプロダクト ID が含まれている、NULL 文字で終わるストリングを指します。注: プロダクト ID について、最大で CWBSV_MAX_PRODUCT_ID 文字が記録されます。これより長いストリングは切り捨てられます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この値は、対応するログ関数 cwbSV_LogAPIEntry() および cwbSV_LogAPIExit() を呼び出す前に設定してください。階層には、あるプロダクト ID を定義し、その下に 1 つまたは複数の構成要素を定義することをお勧めします。

保守容易性 API のリスト: サービス・ファイルの読み取り

これらの System i Access for Windows API を使用して、サービス・ファイル、サービス・ファイル・レコード、およびサービス・ファイルのヘッダー情報を読み取ります。 さらに、ヒストリー・ログ・サービス・レコード、詳細トレース・ファイル・サービス・レコード、およびエントリー/エグジット・トレース・ファイル・サービス・レコードを読み取ることができます。

cwbSV_ClearServiceFile:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルで識別される保守ファイルを削除します。

構文

```
unsigned int CWB_ENTRY cwbSV_ClearServiceFile(
                            cwbSV_ServiceFileHandle serviceFile,
                            cwbSV_ErrHandle         errorHandle);
```

パラメーター

cwbSV_ServiceFileHandle serviceFileHandle - input
以前の cwbSV_OpenServiceFile() 関数の呼び出しによって戻されたハンドル。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_FILE_IO_ERROR
ファイルはクローズできませんでした。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

なし (None)

cwbSV_CloseServiceFile:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守ファイルをクローズします。

構文

```
unsigned int CWB_ENTRY cwbSV_CloseServiceFile(
                            cwbSV_ServiceFileHandle serviceFile,
                            cwbSV_ErrHandle         errorHandle);
```

パラメーター

cwbSV_ServiceFileHandle serviceFileHandle - input
以前の cwbSV_OpenServiceFile() 関数の呼び出しによって戻されたハンドル。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、

cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_FILE_IO_ERROR
ファイルは消去できませんでした。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

なし (None)

cwbSV_CreateServiceRecHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は保守レコード・オブジェクトを作成し、そのオブジェクトへのハンドルを戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_CreateServiceRecHandle(
                            cwbSV_ServiceRecHandle *serviceRecHandle);
```

パラメーター

cwbSV_ServiceRecHandle * serviceRecHandle - input/output
ハンドルが戻される先の cwbSV_ServiceRecordHandle を指すポインター。このハンドルは、これ以降の保守レコード関数呼び出しで使用する必要があります。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ハンドル・アドレスとして NULL が渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリーが十分にないため、ハンドルを作成できません。

使用法

このハンドルをユーザー・プログラムで使用すると、オープンされた保守ファイルからレコードを読み取ってそのレコードから情報を取り出すことができます。

cwbSV_DeleteServiceRecHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、与えられたハンドルで識別される保守レコード・オブジェクトを削除します。

構文

```
unsigned int CWB_ENTRY cwbSV_DeleteServiceRecHandle(
                            cwbSV_ServiceRecHandle serviceRecHandle);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この呼び出しは、ハンドルが必要でなくなった際に行ってください。

cwbSV_GetComponent:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコード・オブジェクトの構成要素 ID を戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetComponent(
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            char                   *componentID,
                            unsigned long           componentIDLength,
                            unsigned long          *returnLength);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

char * componentID - input/output
ハンドルによって識別されるレコードに保管された、構成要素 ID を受け取るバッファーを指すポインター。

unsigned long componentIDLength - input
渡される受信バッファーの長さ。これには、NULL 終了文字を入れるためのスペースを含める必要があります。バッファーが小さすぎると、値が切り捨てられて CWB_BUFFER_OVERFLOW と returnLength が設定されます。注: 推奨サイズは CWBSV_MAX_COMP_ID です。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に、出力ストリングを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

このルーチンを呼び出す前に、読み取り関数呼び出しによって保守レコード・ハンドルを取得する必要があります。これを実行しないと、NULL ストリングが戻されます。この関数は、すべての保守レコード・タイプで有効です。

cwbSV_GetDateStamp:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコードの (地域化された形式の) 日付スタンプを戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetDateStamp(
                           cwbSV_ServiceRecHandle  serviceRecHandle,
                           char                   *dateStamp,
                           unsigned long           dateStampLength,
                           unsigned long          *returnLength);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

char * dateStamp - input/output
ハンドルによって識別されるレコードに保管された、日付スタンプを受け取るバッファーを指すポインター。

unsigned long dateStampLength - input
渡される受信バッファーの長さ。これには、NULL 終了文字を入れるためのスペースを含める必要があります。バッファーが小さすぎると、値が切り捨てられて CWB_BUFFER_OVERFLOW と returnLength が設定されます。注: 推奨サイズは CWBSV_MAX_DATE_VALUE です。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に、出力ストリングを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

このルーチンを呼び出す前に、読み取り関数呼び出しによって保守レコード・ハンドルを取得する必要があります。これを実行しないと、NULL ストリングが戻されます。この関数は、すべての保守レコード・タイプで有効です。

cwbSV_GetMaxRecordSize:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたファイル・ハンドルによって識別される保守ファイルの中の最大レコードのサイズ (バイト数) を戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetMaxRecordSize(
                            cwbSV_ServiceFileHandle  serviceFile,
                            unsigned long           *maxRecordSize);
```

パラメーター

cwbSV_ServiceFileHandle serviceFileHandle - input
以前の cwbSV_OpenServiceFile 関数の呼び出しによって戻されたハンドル。

unsigned long * recordCount - input/output
ファイルの中の最大レコード・サイズを受け取る変数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

なし (None)

cwbSV_GetMessageText:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコード・オブジェクトのメッセージ・テキスト部分を戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetMessageText(
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            char                   *messageText,
                            unsigned long           messageTextLength,
                            unsigned long          *returnLength);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

char * messageText - input/output
ハンドルによって識別されるレコードに保管された、メッセージ・テキストを受け取るバッファーを指すポインター。

unsigned long messageTextLength - input
渡される受信バッファーの長さ。バッファーが小さすぎると、値が切り捨てられて CWB_BUFFER_OVERFLOW と returnLength が設定されます。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に出力データを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_INVALID_RECORD_TYPE
タイプが CWBSV_MESSAGE_REC ではありません。

使用法

レコード・タイプが CWBSV_MESSAGE_REC ではない場合、戻りコード CWBSV_INVALID_RECORD_TYPE が戻されます。(注: cwbSV_GetServiceType() は現行のレコード・タイプを戻します。)

cwbSV_GetProduct:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコード・オブジェクトのプロダクト ID 値を戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetProduct(
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            char                   *productID,
                            unsigned long           productIDLength,
                            unsigned long          *returnLength);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

char * productID - input/output
ハンドルによって識別されるレコードに保管された、プロダクト ID を受け取るバッファーを指すポインター。

unsigned long productIDLength - input
渡される受信バッファーの長さ。これには、NULL 終了文字を入れるためのスペースを含める必要があります。バッファーが小さすぎると、値が切り捨てられて CWB_BUFFER_OVERFLOW と returnLength が設定されます。注: 推奨サイズは CWBSV_MAX_PRODUCT_ID です。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に、出力ストリングを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

このルーチンを呼び出す前に、読み取り関数呼び出しによって保守レコード・ハンドルを取得する必要があります。これを実行しないと、NULL ストリングが戻されます。この関数は、すべての保守レコード・タイプで有効です。

cwbSV_GetRecordCount:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたファイル・ハンドルによって識別される保守ファイル中のレコード数の合計が戻されます。

構文

```
unsigned int CWB_ENTRY cwbSV_GetRecordCount(
                           cwbSV_ServiceFileHandle  serviceFile,
                           unsigned long           *recordCount);
```

パラメーター

cwbSV_ServiceFileHandle serviceFileHandle - input
以前の cwbSV_OpenServiceFile 関数の呼び出しによって戻されたハンドル。

unsigned long * recordCount - input/output
ファイルの中のレコード数の合計を受け取る変数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

なし (None)

cwbSV_GetServiceFileName:

これは、System i Access for Windows 製品で使用する API です。

目的

特定のファイル・タイプについて保守レコードが記録されている場所の完全修飾パス名および完全修飾ファイル名を戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetServiceFileName(
                            cwbSV_ServiceFileType  serviceFileType,
                            char                  *fileName,
                            unsigned long          fileNameLength,
                            unsigned long         *returnLength);
```

パラメーター

cwbSV_ServiceFileType serviceFileType - input
入手したい保守ファイル名を示す値。- CWBSV_HISTORY_LOG - CWBSV_PROBLEM_LOG - CWBSV_DETAIL_TRACE_FILE - CWBSV_ENTRY_EXIT_TRACE_FILE

char * fileName - input/output
要求した関連の保守ファイル名を受け取るバッファーを指すポインター。

unsigned long fileNameLength - input
渡される受信バッファーの長さ。これには、NULL 終了文字を入れるためのスペースを含める必要があります。バッファーが小さすぎると、値が切り捨てられて CWB_BUFFER_OVERFLOW と returnLength が設定されます。注: 推奨サイズは CWBSV_MAX_FILE_PATH です。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に、出力ストリングを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWBSV_INVALID_FILE_TYPE
渡されたファイル・タイプは使用できないタイプです。

使用法

戻されるファイル名ストリングは、cwbSV_OpenServiceFile() ルーチンへの入力として使用することができます。

cwbSV_GetServiceType:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコードのタイプ (トレース、メッセージ、エントリー/エグジットなど) を戻します。注: この関数を呼び出す前に、読み取り関数呼び出しによって保守レコードを取得する必要があります。

構文

```
unsigned int CWB_ENTRY cwbSV_GetServiceType(
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            cwbSV_ServiceRecType   *serviceType,
                            cwbSV_ErrHandle         errorHandle);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

cwbSV_ServiceRecType * serviceType - output
serviceType を戻す先である cwbSV_ServiceRecType を指すポインター。- CWBSV_MESSAGE_REC - CWBSV_PROBLEM_REC - CWBSV_DATA_TRACE_REC - CWBSV_API_TRACE_REC - CWBSV_SPI_TRACE_REC

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_INVALID_RECORD_TYPE
無効なレコード・タイプが検出されました。

使用法

このルーチンを呼び出す前に、読み取り関数呼び出しによって保守レコード・ハンドルを取得する必要があります。これを実行しないと、CWBSV_INVALID_RECORD_TYPE が戻されます。

cwbSV_GetTimeStamp:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコードの (地域化された形式の) タイム・スタンプを戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetTimeStamp(
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            char                   *timeStamp,
                            unsigned long           timeStampLength,
                            unsigned long          *returnLength);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

char * timeStamp - input/output
ハンドルによって識別されるレコードに保管された、タイム・スタンプを受け取るバッファーを指すポインター。

unsigned long timeStampLength - input
渡される受信バッファーの長さ。これには、NULL 終了文字を入れるためのスペースを含める必要があります。バッファーが小さすぎると、値が切り捨てられて CWB_BUFFER_OVERFLOW と returnLength が設定されます。注: 推奨サイズは CWBSV_MAX_TIME_VALUE です。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に、出力ストリングを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

このルーチンを呼び出す前に、読み取り関数呼び出しによって保守レコード・ハンドルを取得する必要があります。これを実行しないと、NULL ストリングが戻されます。この関数は、すべての保守レコード・タイプで有効です。

cwbSV_GetTraceData:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコード・オブジェクトのトレース・データ部分を戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetTraceData(
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            char                   *traceData,
                            unsigned long           traceDataLength,
                            unsigned long          *returnLength);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

char * traceData - input/output
ハンドルによって識別されるレコードに保管された、トレース・データを受け取るバッファーを指すポインター。注: 戻されるデータは 2 進数であるため、ASCIIZ ストリングとしては戻されません。

unsigned long traceDataLength - input
渡される受信バッファーの長さ。バッファーが小さすぎると、値が切り捨てられて CWB_BUFFER_OVERFLOW と returnLength が設定されます。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に出力データを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_INVALID_RECORD_TYPE
タイプが CWBSV_DATA_TRACE_REC ではありません。

使用法

レコード・タイプが CWBSV_TRACE_DATA_REC ではない場合、戻りコード CWBSV_INVALID_RECORD_TYPE が戻されます。(注: cwbSV_GetServiceType() は現行のレコード・タイプを戻します。)

cwbSV_GetTraceAPIData:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコードの API トレース・データ部分を戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetTraceAPIData(
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            char                   *apiData,
                            unsigned long           apiDataLength,
                            unsigned long          *returnLength);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

char * apiData - input/output
ハンドルによって識別されるレコードに保管された API トレース・データを受け取るバッファーを指すポインター。注: 戻されるデータは 2 進数であるため、ASCIIZ ストリングとしては戻されません。

unsigned long apiDataLength - input
渡される受信バッファーの長さ。バッファーが小さすぎると、値が切り捨てられて CWB_BUFFER_OVERFLOW と returnLength が設定されます。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に出力データを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_INVALID_RECORD_TYPE
タイプが CWBSV_API_REC ではありません。

使用法

レコード・タイプが CWBSV_API_TRACE_REC ではない場合、戻りコード CWBSV_INVALID_RECORD_TYPE が戻されます。(注: cwbSV_GetServiceType() は現行のレコード・タイプを戻します。)

cwbSV_GetTraceAPIID:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコード・オブジェクトの API イベント ID を戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetTraceAPIID(
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            char                   *apiID);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

char * apiID - input/output
API イベント ID を受け取る、1 バイトのフィールドを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_INVALID_RECORD_TYPE
タイプが CWBSV_API_REC ではありません。

使用法

レコード・タイプが CWBSV_API_TRACE_REC ではない場合、戻りコード CWBSV_INVALID_RECORD_TYPE が戻されます。(注: cwbSV_GetServiceType() は現行のレコード・タイプを戻します。)

cwbSV_GetTraceAPIType:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコード・オブジェクトの API イベント・タイプを戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetTraceAPIType(
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            cwbSV_EventType        *eventType,
                            cwbSV_ErrHandle         errorHandle);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

cwbSV_EventType * eventType - output
eventType を戻す先である cwbSV_EventType を指すポインター。- CWBSV_ENTRY_POINT - CWBSV_EXIT_POINT

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_INVALID_RECORD_TYPE
タイプが CWBSV_API_REC ではありません。

CWBSV_INVALID_EVENT_TYPE
使用できないイベント・タイプが見付かりました。

使用法

レコード・タイプが CWBSV_API_TRACE_REC ではない場合、戻りコード CWBSV_INVALID_RECORD_TYPE が戻されます。(注: cwbSV_GetServiceType() は現行のレコード・タイプを戻します。)

cwbSV_GetTraceSPIData:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルで識別される保守レコードの SPI トレース・データ部分を戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetTraceSPIData(
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            char                   *spiData,
                            unsigned long           spiDataLength,
                            unsigned long          *returnLength);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

char * spiData - input/output
ハンドルによって識別されるレコードに保管された、SPI トレース・データを受け取るバッファーを指すポインター。注: 戻されるデータは 2 進数であるため、ASCIIZ ストリングとしては戻されません。

unsigned long spiDataLength - input
渡される受信バッファーの長さ。バッファーが小さすぎると、値が切り捨てられて CWB_BUFFER_OVERFLOW と returnLength が設定されます。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に出力データを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_INVALID_RECORD_TYPE
タイプが CWBSV_SPI_TRACE_REC ではありません。

使用法

レコード・タイプが CWBSV_SPI_TRACE_REC ではない場合、戻りコード CWBSV_INVALID_RECORD_TYPE が戻されます。(注: cwbSV_GetServiceType() は現行のレコード・タイプを戻します。)

cwbSV_GetTraceSPIID:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコード・オブジェクトの SPI イベント ID を戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetTraceSPIID(
                           cwbSV_ServiceRecHandle  serviceRecHandle,
                           char                   *spiID);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

char * spiID - input/output
SPI イベント ID を受け取る 1 バイトのフィールドを指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_INVALID_RECORD_TYPE
タイプが CWBSV_SPI_TRACE_REC ではありません。

使用法

レコード・タイプが CWBSV_SPI_TRACE_REC ではない場合、戻りコード CWBSV_INVALID_RECORD_TYPE が戻されます。(注: cwbSV_GetServiceType() は現行のレコード・タイプを戻します。)

cwbSV_GetTraceSPIType:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたハンドルによって識別される保守レコード・オブジェクトの SPI イベント・タイプを戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetTraceSPIType(
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            cwbSV_EventType        *eventType,
                            cwbSV_ErrHandle         errorHandle);
```

パラメーター

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

cwbSV_EventType * eventType - output
eventType を戻す先である cwbSV_EventType を指すポインター。- CWBSV_ENTRY_POINT - CWBSV_EXIT_POINT

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、

cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_INVALID_RECORD_TYPE
タイプが CWBSV_SPI_TRACE_REC ではありません。

CWBSV_INVALID_EVENT_TYPE
使用できないイベント・タイプが見付かりました。

使用法

レコード・タイプが CWBSV_SPI_TRACE_REC ではない場合、戻りコード CWBSV_INVALID_RECORD_TYPE が戻されます。(注: cwbSV_GetServiceType() は現行のレコード・タイプを戻します。)

cwbSV_OpenServiceFile:

これは、System i Access for Windows 製品で使用する API です。

目的

読み取りアクセスのために指定された保守ファイル (ヒストリー・ログ、トレース・ファイルなど) をオープンし、そのハンドルを戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_OpenServiceFile(
                            char                    *serviceFileName,
                            cwbSV_ServiceFileHandle *serviceFileHandle,
                            cwbSV_ErrHandle          errorHandle);
```

パラメーター

char * serviceFileName - input
オープンする保守ファイルの完全修飾名 (例えば、c:¥path¥filename.ext) が含まれているバッファーを指します。

cwbSV_ServiceFileHandle * serviceFileHandle - input/output
ハンドルが戻される先の cwbSV_ServiceFileHandle を指すポインター。このハンドルは、これ以降の保守ファイル関数呼び出しで使用する必要があります。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、

cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ハンドル・アドレスとして NULL が渡されました。

CWB_FILE_IO_ERROR
ファイルはオープンできませんでした。

CWB_NOT_ENOUGH_MEMORY
メモリーが十分にないため、ハンドルを作成できません。

使用法

なし (None)

cwbSV_ReadNewestRecord:

これは、System i Access for Windows 製品で使用する API です。

目的

保守ファイル中の最新レコードを、与えられたレコード・ハンドルに読み取ります。以降の呼び出しで、このレコードに保管されている情報を検索することができます (例えば、GetProduct()、 GetDateStamp() など)。注: このレコードは、ファイルの中で最新の日付と時刻のスタンプを持つレコードです。

構文

```
unsigned int CWB_ENTRY cwbSV_ReadNewestRecord(
                           cwbSV_ServiceFileHandle serviceFileHandle,
                           cwbSV_ServiceRecHandle  serviceRecHandle,
                           cwbSV_ErrHandle         errorHandle);
```

パラメーター

cwbSV_ServiceFileHandle serviceFileHandle - input
以前の cwbSV_OpenServiceFile 関数の呼び出しによって戻されたハンドル。

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_END_OF_FILE
ファイルの終わりに達しました。

CWB_FILE_IO_ERROR
レコードが読み取れません。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この読み取りは、ファイル終わり標識が戻されるまで、一連の cwbSV_ReadPrevRecord() 呼び出しを出す前の「準備タイプ」の読み取りとして使用することが考えられます。

cwbSV_ReadNextRecord:

これは、System i Access for Windows 製品で使用する API です。

目的

保守ファイル中の次のレコードを、与えられたレコード・ハンドルに読み取ります。以降の呼び出しで、このレコードに保管されている情報を検索することができます (例えば、GetProduct()、 GetDateStamp() など)。

構文

```
unsigned int CWB_ENTRY cwbSV_ReadNextRecord(
                           cwbSV_ServiceFileHandle serviceFileHandle,
                           cwbSV_ServiceRecHandle  serviceRecHandle,
                           cwbSV_ErrHandle         errorHandle);
```

パラメーター

cwbSV_ServiceFileHandle serviceFileHandle - input
以前の cwbSV_OpenServiceFile 関数の呼び出しによって戻されたハンドル。

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_END_OF_FILE
ファイルの終わりに達しました。

CWB_FILE_IO_ERROR
レコードが読み取れません。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この読み取りは通常、いったん準備読み取り ReadOldestRecord() を実行してから使用します。

cwbSV_ReadOldestRecord:

これは、System i Access for Windows 製品で使用する API です。

目的

保守ファイル中の最も古いレコードを、与えられたレコード・ハンドルに読み取ります。以降の呼び出しで、このレコードに保管されている情報を検索することができます (例えば、GetProduct()、GetDateStamp() など)。注: このレコードは、ファイルの中で最も古い日付と時刻のスタンプを持つレコードです。

構文

```
unsigned int CWB_ENTRY cwbSV_ReadOldestRecord(
                            cwbSV_ServiceFileHandle serviceFileHandle,
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            cwbSV_ErrHandle         errorHandle);
```

パラメーター

cwbSV_ServiceFileHandle serviceFileHandle - input
以前の cwbSV_OpenServiceFile 関数の呼び出しによって戻されたハンドル。

cwbSV_ServiceRecHandle serviceRecHandle - input
以前の cwbSV_CreateServiceRecHandle() 関数の呼び出しによって戻されたハンドル。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_END_OF_FILE
ファイルの終わりに達しました。

CWB_FILE_IO_ERROR
レコードが読み取れません。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この読み取りは、ファイル終わり標識が戻されるまで、一連の cwbSV_ReadNextRecord() 呼び出しを出す前の「準備タイプ」の読み取りとして使用することが考えられます。

cwbSV_ReadPrevRecord:

これは、System i Access for Windows 製品で使用する API です。

目的

保守ファイル中の 1 行前のレコードを、与えられたレコード・ハンドルに読み取ります。以降の呼び出しで、このレコードに保管されている情報を検索することができます (例えば、GetProduct()、GetDateStamp() など)。

構文

```
unsigned int CWB_ENTRY cwbSV_ReadPrevRecord(
                            cwbSV_ServiceFileHandle serviceFileHandle,
                            cwbSV_ServiceRecHandle  serviceRecHandle,
                            cwbSV_ErrHandle         errorHandle);
```

パラメーター

cwbSV_ServiceFileHandle serviceFileHandle - input
以前の cwbSV_OpenServiceFile 関数の呼び出しによって戻されたハンドル。

cwbSV_ErrHandle errorHandle - output
戻されたメッセージはすべてこのオブジェクトに書き込まれます。このオブジェクトは、cwbSV_CreateErrHandle API を使用して作成されます。メッセージは、cwbSV_GetErrText API を介して検索することができます。パラメーターがゼロに設定されている場合は、メッセージは検索されません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_END_OF_FILE
ファイルの終わりに達しました。

CWB_FILE_IO_ERROR
レコードが読み取れません。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この読み取りは通常、準備読み取り ReadNewestRecord() を実行してから使用します。

保守容易性 API のリスト: メッセージ・テキストの検索

これらの System i Access for Windows API を使用して、エラー・ハンドルに関連したメッセージ・テキストを検索します。

cwbSV_CreateErrHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数はエラー・メッセージ・オブジェクトを作成し、そのオブジェクトへのハンドルを戻します。このエラー・ハンドルは、そのハンドルをサポートする System i Access for Windows API に渡すことができます。これらの API のいずれかでエラーが起こった場合、エラー・ハンドルを使用して、その API エラーに関連するエラー・メッセージ・テキストを検索することができます。

構文

```
unsigned int CWB_ENTRY cwbSV_CreateErrHandle(
                            cwbSV_ErrHandle *errorHandle);
```

パラメーター

cwbSV_ErrHandle *errorHandle - input/output
ハンドルが戻される先の cwbSV_ErrHandle を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
ハンドル・アドレスとして NULL が渡されました。

CWB_NOT_ENOUGH_MEMORY
メモリーが十分にないため、ハンドルを作成できません。

使用法

なし (None)

cwbSV_DeleteErrHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

この関数は、与えられたハンドルで識別されるエラー・メッセージ・オブジェクトを削除します。

構文

```
unsigned int CWB_ENTRY cwbSV_DeleteErrHandle(
                            cwbSV_ErrHandle errorHandle);
```

パラメーター

cwbSV_ErrHandle errorHandle - output
以前の cwbSV_CreateErrHandle() 関数の呼び出しによって戻されたハンドル。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

この呼び出しは、ハンドルが必要でなくなった際に行ってください。

cwbSV_GetErrClass:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたエラー・ハンドルによって識別される、最上位の (最新の) エラーに関連するメッセージ・クラスを戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetErrClass(
                            cwbSV_ErrHandle  errorHandle,
                            unsigned long   *errorClass);
```

パラメーター

cwbSV_ErrHandle errorHandle - input
以前の cwbSV_CreateErrHandle() 関数の呼び出しによって戻されたハンドル。

unsigned long * errorClass - output
ハンドルによって識別されるエラーに保管された、エラー・クラスを受け取る変数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_NO_ERROR_MESSAGES
エラー・ハンドルに関連するエラー・メッセージがありません。

使用法

なし (None)

cwbSV_GetErrClassIndexed:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたエラー指標に関連するメッセージ・クラスを戻します。指標値が 1 であれば、エラー・ハンドルに関連する最下位の (例えば、最も古い) メッセージを検索します。指標値が「cwbSV_GetErrCount() が戻した errorCount」であれば、エラー・ハンドルに関連する最上位の (例えば、最新の) メッセージを検索します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetErrClassIndexed(
                            cwbSV_ErrHandle  errorHandle,
                            unsigned long    errorIndex,
                            unsigned long   *errorClass);
```

パラメーター

cwbSV_ErrHandle errorHandle - input
以前の cwbSV_CreateErrHandle() 関数の呼び出しによって戻されたハンドル。

unsigned long errorIndex - input
複数のエラーがエラー・ハンドルに関連する場合、どのエラー・テキストを戻すかを示す指標値。

unsigned long * errorClass - output
指標によって識別されるエラーに保管されたエラー・クラスを受け取る変数を指すポインター。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_NO_ERROR_MESSAGES
エラー・ハンドルに関連するエラー・メッセージがありません。

使用法

有効な指標値は 1 から cwbSV_GetErrCount() の戻り値までです。1 よりも小さい指標値は、 1 が渡された場合と同様に動作します。 cwbSV_GetErrCount() よりも大きい指標値は、 errorCount が渡された場合と同様に動作します。

cwbSV_GetErrCount:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたエラー・ハンドルに関連するメッセージ数を戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetErrCount(
                           cwbSV_ErrHandle  errorHandle,
                           unsigned long   *errorCount);
```

パラメーター

cwbSV_ErrHandle errorHandle - input
以前の cwbSV_CreateErrHandle() 関数の呼び出しによって戻されたハンドル。

unsigned long * errorCount - input/output
このエラー・ハンドルに関連するメッセージ数を受け取る変数を指すポインター。ゼロが戻された場合は、エラー・ハンドルに関連するエラーはありません。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

使用法

なし (None)

cwbSV_GetErrFileName:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたエラー・ハンドルに追加される最上位 (最新) メッセージのメッセージ・ファイル名を戻します。このメッセージ属性は、System i メッセージにのみ関連します。このファイル名は、メッセージが収められている System i メッセージ・ファイルの名前になります。

構文

```
unsigned int CWB_ENTRY cwbSV_GetErrFileName(
                            cwbSV_ErrHandle  errorHandle,
                            char            *fileName,
                            unsigned long    fileNameLength,
                            unsigned long   *returnLength);
```

パラメーター

cwbSV_ErrHandle errorHandle - input
以前の cwbSV_CreateErrHandle() API への呼び出しによって戻されたハンドル。

char * fileName - input/output
ハンドルによって識別されるエラーに保管されたメッセージ・ファイル名を受け取るバッファーを指すポインター。戻される値は ASCIIZ ストリングです。

unsigned long fileNameLength - input
渡される受信バッファーの長さ。これには、NULL 終了文字を入れるためのスペースを含める必要があります。バッファーが小さすぎると、値が切り捨てられ、CWB_BUFFER_OVERFLOW と returnLength が設定されます。注: 推奨サイズは、CWBSV_MAX_MSGFILE_NAME です。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に、出力ストリングを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_NO_ERROR_MESSAGES
エラー・ハンドルにメッセージは入っていません。

CWBSV_ATTRIBUTE_NOT_SET
属性が現行メッセージ内に設定されていません。

使用法

cwbRC_CallPgm() API および cwbRC_RunCmd() API の使用時に、System i メッセージがエラー・ハンドルに追加されることがあります。そのような場合には、この API を使用して、エラー・ハンドルに収められた System i メッセージのメッセージ・ファイル名を検索できます。そのメッセージのメッセージ・ファイル名属性がなければ、戻りコード CWBSV_ATTRIBUTE_NOT_SET が戻されます。

cwbSV_GetErrFileNameIndexed:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられた指標によって識別されるメッセージのメッセージ・ファイル名を戻します。このメッセージ属性は、System i から戻されるメッセージにのみ関連します。このファイル名は、メッセージが収められている System i メッセージ・ファイルの名前になります。

構文

```
unsigned int CWB_ENTRY cwbSV_GetErrFileNameIndexed(
                            cwbSV_ErrHandle  errorHandle,
                            unsigned long    index,
                            char            *fileName,
                            unsigned long    fileNameLength,
                            unsigned long   *returnLength);
```

パラメーター

cwbSV_ErrHandle errorHandle - input
以前の cwbSV_CreateErrHandle() API への呼び出しによって戻されたハンドル。

unsigned long index - input
複数のエラーがエラー・ハンドルに関連する場合、戻すメッセージ・ファイル名を示す指標値。指標の有効な範囲は、1 から「エラー・ハンドルに入っているメッセージ数」です。メッセージ数は、cwbSV_GetErrCount() API を呼び出して入手することができます。

char * fileName - input/output
指標によって識別されるエラーに保管されたメッセージ・ファイル名を受け取るバッファーを指すポインター。戻される値は ASCIIZ ストリングです。

unsigned long fileNameLength - input
渡される受信バッファーの長さ。これには、NULL 終了文字を入れるためのスペースを含める必要があります。バッファーが小さすぎると、値が切り捨てられ、CWB_BUFFER_OVERFLOW と returnLength が設定されます。注: 推奨サイズは、CWBSV_MAX_MSGFILE_NAME です。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に、出力ストリングを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_NO_ERROR_MESSAGES
エラー・ハンドルにメッセージは入っていません。

CWBSV_ATTRIBUTE_NOT_SET
属性が現行メッセージ内に設定されていません。

使用法

cwbRC_CallPgm() API および cwbRC_RunCmd() API の使用時に、System i メッセージがエラー・ハンドルに追加されることがあります。そのような場合には、この API を使用して、エラー・ハンドルに収められた System i メッセージのメッセージ・ファイル名を検索できます。そのメッセージのメッセージ・ファイル名属性がなければ、戻りコード CWBSV_ATTRIBUTE_NOT_SET が戻されます。指標値 1 では、エラー・ハンドル内の最下位 (つまり最も古い) メッセージを処理します。cwbSV_GetErrCount() API によって戻されたカウントと同じ指標値では、エラー・ハンドル内の最上位 (つまり最新) メッセージを処理します。1 よりも小さい指標値を指定すると、1 が渡された場合のように動作します。エラー・ハンドルに入っているメッセージ数より大きい指標値を指定すると、cwbSV_GetErrCount() API から戻されたカウント値が渡された場合のように動作します。

cwbSV_GetErrLibName:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたエラー・ハンドルに追加される最上位 (つまり、最新) メッセージのメッセージ・ファイル・ライブラリー名を戻します。このメッセージ属性は、System i から戻されるメッセージにのみ関連します。このライブラリー名は、メッセージのメッセージ・ファイルが収められている System i ライブラリーの名前になります。

構文

```
unsigned int CWB_ENTRY cwbSV_GetErrLibName(
                            cwbSV_ErrHandle  errorHandle,
                            char            *libraryName,
                            unsigned long    libraryNameLength,
                            unsigned long   *returnLength);
```

パラメーター

cwbSV_ErrHandle errorHandle - input
以前の cwbSV_CreateErrHandle() API への呼び出しによって戻されたハンドル。

char * libraryName - input/output
ハンドルによって識別されるエラーで保管されたメッセージ・ファイル・ライブラリー名を受け取るバッファーを指すポインター。戻される値は ASCIIZ ストリングです。

unsigned long libraryNameLength - input
渡される受信バッファーの長さ。これには、NULL 終了文字を入れるためのスペースを含める必要があります。バッファーが小さすぎると、値が切り捨てられ、CWB_BUFFER_OVERFLOW と returnLength が設定されます。注: 推奨サイズは、CWBSV_MAX_MSGFILE_LIBR です。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に、出力ストリングを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_NO_ERROR_MESSAGES
エラー・ハンドルにメッセージは入っていません。

CWBSV_ATTRIBUTE_NOT_SET
属性が現行メッセージ内に設定されていません。

使用法

cwbRC_CallPgm() API および cwbRC_RunCmd() API の使用時に、System i メッセージがエラー・ハンドルに追加されることがあります。そのような場合には、この API を使用して、エラー・ハンドルに収められた System i メッセージのメッセージ・ファイル・ライブラリーの名前を検索できます。そのメッセージのメッセージ・ファイル・ライブラリー名属性がなければ、戻りコード CWBSV_ATTRIBUTE_NOT_SET が戻されます。

cwbSV_GetErrLibNameIndexed:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられた指標によって識別されるメッセージのメッセージ・ファイル・ライブラリー名を戻します。このメッセージ属性は、System i から戻されるメッセージにのみ関連します。このライブラリー名は、メッセージのメッセージ・ファイルが収められている System i ライブラリーの名前になります。

構文

```
unsigned int CWB_ENTRY cwbSV_GetErrLibNameIndexed(
                           cwbSV_ErrHandle  errorHandle,
                           unsigned long    index,
                           char            *libraryName,
                           unsigned long    libraryNameLength,
                           unsigned long   *returnLength);
```

パラメーター

cwbSV_ErrHandle errorHandle - input
以前の cwbSV_CreateErrHandle() API への呼び出しによって戻されたハンドル。

unsigned long index - input
複数のエラーがエラー・ハンドルに関連する場合、戻すメッセージ・ファイル・ライブラリー名を示す指標値。指標の有効な範囲は、1 から「エラー・ハンドルに入っているメッセージ数」です。メッセージ数は、cwbSV_GetErrCount() API を呼び出して入手することができます。

char * libraryName - input/output
指標によって識別されるエラーに保管されたメッセージ・ファイル・ライブラリー名を受け取るバッファーを指すポインター。戻される値は ASCIIZ ストリングです。

unsigned long libraryNameLength - input
渡される受信バッファーの長さ。これには、NULL 終了文字を入れるためのスペースを含める必要があります。バッファーが小さすぎると、値が切り捨てられ、CWB_BUFFER_OVERFLOW と returnLength が設定されます。注: 推奨サイズは、CWBSV_MAX_MSGFILE_LIBR です。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に、出力ストリングを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_NO_ERROR_MESSAGES
エラー・ハンドルにメッセージは入っていません。

CWBSV_ATTRIBUTE_NOT_SET
属性が現行メッセージ内に設定されていません。

使用法

cwbRC_CallPgm() API および cwbRC_RunCmd() API の使用時に、System i メッセージがエラー・ハンドルに追加されることがあります。そのような場合には、この API を使用して、エラー・ハンドルに収められた System i メッセージのメッセージ・ファイル・ライブラリーの名前を検索できます。そのメッセージのメッセージ・ファイル・ライブラリー名属性がなければ、戻りコード CWBSV_ATTRIBUTE_NOT_SET が戻されます。指標値 1 では、エラー・ハンドル内の最下位 (つまり最も古い) メッセージを処理します。cwbSV_GetErrCount() API によって戻されたカウントと同じ指標値では、エラー・ハンドル内の最上位 (つまり最新) メッセージを処理します。1 よりも小さい指標値を指定すると、1 が渡された場合のように動作します。エラー・ハンドルに入っているメッセージ数より大きい指標値を指定すると、cwbSV_GetErrCount() API から戻されたカウント値が渡された場合のように動作します。

cwbSV_GetErrSubstText:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたエラー・ハンドルで識別される最上位 (最新) のメッセージのメッセージ置換データを戻します。このメッセージ属性は、System i から戻されるメッセージにのみ関連します。置換データは、メッセージに対して定義されている置換変数フィールドに挿入されます。

構文

```
unsigned int CWB_ENTRY cwbSV_GetErrSubstText(
                            cwbSV_ErrHandle  errorHandle,
                            char            *substitutionData,
                            unsigned long    substitutionDataLength,
                            unsigned long   *returnLength);
```

パラメーター

cwbSV_ErrHandle errorHandle - input
以前の cwbSV_CreateErrHandle() API への呼び出しによって戻されたハンドル。

char * substitutionData - input/output
ハンドルで識別されるメッセージの置換データを受け取るバッファーを指すポインター。注: 戻されるデータは 2 進数であるため、ASCIIZ ストリングとしては戻されません。置換データに含まれる文字ストリングは、すべて EBCDIC 値として戻されます。

unsigned long substitutionDataLength - input
渡される受信バッファーの長さ。バッファーが小さすぎると、値が切り捨てられ、CWB_BUFFER_OVERFLOW と returnLength が設定されます。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に出力データを保持するために必要なバイト数を保管するための戻りアドレス。これは、正常に終了した時点で戻される出力データの実際のバイト数に設定されます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_NO_ERROR_MESSAGES
エラー・ハンドルにメッセージは入っていません。

CWBSV_ATTRIBUTE_NOT_SET
属性が現行メッセージ内に設定されていません。

使用法

cwbRC_CallPgm() API および cwbRC_RunCmd() API の使用時に、System i メッセージがエラー・ハンドルに追加されることがあります。そのような場合には、この API を使用して、エラー・ハンドルに収められた System i メッセージの置換データを検索できます。メッセージの置換データがない場合は、戻りコード CWBSV_ATTRIBUTE_NOT_SET が戻されます。戻りコードが CWB_OK である場合は、returnLength パラメーターを使用して、置換データ内に戻された実際のバイト数を判別してください。この API で戻された置換データを、後続のホスト検索メッセージ API 呼び出し (QSYS/QMHRTVM) で使用して、置換デ

ータの形式を検索したり、置換データを追加した 2 次ヘルプ・テキストを戻したりすることができます。ホスト API は、cwbRC_CallPgm() API を使用して呼び出します。

cwbSV_GetErrSubstTextIndexed:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられた指標で識別されるメッセージのメッセージ置換データを戻します。このメッセージ属性は、System i から戻されるメッセージにのみ関連します。置換データは、メッセージに対して定義されている置換変数フィールドに挿入されるデータです。

構文

```
unsigned int CWB_ENTRY cwbSV_GetErrSubstTextIndexed(
                            cwbSV_ErrHandle  errorHandle,
                            unsigned long    index,
                            char            *substitutionData,
                            unsigned long    substitutionDataLength,
                            unsigned long   *returnLength);
```

パラメーター

cwbSV_ErrHandle errorHandle - input
以前の cwbSV_CreateErrHandle() API への呼び出しによって戻されたハンドル。

unsigned long index - input
複数のエラーがエラー・ハンドルに関連する場合に戻す置換データを示す指標値。指標の有効な範囲は、1 から「エラー・ハンドルに入っているメッセージ数」です。メッセージ数は、cwbSV_GetErrCount() API を呼び出して入手することができます。

char * substitutionData - input/output
指標で識別されるエラーに保管された置換データを受け取るバッファーを指すポインター。注: 戻されるデータは 2 進数であるため、ASCIIZ ストリングとしては戻されません。置換データに含まれる文字ストリングは、すべて EBCDIC 値として戻されます。

unsigned long substitutionDataLength - input
渡される受信バッファーの長さ。バッファーが小さすぎると、値が切り捨てられ、CWB_BUFFER_OVERFLOW と returnLength が設定されます。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に出力データを保持するために必要なバイト数を保管するための戻りアドレス。これは、正常に終了した時点で戻される出力データの実際のバイト数に設定されます。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_NO_ERROR_MESSAGES
エラー・ハンドルにメッセージは入っていません。

CWBSV_ATTRIBUTE_NOT_SET
属性が現行メッセージ内に設定されていません。

使用法

cwbRC_CallPgm() API および cwbRC_RunCmd() API の使用時に、System i メッセージがエラー・ハンドルに追加されることがあります。そのような場合には、この API を使用して、エラー・ハンドルに収められた System i メッセージの置換データを検索できます。メッセージの置換データがない場合は、戻りコード CWBSV_ATTRIBUTE_NOT_SET が戻されます。指標値 1 では、エラー・ハンドル内の最下位 (つまり最も古い) メッセージを処理します。cwbSV_GetErrCount() API によって戻されたカウントと同じ指標値では、エラー・ハンドル内の最上位 (つまり最新) メッセージを処理します。1 よりも小さい指標値を指定すると、1 が渡された場合のように動作します。エラー・ハンドルに入っているメッセージ数より大きい指標値を指定すると、cwbSV_GetErrCount() API から戻されたカウント値が渡された場合のように動作します。戻りコードが CWB_OK である場合は、returnLength パラメーターを使用して、置換データ内に戻された実際のバイト数を判別してください。この API で戻された置換データを、後続のホスト検索メッセージ API 呼び出し (QSYS/QMHRTVM) で使用して、置換データの形式を検索したり、置換データを追加した 2 次ヘルプ・テキストを戻したりすることができます。ホスト API は、cwbRC_CallPgm() API を使用して呼び出します。

cwbSV_GetErrText:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたエラー・ハンドルによって識別される、最上位の (例えば、最新の) エラーに関連するメッセージ・テキストを戻します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetErrText(
                           cwbSV_ErrHandle  errorHandle,
                           char            *errorText,
                           unsigned long    errorTextLength,
                           unsigned long   *returnLength);
```

パラメーター

cwbSV_ErrHandle errorHandle - input
以前の cwbSV_CreateErrHandle() 関数の呼び出しによって戻されたハンドル。

char * errorText - input/output
ハンドルによって識別されるエラーに保管された、エラー・メッセージ・テキストを受け取るバッファーを指すポインター。

unsigned long errorTextLength - input
渡される受信バッファーの長さ。これには、NULL 終了文字を入れるためのスペースを含める必要があります。バッファーが小さすぎると、値が切り捨てられて CWB_BUFFER_OVERFLOW と returnLength が設定されます。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に、出力ストリングを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_NO_ERROR_MESSAGES
エラー・ハンドルに関連するエラー・メッセージがありません。

使用法

なし (None)

cwbSV_GetErrTextIndexed:

これは、System i Access for Windows 製品で使用する API です。

目的

与えられたエラー指標に関連するメッセージ・テキストを戻します。指標値が 1 であれば、エラー・ハンドルに関連する最下位の (例えば、最も古い) メッセージを検索します。指標値が「cwbSV_GetErrCount() が戻した errorCount」であれば、エラー・ハンドルに関連する最上位の (例えば、最新の) メッセージを検索します。

構文

```
unsigned int CWB_ENTRY cwbSV_GetErrTextIndexed(
                            cwbSV_ErrHandle  errorHandle,
                            unsigned long    errorIndex,
                            char            *errorText,
                            unsigned long    errorTextLength,
                            unsigned long   *returnLength);
```

パラメーター

cwbSV_ErrHandle errorHandle - input
以前の cwbSV_CreateErrHandle() 関数の呼び出しによって戻されたハンドル。

unsigned long errorIndex - input
複数のエラーがエラー・ハンドルに関連する場合、どのエラー・テキストを戻すかを示す指標値。

char * errorText - input/output
指標によって識別されるエラーに保管された、エラー・メッセージ・テキストを受け取るバッファーを指すポインター。

unsigned long errorTextLength - input
渡される受信バッファーの長さ。これには、NULL 終了文字を入れるためのスペースを含める必要があります。バッファーが小さすぎると、値が切り捨てられて CWB_BUFFER_OVERFLOW と returnLength が設定されます。

unsigned long * returnLength - input/output
オプションであり、NULL でも構いません。受信バッファーが小さすぎる場合に、出力ストリングを保持するために必要なバイト数を保管するための戻りアドレス。

戻りコード

以下は、共通の戻り値です。

CWB_OK
正常終了。

CWB_BUFFER_OVERFLOW
出力バッファーが小さすぎるため、データは切り捨てられました。

CWB_INVALID_POINTER
NULL が出力パラメーターに渡されました。

CWB_INVALID_HANDLE
無効なハンドル。

CWBSV_NO_ERROR_MESSAGES
エラー・ハンドルに関連するエラー・メッセージがありません。

使用法

有効な指標値は 1 から cwbSV_GetErrCount() の戻り値までです。1 よりも小さい指標値は、 1 が渡された場合と同様に動作します。 cwbSV_GetErrCount() よりも大きい指標値は、 errorCount が渡された場合と同様に動作します。

例: System i Access for Windows 保守容易性 API の使用法

System i Access for Windows の保守容易性 API を使用して、System i Access for Windows のヒストリー・ログにメッセージ・ストリングを記録する方法を、以下の例で説明します。

```
#include <stdio.h>
#include "CWBSV.H"

unsigned int logMessageText(char *msgtxt)
/* Write a message to the active message log.                    */
    {
    cwbSV_MessageTextHandle messageTextHandle;
    unsigned int    rc;

    /* Create a handle to a message text object, so that we may write  */
    /* message text to the active message log.                         */
    if ((rc = cwbSV_CreateMessageTextHandle("ProductID", "ComponentID",
            &messageTextHandle)) != CWB_OK)
        return(rc);
```

```
    /* Log the supplied message text to the active message log.        */
    rc = cwbSV_LogMessageText(messageTextHandle, msgtxt, strlen(msgtxt));

    /* Delete the message text object identified by the handle provided.*/
    cwbSV_DeleteMessageTextHandle(messageTextHandle);

    return(rc);
}

unsigned int readMessageText(char **bufptr, cwbSV_ErrHandle errorHandle)
/* Read a message from the active message log.                    */
    {
    cwbSV_ServiceFileHandle serviceFileHandle;
    cwbSV_ServiceRecHandle  serviceRecHandle;
    static char buffer[BUFSIZ];
    unsigned int    rc;

    /* Retrieve the fully-qualified path and file name of the active  */
    /* message log.                                                   */
    if ((rc = cwbSV_GetServiceFileName(CWBSV_HISTORY_LOG, buffer, BUFSIZ,
            NULL)) != CWB_OK)
        return(rc);

    /* Open the active message log for READ access and return a handle */
    /*  to it.                                                         */
    if ((rc = cwbSV_OpenServiceFile(buffer, &serviceFileHandle, errorHandle))
            != CWB_OK)
        return(rc);

    /* Create a service record object and return a handle to it.     */
    if ((rc = cwbSV_CreateServiceRecHandle(&serviceRecHandle)) != CWB_OK) {
        cwbSV_CloseServiceFile(serviceFileHandle, 0);
        return(rc);
    }

    /* Read the newest record in the active message log into the      */
    /* record handle provided.                                        */
    if ((rc = cwbSV_ReadNewestRecord(serviceFileHandle, serviceRecHandle,
            errorHandle)) != CWB_OK) {
        cwbSV_DeleteServiceRecHandle(serviceRecHandle);
        cwbSV_CloseServiceFile(serviceFileHandle, 0);
        return(rc);
    }

    /* Retrieve the message text portion of the service record object */
    /* identified by the handle provided.                             */
    if ((rc = cwbSV_GetMessageText(serviceRecHandle, buffer, BUFSIZ, NULL))
            == CWB_OK || rc == CWB_BUFFER_OVERFLOW) {
        *bufptr = buffer;
        rc = CWB_OK;
    }

    /* Delete the service record object identified by the             */
    /* handle provided.                                               */
    cwbSV_DeleteServiceRecHandle(serviceRecHandle);

    /* Close the active message log identified by the handle provided.*/
    cwbSV_CloseServiceFile(serviceFileHandle, errorHandle);

    return(rc);
}

void main(int argc, char *argv[ [)
{
    cwbSV_ErrHandle errorHandle;
    char   *msgtxt = NULL, errbuf[BUFSIZ];
```

```
    unsigned int    rc;

    /* Write a message to the active message log.                    */
    if (logMessageText("Sample message text") != CWB_OK)
        return;

    /* Create an error message object and return a handle to it.     */
    cwbSV_CreateErrHandle(&errorHandle);

    /* Read a message from the active message log.                   */
    if (readMessageText(&msgtxt, errorHandle) != CWB_OK) {
        if ((rc = cwbSV_GetErrText(errorHandle, errbuf, BUFSIZ, NULL)) ==
                CWB_OK || rc == CWB_BUFFER_OVERFLOW)
            fprintf(stdout, "%s¥n", errbuf);
    }
    else if (msgtxt)
        fprintf(stdout, "Message text: ¥"%s¥"¥n", msgtxt);

    /* Delete the error message object identified by the             */
    /* handle provided.                                              */
    cwbSV_DeleteErrHandle(errorHandle);
}
```

System i Access for Windows システム・オブジェクト・アクセス (SOA) API

システム・オブジェクト・アクセスを使用することで、グラフィカル・ユーザー・インターフェースを介して、システム・オブジェクトの表示および操作を行うことができます。

System i Access for Windows のシステム・オブジェクト・アクセスのアプリケーション・プログラミング・インターフェース (API) によって、オブジェクト属性に直接アクセスすることが可能になります。例えば、一連の SOA API を呼び出して任意のスプール・ファイルのコピー数を取得し、必要に応じてその値を変更することができます。

System i Access for Windows のシステム・オブジェクト・アクセス API に必要なファイル

インターフェース定義ファイル	インポート・ライブラリー	ダイナミック・リンク・ライブラリー
cwbsoapi.h	cwbapi.lib	cwbsoapi.dll

Programmer's Toolkit:

Programmer's Toolkit には、システム・オブジェクト・アクセスの資料、cwbsoapi.h ヘッダー・ファイルへのアクセス、およびプログラム例へのリンクが用意されています。この情報にアクセスするには、Programmer's Toolkit をオープンして「**System i オペレーション (System i Operations)**」 → 「**C/C++ API**」と選択します。

注: コード例を使用することで、 633 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』の条件に同意します。

関連資料

33 ページの『システム・オブジェクト・アクセス API 戻りコード』
System i Access for Windows の SOA API 戻りコードを示します。

7 ページの『接続 API 用の System i 名の形式』
パラメーターとして System i 名を取得する API では、3 つの異なる形式の名前を使用できます。

SOA オブジェクト

システム・オブジェクト・アクセスを使用して、次の System i オブジェクトの表示および操作を行います。

表示および操作可能なオブジェクト

- ジョブ
- プリンター
- 印刷出力
- メッセージ
- スプール・ファイル

操作のみ可能なオブジェクト

- ユーザーとグループ
- TCP/IP インターフェース
- TCP/IP 経路
- イーサネット回線
- トークンリング回線
- ハードウェア資源
- ソフトウェア資源
- QSYS のライブラリー

システム・オブジェクト・ビュー

System i Access for Windows では、次の 2 つのタイプのシステム・オブジェクト・ビューが提供されています。

リスト・ビュー
選択したシステム・オブジェクトに対して、カスタマイズ可能なグラフィック・リスト・ビューを表示します。ユーザーは 1 つまたは複数のオブジェクトに対して、さまざまなアクションを行うことができます。

プロパティー・ビュー
特定のシステム・オブジェクトの属性について、詳細なグラフィック・ビューを表示します。ユーザーは、必要であればすべての属性を表示させることができ、変更可能な属性に変更を加えることができます。

System i Access for Windows 用システム・オブジェクト・アクセス API の一般的な使用法

ここでは、システム・オブジェクト・アクセス API の 3 つの使用例を示します。

各例は、2 回ずつ示してあります。API 呼び出しの一般的な順序を要約形式で示し、次に実際の C 言語サンプル・プログラムを示しています。要約では、どの API が必須 (R) か、およびどれがオプション (O) かを示しています。通常、各関数呼び出しにはエラーの検査および処理のための追加のコードが必要です。紙面の都合上、ここでは、それらのコードについては省略しています。

システム・オブジェクトのカスタマイズ・リストの表示:

この例では、System i スプール・ファイル・オブジェクトのリストを作成します。希望するソートおよびフィルター基準を設定した後、ある種のユーザー・アクションが使用不可になるようユーザー・インターフェースがカスタマイズされた状態で、リストがユーザーに表示されます。

ユーザーがリストを見終わった後、フィルター基準はアプリケーション・プロファイルに保管され、プログラムは終了します。

システム・オブジェクトのカスタマイズ・リストの表示 (要約)

```
(O)  cwbRC_StartSys              System i の会話を開始する。
(R)  CWBSO_CreateListHandle      システム・オブジェクトのリストを作成する。
(O)  CWBSO_SetListProfile        アプリケーションの名前を設定する。
(O)  CWBSO_ReadListProfile       アプリケーションの設定をロードする。
(O)  CWBSO_SetListFilter         リスト・フィルターの基準を設定する。
(O)  CWBSO_SetListSortFields     リスト・ソートの基準を設定する。
(O)  CWBSO_DisallowListFilter    ユーザーにフィルター基準の変更を許可しない。
(O)  CWBSO_DisallowListActions   選択されたリスト・アクションを許可しない。
(O)  CWBSO_SetListTitle          リストのタイトルを設定する。
(R)  CWBSO_CreateErrorHandle     エラー・オブジェクトを作成する。
(R)  CWBSO_DisplayList           カスタマイズ・リストを表示する。
(O)  CWBSO_DisplayErrMsg         エラーが発生した場合、エラー・メッセージを表示する。
(O)  CWBSO_WriteListProfile      リスト・フィルター基準を保管する。
(R)  CWBSO_DeleteErrorHandle     エラー・オブジェクトを削除する。
(R)  CWBSO_DeleteListHandle      リストを削除する。
(O)  cwbRC_StopSys               System i の会話を終了する。
```

注: コード例を使用することで、 633 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』の条件に同意します。

サンプル・プログラム: システム・オブジェクトのカスタマイズ・リストの表示:

このサンプル・プログラムを使用して、System i オブジェクトを表示します。

```
#ifdef UNICODE
   #define _UNICODE
#endif
#include  <windows.h>               // Windows APIs and datatypes
#include  "cwbsoapi.h"              // System Object Access APIs
#include  "cwbrc.h"                 // System i DPC APIs
#include  "cwbun.h"                 // System i Navigator APIs

#define  APP_PROFILE  "APPPROF"     // Application profile name

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                       LPSTR lpszCmdLine, int nCmdShow)
{
MSG              msg;                          // Message structure
HWND             hWnd;                         // Window handle
```

```
cwbRC_SysHandle   hSystem;                      // System handle
CWBSO_LIST_HANDLE hList  = CWBSO_NULL_HANDLE;   // List handle
CWBSO_ERR_HANDLE  hError = CWBSO_NULL_HANDLE;   // Error handle
cwbCO_SysHandle   hSystemHandle;                // System object handle
unsigned int      rc;                           // System Object Access return codes

unsigned short    sortIDs[] = { CWBSO_SFL_SORT_UserData,
                                CWBSO_SFL_SORT_Priority };
                                                // Array of sort IDs
unsigned short    actionIDs[] = { CWBSO_ACTN_PROPERTIES };
                                                // Array of action IDs

//*****************************************************************
// Start a conversation with System i SYSNAME.  Specify
// application name APPNAME.
//*****************************************************************
cwbUN_GetSystemHandle((char *)"SYSNAME", (char *)"APPNAME", &hSystemHandle);

cwbRC_StartSysEx(hSystemHandle, &hSystem);

//*****************************************************************
// Create a list of spooled files.  Set desired sort/filter criteria.

// Create a list of spooled files on system SYSNAME
CWBSO_CreateListHandleEx(hSystemHandle,
                         CWBSO_LIST_SFL,
                         &hList);

// Identify the name of the application profile
CWBSO_SetListProfile(hList, APP_PROFILE);

// Create an error handle
CWBSO_CreateErrorHandle(&hError);

// Load previous filter criteria
CWBSO_ReadListProfile(hList, hError);

// Only show spooled files on printer P3812 for user TLK
CWBSO_SetListFilter(hList, CWBSO_SFLF_DeviceFilter, "P3812");
CWBSO_SetListFilter(hList, CWBSO_SFLF_UserFilter, "TLK");

// Sort by 'user specified data', then by 'output priority'
CWBSO_SetListSortFields(hList, sortIDs, sizeof(sortIDs) / sizeof(short));

//*****************************************************************
// Customize the UI by disabling selected UI functions.  Set the list title.
//*****************************************************************

// Do not allow users to change list filter
CWBSO_DisallowListFilter(hList);

// Do not allow the 'properties' action to be selected
CWBSO_DisallowListActions(hList, actionIDs, sizeof(actionIDs) / sizeof(short));

// Set the string that will appear in the list title bar
CWBSO_SetListTitle(hList, "Application Title");

//*****************************************************************
// Display the list.
//*****************************************************************

// Display the customized list of spooled files
rc = CWBSO_DisplayList(hList, hInstance, nCmdShow, &hWnd, hError);

// If an error occurred, display a message box
if (rc == CWBSO_ERROR_OCCURRED)
  CWBSO_DisplayErrMsg(hError);
```

```
else
{
  // Dispatch messages for the list window
  while(GetMessage(&msg, NULL, 0, 0))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }

  // List window has been closed - save filter criteria in application profile
  CWBSO_WriteListProfile(hList, hError);
}

//*********************************************************************
// Processing complete - clean up and exit.
//*********************************************************************

// Clean up handles
CWBSO_DeleteErrorHandle(hError);
CWBSO_DeleteListHandle(hList);

// End the conversation started by EHNDP_StartSys
cwbRC_StopSys(hSystem);

//*********************************************************************
// Return from WinMain.
//*********************************************************************

return rc;
}
```

システム・オブジェクトのプロパティー・ビューの表示:

System i スプール・ファイルのリスト用のリスト・オブジェクトが作成されます。希望するフィルター基準を設定した後、リストがオープンされ、リスト中の最初のオブジェクトのハンドルが獲得されます。このオブジェクトの属性を示すプロパティー・ビューがユーザーに表示されます。

オブジェクトのプロパティー・ビューの表示 (要約)

(O)	cwbRC_StartSys	System i の会話を開始する。
(R)	CWBSO_CreateListHandle	システム・オブジェクトのリストを作成する。
(O)	CWBSO_SetListFilter	リスト・フィルターの基準を設定する。
(R)	CWBSO_CreateErrorHandle	エラー・オブジェクトを作成する。
(R)	CWBSO_OpenList	リストをオープンする (System i リストを作成する)。
(O)	CWBSO_DisplayErrMsg	エラーが発生した場合、エラー・メッセージを表示する。
(O)	CWBSO_GetListSize	リスト内のオブジェクトの数を取得する。
(R)	CWBSO_GetObjHandle	リストからオブジェクトを取得する。
(R)	CWBSO_DisplayObjAttr	オブジェクトのプロパティー・ビューを表示する。
(R)	CWBSO_DeleteObjHandle	オブジェクトを削除する。
(O)	CWBSO_CloseList	リストをクローズする。
(R)	CWBSO_DeleteErrorHandle	エラー・オブジェクトを削除する。

(R) CWBSO_DeleteListHandle リストを削除する。

(O) cwbRC_StopSys System i の会話を終了する。

注: コード例を使用することで、 633 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』の条件に同意します。

サンプル・プログラム: オブジェクトのプロパティー・ビューの表示:

このサンプル・プログラムを System i Access for Windows で使用して、プロパティー・ビューを表示します。

```
#ifdef UNICODE
   #define _UNICODE
#endif
#include  <windows.h>                    // Windows APIs and datatypes
#include  "cwbsoapi.h"                   // System Object Access APIs
#include  "cwbrc.h"                      // System i DPC APIs
#include  "cwbun.h"                      // System i Navigator APIs

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpszCmdLine, int nCmdShow)
{
MSG               msg;                          // Message structure
HWND              hWnd;                         // Window handle
cwbRC_SysHandle   hSystem;                      // System handle
CWBSO_LIST_HANDLE hList  = CWBSO_NULL_HANDLE;   // List handle
CWBSO_ERR_HANDLE  hError = CWBSO_NULL_HANDLE;   // Error handle
CWBSO_OBJ_HANDLE  hObject = CWBSO_NULL_HANDLE;  // Object handle
cwbCO_SysHandle   hSystemHandle;                // System object handle
unsigned long     listSize = 0;                 // List size
unsigned short    listStatus = 0;               // List status
unsigned int      rc;                           // System Object Access return codes

//*********************************************************************
// Start a conversation with System i SYSNAME.  Specify
// application name APPNAME.
//*********************************************************************

cwbUN_GetSystemHandle((char *)"SYSNAME", (char *)"APPNAME", &hSystemHandle);

cwbRC_StartSysEx(hSystemHandle, &hSystem);

//*********************************************************************
// Create a list of spooled files.  Set desired filter criteria.
//*********************************************************************

// Create a list of spooled files on system SYSNAME
CWBSO_CreateListHandleEx(hSystemHandle,
                         CWBSO_LIST_SFL,
                         &hList);

// Only include spooled files on printer P3812 for user TLK
CWBSO_SetListFilter(hList, CWBSO_SFLF_DeviceFilter, "P3812");
CWBSO_SetListFilter(hList, CWBSO_SFLF_UserFilter, "TLK");

//*********************************************************************
// Open the list.
//*********************************************************************

// Create an error handle
CWBSO_CreateErrorHandle(&hError);

// Open the list of spooled files
rc = CWBSO_OpenList(hList, hError);
```

```
// If an error occurred, display a message box
if (rc == CWBSO_ERROR_OCCURRED)
  CWBSO_DisplayErrMsg(hError);
else
{
  //*****************************************************************
  // Display the properties of the first object in the list
  //*****************************************************************

  // Get the number of objects in the list
  CWBSO_GetListSize(hList, &listSize, &listStatus, hError);

  if (listSize > 0)
  {
    // Get the first object in the list
    CWBSO_GetObjHandle(hList, 0, &hObject, hError);

    // Display the properties window for this object
    CWBSO_DisplayObjAttr(hObject, hInstance, nCmdShow, &hWnd, hError);

    // Dispatch messages for the properties window
    while(GetMessage(&msg, NULL, 0, 0))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
 }

    // Properties window has been closed - delete object handle
    CWBSO_DeleteObjHandle(hObject);
  }
}

//*****************************************************************
// Processing complete - clean up and exit.
//*****************************************************************

// Close the list
CWBSO_CloseList(hList, hError);

// Clean up handles
CWBSO_DeleteErrorHandle(hError);
CWBSO_DeleteListHandle(hList);

// End the conversation started by EHNDP_StartSys
cwbRC_StopSys(hSystem);

//*****************************************************************
// Return from WinMain.
//*****************************************************************

return rc;
}
```

システム・オブジェクトのデータのアクセスと更新:

System i スプール・ファイルのリスト・オブジェクトが作成されます。希望するフィルター基準を設定した後、リストがオープンされます。パラメーター・オブジェクトが作成され、これを使用してリスト内の各スプール・ファイルの出力優先順位が変更されます。

希望する出力優先順位値の ″9″ をパラメーター・オブジェクトに保管した後、ループに入ります。リスト内の各オブジェクトが順番に調べられ、10 ページ以上あるスプール・ファイルが見付かると、その出力優先順位が変更されます。

この例では、装置 P3812 のスプール・ファイルのうち、10 ページ以上のものは、出力優先順位が 9 に変更され、それより小さいファイルより前に印刷しないようにしています。

システム・オブジェクトのデータのアクセスおよび更新 (要約)

```
(R)  CWBSO_CreateListHandle      システム・オブジェクトのリストを作成する。

(O)  CWBSO_SetListFilter         リスト・フィルターの基準を設定する。

(R)  CWBSO_CreateErrorHandle     エラー・オブジェクトを作成する。

(R)  CWBSO_OpenList              リストをオープンする
                                 (System i の会話を自動的に開始する)。

(O)  CWBSO_DisplayErrMsg         エラーが発生した場合、エラー・メッセージを表示する。

(R)  CWBSO_CreateParmObjHandle   パラメーター・オブジェクトを作成する。

(R)  CWBSO_SetParameter          1 つまたは複数のオブジェクト属性用に新規の値を設定する。


(R)  CWBSO_WaitForObj            最初のオブジェクトが使用可能になるまで待機する。

. . . すべてのオブジェクトをループする。
.
. (R)  CWBSO_GetObjHandle        リストからオブジェクトを取得する。
.
. (R)  CWBSO_GetObjAttr          特定の属性データを読み取る。
.
. (R)  CWBSO_SetObjAttr          System i 属性を更新する。
.
. (R)  CWBSO_DeleteObjHandle     オブジェクト・ハンドルを終結処理する。
.
. (R)  CWBSO_WaitForObj          リストの次のオブジェクトを待機する。
.
..............


(R)  CWBSO_DeleteParmObjHandle   パラメーター・オブジェクトを削除する。

(O)  CWBSO_CloseList             リストをクローズする。

(R)  CWBSO_DeleteErrorHandle     エラー・オブジェクトを削除する。

(R)  CWBSO_DeleteListHandle      リストを削除する
                                 (System i の会話を自動的に終了する)。
```

注: コード例を使用することで、 633 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』の条件に同意します。

サンプル・プログラム: システム・オブジェクトのデータのアクセスと更新:

この System i Access for Windows サンプル・プログラムを使用して、システム・オブジェクトを更新します。

```
#include  <windows.h>                  // Windows APIs and datatypes
#include  <stdlib.h>                   // For atoi
#include  "cwbsoapi.h"                 // System Object Access APIs

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                       LPSTR lpszCmdLine, int nCmdShow)
{
CWBSO_LIST_HANDLE hList  = CWBSO_NULL_HANDLE;   // List handle
CWBSO_ERR_HANDLE  hError = CWBSO_NULL_HANDLE;   // Error handle
CWBSO_PARMOBJ_HANDLE  hParmObject = CWBSO_NULL_HANDLE; // Parm object
```

```
CWBSO_OBJ_HANDLE  hObject = CWBSO_NULL_HANDLE;  // Object handle
unsigned int      rc, setRC;                    // System Object Access return codes
unsigned long     bytesNeeded = 0;              // Bytes needed
unsigned short    errorIndex = 0;          // Error index (SetObjAttr)
char              szString[100]; // Buffer for formatting
int               totalPages = 0;          // Total pages
int               i = 0;                   // Loop counter
int               nNbrChanged = 0;         // Count of changed objects

MessageBox(GetFocus(), "Start of Processing", "PRIORITY", MB_OK);

//*********************************************************************
// Create a list of spooled files.  Set desired filter criteria.
//*********************************************************************

// Create a list of spooled files on system SYSNAME
CWBSO_CreateListHandle("SYSNAME",
                       "APPNAME",
                       CWBSO_LIST_SFL,
                       &hList);

// Only include spooled files for device P3812
CWBSO_SetListFilter(hList, CWBSO_SFLF_DeviceFilter, "P3812");

//*********************************************************************
// Open the list.
//*********************************************************************

// Create an error handle
CWBSO_CreateErrorHandle(&hError);

// Open the list of spooled files
rc = CWBSO_OpenList(hList, hError);


// If an error occurred, display a message box
if (rc == CWBSO_ERROR_OCCURRED)
  CWBSO_DisplayErrMsg(hError);
else
{
  //*******************************************************************
  // Set up to change output priority for all objects in the list.
  //*******************************************************************

  // Create a parameter object to hold the attribute changes
  CWBSO_CreateParmObjHandle(&hParmObject);

  // Set the parameter to change the output priority to '9'
  CWBSO_SetParameter(hParmObject,
                     CWBSO_SFL_OutputPriority,
                     "9",
                     hError);

  //*******************************************************************
  // Loop through the list, changing the output priority for any
  // files that have more than 10 total pages.  Loop will
  // terminate when CWBSO_WaitForObj
  // returns CWBSO_BAD_LIST_POSITION, indicating that there
  // are no more objects in the list.
  //*******************************************************************

  // Wait for first object in the list
  rc = CWBSO_WaitForObj(hList, i, hError);

  // Loop through entire list
  while (rc == CWBSO_NO_ERROR)
  {
```

```
    // Get the list object at index i
    CWBSO_GetObjHandle(hList, i, &hObject, hError);

    // Get the total pages attribute for this spooled file
    CWBSO_GetObjAttr(hObject,
                     CWBSO_SFL_TotalPages,
                     szString,
                     sizeof(szString),
                     &bytesNeeded,;
                     hError);

    totalPages = atoi(szString);

    // Update the output priority if necessary
    if (totalPages > 10)
    {
      // Change the spool file's output priority to '9'
      setRC = CWBSO_SetObjAttr(hObject, hParmObject, &errorIndex, hError);
      if (setRC == CWBSO_NO_ERROR)
        nNbrChanged++;
    }


    // Delete the object handle
    CWBSO_DeleteObjHandle(hObject);

    // Increment list item counter
    i++;

    // Wait for next list object
    rc = CWBSO_WaitForObj(hList, i, hError);

  } /* end while */

  // Parameter object no longer needed
  CWBSO_DeleteParmObjHandle(hParmObject);

  } /* end if */


// Display the number of spooled files that had priority changed
wsprintf (szString, "Number of spool files changed:  %d", nNbrChanged);
MessageBox(GetFocus(), szString, "PRIORITY", MB_OK);

//*********************************************************************
// Processing complete - clean up and exit.
//*********************************************************************

// Close the list
CWBSO_CloseList(hList,hError);

// Clean up handles
CWBSO_DeleteErrorHandle(hError);
CWBSO_DeleteListHandle(hList);

//*********************************************************************
// Return from WinMain.
//*********************************************************************

return 0;
}
```

System i Access for Windows システム・オブジェクト・アクセスのプログラミングに関する考慮事項

SOA のプログラミングに関する重要な考慮事項については、以下のトピックを参照してください。

システム・オブジェクト・アクセスのエラー:

System i Access for Windows 関数では、戻りコードを使用してエラー条件を報告するシステム・オブジェクト・アクセス API を、すべてサポートしています。

関数呼び出しごとに、エラーがないかチェックが行われます。それに加え、特定の API では、「エラー・オブジェクト」のハンドルをそれ自体のインターフェースの中に組み入れています。エラー・オブジェクトは、要求の処理中に発生したエラーについての追加情報を提供するために使用します。 System i オペレーティング・システムとの対話中に、このようなエラーがしばしば発生しますが、その場合、エラー・オブジェクトにエラー・メッセージ・テキストが組み込まれます。

関数呼び出しが CWBSO_ERROR_OCCURRED を戻す場合、エラー・オブジェクトには、エラーを記述する情報が入れられます。エラー・メッセージ・テキストを検索するには、CWBSO_GetErrMsgText を使用します。メッセージは、ユーザーの実行環境で指定された言語に翻訳されます。もう 1 つの方法として、CWBSO_DisplayErrMsg を呼び出すことによって、エラー・メッセージをユーザーに直接表示することができます。

内部処理エラーが発生した場合、エラー・オブジェクトは、System i Access for Windows のインストール・ディレクトリー内に収められているシステム・オブジェクト・アクセス・ログ・ファイル `soa.log` の中に項目を自動的に記録します。このファイルは英語のみで、問題分析のために IBM の担当者が使用するためのものです。

関連資料

33 ページの『システム・オブジェクト・アクセス API 戻りコード』
System i Access for Windows の SOA API 戻りコードを示します。

システム・オブジェクト・アクセスのアプリケーション・プロファイル:

System i Access for Windows のアプリケーション・プロファイルを使用します。

デフォルトでは、ユーザー指定のリスト・フィルター基準はディスクに保管されません。システム・オブジェクト・アクセスによって、以下の API が提供されます。

- レジストリーから指定のリスト・オブジェクトの中にフィルター・データをロードするための、アプリケーション固有のレジストリー・キーの使用を要求する API
- 特定のリスト・オブジェクトのデータを、レジストリーに保管する API

System i 名ごとにデータが保存され、システム名内ではオブジェクト・タイプごとにデータが保存されます。プロファイル・データの読み取りまたは書き込みを行うためには、システム名が、リスト・オブジェクトについての CWBSO_CreateListHandle 呼び出しで指定されている必要があります。

アプリケーション・プログラムのための System i 通信セッションの管理:

System i Access for Windows のシステム・オブジェクト・アクセス API は、1 つ以上のクライアント/サーバー会話を介してシステムと通信します。

1 つの会話を確立するために数秒を要することが多いため、リストが初めてオープンされる際に、アプリケーションに遅延が発生することがあります。このトピックでは、会話の開始を制御および管理して、アプリケーション・プログラムに対するパフォーマンスの影響を最小にする方法について説明します。

システム・オブジェクト・アクセスのデフォルトの動作は、次のように要約されます。

- CWBSO_CreateListHandleEx API で識別される System i オブジェクトとの間に何も会話が確立していない場合は、リストがオープンまたは表示される際に会話が自動的に開始されます。 System i Access for Windows が指定のシステムとの通信をまだ確立していない場合は、ダイアログ・ボックスが表示され、適切なユーザー ID とパスワードを入力するようユーザーに求めるプロンプトが表示されます。
- アプリケーション・プログラムの別のインスタンスが開始されると、上記の過程が繰り返されます。異なるプロセスで (つまり、異なるインスタンス・ハンドルによって) 稼働しているアプリケーション・プログラム間では、会話の共用は行われません。
- アプリケーション・プログラムが最後のシステム・オブジェクト・アクセス・リストを削除すると、System i の会話は自動的に終了します (CWBSO_CloseList の場合は System i の会話を終了しないので、注意してください)。

システム・オブジェクト・アクセスの会話は、cwbRC_StartSysEx API を使用して開始することができます。この API は、System i オブジェクトをパラメーターとして受け入れ、システム・ハンドルを戻します。このハンドルを保存して、後で cwbRC_StopSys API で使用してください (アプリケーションが終了し、System i の会話を終了させるとき)。

cwbRC_StartSysEx API が呼び出されると、会話が確立するまでアプリケーションは停止します。そのため、呼び出しのすぐ前に、接続が行われようとしていることをユーザーに通知することをお勧めします。呼び出しに対して応答が戻された時点で、会話は開始済みの状態になります。システム・オブジェクト・アクセスのリスト処理では、新規の会話を開始する代わりに、この会話を使用します。

cwbRC_StartSysEx がこのように使用される場合、最後のリストが削除されても会話は終了されません。アプリケーションを終了する前に、cwbRC_StopSys を明示的に呼び出す必要があります。

System i Access for Windows のシステム・オブジェクト・アクセス API のリスト

以下のリストに、System i Access for Windows のシステム・オブジェクト・アクセス API がアルファベット順に掲載されています。

SOA イネーブラー

システム・オブジェクト・アクセスには、イネーブラー (API) も含まれます。アプリケーションは、これらのイネーブラーを使用してシステム・オブジェクト内のデータにアクセスしたり、グラフィカル・リスト、ならびに、オブジェクト・データの属性ビューを要求することができます。オブジェクトのリストを操作する API は、正しい順序で呼び出す必要があります。基本的なフローは以下のとおりです。

```
CreateErrorHandle - 他の API に渡すエラー・オブジェクトのハンドルを作成する。
CreateListHandle -- クライアント上のリスト・オブジェクトのインスタンスを作成する。
OpenList -- クライアント・リストに関連した System i リストを構築する。
(さまざまな汎用のサブクラス API を使用して、リストとそのオブジェクトを操作する)
CloseList -- リストをクローズして System i 資源を解放する。
DeleteListHandle  - クライアント上のリスト・オブジェクトを破棄する。
```

CWBSO_CreateListHandleAPI を呼び出してリストを作成した後に、他のリスト API を呼び出すようにする必要があります。 CWBSO_CreateListHandle API は、リスト・ハンドルを呼び出し側に戻します。リスト・ハンドルは、他のすべてのリスト API への入力として渡されなければなりません。

リストが割り振られた後で、CWBSO_SetListFilterAPI を呼び出してそのリストのフィルター基準を変更することができます。 CWBSO_SetListFilter はオプションで、これを呼び出さなかった場合は、デフォルトのフィルター基準を使用してリストが作成されます。同様に、CWBSO_SetListSortFields API を呼び出して、リストのソートの基準に使用される属性を定義することができます。これが呼び出されないと、リストはソートされません。

オブジェクトのリストを作成するには、CWBSO_OpenList API を呼び出す必要があります。これによって作成された要求が、システムに送信されます。リストは、そのシステム上に作成され、リスト内の一部またはすべてのオブジェクト (レコード) は、バッファーに入れられ、クライアントのリスト内に収められます。リストにあるオブジェクトすべてがクライアントでキャッシュされるとは限りませんが、API は、すべてがキャッシュされるかのように動作します。CWBSO_OpenList API が正常に呼び出された後、次の API を呼び出すことができます。

CWBSO_GetObjHandle
リスト内の特定のオブジェクトのハンドルを検索します。このオブジェクト・ハンドルはその後、特定のオブジェクトを操作するために使用することができます。

CWBSO_DeleteObjHandle
CWBSO_GetObjHandle によって戻されたハンドルを解放します。

CWBSO_DisplayList
リストのスプレッドシート・ビューを表示します。

CWBSO_GetListSize
リスト内のオブジェクト数を検索します。

CWBSO_CloseList
System i リストをクローズし、リスト内のクライアント・オブジェクトをすべて破棄します。リストがクローズされた後は、CWBSO_GetListObject によって戻されたオブジェクト・ハンドルはすべて無効になります。リストがクローズされた後は、CWBSO_OpenList API を再度呼び出すまでそのリスト内の API を呼び出すことはできません。リスト・オブジェクトを破棄するには、CWBSO_DeleteListHandle API を呼び出す必要があります。

CWBSO_CloseList:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトのリストをクローズし、System i に割り振られた資源を解放します。

構文

```
unsigned int CWB_ENTRY CWBSO_CloseList(
                CWBSO_LIST_HANDLE listHandle,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

CWBSO_ERR_HANDLE errorHandle - input
以前の CWBSO_CreateErrorHandle の呼び出しによって戻されたエラーのハンドル。この API で戻さ

れた値が CWBSO_ERROR_OCCURRED である場合、エラー・ハンドルを使って、エラー・メッセージ・テキストの検索、またはユーザーへのエラーの表示を行うことができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。リストは現在オープンされている必要があります。リストは、CWBSO_OpenList を呼び出してオープンします。この API によって、System i の会話を終了させることはできません。会話を終了させるには、CWBSO_DeleteListHandle を使用して、リストを削除する必要があります。

CWBSO_CopyObjHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトの新しいインスタンスを作成し、新しいインスタンスへのハンドルを戻します。これは、新規システム・オブジェクトを作成するものではなく、単にクライアント上にシステム・オブジェクトの追加インスタンスを作成するものです。 CWBSO_GetObjHandle によって戻されるオブジェクト・ハンドルは、そのオブジェクトが収められているリストがクローズされる時点で必ず破棄されます。この API によって、リストがクローズされた後も持続するオブジェクトのインスタンスの作成が可能になります。この API によって作成されたオブジェクト・インスタンスは、リストのオブジェクトと同期が保たれます。言いかえれば、オブジェクトの 1 つが変更される場合、その変更は別のオブジェクトでも見ることができます。

構文

```
unsigned int CWB_ENTRY CWBSO_CopyObjHandle(
                CWBSO_OBJ_HANDLE objectHandle,
                CWBSO_OBJ_HANDLE far* lpNewObjectHandle);
```

パラメーター

CWBSO_OBJ_HANDLE objectHandle - input
以前の CWBSO_GetObjHandle または CWBSO_CopyObjHandle の呼び出しによって戻されたオブジェクトのハンドル。

CWBSO_OBJ_HANDLE far* lpNewObjectHandle - output
同じシステム・オブジェクトの新しいハンドルに設定されるハンドルを指す、long 型のポインター。このハンドルは、オブジェクト・ハンドルを受け入れる他の API でも使用できますが、API によっては特定のタイプのオブジェクトにしか機能しないものもあります。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_BAD_OBJ_HANDLE
指定されたオブジェクト・ハンドルが無効。

使用法

この API を呼び出す前に、CWBSO_GetObjHandle または CWBSO_CopyObjHandle を呼び出す必要があります。CWBSO_GetObjHandle または CWBSO_CopyObjHandle で戻されたオブジェクト・ハンドルを、この API への入力として渡す必要があるためです。オブジェクトが不要になったら、呼び出し側プログラムでは、次の操作を行う必要があります。

- CWBSO_DeleteObjHandle を呼び出して、クライアントに割り振られている資源を解放する。

CWBSO_CreateErrorHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

エラー・ハンドルを作成します。エラー・ハンドルは、他の API から戻されたエラー・メッセージを入手するために使用します。エラー・ハンドルを使用して、エラーをダイアログで表示したり、関連するエラー・メッセージ・テキストを検索することができます。

構文

```
unsigned int CWB_ENTRY CWBSO_CreateErrorHandle(
               CWBSO_ERR_HANDLE far* lpErrorHandle);
```

パラメーター

CWBSO_ERR_HANDLE far* lpErrorHandle - output
エラーのためのハンドルに設定されるハンドルを指す long 型のポインター。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

使用法

エラー・ハンドルが不要になった後、呼び出したプログラムでは、次の操作を行う必要があります。

- CWBSO_DeleteErrorHandle を呼び出して、クライアントに割り振られている資源を解放する。

CWBSO_CreateListHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

新しいリストを作成し、そのリストのハンドルを戻します。

構文

```
unsigned int CWB_ENTRY CWBSO_CreateListHandle(
                char far* lpszSystemName,
                char far* lpszApplicationName,
                CWBSO_LISTTYPE type,
                CWBSO_LIST_HANDLE far* lpListHandle);
```

パラメーター

char far* lpszSystemName - input
リストが作成される System i の名前。この名前には、構成済みシステムの名前を指定する必要があります。現時点でクライアントが対象のシステムに接続されていない場合には、リストのオープン時に System i 接続が確立されます。システム名に NULL が指定されていると、現行の System i Access のデフォルト・システムが使用されます。

char far* lpszApplicationName - input
リストと対話するアプリケーションを識別する文字ストリング。このストリングの最大長は、NULL 終了文字を除いた 10 文字です。

CWBSO_LISTTYPE type - input
作成するリストのタイプ。下記のうちのいずれかを指定します。

CWBSO_LIST_JOB
ジョブのリスト。

CWBSO_LIST_SJOB
サーバー・ジョブのリスト。

CWBSO_LIST_SJOB
サーバー・ジョブのリスト。

CWBSO_LIST_MSG
メッセージのリスト。

CWBSO_LIST_PRT
プリンターのリスト。

CWBSO_LIST_SFL
スプール・ファイルのリスト。

CWBSO_LIST_IFC
インターフェースのリスト。

CWBSO_LIST_ELN
イーサネット回線のリスト。

CWBSO_LIST_TLN
トークンリング回線のリスト。

CWBSO_LIST_HWL
ハードウェア資源のリスト。

CWBSO_LIST_SW
ソフトウェア・プロダクトのリスト。

CWBSO_LIST_RTE
TCP/IP 経路のリスト。

CWBSO_LIST_PRF
ユーザー・プロファイルのリスト。

CWBSO_LIST_SMP
QSYS のライブラリーのリスト。

CWBSO_LIST_HANDLE far* lpListHandle - output
新しく作成されたリストのハンドルに設定されるハンドルを指す long 型のポインター。このハンドルは、リスト・ハンドルを受け入れる他の API で使用できます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LISTTYPE
リストのタイプに指定した値が無効です。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_BAD_SYSTEM_NAME
指定されたシステム名は、有効な System i 名ではありません。

使用法

リストが必要ではなくなったとき、呼び出し側プログラムでは次の操作を行う必要があります。

- CWBSO_DeleteListHandle を呼び出して、クライアントに割り振られている資源を解放する。

CWBSO_CreateListHandleEx:

これは、System i Access for Windows 製品で使用する API です。

目的

新しいリストを作成し、そのリストのハンドルを戻します。

構文

```
unsigned int CWB_ENTRY CWBSO_CreateListHandleEx(
                cwbCO_SysHandle systemObjectHandle,
                CWBSO_LISTTYPE type,
                CWBSO_LIST_HANDLE far* lpListHandle);
```

パラメーター

cwbCO_SysHandle systemObjectHandle - input
リストの作成場所であるシステムを表す、システム・オブジェクトのハンドル。この System i ハンドルは、構成済みのシステムに対するものでなければなりません。

CWBSO_LISTTYPE
作成するリストのタイプ。下記のうちのいずれかを指定します。

CWBSO_LIST_JOB
ジョブのリスト。

CWBSO_LIST_SJOB
サーバー・ジョブのリスト。

CWBSO_LIST_SJOB
サーバー・ジョブのリスト。

CWBSO_LIST_MSG
メッセージのリスト。

CWBSO_LIST_PRT
プリンターのリスト。

CWBSO_LIST_SFL
スプール・ファイルのリスト。

CWBSO_LIST_IFC
インターフェースのリスト。

CWBSO_LIST_ELN
イーサネット回線のリスト。

CWBSO_LIST_TLN
トークンリング回線のリスト。

CWBSO_LIST_HWL
ハードウェア資源のリスト。

CWBSO_LIST_SW
ソフトウェア・プロダクトのリスト。

CWBSO_LIST_RTE
TCP/IP 経路のリスト。

CWBSO_LIST_PRF
ユーザー・プロファイルのリスト。

CWBSO_LIST_SMP
QSYS のライブラリーのリスト。

CWBSO_LIST_HANDLE far* lpListHandle - output
新しく作成されたリストのハンドルに設定されるハンドルを指す long 型のポインター。このハンドルは、リスト・ハンドルを受け入れる他の API で使用できます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LISTTYPE
リストのタイプに指定した値が無効です。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_BAD_SYSTEM_NAME
指定されたシステム名は、有効な System i 名ではありません。

使用法

リストが必要ではなくなったとき、呼び出し側プログラムでは次の操作を行う必要があります。

- CWBSO_DeleteListHandle を呼び出して、クライアントに割り振られている資源を解放する。

CWBSO_CreateObjHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

新規のオブジェクト・ハンドルを作成し、そのオブジェクトのハンドルを戻します。この API は、リスト形式に従っていないリモート・オブジェクトにアクセスする場合に使用します。

構文

```
unsigned int CWB_ENTRY CWBSO_CreateObjHandle(
                char far* lpszSystemName,
                char far* lpszApplicationName,
                CWBSO_OBJTYPE type,
                CWBSO_OBJ_HANDLE far* lpObjHandle);
```

パラメーター

char far* lpszSystemName - input
オブジェクトが作成されるシステムの名前。この名前には、構成済みシステムの名前を指定する必要があります。現時点でクライアントが接続されていない場合には、リストのオープン時に System i 接続が確立されます。システム名に NULL が指定されていると、現行の System i のデフォルト・システムが使用されます。

char far* lpszApplicationName - input
リストと対話するアプリケーションを識別する文字ストリング。このストリングの最大長は、NULL 終了文字を除いた 10 文字です。

CWBSO_OBJTYPE type - input
作成するオブジェクトのタイプ。次のように指定します。

- CWBSO_OBJ_TCIPATTR - TCP/IP 属性

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_BAD_SYSTEM_NAME
指定されたシステム名は、有効な System i 名ではありません。

使用法

リストが必要ではなくなったとき、呼び出し側プログラムでは次の操作を行う必要があります。

- CWBSO_DeleteObjHandle を呼び出して、クライアントに割り振られている資源を解放する。

CWBSO_CreateParmObjHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

パラメーター・オブジェクトを作成し、そのオブジェクトのハンドルを戻します。パラメーター・オブジェクトには、一連のパラメーター ID および他の API への入力として渡すことができる値が入っています。

構文

```
unsigned int CWB_ENTRY CWBSO_CreateParmObjHandle(
                CWBSO_PARMOBJ_HANDLE far* lpParmObjHandle);
```

パラメーター

CWBSO_PARMOBJ_HANDLE far* lpParmObjHandle - output
新しいパラメーター・オブジェクトのためのハンドルに設定されるハンドルを指す long 型のポインター。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

使用法

パラメーター・オブジェクトが必要でなくなった後、呼び出したプログラムでは次の操作を行う必要があります。

- CWBSO_DeleteParmObjHandle を呼び出して、クライアントに割り振られている資源を解放する。

CWBSO_DeleteErrorHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

エラー・ハンドルを削除し、クライアントに割り振られた資源を解放します。

構文

```
unsigned int CWB_ENTRY CWBSO_DeleteErrorHandle(
               CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_ERR_HANDLE errorHandle - input
以前の CWBSO_CreateErrorHandle の呼び出しによって戻されるエラー・ハンドル。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

使用法

この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。
CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。

CWBSO_DeleteListHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトのリストを削除し、クライアントに割り振られた資源を解放します。

構文

```
unsigned int CWB_ENTRY CWBSO_DeleteListHandle(
               CWBSO_LIST_HANDLE listHandle);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。

CWBSO_DeleteObjHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

以前の CWBSO_GetObjHandle または CWBSO_CopyObjHandle の呼び出しから戻されたオブジェクト・ハンドルを削除します。

構文

```
unsigned int CWB_ENTRY CWBSO_DeleteObjHandle(
               CWBSO_OBJ_HANDLE objectHandle);
```

パラメーター

CWBSO_OBJ_HANDLE objectHandle - input
以前の CWBSO_GetObjHandle または CWBSO_CopyObjHandle の呼び出しによって戻されたオブジェクトのハンドル。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_OBJ_HANDLE
指定されたオブジェクト・ハンドルが無効。

使用法

この API を呼び出す前に、CWBSO_GetObjHandle または CWBSO_CopyObjHandle を呼び出す必要があります。CWBSO_GetObjHandle または CWBSO_CopyObjHandle で戻されたオブジェクト・ハンドルを、この API への入力として渡す必要があるためです。

CWBSO_DeleteParmObjHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

パラメーター・オブジェクト・ハンドルを削除し、クライアントに割り振られた資源を解放します。

構文

```
unsigned int CWB_ENTRY CWBSO_DeleteParmObjHandle(
                CWBSO_PARMOBJ_HANDLE parmObjHandle);
```

パラメーター

CWBSO_PARMOBJ_HANDLE parmObjHandle - input
以前の CWBSO_CreateParmObjHandle の呼び出しによって戻されたパラメーター・オブジェクトのハンドル。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_PARMOBJ_HANDLE
指定されたパラメーター・オブジェクト・ハンドルが無効です。

使用法

この API を呼び出す前に、CWBSO_CreateParmObjHandle を呼び出す必要があります。CWBSO_CreateParmObjHandle で戻されたパラメーター・オブジェクト・ハンドルをこの API への入力として渡す必要があるためです。

CWBSO_DisallowListActions:

これは、System i Access for Windows 製品で使用する API です。

目的

リスト内のオブジェクトに対するユーザーの実行が許可されないアクションを設定します。これは、CWBSO_DisplayList を呼び出してリストを表示する時点で使用可能になるアクションに影響します。使用禁止にされたアクションは、メニュー・バー、ツールバー、またはオブジェクトのポップアップ・メニューには表示されません。この API は 1 つのリストにつき 1 回のみ呼び出すことができ、リストを表示する前に呼び出さなくてはなりません。

構文

```
unsigned int CWB_ENTRY CWBSO_DisallowListActions(
                CWBSO_LIST_HANDLE listHandle,
                unsigned short far* lpusActionIDs,
                unsigned short usCount);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

unsigned short far* lpusActionIDs - input
アクション識別コードの値の配列を指す long 型のポインター。これらの値は、ユーザーによる実行が

認められていないアクションを示します。このパラメーターの有効な値は、リストのオブジェクトのタイプによって決まります。有効な値については、下記の該当するヘッダー・ファイルを参照してください。

- cwbsojob.h
- cwbsomsg.h
- cwbsoprt.h
- cwbsosfl.h

unsigned short usCount - input
指定されたアクション識別コードの値の数。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_ACTION_ID
指定されたアクション ID がこのリストのタイプに対しては無効です。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_NOT_ALLOWED_NOW
要求されたアクションは、現在許可されていません。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。

CWBSO_DisallowListFilter:

これは、System i Access for Windows 製品で使用する API です。

目的

リストのフィルター値をユーザーが変更できないようにリストを設定します。これによって、リストが表示されるときに「オプション」のプルダウン・メニューから「組み込み」を選択できなくなります。リストは、CWBSO_DisplayList を呼び出して表示します。この API が意味を持つのは、CWBSO_DisplayList API を使用して表示されるリストについてのみです。この API は 1 つのリストにつき 1 回のみ呼び出すことができ、リストを表示する前に呼び出さなくてはなりません。

構文

```
unsigned int CWB_ENTRY CWBSO_DisallowListFilter(
               CWBSO_LIST_HANDLE listHandle);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。

CWBSO_DisplayErrMsg:

これは、System i Access for Windows 製品で使用する API です。

目的

エラー・メッセージをダイアログ・ボックスに表示します。この API は、別の API の呼び出しからの戻り値として CWBSO_ERROR_OCCURRED が返されたときにのみ呼び出してください。この場合、エラー・ハンドルに関連したエラー・メッセージがあります。

構文

```
unsigned int CWB_ENTRY CWBSO_DisplayErrMsg(
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_ERR_HANDLE errorHandle - input
エラーのハンドル。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_NO_ERROR_MESSAGE
指定されたエラー・ハンドルにエラー・メッセージが入っていません。

CWBSO_DISP_MSG_FAILED
メッセージの表示要求が失敗しました。

使用法

この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。 CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。

CWBSO_DisplayList:

これは、System i Access for Windows 製品で使用する API です。

目的

リストをウィンドウに表示します。ユーザーは、このウィンドウから、リスト内のオブジェクトに対してアクションを行うことができます。

構文

```
unsigned int CWB_ENTRY CWBSO_DisplayList(
                CWBSO_LIST_HANDLE listHandle,
                HINSTANCE hInstance,
                int nCmdShow,
                HWND far* lphWnd ,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

HINSTANCE hInstance - input
呼び出し側プログラムの WinMain プロシージャーに渡されたプログラム・インスタンス。

int nCmdShow - input
呼び出し側プログラムの WinMain プロシージャーに渡されたウィンドウ表示パラメーター。その代わりとして、Windows API ShowWindow() 用に定義された定数のいずれかを使用することができます。

HWND far* lphWnd - output
ウィンドウ・ハンドルを指す long 型のポインター。これは、リストが表示されるウィンドウのハンドルに設定されます。

CWBSO_ERR_HANDLE errorHandle - input
エラー・オブジェクトのハンドル。エラー・テキストのあるエラーが発生した場合、このハンドルを使用して、エラー・メッセージ・テキストを検索したり、エラーをユーザーに表示したりすることができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_DISPLAY_FAILED
ウィンドウが作成できませんでした。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。この API を使用する場合、CWBSO_OpenList または CWBSO_CloseList を呼び出す必要はありません。CWBSO_DisplayList が、リストのオープンとクローズの両方を処理します。システム・オブジェクト・リストの使用中に送られる Windows メッセージを受け取るためには、プログラムにメッセージ・ループが必要です。

この API は、ジョブ、メッセージ、プリンター、プリンター出力、およびスプール・ファイルの各リスト・タイプにのみ適用されます。

CWBSO_DisplayObjAttr:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトについて属性ウィンドウを表示します。このウィンドウから、ユーザーはオブジェクトの属性を表示したり、変更可能な属性を変更することが可能になります。

構文

```
unsigned int CWB_ENTRY CWBSO_DisplayObjAttr(
               CWBSO_OBJ_HANDLE objectHandle,
               HINSTANCE hInstance,
               int nCmdShow,
               HWND far* lphWnd ,
               CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_OBJ_HANDLE objectHandle - input
以前の CWBSO_GetObjHandle または CWBSO_CopyObjHandle の呼び出しによって戻されたオブジェクトのハンドル。

HINSTANCE hInstance - input
呼び出し側プログラムの WinMain プロシージャーに渡されたプログラム・インスタンス。

int nCmdShow - input
呼び出し側プログラムの WinMain プロシージャーに渡されたウィンドウ表示パラメーター。その代わりとして、Windows API ShowWindow() 用に定義された定数のいずれかを使用することができます。

HWND far* lphWnd - output
ウィンドウ・ハンドルを指す long 型のポインター。これは、オブジェクト属性が表示されるウィンドウのハンドルに設定されます。

CWBSO_ERR_HANDLE errorHandle - input
エラー・オブジェクトのハンドル。エラー・テキストのあるエラーが発生した場合、このハンドルを使用してエラー・メッセージおよびメッセージ・ヘルプを検索することができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_OBJ_HANDLE
指定されたオブジェクト・ハンドルが無効。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_DISPLAY_FAILED
ウィンドウが作成できませんでした。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_GetObjHandle または CWBSO_CopyObjHandle を呼び出す必要があります。CWBSO_GetObjHandle または CWBSO_CopyObjHandle で戻されたオブジェクト・ハンドルを、この API への入力として渡す必要があるためです。この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。システム・オブジェクトの属性ウィンドウの使用中に送られる Windows メッセージを受け取るためには、プログラムにメッセージ・ループが必要です。

この API は、ジョブ、メッセージ、プリンター、プリンター出力、およびスプール・ファイルの各リスト・タイプにのみ適用されます。

CWBSO_GetErrMsgText:

これは、System i Access for Windows 製品で使用する API です。

目的

エラー・ハンドルからメッセージ・テキストを検索します。この API は、別の API の呼び出しからの戻り値として CWBSO_ERROR_OCCURRED が返されたときにのみ呼び出してください。この場合、エラー・ハンドルに関連したエラー・メッセージがあります。

構文

```
unsigned int CWB_ENTRY CWBSO_GetErrMsgText(
                CWBSO_ERR_HANDLE errorHandle ,
                char far* lpszMsgBuffer ,
                unsigned long ulBufferLength,
                unsigned long far* lpulBytesNeeded);
```

パラメーター

CWBSO_ERR_HANDLE errorHandle - input
エラー・オブジェクトのハンドル。エラー・テキストのあるエラーが発生した場合、このハンドルを使用してエラー・メッセージおよびメッセージ・ヘルプを検索することができます。

char far* lpszMsgBuffer - output
メッセージ・テキストが入れられる出力バッファーを指す long 型のポインター。この API によって戻されたメッセージ・テキストは、変換されたテキストです。戻りコードが CWBSO_NO_ERROR に設定されない場合、出力バッファーは変更されません。

unsigned long ulBufferLength - input
出力バッファー引数のバイトでのサイズ。

unsigned long far* lpulBytesNeeded - output
出力バッファーにメッセージ・テキスト全体を入れるために必要なバイト数に設定される、符号なし長精度整数を指す long 型のポインター。この値が、指定された出力バッファーのサイズと等しいかこれより小さいと、メッセージ・テキスト全体が出力バッファーに入れられます。この値が、指定された出力バッファーのサイズより大きいと、出力バッファーには NULL ストリングが入ります。メッセージ・テキストに必要なバイト数を超えて、出力バッファーが変更されることはありません。戻りコードが CWBSO_NO_ERROR に設定されない場合、この値はゼロに設定されます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_NO_ERROR_MESSAGE
指定されたエラー・ハンドルにエラー・メッセージが入っていません。

CWBSO_GET_MSG_FAILED
エラー・メッセージのテキストを検索できませんでした。

使用法

この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。 CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。 System i エラーに関するメッセージ・テキストは、ユーザーの実行環境用に指定された言語で表示されます。その他のメッセージ・テキストはすべて、ユーザーのパーソナル・コンピューターの Windows コントロール・パネルで指定された言語で表示されます。

CWBSO_GetListSize:

これは、System i Access for Windows 製品で使用する API です。

目的

リスト内のオブジェクトの数を検索します。

構文

```
unsigned int CWB_ENTRY CWBSO_GetListSize(
                CWBSO_LIST_HANDLE listHandle,
                unsigned long  far* lpulSize,
                unsigned short far* lpusStatus,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

unsigned long far* lpulSize - output
現在リスト内にある項目の数に設定される無符号長精度整数を指す long 型のポインター。リスト状況がリストが完全に作成されていることを示している場合、この値はリストのオブジェクトの合計数を表します。リスト状況がリストが完全に作成されていないことを示している場合、この値は、現在ホストから利用できるオブジェクトの数を表しており、これ以降にこの API を呼び出すと、これより多くの項目が利用可能であると示される可能性があります。

unsigned short far* lpusStatus - output
リストが完全に作成されているかどうかを示すために設定される符号なしの短精度整数を指す long 型ポインター。リストが完全に作成されていない場合、この値は 0 に設定され、リストが完全に作成されている場合、値は 1 に設定されます。

CWBSO_ERR_HANDLE errorHandle - input
エラー・オブジェクトのハンドル。エラー・テキストのあるエラーが発生した場合、このハンドルを使用してエラー・メッセージおよびメッセージ・ヘルプを検索することができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。
CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるた

めです。この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。リストは現在オープンされている必要があります。リストは、CWBSO_OpenList を呼び出してオープンします。 CWBSO_CloseList を呼び出してリストをクローズする場合は、CWBSO_OpenList を再度呼び出さなければ、この API を呼び出すことができません。

CWBSO_GetObjAttr:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトから属性の値を検索します。

構文

```
unsigned int CWB_ENTRY CWBSO_GetObjAttr(
                CWBSO_OBJ_HANDLE objectHandle,
                unsigned short usAttributeID,
                char far* lpszBuffer,
                unsigned long ulBufferLength,
                unsigned long far* lpulBytesNeeded,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_OBJ_HANDLE objectHandle - input
以前の CWBSO_GetObjHandle または CWBSO_CopyObjHandle の呼び出しによって戻されたオブジェクトのハンドル。

unsigned short usAttributeID - input
検索すべき属性の識別コード。このパラメーターの有効な値は、オブジェクトのタイプによって決まります。有効な値については、下記の該当するヘッダー・ファイルを参照してください。

- cwbsojob.h
- cwbsomsg.h
- cwbsoprt.h
- cwbsosfl.h

char far* lpszBuffer - output
属性値が入れられる出力バッファーを指す long 型のポインター。この API によって戻された値は、変換されたストリングではありません。例えば、スプール・ファイルの終了ページ属性の場合、「終了ページ」ではなく、「*END」が戻されます。それぞれのオブジェクトのタイプごとに戻される可能性のある特殊値については、496 ページの『SOA 属性の特殊値』を参照してください。戻りコードが CWBSO_NO_ERROR に設定されない場合、出力バッファーは変更されません。

unsigned long ulBufferLength - input
出力バッファー引数のバイトでのサイズ。

unsigned long far* lpulBytesNeeded - output
出力バッファーに属性値全体を入れるのに必要なだけのバイト数に設定される無符号長精度整数を指す long 型のポインター。この値が、指定された出力バッファーのサイズと等しいかこれより小さいと、属性値全体が出力バッファーに入れられます。この値が、指定された出力バッファーのサイズより大き

いと、出力バッファーには NULL ストリングが入ります。属性値に必要なバイト数を超えて、出力バッファーが変更されることはありません。戻りコードが CWBSO_NO_ERROR に設定されない場合、この値はゼロに設定されます。

CWBSO_ERR_HANDLE errorHandle - input
エラー・オブジェクトのハンドル。エラー・テキストのあるエラーが発生した場合、このハンドルを使用してエラー・メッセージおよびメッセージ・ヘルプを検索することができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_OBJ_HANDLE
指定されたオブジェクト・ハンドルが無効。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_BAD_ATTRIBUTE_ID
属性キーがこのオブジェクトに対して無効。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_GetObjHandle または CWBSO_CopyObjHandle を呼び出す必要があります。CWBSO_GetObjHandle または CWBSO_CopyObjHandle で戻されたオブジェクト・ハンドルを、この API への入力として渡す必要があるためです。この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。

CWBSO_GetObjHandle:

これは、System i Access for Windows 製品で使用する API です。

目的

リスト内のオブジェクトのハンドルを取得します。この API によって戻されたオブジェクト・ハンドルは、リストがクローズされるまで、またはオブジェクト・ハンドルが削除されるまで有効です。このオブジェクト・ハンドルは、以下の API を呼び出す際に使用されます。

- CWBSO_CopyObjHandle
- CWBSO_DeleteObjHandle
- CWBSO_DisplayObjAttr
- CWBSO_GetObjAttr
- CWBSO_RefreshObj
- CWBSO_SetObjAttr

- CWBSO_WaitForObj

構文

```
unsigned int CWB_ENTRY CWBSO_GetObjHandle(
                CWBSO_LIST_HANDLE listHandle,
                unsigned long ulPosition,
                CWBSO_OBJ_HANDLE far* lpObjectHandle,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

unsigned long ulPosition - input
ハンドルが必要な、リスト内のオブジェクトの位置。注: リスト内の最初のオブジェクトは、位置 0 と見なされます。

CWBSO_OBJ_HANDLE far* lpObjectHandle - output
System i オブジェクトのハンドルに設定されるハンドルを指す、long 型のポインター。このハンドルは、オブジェクト・ハンドルを受け入れる他の API でも使用できますが、API によっては特定のタイプのオブジェクトにしか機能しないものもあります。

CWBSO_ERR_HANDLE errorHandle - input
エラー・オブジェクトのハンドル。エラー・テキストのあるエラーが発生した場合、このハンドルを使用してエラー・メッセージおよびメッセージ・ヘルプを検索することができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_BAD_LIST_POSITION
指定されたリスト内の位置が無効です。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。
CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。
CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があり

ます。リストは現在オープンされている必要があります。リストは、CWBSO_OpenList を呼び出してオープンします。 CWBSO_CloseList を呼び出してリストをクローズする場合は、CWBSO_OpenList を再度呼び出さなければ、この API を呼び出すことができません。この API を使用するとき、オブジェクトがリストに組み込まれるまでそのオブジェクトにアクセスすることはできません。例えば、CWBSO_OpenList を呼び出した直後に、位置 100 にあるオブジェクトをこの API を出して取得しようとしても、オブジェクトはすぐには利用可能とならない場合があります。そのような場合には、CWBSO_WaitForObj を使用し、オブジェクトが利用可能になるまで待機します。この API によって戻されるオブジェクト・ハンドルは、後続の CWBSO_DeleteObjHandle の呼び出しによって削除する必要があります。

CWBSO_OpenList:

これは、System i Access for Windows 製品で使用する API です。

目的

リストをオープンします。リスト作成の要求が、システムに送信されます。

構文

```
unsigned int CWB_ENTRY CWBSO_OpenList(
                CWBSO_LIST_HANDLE listHandle,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

CWBSO_ERR_HANDLE errorHandle - input
以前の CWBSO_CreateErrorHandle の呼び出しによって戻されたエラーのハンドル。この API で戻された値が CWBSO_ERROR_OCCURRED である場合、エラー・ハンドルを使って、エラー・メッセージ・テキストの検索、またはユーザーへのエラーの表示を行うことができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。リストが必要ではなくなったとき、呼び出し側プログラムでは次の操作を行う必要があります。

- CWBSO_CloseList を呼び出して、リストをクローズし、System i に割り振られている資源を解放する。
- CWBSO_DeleteListHandle を呼び出して、クライアントに割り振られている資源を解放する。

CWBSO_ReadListProfile:

これは、System i Access for Windows 製品で使用する API です。

目的

リストに関するフィルター情報を、Windows レジストリーから読み取ります。ユーザーは、CWBSO_SetListProfile API を使用してアプリケーション名を設定しておかなければなりません。この API は、CWBSO_OpenList または CWBSO_DisplayList API を使用して、リストをオープンする前に呼び出す必要があります。

構文

```
unsigned int CWB_ENTRY CWBSO_ReadListProfile(
                CWBSO_LIST_HANDLE listHandle,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

CWBSO_ERR_HANDLE errorHandle - input
以前の CWBSO_CreateErrorHandle の呼び出しによって作成されたエラー・オブジェクトのハンドル。この API で戻された値が CWBSO_ERROR_OCCURRED である場合、エラー・ハンドルを使って、エラー・メッセージ・テキストの検索、またはユーザーへのエラーの表示を行うことができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_SYSTEM_NAME_DEFAULTED
そのリストに関する CWBSO_CreateListHandle 呼び出しでシステム名が指定されませんでした。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。この API を呼び出す前に、CWBSO_SetListProfile を呼び出す必要があります。この API は、既にオープンされているリストに対しては有効とはなりません。プロファイルのフィルター基準を有効にするためには、この API を呼び出した後でリストをオープンする必要があります。

CWBSO_RefreshObj:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトの System i 属性をリフレッシュします。オブジェクトについてオープンしているシステム・オブジェクト・アクセス・ビューをすべてリフレッシュします。

構文

```
unsigned int CWB_ENTRY CWBSO_RefreshObj(
                CWBSO_OBJ_HANDLE objectHandle,
                HWND hWnd ,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_OBJ_HANDLE objectHandle - input
以前の CWBSO_GetObjHandle または CWBSO_CopyObjHandle の呼び出しによって戻されたオブジェクトのハンドル。

HWND hWnd - input
リフレッシュが完了した後にフォーカスを受け取るウィンドウのハンドル。このパラメーターは NULL にすることができます。この API がアプリケーション・ウィンドウ・プロシージャーから呼び出されていた場合は、現行のウィンドウ・ハンドルを与える必要があります。これを実行しない場合、フォーカスは、最後にオープンされたオープン状態のシステム・オブジェクト・アクセス・ウィンドウにシフトします。

CWBSO_ERR_HANDLE errorHandle - input
エラー・オブジェクトのハンドル。エラー・テキストのあるエラーが発生した場合、このハンドルを使用してエラー・メッセージおよびメッセージ・ヘルプを検索することができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_OBJ_HANDLE
指定されたオブジェクト・ハンドルが無効。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_GetObjHandle または CWBSO_CopyObjHandle を呼び出す必要があります。CWBSO_GetObjHandle または CWBSO_CopyObjHandle で戻されたオブジェクト・ハンドルを、この API への入力として渡す必要があるためです。この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。

CWBSO_ResetParmObj:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトから属性値を取り除くために、パラメーター・オブジェクトをリセットします。

構文

```
unsigned int CWB_ENTRY CWBSO_ResetParmObj(
               CWBSO_PARMOBJ_HANDLE parmObjHandle);
```

パラメーター

CWBSO_PARMOBJ_HANDLE parmObjHandle - input
以前の CWBSO_CreateParmObjHandle の呼び出しによって戻されたパラメーター・オブジェクトのハンドル。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_PARMOBJ_HANDLE
パラメーター・オブジェクト・ハンドルが無効です。

使用法

この API を呼び出す前に、CWBSO_CreateParmObjHandle を呼び出す必要があります。CWBSO_CreateParmObjHandle で戻されたパラメーター・オブジェクト・ハンドルをこの API への入力として渡す必要があるためです。

CWBSO_SetListFilter:

これは、System i Access for Windows 製品で使用する API です。

目的

リストのフィルター値を設定します。リストのタイプによって、さまざまなフィルター値の設定が可能です。フィルター値では、CWBSO_OpenList によってリストが作成される時点で、そのリストに組み込むオブジェクトを制御します。

構文

```
unsigned int CWB_ENTRY CWBSO_SetListFilter(
                CWBSO_LIST_HANDLE listHandle,
                unsigned short usFilterID,
                char far* lpszValue);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

unsigned short usFilterID - input
フィルターのどの部分が設定されるかを指定するフィルター識別コード。このパラメーターの有効な値は、リストのオブジェクトのタイプによって決まります。有効な値については、下記の該当するヘッダー・ファイルを参照してください。

- cwbsojob.h
- cwbsomsg.h
- cwbsoprt.h
- cwbsosfl.h

char far* lpszValue - input
フィルター属性の値。複数の項目を指定する場合、それらをコンマで区切らなければなりません。システム・オブジェクト名を指定するフィルター値項目は、大文字でなければなりません。修飾オブジェクト名は、ライブラリー / オブジェクトの形式にする必要があります。修飾ジョブ名は、ジョブ番号 / ユーザー / ジョブ名の形式にする必要があります。特殊値 (アスタリスクで始まる) を指定するフィルター値項目は、大文字で指定する必要があります。それぞれのオブジェクトのタイプごとに指定できる特殊値については、496 ページの『SOA 属性の特殊値』を参照してください。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_FILTER_ID
指定のフィルター ID がこのリストのタイプに対して無効です。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。
CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。この API は、既にオープンされているリストに対しては有効とはなりません。フィルター基準を

有効とするためには、この API を呼び出した後でリストをオープンする必要があります。複雑なフィルターを要求すると、リストのパフォーマンスを低下させることがあるため、注意が必要です。

CWBSO_SetListProfile:

これは、System i Access for Windows 製品で使用する API です。

目的

アプリケーション名を Windows レジストリーに追加することによって、プロファイル名を設定します。リストを表示する前に、CWBSO_ReadListProfile を使用して、レジストリーからフィルター情報を読み取ります。また、リストを削除する前に、CWBSO_WriteListProfile を使用して、更新済みのフィルター情報をレジストリーに書き込みます。この API を呼び出さないと、CWBSO_ReadListProfile と CWBSO_WriteListProfile は有効となりません。

構文

```
unsigned int CWB_ENTRY CWBSO_SetListProfile(
                CWBSO_LIST_HANDLE listHandle,
                char far* lpszKey);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

char far* lpszKey - input
リストに関する Windows レジストリー内でのキーとして使用されるストリングを指す long 型のポインター。この名前は、アプリケーション名の場合もあります。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_PROFILE_NAME
指定されたプロファイル名が無効です。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。

CWBSO_SetListSortFields:

これは、System i Access for Windows 製品で使用する API です。

目的

リストのソート基準を設定します。ソート基準では、CWBSO_OpenList の呼び出しによってリストが作成される時点で、オブジェクトがそのリスト内に表示される順序を決定します。この API は、ジョブのリストおよびスプール・ファイルのリストについてのみ有効です。この API は、メッセージのリストおよびプリンターのリストには許可されていません。

構文

```
unsigned int CWB_ENTRY CWBSO_SetListSortFields(
                CWBSO_LIST_HANDLE listHandle,
                unsigned short far* lpusSortIDs,
                unsigned short usCount);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

unsigned short far* lpusSortIDs - input
ソート列識別コードの配列を指す long 型のポインター。指定されたソート ID は、リストの現行のソート基準を置換します。このパラメーターの有効な値は、リストのオブジェクトのタイプによって決まります。有効な値については、下記の該当するヘッダー・ファイルを参照してください。

- cwbsojob.h
- cwbsosfl.h

注: 複数のソート ID が指定される場合、配列内でのソート ID の順序によって、ソートが行われる順序が定義されます。

unsigned short usCount - input
指定されたソート列識別コードの数。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_SORT_ID
指定のソート ID はこのリストのタイプに対しては無効です。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_SORT_NOT_ALLOWED
このリストのタイプに対するソートは許可されていません。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。
CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるた

めです。この API は、既にオープンされているリストに対しては有効とはなりません。ソート基準を有効とするためには、この API を呼び出した後でリストをオープンする必要があります。複雑なソートを要求すると、リストのパフォーマンスが低下することがあるため、注意が必要です。

CWBSO_SetListTitle:

これは、System i Access for Windows 製品で使用する API です。

目的

リストのタイトルを設定します。このタイトルは、CWBSO_DisplayList の呼び出しによってリストが表示される時点で、ウィンドウのタイトル・バーに表示されます。

構文

```
unsigned int CWB_ENTRY CWBSO_SetListTitle(
               CWBSO_LIST_HANDLE listHandle ,
               char far* lpszTitle);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

char far* lpszTitle - input
リストのタイトルに使用されるストリングを指す long 型のポインター。ストリングの長さは、79 以下でなければなりません。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_TITLE
指定されたタイトルが無効です。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。

CWBSO_SetObjAttr:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトの 1 つまたは複数の属性の値を設定します。

構文

```
unsigned int CWB_ENTRY CWBSO_SetObjAttr(
                CWBSO_OBJ_HANDLE objectHandle,
                CWBSO_PARMOBJ_HANDLE parmObjHandle,
                unsigned short far* lpusErrorIndex,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_OBJ_HANDLE objectHandle - input
以前の CWBSO_GetObjHandle または CWBSO_CopyObjHandle の呼び出しによって戻されたオブジェクトのハンドル。

CWBSO_PARMOBJ_HANDLE parmObjHandle - input
以前の CWBSO_CreateParmObjHandle の呼び出しによって戻されたパラメーター・オブジェクトのハンドル。パラメーター・オブジェクトには、そのオブジェクトについて変更すべき属性が入っています。

unsigned short far* lpusErrorIndex - output
エラーが発生した場合、この値が、エラーを引き起こしたパラメーター項目の指標に設定されます。最初のパラメーター項目は 1 です。パラメーター項目のいずれもエラーではない場合、この値は 0 に設定されます。

CWBSO_ERR_HANDLE errorHandle - input
エラー・オブジェクトのハンドル。エラー・テキストのあるエラーが発生した場合、このハンドルを使用してエラー・メッセージおよびメッセージ・ヘルプを検索することができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_OBJECT_HANDLE
指定されたオブジェクト・ハンドルが無効。

CWBSO_BAD_PARMOBJ_HANDLE
指定されたパラメーター・オブジェクト・ハンドルが無効です。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_CANNOT_CHANGE_ATTRIBUTE
属性は現時点では変更できません。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_GetObjHandle または CWBSO_CopyObjHandle を呼び出す必要があります。CWBSO_GetObjHandle または CWBSO_CopyObjHandle で戻されたオブジェクト・ハンドルを、この API への入力として渡す必要があるためです。この API を呼び出す前に、 CWBSO_CreateErrorHandle を

呼び出す必要があります。CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。

CWBSO_SetParameter:

これは、System i Access for Windows 製品で使用する API です。

目的

オブジェクトの 1 つの属性の値を設定します。CWBSO_SetObjAttr を呼び出す前に、この API を複数回呼び出すことができます。これにより、1 つの特定のオブジェクトについて CWBSO_SetObjAttr の一度の呼び出しで複数の属性を変更することができます。

構文

```
unsigned int CWB_ENTRY CWBSO_SetParameter(
                CWBSO_PARMOBJ_HANDLE parmObjHandle,
                unsigned short usAttributeID,
                char far* lpszValue,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_PARMOBJ_HANDLE parmObjHandle - input
以前の CWBSO_CreateParmObjHandle の呼び出しによって戻されたパラメーター・オブジェクトのハンドル。

unsigned short usAttributeID - input
設定されるパラメーターの属性 ID。このパラメーターの有効な値は、オブジェクトのタイプによって決まります。有効な値については、下記の該当するヘッダー・ファイルを参照してください。

- cwbsojob.h
- cwbsomsg.h
- cwbsoprt.h
- cwbsosfl.h

char far* lpszValue - input
属性値を指す long 型のポインター。ASCIIZ ストリングのみが受け入れられることに注意してください。2 進値は、適切なライブラリー関数を使用してストリングに変換する必要があります。それぞれのオブジェクトのタイプごとに指定できる特殊値については、 496 ページの『SOA 属性の特殊値』を参照してください。

CWBSO_ERR_HANDLE errorHandle - input
エラー・オブジェクトのハンドル。エラー・テキストのあるエラーが発生した場合、このハンドルを使用してエラー・メッセージおよびメッセージ・ヘルプを検索することができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_PARMOBJ_HANDLE
指定されたパラメーター・オブジェクト・ハンドルが無効です。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_CreateParmObjHandle を呼び出す必要があります。CWBSO_CreateParmObjHandle で戻されたパラメーター・オブジェクト・ハンドルをこの API への入力として渡す必要があるためです。この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。この API を呼び出しても、System i オブジェクトの属性は更新されません。指定したオブジェクトの System i 属性値 (複数の場合あり) を実際に更新するには、CWBSO_SetObjAttr を呼び出す必要があります。

CWBSO_WaitForObj:

これは、System i Access for Windows 製品で使用する API です。

目的

非同期で作成されているリストでオブジェクトが使用可能になるまで待機します。

構文

```
unsigned int CWB_ENTRY CWBSO_WaitForObj(
                CWBSO_LIST_HANDLE listHandle,
                unsigned long ulPosition,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

unsigned long ulPosition - input
リスト内の、使用したいオブジェクトの位置。注: リスト内の最初のオブジェクトは、位置 0 と見なされます。

CWBSO_ERR_HANDLE errorHandle - input
エラー・オブジェクトのハンドル。エラー・テキストのあるエラーが発生した場合、このハンドルを使用してエラー・メッセージおよびメッセージ・ヘルプを検索することができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_BAD_LIST_POSITION
指定されたリスト内の位置が存在しません。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。この API を呼び出す前に、 CWBSO_CreateErrorHandle を呼び出す必要があります。CWBSO_CreateErrorHandle によって戻されたエラー・ハンドルを、この API の入力として渡す必要があります。

CWBSO_WriteListProfile:

これは、System i Access for Windows 製品で使用する API です。

目的

Windows レジストリー内の指定されたキーに、リストに関するフィルター情報を書き込みます。キー名は、CWBSO_SetListProfile API を使用して、前もって設定されていなければなりません。この API は、リストを削除する前に呼び出す必要があります。そうすることで、CWBSO_DisplayList API 使用時にユーザーが変更したフィルター基準が、すべて保管されます。システムごと、およびリストのタイプごとに、フィルター情報がレジストリーに保管されます。例えば、アプリケーションで 2 つの異なるシステムからオブジェクトにアクセスし、4 つのリスト・タイプをすべて表示した場合、レジストリーには、フィルター情報を指定する 8 つの個別セクションがあることになります。

構文

```
unsigned int CWB_ENTRY CWBSO_WriteListProfile(
                CWBSO_LIST_HANDLE listHandle,
                CWBSO_ERR_HANDLE errorHandle);
```

パラメーター

CWBSO_LIST_HANDLE listHandle - input
以前の CWBSO_CreateListHandle または CWBSO_CreateListHandleEx の呼び出しによって戻されたリストのハンドル。

CWBSO_ERR_HANDLE errorHandle - input
以前の CWBSO_CreateErrorHandle の呼び出しによって作成されたエラー・オブジェクトのハンドル。この API で戻された値が CWBSO_ERROR_OCCURRED である場合、エラー・ハンドルを使って、エラー・メッセージ・テキストの検索、またはユーザーへのエラーの表示を行うことができます。

戻りコード

以下は、共通の戻り値です。

CWBSO_NO_ERROR
エラーは起こりませんでした。

CWBSO_BAD_LIST_HANDLE
指定されたリスト・ハンドルが無効。

CWBSO_BAD_ERR_HANDLE
指定されたエラー・ハンドルが無効。

CWBSO_SYSTEM_NAME_DEFAULTED
そのリストに関する CWBSO_CreateListHandle 呼び出しでシステム名が指定されませんでした。

CWBSO_LOW_MEMORY
要求に対する十分なメモリーがありませんでした。

CWBSO_ERROR_OCCURRED
エラーが発生しました。詳細な情報を入手するには、エラー・ハンドルを使用してください。

使用法

この API を呼び出す前に、CWBSO_CreateListHandle を呼び出す必要があります。CWBSO_CreateListHandle によって戻されたリスト・ハンドルをこの API の入力として渡す必要があるためです。この API を呼び出す前に、CWBSO_SetListProfile を呼び出す必要があります。

SOA 属性の特殊値:

以下のリストにある System i Access for Windows のトピックでは、オブジェクトのタイプごとに、CWBSO_GetObjAttr によって戻される特殊値、および CWBSO_SetObjAttr で指定される特殊値について説明します。また、リスト・オブジェクトのタイプごとに、CWBSO_SetListFilter で指定される特殊値についても説明します。

特別な考慮事項

- 通常、数値である属性については、System i API は負の数値を戻して、どの特殊値 (存在する場合) がオブジェクト属性に含まれているかを示します。システム・オブジェクト・アクセスでは、自動的にこれらの負の数値をそれに対応する特殊値ストリングにマップします。例えば、スプール・ファイル属性の検索 (QUSRSPLA) API では、出力縮小が自動的に行われる場合のページ回転について「-1」を戻します。CWBSO_GetObjAttr は、「*AUTO」を戻します。
- いくつかのリスト・フィルター基準は複数の値を受け入れます。例えば、複数のプリンター名についてプリンターのリストをフィルター処理することが可能です。そのような場合、指定する値はコンマで区切らなければなりません。

属性の特殊値についての追加情報の参照

i5/OS Information Center のトピック『i5/OS API』を参照してください。

ジョブ属性:

システム・オブジェクト・アクセスは、System i API である ジョブのリスト (QUSLJOB) およびジョブ情報の検索 (QUSRJOBI) を使用して、ジョブの属性を検索します。

指定できる特殊値は、i5/OS Information Center の『i5/OS API: Work Management API』トピックで説明されているものと同じです。以下の特殊値マッピングは、明示的には文書化されていません。

CWBSO_JOB_CpuTimeUsed
フィールドが実際の結果を保持するには十分な大きさではない場合、QUSRJOBI は -1 を戻します。システム・オブジェクト・アクセスは「++++」を戻します。

CWBSO_JOB_MaxCpuTimeUsed,

CWBSO_JOB_MaxTemporaryStorage,

CWBSO_JOB_DefaultWaitTime
値が *NOMAX の場合、QUSRJOBI は -1 を戻します。システム・オブジェクト・アクセスは「*NOMAX」を戻します。

CWBSO_SetListFilter は、ジョブのリスト (QUSLJOB) API でサポートされるすべての特殊値を受け入れます。

メッセージ属性:

システム・オブジェクト・アクセスは、非プログラム・メッセージのリスト (QMHLSTM) i5/OS API を使用して、メッセージの属性を検索します。

指定できる特殊値は、i5/OS Information Center の『i5/OS API: Message Handling API』トピックで説明されているものと同じです。

重大度基準については、CWBSO_SetListFilter は、非プログラム・メッセージのリスト (QMHLSTM) API でサポートされる特殊値を受け入れます。さらに、CWBSO_MSGF_UserName フィルター ID を指定することによって、10 文字のユーザー名を与えることができます。「*CURRENT」を使用して、現行ユーザーについてのメッセージのリストを入手することができます。

プリンター属性:

システム・オブジェクト・アクセスは文書化されていない System i API を使用して、プリンター・オブジェクトについての属性を検索します。

プリンターは「論理」オブジェクトであり、実際には装置記述、書き出しプログラム、および出力待ち行列を組み合わせたものです。この属性および指定できる値は以下のとおりです。

CWBSO_PRT_AdvancedFunctionPrinting
プリンターが高機能印刷™ (AFP) をサポートするかどうか。

***NO** プリンターは高機能印刷をサポートしません。

***YES** プリンターは高機能印刷をサポートします。

CWBSO_PRT_AllowDirectPrinting
プリンターに直接印刷するジョブに、プリンターを割り振ることを印刷装置書き出しプログラムが許可するかどうか。

***NO** 直接印刷は許可されません。

***YES** 直接印刷は許可されます。

CWBSO_PRT_BetweenCopiesStatus
複数コピー・スプール・ファイルのコピーとコピーの間に書き出しプログラムが使用可能かどうか。指定できる値は、Y (はい) または N (いいえ) です。

CWBSO_PRT_BetweenFilesStatus
書き出しプログラムがスプール・ファイル間で使用可能かどうか。指定できる値は、Y (はい) または N (いいえ) です。

CWBSO_PRT_ChangesTakeEffect
書き出しプログラムに対する保留中の変更が効力を持つ時点。指定できる値は以下のとおりです。

***NORDYF**
現行の有資格ファイルのすべてが印刷されるとき。

***FILEEND**
現行のスプール・ファイルの印刷が行われるとき。

ブランク
書き出しプログラムに対する保留中の変更はありません。

CWBSO_PRT_CopiesLeftToProduce
まだ印刷していないコピー数。印刷するファイルがない場合、このフィールドは 0 に設定されます。

CWBSO_PRT_CurrentPage
現在、書き出しプログラムによって処理中の、スプール・ファイルのページ番号。示されたページ番号は、印刷されている実際のページ番号より前かまたは後である場合があります。これは、システムによって行われるバッファー方式のためです。印刷されるスプール・ファイルがない場合、このフィールドは 0 に設定されます。

CWBSO_PRT_Description
プリンターのテキスト記述。

CWBSO_PRT_DeviceName
プリンターの名前。

CWBSO_PRT_DeviceStatus
プリンターの状況。指定できる値は、構成状況の検索 (QDCRCFGS) API で戻される装置状況と同じです。

CWBSO_PRT_EndAutomatically
書き出しプログラムが、自動的に終了する場合にいつ終了させるか。

***NORDYF**
書き出しプログラムが印刷すべきファイルを選択する出力待ち行列に、印刷準備状態のファイルがないとき。

***FILEEND**
現行のスプール・ファイルの印刷終了時。

***NO** 書き出しプログラムは終了せず、さらにスプール・ファイルを待機します。

CWBSO_PRT_EndPendingStatus
書き出しプログラム終了 (ENDWTR) コマンドが、この書き出しプログラムに対して出されたかどうか。指定できる値は以下のとおりです。

N ENDWTR コマンドは出されませんでした。

I *IMMED: 出力バッファーが空になるとすぐに書き出しプログラムは終了します。

C *CNTRLD: スプール・ファイルの現行コピーが印刷された後で書き出しプログラムは終了します。

P *PAGEEND: 書き出しプログラムはページの終わりで終了します。

CWBSO_PRT_FileName
現在、書き出しプログラムによって処理中の、スプール・ファイル名。印刷しているファイルがない場合、このフィールドはブランクです。

CWBSO_PRT_FileNumber
現在、書き出しプログラムによって処理中の、スプール・ファイルの番号。印刷されるスプール・ファイルがない場合、このフィールドは 0 に設定されます。

CWBSO_PRT_FormsAlignment
用紙位置決めメッセージが送信される時点。指定できる値は以下のとおりです。

***WTR** 書き出しプログラムがメッセージをいつ送信するかを決定します。

***FILE** ページ位置決めの制御は、各ファイルによって指定されます。

CWBSO_PRT_FormType
スプール・ファイルの印刷に使用している用紙のタイプ。指定できる値は以下のとおりです。

***ALL** いかなる用紙タイプであってもすべてのスプール・ファイルを印刷するというオプションで、書き出しプログラムが開始されます。

***FORMS**
異なる用紙タイプを使用する前に、同じ用紙タイプ指定を持つすべてのスプール・ファイルを印刷するというオプションで、書き出しプログラムが開始されます。

***STD** 用紙タイプ指定が *STD である、すべてのスプール・ファイルを印刷するというオプションで、書き出しプログラムが開始されます。

用紙タイプ名
ユーザーが指定した用紙タイプを持つすべてのスプール・ファイルを印刷するというオプションで、書き出しプログラムが開始されます。

CWBSO_PRT_FormTypeNotification
この用紙の終了時にメッセージ待ち行列へメッセージを送信するためのメッセージ・オプション。指定できる値は以下のとおりです。

***MSG** メッセージがメッセージ待ち行列へ送信されます。

***NOMSG**
メッセージはメッセージ待ち行列へ送信されません。

***INFOMSG**
通知メッセージがメッセージ待ち行列へ送信されます。

***INQMSG**
照会メッセージがメッセージ待ち行列へ送信されます。

CWBSO_PRT_HeldStatus
書き出しプログラムが保留されるかどうか。指定できる値は、Y (はい) または N (いいえ) です。

CWBSO_PRT_HoldPendingStatus
書き出しプログラムの保留 (HLDWTR) コマンドがこの書き出しプログラムについて出されたかどうか。指定できる値は以下のとおりです。

N HLDWTR コマンドは出されませんでした。

I *IMMED: 出力バッファーが空になるとすぐに書き出しプログラムは保留されます。

C *CNTRLD: ファイルの現行コピーが印刷された後で書き出しプログラムは保留されます。

P *PAGEEND: ページの終わりで書き出しプログラムは保留されます。

CWBSO_PRT_JobName
現在、書き出しプログラムによって処理中のスプール・ファイルを作成したジョブの名前。印刷しているスプール・ファイルがない場合、このフィールドはブランクです。

CWBSO_PRT_JobNumber
現在、書き出しプログラムによって処理中のスプール・ファイルを作成したジョブの番号。印刷しているスプール・ファイルがない場合、このフィールドはブランクです。

CWBSO_PRT_MessageKey
書き出しプログラムが応答を待っているメッセージへのキー。書き出しプログラムが照会メッセージへの応答を待っていない場合、このフィールドはブランクになります。

CWBSO_PRT_MessageQueueLibrary
メッセージ待ち行列が入っているライブラリーの名前。

CWBSO_PRT_MessageQueueName
この書き出しプログラムが操作上のメッセージに使用するメッセージ待ち行列の名前。

CWBSO_PRT_MessageWaitingStatus
照会メッセージへの応答を書き出しプログラムが待っているかどうか。指定できる値は、Y (はい) または N (いいえ) です。

CWBSO_PRT_NextFormType
次に印刷する用紙タイプの名前。指定できる値は以下のとおりです。

***ALL** いずれの用紙タイプであってもすべてのスプール・ファイルを印刷するというオプションに、書き出しプログラムが変更されます。

***FORMS**
異なる用紙タイプを使用する前に、同じ用紙タイプ指定を持つすべてのスプール・ファイルを印刷するというオプションに、書き出しプログラムが変更されます。

***STD** 用紙タイプ指定が *STD である、すべてのスプール・ファイルを印刷するというオプションに、書き出しプログラムが変更されます。

用紙タイプ名
ユーザーが指定した用紙タイプを持つすべてのスプール・ファイルを印刷するというオプションに、書き出しプログラムが変更されます。

ブランク
この書き出しプログラムに対して変更は行われませんでした。

CWBSO_PRT_NextFormTypeNotification
次の用紙タイプの終了時に、メッセージ待ち行列へメッセージを送信するためのメッセージ・オプション。指定できる値は以下のとおりです。

***MSG** メッセージがメッセージ待ち行列へ送信されます。

***NOMSG**
メッセージはメッセージ待ち行列へ送信されません。

***INFOMSG**
通知メッセージがメッセージ待ち行列へ送信されます。

***INQMSG**
照会メッセージがメッセージ待ち行列へ送信されます。

ブランク
この書き出しプログラムに対して変更は行われませんでした。

CWBSO_PRT_NextOutputQueueLibrary
次の出力待ち行列が入っているライブラリーの名前。書き出しプログラムに対して変更が行われなかった場合、このフィールドはブランクです。

CWBSO_PRT_NextOutputQueueName
次に処理する出力待ち行列の名前。書き出しプログラムに対して変更が行われなかった場合、このフィールドはブランクです。

CWBSO_PRT_NextSeparatorDrawer
この値は、書き出しプログラムに対する変更がある場合に、分離ページを取り出す用紙入れを示します。指定できる値は以下のとおりです。

***FILE** 区切りページは、スプール・ファイルが印刷されるのと同じ用紙入れから印刷されます。色付きまたは異なるタイプの用紙が入っている、スプール・ファイルとは異なる用紙入れをユーザーが指定すれば、区切りページがさらに識別しやすくなります。

***DEVD**
区切りページは、プリンター記述で指定された区切りページ用紙入れから印刷されます。

空ストリング
書き出しプログラムに対する保留中の変更はありません。

1 1 番目の用紙入れ。

2 2 番目の用紙入れ。

3 3 番目の用紙入れ。

CWBSO_PRT_NextSeparators
書き出しプログラムに対する変更が行われるときに印刷される区切りページの次の数。指定できる値は以下のとおりです。

***FILE** 区切りページの数はファイルごとに指定されます。

空ストリング
書き出しプログラムに対する保留中の変更はありません。

区切りページの数
印刷される区切りページの数。

CWBSO_PRT_NumberOfSeparators
印刷される区切りページの数。指定できる値は以下のとおりです。

***FILE** 区切りページの数はファイルごとに指定されます。

区切りページの数
印刷される区切りページの数。

CWBSO_PRT_OnJobQueueStatus
書き出しプログラムがジョブ待ち行列にあり、そのため現在実行中ではないかどうか。指定できる値は、Y (はい) または N (いいえ) です。

CWBSO_PRT_OutputQueueLibrary
スプール・ファイルが印刷のために選択される、出力待ち行列が入っているライブラリーの名前。

CWBSO_PRT_OutputQueueName
スプール・ファイルが印刷のために選択される、出力待ち行列の名前。

CWBSO_PRT_OutputQueueStatus
スプール・ファイルが印刷のために選択される、出力待ち行列の状況。指定できる値は以下のとおりです。

H 出力待ち行列は保留されています。

R 出力待ち行列は解放されています。

CWBSO_PRT_PrinterDeviceType
スプール・ファイルの印刷に使用されているプリンターのタイプ。有効な値は以下のとおりです。

***SCS** SNA (システム・ネットワーク体系) 文字ストリーム

***IPDS** Intelligent Printer Data Stream

CWBSO_PRT_SeparatorDrawer
ジョブおよびファイルの区切りページが取り出される用紙入れを識別します。指定できる値は以下のとおりです。

***FILE** ファイルが印刷される場合と同じ用紙入れから、区切りページは印刷されます。色付きまたは異なるタイプの用紙が入っている、ファイルとは異なる用紙入れをユーザーが指定すれば、区切りページがさらに識別しやすくなります。

***DEVD**
区切りページは、プリンター記述で指定された区切りページ用紙入れから印刷されます。

1 1 番目の用紙入れ。

2 2 番目の用紙入れ。

3 3 番目の用紙入れ。

CWBSO_PRT_StartedByUser
書き出しプログラムを開始したユーザーの名前。

CWBSO_PRT_Status
論理プリンターの全体的な状況。このフィールドは、プリンター状況 (構成状況の検索 QDCRCFGS API からのもの)、出力待ち行列状況 (プリンターと書き出しプログラム状況のリスト、および XPF マクロからのもの)、および書き出しプログラム状況 (書き出しプログラム情報の検索、QSPRWTRI API からのもの) から取り込まれます。指定できる値は以下のとおりです。

1 使用不可

2 電源オフまたはまだ使用不可

3 停止状態

4 メッセージ待ち状態

5 保留

6 停止 (保留中)

7 保留 (保留中)

8 プリンターを待機中

9 開始を待機中

10 印刷中

11 プリンター出力の待機中

12 接続保留中

13 電源オフ

14 使用不可

15 サービス中

999 認識不能

CWBSO_PRT_TotalCopies
印刷されるコピーの合計数。

CWBSO_PRT_TotalPages
スプール・ファイル内のページの合計数。指定できる値は以下のとおりです。

数値 スプール・ファイル内のページ数。

0 印刷中のスプール・ファイルはありません。

CWBSO_PRT_User
現在、書き出しプログラムによって処理中のスプール・ファイルを作成したユーザーの名前。印刷しているファイルがない場合、このフィールドはブランクです。

CWBSO_PRT_UserSpecifiedData
現在、書き出しプログラムによって処理中のファイルを記述しているユーザー指定のデータ。印刷しているファイルがない場合、このフィールドはブランクです。

CWBSO_PRT_WaitingForDataStatus
書き出しプログラムが、現在スプール・ファイルにあるすべてのデータを書き込み済みで、さらにデータを待っているかいないか。指定できる値は以下のとおりです。

N 書き出しプログラムは、それ以上データを待っていません。

Y 書き出しプログラムは現在スプール・ファイルにあるすべてのデータを書き込み済みで、さらにデータを待っています。この条件が発生するのは、書き出しプログラムが、SCHEDULE(*IMMED) を指定したオープン・スプール・ファイルを生成しているときです。

CWBSO_PRT_WaitingForDeviceStatus
プリンターに直接印刷を行っているジョブから装置を獲得するのを書き出しプログラムが待っているかどうか。

N 書き出しプログラムは装置を待っていません。

Y 書き出しプログラムは装置を待っています。

CWBSO_PRT_WriterJobName
印刷装置書き出しプログラムのジョブ名。

CWBSO_PRT_WriterJobNumber
印刷装置書き出しプログラムのジョブ番号。

CWBSO_PRT_WriterJobUser
システム・ユーザーの名前。

CWBSO_PRT_WriterStarted
このプリンターに対して書き出しプログラムが開始しているかどうかを指示します。指定できる値は以下のとおりです。

0 書き出しプログラムは開始されていません。

1 書き出しプログラムは開始されています。

CWBSO_PRT_WriterStatus
このプリンターについての書き出しプログラムの状況。指定できる値は以下のとおりです。

X'01' 開始済み

X'02' 終了済み

X'03' ジョブ待ち行列中

X'04' 保留

X'05' メッセージ待ち状態

CWBSO_PRT_WritingStatus
印刷装置書き出しプログラムが書き込み状況にあるかどうか。指定できる値は以下のとおりです。

Y 書き出しプログラムは書き込み状況にあります。

N 書き出しプログラムは書き込み状況にありません。

S 書き出しプログラムはファイル区切りを書き込み中です。

システム・オブジェクト・アクセスは、コンマで区切られたプリンター名のリストを受け入れます。最高 100 個のプリンター名の指定が可能です。 System i の全プリンターのリストを要求するには、特殊値の「*ALL」を指定します。

プリンター出力属性:

システム・オブジェクト・アクセスは、System i API であるスプール・ファイルのリスト (QUSLSPL) およびスプール・ファイル属性の検索 (QUSRSPLA) を使用して、プリンター出力の属性を検索します。

指定できる特殊値は、i5/OS Information Center の『i5/OS APIs: Spooled File APIs』トピックで説明されているものと同じです。以下の特殊値マッピングは、明示的には文書化されていません。

CWBSO_SFL_StartingPage
終了ページの値が使用される場合、QUSRSPLA は -1 を戻します。システム・オブジェクト・アクセスは「*ENDPAGE」を戻します。

CWBSO_SFL_EndingPage
最後のページが終了ページになる場合、QUSRSPLA は 0 または 2147483647 を戻します。システム・オブジェクト・アクセスは「*END」を戻します。

CWBSO_SFL_MaximumRecords
最大がない場合、QUSRSPLA は 0 を戻します。システム・オブジェクト・アクセスは「*NOMAX」を戻します。

CWBSO_SFL_PageRotation
回転が行われない場合、QUSRSPLA は 0 を戻します。システム・オブジェクト・アクセスは「*NONE」を戻します。

1 つの文書化されていない API が、スプール・ファイルの 1 つまたは複数のプリンター名を検索するために使用されます。その属性および指定できる値について以下で説明します。

CWBSO_SFL_DeviceNames
ファイルを印刷するプリンターの名前。プリンター出力が複数のプリンターに割り当てられている場合、このフィールドには、プリンター・グループ内のすべてのプリンター名が入ります。指定できる値は以下のとおりです。

プリンター名
プリンター出力が割り当てられているプリンターの名前。

プリンター名のリスト
プリンター出力が割り当てられているグループ内のプリンターの名前。プリンター名はコンマで区切られます。

空ストリング
プリンター出力がプリンターまたはプリンター・グループに割り当てられていません。

CWBSO_SetListFilter は、スプール・ファイルのリスト (QUSLSPL) API でサポートされるすべての特殊値を受け入れます。

TCP/IP インターフェース属性:

システム・オブジェクト・アクセスは、System i API である ネットワーク・インターフェースのリスト (QtocLstNetIfc) を使用して、TCP/IP インターフェースの属性を検索します。

システム・オブジェクト・アクセスを使用して TCP/IP インターフェースの属性を検索するには、以下の API のいずれか 1 つを使用します。

- 変更 IPv4 インターフェース (QTOCC4IF) API
  - この API はプログラム一時修正 (PTF) によって文書化されています。PTF の詳細については、以下のページの検索機能に SI17284 を入力して参照してください。
    - System i Access for Windows Service Packs (http://www.ibm.com/servers/eserver/iseries/access/casp.htm)
- リスト・ネットワーク・インターフェース (QtocLstNetIfc) API

イーサネット回線属性:

イーサネット回線に関する情報は、System i Access for Windows 構成 API のトピックにあります。

i5/OS Information Center の『Configuration APIs』を参照してください。

トークンリング回線属性:

トークンリング回線に関する情報は、System i Access for Windows 構成 API のトピックにあります。

i5/OS Information Center の『Configuration APIs』を参照してください。

ハードウェア資源属性:

ハードウェア資源に関する情報は、System i Access for Windows ハードウェア資源 API のトピックにあります。

i5/OS Information Center の『Hardware Resource APIs』のトピックを参照してください。

ソフトウェア・プロダクト属性:

ソフトウェア・プロダクトに関する情報は、System i Access for Windows ソフトウェア・プロダクト API のトピックにあります。

i5/OS Information Center の『Software Product APIs』トピックを参照してください。

TCP/IP 経路属性:

システム・オブジェクト・アクセスは、System i API である TCP/IP 経路 (QTOCRTEU) を使用して、TCP/IP 経路の属性を検索します。

指定できる特殊値は以下のとおりです。

CWBSO_RTE_TCPIPNetworkName

CWBSO_RTE_InternetAddress

CWBSO_RTE_BinaryInternetAddress
*RTVxxxLST の場合のみ - 入出力変数ヘッダーの直後に、戻された経路のリストが表示されます。インターフェース構造体は、戻された経路ごとに繰り返されます。

CWBSO_RTE_SubnetMask

CWBSO_RTE_BinarySubnetMask
*RTVxxxLST の場合のみ - 入出力変数ヘッダーの直後に、戻された経路のリストが表示されます。インターフェース構造体は、戻された経路ごとに繰り返されます。

CWBSO_RTE_NextHopAddress

CWBSO_RTE_BinaryNextHop
*RTVxxxLST の場合のみ - 入出力変数ヘッダーの直後に、戻された経路のリストが表示されます。インターフェース構造体は、戻された経路ごとに繰り返されます。

CWBSO_RTE_BindingInterface

CWBSO_RTE_BinaryBindingIP
*RTVxxxLST の場合のみ - 入出力変数ヘッダーの直後に、戻された経路のリストが表示されます。インターフェース構造体は、戻された経路ごとに繰り返されます。

CWBSO_RTE_MaximumTransmissionUnit

CWBSO_RTE_TypeOfService

- 1=通常
- 2=最小遅延
- 3=最大スループット
- 4=最大信頼性
- 5=最小コスト

CWBSO_RTE_RoutePrecedence

CWBSO_RTE_RIPMetric

CWBSO_RTE_RIPRedistribution

- 1=はい
- 2=いいえ

CWBSO_RTE_PPPProfile
*xxxRTE には無効

CWBSO_RTE_PPPCallerUserid
*xxxRTE には無効

CWBSO_RTE_PPPCallerIP
*xxxRTE には無効

CWBSO_RTE_ApplicationDefined

ユーザーとグループ属性:

このリストを使用して、System i ユーザーおよびグループの有効な特殊値を識別します。

- CWBSO_USR_ProfileName
- CWBSO_USR_ProfileOrGroupIndicator
- CWBSO_USR_GroupHasMembers
- CWBSO_USR_TextDescription
- CWBSO_USR_PreviousSignonDate
- CWBSO_USR_PreviousSignonTime
- CWBSO_USR_SignonAttemptsNotValid
- CWBSO_USR_Status
- CWBSO_USR_PasswordChangeDate
- CWBSO_USR_NoPasswordIndicator
- CWBSO_USR_PasswordExpirationInterval
- CWBSO_USR_DatePasswordExpires
- CWBSO_USR_DaysUntilPasswordExpires
- CWBSO_USR_SetPasswordToExpire
- CWBSO_USR_DisplaySignonInformation
- CWBSO_USR_UserClassName
- CWBSO_USR_AllObjectAccess
- CWBSO_USR_SecurityAdministration
- CWBSO_USR_JobControl
- CWBSO_USR_SpoolControl
- CWBSO_USR_SaveAndRestore
- CWBSO_USR_SystemServiceAccess
- CWBSO_USR_AuditingControl
- CWBSO_USR_SystemConfiguration
- CWBSO_USR_GroupProfileName
- CWBSO_USR_Owner
- CWBSO_USR_GroupAuthority
- CWBSO_USR_LimitCapabilities
- CWBSO_USR_GroupAuthorityType
- CWBSO_USR_SupplementalGroups
- CWBSO_USR_AssistanceLevel
- CWBSO_USR_CurrentLibraryName

- CWBSO_USR_InitialMenuName
- CWBSO_USR_InitialMenuLibraryName
- CWBSO_USR_InitialProgramName
- CWBSO_USR_InitialProgramLibraryName
- CWBSO_USR_LimitDeviceSessions
- CWBSO_USR_KeyboardBuffering
- CWBSO_USR_MaximumAllowedStorage
- CWBSO_USR_StorageUsed
- CWBSO_USR_HighestSchedulingPriority
- CWBSO_USR_JobDescriptionName
- CWBSO_USR_JobDescriptionNameLibrary
- CWBSO_USR_AccountingCode
- CWBSO_USR_MessageQueueName
- CWBSO_USR_MessageQueueLibraryName
- CWBSO_USR_MessageQueueDeliveryMethod
- CWBSO_USR_MessageQueueSeverity
- CWBSO_USR_OutputQueue
- CWBSO_USR_OutputQueueLibrary
- CWBSO_USR_PrintDevice
- CWBSO_USR_SpecialEnvironment
- CWBSO_USR_AttentionKeyHandlingProgramName
- CWBSO_USR_AttentionKeyHandlingProgramLibrary
- CWBSO_USR_LanguageID
- CWBSO_USR_CountryID
- CWBSO_USR_CharacterCodeSetID
- CWBSO_USR_ShowParameterKeywords
- CWBSO_USR_ShowAllDetails
- CWBSO_USR_DisplayHelpOnFullScreen
- CWBSO_USR_ShowStatusMessages
- CWBSO_USR_DoNotShowStatusMessages
- CWBSO_USR_ChangeDirectionOfRollkey
- CWBSO_USR_SendMessageToSpoolFileOwner
- CWBSO_USR_SortSequenceTableName
- CWBSO_USR_SortSequenceTableLibraryName
- CWBSO_USR_DigitalCertificateIndicator
- CWBSO_USR_CharacterIDControl
- CWBSO_USR_ObjectAuditValue
- CWBSO_USR_CommandUsage
- CWBSO_USR_ObjectCreation
- CWBSO_USR_ObjectDeletion

- CWBSO_USR_JobTasks
- CWBSO_USR_ObjectManagement
- CWBSO_USR_OfficeTasks
- CWBSO_USR_ProgramAdoption
- CWBSO_USR_SaveAndRestoreTasks
- CWBSO_USR_SecurityTasks
- CWBSO_USR_ServiceTasks
- CWBSO_USR_SpoolManagement
- CWBSO_USR_SystemManagement
- CWBSO_USR_OpticalTasks
- CWBSO_USR_UserIDNumber
- CWBSO_USR_GroupIDNumber
- CWBSO_USR_DoNotSetAnyJobAttributes
- CWBSO_USR_UseSystemValue
- CWBSO_USR_CodedCharacterSetID
- CWBSO_USR_DateFormat
- CWBSO_USR_DateSeparator
- CWBSO_USR_SortSequenceTable
- CWBSO_USR_TimeSeparator
- CWBSO_USR_DecimalFormat
- CWBSO_USR_HomeDirectoryDelimiter
- CWBSO_USR_HomeDirectory
- CWBSO_USR_Locale
- CWBSO_USR_IndirectUser
- CWBSO_USR_PrintCoverPage
- CWBSO_USR_MailNotification
- CWBSO_USR_UserID
- CWBSO_USR_LocalDataIndicator
- CWBSO_USR_UserAddress
- CWBSO_USR_SystemName
- CWBSO_USR_SystemGroup
- CWBSO_USR_UserDescription
- CWBSO_USR_FirstName
- CWBSO_USR_PreferredName
- CWBSO_USR_MiddleName
- CWBSO_USR_LastName
- CWBSO_USR_FullName
- CWBSO_USR_JobTitle
- CWBSO_USR_CompanyName
- CWBSO_USR_DepartmentName

- CWBSO_USR_NetworkUserID
- CWBSO_USR_PrimaryTelephoneNumber
- CWBSO_USR_SecondaryTelephoneNumber
- CWBSO_USR_FaxNumber
- CWBSO_USR_Location
- CWBSO_USR_BuildingNumber
- CWBSO_USR_OfficeNumber
- CWBSO_USR_MailingAddress
- CWBSO_USR_MailingAddress2
- CWBSO_USR_MailingAddress3
- CWBSO_USR_MailingAddress4
- CWBSO_USR_CCMailAddress
- CWBSO_USR_CCMailComment
- CWBSO_USR_MailServerFrameworkServiceLevel
- CWBSO_USR_PreferredAddressFieldName
- CWBSO_USR_PreferredAddressProductID
- CWBSO_USR_PreferredAddressTypeValue
- CWBSO_USR_PreferredAddressTypeName
- CWBSO_USR_PreferredAddress
- CWBSO_USR_ManagerCode
- CWBSO_USR_SMTPUserID
- CWBSO_USR_SMTPDomain
- CWBSO_USR_SMTPRoute
- CWBSO_USR_GroupMemberIndicator

注: V4R4 以降では、Lotus Notes® が System i プラットフォームにインストールされている場合にのみ、次の属性が有効になります。

- CWBSO_USR_NotesServerName
- CWBSO_USR_NotesCertifierID
- CWBSO_USR_MailType
- CWBSO_USR_NotesMailFileName
- CWBSO_USR_CreateMailFiles
- CWBSO_USR_NotesForwardingAddress
- CWBSO_USR_SecurityType
- CWBSO_USR_LicenseType
- CWBSO_USR_MinimumNotesPasswordLength
- CWBSO_USR_UpdateExistingNotesUser
- CWBSO_USR_NotesMailServer
- CWBSO_USR_LocationWhereUserIDIsStored
- CWBSO_USR_ReplaceExistingNotesID

- CWBSO_USR_NotesComment
- CWBSO_USR_NotesUserLocation
- CWBSO_USR_UserPassword
- CWBSO_USR_NotesUserPassword
- CWBSO_USR_NotesCertifierPassword
- CWBSO_USR_ShortName

QSYS のライブラリー属性:

QSYS のライブラリーに関する情報は、System i Access for Windows オブジェクト API のトピックにあります。

i5/OS Information Center の『オブジェクト API (Object APIs)』のトピックを参照してください。

System i Access for Windows: データベース・プログラミング

System i Access for Windows には、データベース・ファイルにアクセスするためのプログラミング・インターフェースが複数あります。

共通インターフェースを使用して、System i データベースと非 System i データベースの両方にアクセスする、単一のアプリケーションを作成することができます。構造化照会言語 (SQL) を使用して、DB2® for i5/OS のデータベース・ファイルにアクセスできます。また、ストアード・プロシージャーやレコード・レベルのアクセス・インターフェースを使用して、ファイル内の単一レコードにアクセスすることもできます。

以下のトピックでは、サポートされるインターフェースに関する情報を紹介します。また、i5/OS Information Center にある『DB2 for i5/OS SQL 解説書』の一連のトピックから、DB2 for i5/OS の SQL プログラミングに関する資料にアクセスして、詳細を参照してください。

注: コード例を使用することで、 633 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』の条件に同意します。

関連情報

DB2 for i5/OS SQL 解説書

System i Access for Windows .NET Provider

System i Access for Windows .NET Provider によって、.NET 管理対象プログラムから System i データベース・ファイルへのアクセスを、SQL を使用して行うことができます。

System i Access for Windows .NET のサポートについては、次のいずれかの名称が使われることがあります。

- **管理対象のプロバイダー**
- **IBM DB2 for i5/OS .NET Provider**
- **IBM.Data.DB2.iSeries のデータ・プロバイダー**

使用される名称にかかわらず、System i での .NET Data Access Framework 接続時に、このデータ・プロバイダーによって PC-to-System i SQL アプリケーションの開発とサポートが実現されます。このプロバイ

ダーは、ADO.NET アーキテクチャー・モデルで定義され、サポートされている接続、コマンド、DataAdapter、および DataReader の各機能へのアクセスを提供する、クラスとデータ・タイプのセットから構成されます。

IBM.Data.DB2.iSeries Data Provider は、既存の OLE DB データベース・プロバイダーを補完するものです。このプロバイダーにより、Visual Basic および C# を使用して .NET クライアント/サーバー・アプリケーションを開発することが可能になります。このプロバイダーと Programmer's Toolkit を併用すれば、.NET Windows クライアント PC アプリケーションをより短時間かつ簡単に開発することができます。

管理対象のプロバイダーは、.NET Framework が PC に既にインストールされているという要件を含め、管理対象コードの .NET Framework 仕様に準拠します。このフレームワークのインストール後に System i Access for Windows の各フィーチャーのインストールまたは除去を行う場合は、「ユーザーズ・ガイド」を参照してください。

Microsoft の .NET Framework、ADO.NET、Windows Installer、GAC、CLR のアーキテクチャーおよび詳細、ならびに管理対象コードの仕様については、Microsoft の Web サイト (英語) を参照してください。

技術的な詳細へのアクセス

- 「**IBM DB2 for i5/OS .NET Provider Technical Reference**」(System i Access for Windows 製品に同梱されています) には、**管理対象のプロバイダー**のサポートに関する詳細な説明が記載されています。この情報にアクセスする手順は、次のとおりです: **「スタート」 → 「プログラム」 → 「System i Access for Windows」 → 「Programmer's Toolkit」 → 「Programmer's Toolkit」 → 「共通インターフェース」 → 「ADO.NET」**
- **制限**
  - V5R2M0 より前のサーバー上での iDB2CommandBuilder に関する制限が一部あります。

.NET framework

Microsoft の .NET Framework、ADO.NET、Windows Installer、GAC、CLR のアーキテクチャーおよび詳細、ならびに管理対象コードの仕様については、MicrosoftWeb サイト を参照してください。

Programmer's Toolkit をインストールするには、次の手順を実行します。

- Programmer's Toolkit は、System i Access for Windows 製品のインストール時にオプションでインストールすることもできますし、製品のインストール後に「**選択セットアップ**」を実行してインストールすることもできます。『Programmer's Toolkit』を参照してください。

その他の .NET 情報源

- IBM System i Access for Windows .NET Provider の Web サイト (英語)
- IBM Redbook Integrating DB2 for i5/OS with Microsoft ADO .NET (SG24-6440)

V5R2M0 より前のサーバーにおける iDB2CommandBuilder の制限

System i の制限により、V5R2M0 より前のリリースの i5/OS では、各システムにおける iDB2CommandBuilder の使用についてサポートが制限されています。

V5R2M0 より前のサーバーに接続する場合に重要なのは、 iDB2CommandBuilder で使用する iDB2Command オブジェクトで Select コマンド・テキストを正しく指定することです。以下は、V5R2M0 より前のサーバー上で使用する Select ステートメントを作成する場合の推奨ガイドラインです。

- 単純ステートメントが最適な結果を出します。例えば、SELECT * FROM MYSCHEMA.MYTABLE などです。
- テーブル名をそのスキーマで完全修飾してください。例えば、MYSCHEMA.MYTABLE などです。
- 選択フィールドも使用できますが、単純なフォーマットで指定する必要があります。 QUERY テーブルで指定した列のみを使用するようにしてください。例えば、SELECT ID、NAME、BALANCE FROM MYSCHEMA.MYTABLE などです。
- 選択基準で派生フィールドや派生定数を使用しないようにしてください。これらを使用すると予測不能な結果が生じることがあります。例えば、SELECT ID、LENGTH(NAME)、'Name' FROM MYSCHEMA.MYTABLE などです。

System i Access for Windows OLE DB Provider

System i データベース・ファイルへのレコード・レベル・アクセスおよび SQL アクセスをサポートします。このサポートを活用するためには、ActiveX データ・オブジェクト (ADO) および OLE DB インターフェースを使用します。

System i Access for Windows OLE DB Provider を Programmer's Toolkit と組み合わせて使用することによって、Windows クライアント PC で System i クライアント/サーバー・アプリケーション開発を短期間で簡単に行うことができます。 System i Access for Windows OLE DB Provider 構成要素により、System i のプログラマーは、System i の DB2 for i5/OS 論理および物理データベース・ファイルに対する、レコード・レベルでのアクセス・インターフェースを利用することができます。また、SQL、データ待ち行列、プログラム、およびコマンドのサポートも提供されます。

ADO 規格と OLE DB 規格によって、System i のデータとサービスへの一貫性のあるインターフェースがプログラマーに提供されます。 3 つのプロバイダー (**IBMDA400**、**IBMDASQL**、および **IBMDARLA**) すべてにおいて、System i から PC への変換およびデータ・タイプ間の変換がすべて処理されます。

OLE DB Provider のインストール方法

この Provider をインストールする場合には、「ユーザーズ・ガイド」でフィーチャーのインストールと削除に関するトピックを参照してください。

注: System i Access for Windows 製品をインストールする前にコンピューターに MDAC 2.5 以降がインストールされていない場合には、OLE DB Provider はインストールされません。 MDAC は Microsoft Web サイト: www.microsoft.com/data/doc.htm からダウンロードすることができます。

OLE DB Technical Reference へのアクセス

System i Access for Windows の OLE DB Technical Reference (System i Access for Windows 製品に同梱されています) では、OLE DB Provider サポートに関する詳細な資料が提供されています。 Programmer's Toolkit からこれにアクセスするには、「**概要**」 → 「**共通インターフェース**」 → 「**ADO/OLE DB**」と選択します。

Programmer's Toolkit をインストールするには、次の手順を実行します。

この Toolkit をインストールする場合には、「ユーザーズ・ガイド」でフィーチャーのインストールと削除に関するトピックを参照してください。

その他の OLE DB 情報源

- IBM System i Access for Windows OLE DB サポート Web サイト  。
- IBM Redbook Fast Path to iSeries™ Client/Server Using iSeries OLE DB Support: SG24-5183 

関連資料

631 ページの『ActiveX プログラミング』
ActiveX オートメーションは、Microsoft によって定義されたプログラミング・テクノロジーであり、System i Access for Windows 製品でサポートされています。

System i Access ODBC

ODBC は、データベース・アクセス言語として SQL を使用する共通データベース・インターフェースです。System i Access 製品では、このインターフェースにサポートを提供するために、ODBC ドライバーをサポートしています。

ODBC とは

ODBC とはオープン・データベース接続のことです。 ODBC は以下のもので構成されます。

- 適切に定義された関数のセット (アプリケーション・プログラミング・インターフェース)
- SQL 構文用の標準 (推奨されているが課せられてはいない)
- エラー・コード
- データ・タイプ

アプリケーション・プログラミング・インターフェースには、データベース管理システムへの接続、SQL ステートメントの実行、データのリトリーブを行う豊富な関数のセットが用意されています。 API には、データベースの SQL カタログとドライバーの機能を問い合わせる関数も含まれています。

ODBC ドライバーは、標準エラー・コードを戻し、データ・タイプを共通 (ODBC) 標準に変換します。ODBC を使用することによって、アプリケーション開発者は、統合データベースのエラー情報を入手し、アプリケーションを移植可能にする際に生じる最も複雑な問題の一部を回避することができるようになります。

ODBC でユーザーが行えること

ODBC を使用して以下を行うことができます。

- SQL 要求をデータベース管理システム (DBMS) へ送信する。
- 同じプログラムを、再コンパイルせずに使用して、いろいろなデータベース管理システム (DBMS) プロダクトにアクセスする。
- データ通信プロトコルから独立したアプリケーションを作成する。
- アプリケーションに使いやすい形式でデータを処理する。

ODBC の API は柔軟性があるため、(SQL が事前定義されている) トランザクション・ベースの基幹業務アプリケーション、および (Select ステートメントが実行時に作成される) QUERY ツールでも使用することができます。

構造化照会言語 (SQL)

ODBC は動的 SQL を使用するためパフォーマンスが低下することがあります。しかし、パラメーター・マーカーをうまく利用してステートメントを繰り返して使用することにより、静的 SQL と同じようなパフ

ォーマンスを得ることが可能になります。また、System i Access for Windows ODBC ドライバーの特殊機能である拡張動的 SQL を使用すると、準備済み SQL ステートメントによって、静的 SQL に匹敵するパフォーマンスを得ることができます。

SQL について詳しくは、IBM「*SQL 解説書*」ブックを参照してください。i5/OS Information Center にある『DB2 for i5/OS SQL 解説書』の一連のトピックから、上記ブックの HTML オンライン版を表示するか、PDF 版を印刷してください。以下の関連リンクを参照してください。

System i Access ODBC トピック:

注: このページからリンクされている情報は、System i Access for Windows 32 ビット ODBC ドライバーのサポート、System i Access for Windows 64 ビット ODBC ドライバーのサポート、および System i Access for Linux® ODBC ドライバーのサポートに該当します。 System i Access for Linux 環境でのセットアップに関する詳細については、i5/OS Information Center の『System i Access for Linux』の一連のトピックを参照するリンク (以下参照) を選択してください。

ODBC 標準に関する資料を探すには、Microsoft Web サイトで ODBC を検索してください。

注: コード例を使用することで、 633 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』の条件に同意します。

関連情報

DB2 for i5/OS SQL 解説書

System i Access for Windows Linux

Microsoft Web サイト

ODBC アプリケーションの作成に必要なファイル

ODBC アプリケーションの作成に必要な System i Access for Windows ファイルを識別します。

ODBC アプリケーションの作成で使用されるファイルおよびその他の概念の情報については、以下のトピックを選択してください。

注: Programmer's Toolkit は ODBC ドキュメンテーションを提供し、またサンプル・プログラムおよび関連情報にリンクしています。この情報にアクセスするには、Programmer's Toolkit をオープンして、「**データベース**」 → 「**ODBC**」と選択します。

ODBC ドライバーにアクセスするためのインターフェースの選択:

System i Access の ODBC ドライバーでは、さまざまなプログラミング・インターフェースを使用することができます。各インターフェースには、それぞれ長所と短所があります。

共通性の高いプログラミング・インターフェースとして、ActiveX Data Objects (ADO)、Rapid Application Development (RAD) ツール、および ODBC API の 3 つがあります。以下に、これらの 3 つのインターフェースについて、サポートされる言語、使用する理由、および詳しい情報の入手先を示します。

ActiveX Data Objects (ADO)

ADO は ActiveX Data Object の略称であり、Microsoft 社のデータ・アクセス用高水準オブジェクト・モデルです。

- サポートされるプログラミング言語
  - Visual Basic

  - Active Server Pages (ASP)
  - Delphi
  - Visual Basic Script
  - ActiveX または COM をサポートするその他の言語またはスクリプト
- この方式を使用する理由
  - ODBC API のコーディングを行わずに済ませる
  - 必要に応じてプロバイダーの切り替えをサポートする
- 詳細情報の入手先
  - ADO の詳細な使用方法については、MDAC として配布されている ADO 文書 (www.microsoft.com/data/doc.htm) を参照してください。
  - ADO を介した System i Access OLE-DB Provider の使用法については、513 ページの『System i Access for Windows OLE DB Provider』を参照してください。
- 特別な注意事項
  - ADO を介して ODBC を使用するためには、アプリケーションにおいて、接続ストリングで MSDASQL プロバイダーを指定する必要があります。 MSDASQL は、ADO 呼び出しを、ODBC ドライバーと通信する ODBC API 呼び出しに変換します。
  - ADO 接続ストリングの使用例は、以下のとおりです。

```
ConnectionString = ″Provider=MSDASQL;Data Source=MYODBCDS;″
```

Rapid Application Development (RAD) ツール

Rapid Application Development ツールは、アプリケーションを迅速に作成する上で役立つツールです。これらのツールを使用すると、アプリケーションの作成者は、ODBC 仕様に関する詳しい知識が不要になります。

- サポートされるプログラミング言語
  - 使用される RAD ツールによって異なります。
  - 一般的に使用されるツールとしては、Powerbuilder、Delphi、および Seagate Crystal Reports などがあります。
- この方式を使用する理由
  - ODBC API のコーディングを行わずに済ませる
  - 1 つのプログラムを使用して、変更をほとんど、あるいはまったく行わずに複数の ODBC ドライバーを操作する
- 詳細情報の入手先
  - RAD ツールに組み込まれている資料を参照してください。

直接 ODBC API 呼び出し

直接 ODBC API 呼び出しは、アプリケーションが ODBC 仕様に合わせて直接作成される場合に行われます。

- サポートされるプログラム言語

  C/C++
- この方式を使用する理由

- どの ODBC API を呼び出すのかを直接制御することができるため、ADO オブジェクトや RAD ツールを使用する場合よりも迅速な制御が可能
- ドライバー固有の機能を利用するように設計されている

• 詳細情報の入手先
- ODBC の仕様およびサンプルについては、MDAC として配布されている ODBC 文書 (www.microsoft.com/data/doc.htm) を参照してください。
- ドライバー固有の機能については、 538 ページの『ODBC API のインプリメンテーションに関する事項』を参照してください。

ODBC C/C++ アプリケーション・ヘッダー・ファイル:

System i Access の ODBC C/C++ アプリケーションの C/C++ ヘッダー・ファイルを識別します。

ヘッダー・ファイル	インポート・ライブラリー	ダイナミック・リンク・ライブラリー
sql.h sqlext.h sqltypes.h sqlucode.h	odbc32.lib	odbc32.dll

ODBC API: 一般概念:

System i Access の ODBC API に適用される一般概念は、次のとおりです。

環境 ODBC がその実行時情報をモニターできるように、Windows が一部のメモリーを使用可能にできる環境のことです。

接続 環境内では、データ・ソースへの、複数の接続が可能です。別々の物理サーバーに接続することも、同じサーバーに接続することも、あるいはそれらの任意の組み合わせに接続することも可能です。

ステートメント
各接続内で複数のステートメントを実行することができます。

ハンドル
ハンドルとは、ドライバー・マネージャーや個々のドライバーによって割り振られる記憶域の ID のことです。ハンドルには、次の 3 タイプがあります。

環境ハンドル
他のハンドルも含むグローバル情報です。1 つのアプリケーションにつき 1 つのハンドルが許されます。

接続ハンドル
データ・ソースへの接続に関する情報です。環境当たり、複数の接続ハンドルが許されます。

ステートメント・ハンドル
特定の SQL ステートメントに関する情報です。接続当たり、複数のステートメント・ハンドルが許されます。ステートメント・ハンドルは、そのステートメント状態が有効である限り、他の SQL ステートメントでも再利用可能です。

記述子ハンドル
接続ハンドルに関連した明示的な記述子に関する情報です。アプリケーションは、これらを作成し、ステートメント・ハンドルに関連した暗黙的な記述子の代わりに、これらの記述子ハンドルを使用するようにドライバーに依頼します。

基本的には、**ハンドル**は、ODBC に認識されている資源 (この場合は、環境、接続、またはステートメント) の ID と考えることができます。 ODBC が、プログラムで使用することができるこの資源の ID (ハンドル) を与えます。 ODBC が、ハンドルの中に保管する (長整数として保持される) ものが何であるかを正確に把握する必要はありません。注意すべきことは、値を変更することなく、さまざまなハンドルを保持している変数に固有の名前を割り当てることです。

一部の API がハンドル (例えば、ハンドル・タイプが SQL_HANDLE_ENV である、SQLAllocEnv または SQLAllocHandle) を設定するので、変数に対する参照、つまり、ポインターを渡す必要があります。一部の API では、前に設定されたハンドル (例えば **SQLExecute** など) を参照するので、変数を値で渡す必要があります。

パラメーター・マーカー:

パラメーター・マーカーは、System i Access for Windows SQL ステートメントをデータ・ソースで実行するように求めた際に、プログラムが提供した値に対するプレースホルダーとしての役割を果たします。

SQLPrepare を使用することによって、パラメーター・マーカーが指定されているステートメントは、SQL 594 ページの『最適化ルーチン』で準備したデータ・ソースに渡されます。これによって、この最適化ルーチンはステートメントのプランを作成し、後で参照できるようにパラメーター・マーカーを保持します。それぞれのパラメーター・マーカーには、プログラム変数 (厳密には、プログラム変数を指すポインター) を関連付けておく必要があり、このために **SQLBindParameter** を使用します。

SQLBindParameter は使い方が複雑な関数であることから、 t「*Microsoft ODBC Software Development Kit and Programmer's Reference*」ISBN 1-57231-516-4 にある関連するセクションを入念にお読みになることをお勧めします。 **SQLBindParameter** を使用すれば、大部分の SQL ステートメントで関数に入力情報を提供することができますが、ストアード・プロシージャーの場合は、さらに、データを受け取ることも可能になります。

ステートメントの作成とパラメーターのバインドを終えると、**SQLExecute** を使用して、関連した変数の現行値をデータ・ソースに設定することができるようになります。

SQLFetch および SQLGetData:

SQLBindCol の代わりに **SQLGetData** を使うと、各コマンドを System i Access for Windows 関数で使用して、検索済みの行の列からデータを取り出すことができます。これは、配列サイズが 1 の場合に、取り出し API を呼び出してからしか呼び出せません。

概して、**SQLBindCol** の方が **SQLGetData** よりも使用に適しています。 **SQLBindCol** では、データの取り出しごとに実行するのではなく、1 回実行するのみであることから、パフォーマンスのオーバーヘッドが少なくて済みます。ただし、Visual Basic における **SQLBindCol** の使用については、特別な考慮事項があります。

Visual Basic では、メモリーを節約するために、文字ストリングをさまざまな場所に移動します。ストリング変数が 1 つの列にバインドされた場合は、後続の **SQLFetch** で参照されるメモリーがデータを所定の変数に入れない可能性があります。結果として**一般保護違反**になる可能性が高くなります。
SQLBindParameter の場合にも同じような問題が生じることがあります。

Visual Basic でストリングを使用することはお勧めしません。この問題の回避策としては、**バイト配列**を使用する方法があります。バイト配列は固定サイズで、メモリー内の移動は行われません。

もう一つの回避策として、Microsoft Development Library Knowledge Base に記載されている、 Windows メモリー割り振り API 関数を使用する方法があります。ただし、この方法には、Windows 3.1 とそれ以後のリリースの間で完全な互換性がないという、プログラミング上の問題が伴います。

SQLBindCol ではなく **SQLGetData** を使用し、 **SQLParamData** と **SQLPutData** を **SQLBindParameter** と一緒に使用すると、Visual Basic にさらに適合したソフトウェアが作成されます。ただしこの方法には、プログラミング上の問題が伴います。

ODBC API への直接コーディング:

PC アプリケーションの多くで、ユーザーが異種プラットフォーム上のデータにシームレスにアクセスできる ODBC 呼び出しが行われます。 ODBC API で独自の System i Access アプリケーションの開発を始める前に、ODBC アプリケーションをデータベース・サーバーに接続して、サーバーと情報を交換する方法について、理解しておいてください。

以下の処理を行う ODBC API がサポートされています。

- ODBC 環境の設定
- データ・ソースへの接続の確立と切断
- SQL ステートメントの実行
- ODBC 環境のクリーンアップ

関連資料

626 ページの『例: Visual Basic - ストアード・プロシージャーの呼び出しによるデータへのアクセスと戻し』
System i のストアード・プロシージャーの作成、準備、バインド、および呼び出しについて、Visual Basic の例を使って説明します。

ストアード・プロシージャーの呼び出し:

System i Access ODBC アプリケーションのパフォーマンスと機能を高めるためにはストアード・プロシージャーを使用します。

どの System i プログラムもストアード・プロシージャーの機能を果たすことができます。 System i ストアード・プロシージャーは、入力、入出力、および出力のパラメーターをサポートします。さらに、ストアード・プロシージャーでは、単一の結果セットと複数の結果セットの戻りもサポートします。カーソルをリターンへ指定する (組み込み SQL ステートメントから) か、または、値の配列を指定することによって、ストアード・プロシージャー・プログラムが結果セットを戻すことができます。詳しくは、トピック『ストアード・プロシージャー』を参照してください。

ストアード・プロシージャーを呼び出すためには次のステップを行います。

1. SQL ステートメント CREATE PROCEDURE を使用して、ストアード・プロシージャーが宣言されていることを検査する。

   詳細: ストアード・プロシージャーが実行されている間、CREATE PROCEDURE を実行する必要があるのは 1 回のみです。 DROP PROCEDURE を使用すると、プロシージャーのプログラムを削除しないでプロシージャーを削除することができます。 DECLARE PROCEDURE も使用できますが、この方式には不利な点がいくつかあります。「*Database Programming (DB2 UDB サーバー (AS/400 版) データベース・プログラミング)*」ブックに、 DECLARE PROCEDURE の詳細につ

いて記述されています。 i5/OS Information Center にある『DB2 for i5/OS SQL 解説書』のトピックから、上記ブックの HTML オンライン版を表示するか、PDF 版を印刷してください。

2. *SQL Prepare* を使用して、ストアード・プロシージャーの呼び出しを準備する。
3. 入力パラメーターと出力パラメーターをバインドする。
4. ストアード・プロシージャーへ呼び出しを実行する。
5. 結果セットを取り出す (結果が戻された場合)。

この C の例では、デフォルトの System i ライブラリーにある NEWORD という名前の COBOL プログラムを呼び出しています。 *szCustId* という名前のフィールドの値が渡され、*szName* という名前のフィールドに値が戻されます。

```
SQLRETURN rc;
HSTMT hstmt;
SQLCHAR Query[320];
SQLCHAR szCustId[10];
SQLCHAR szName[30];
SQLINTEGER strlen_or_indPtr = SQL_NTS, strlen_or_indPtr2 = SQL_NTS;

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

// Create the stored procedure definition.
// The create procedure could be moved to the application's
// install program so that it is only executed once.
strcpy(Query,"CREATE PROCEDURE NEWORD (:CID IN CHAR(10), :NAME OUT CHAR(30) )");
strcat(Query," (EXTERNAL NAME NEWORD LANGUAGE COBOL GENERAL WITH NULLS)");

// Create the stored procedure
rc = SQLExecDirect(hstmt, (unsigned char *)Query, SQL_NTS);

strcpy(Query, "CALL NEWORD(?,?)");

// Prepare the stored procedure call
rc = SQLPrepare(hstmt, (unsigned char *)Query, SQL_NTS);

// Bind the parameters
rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
                                    10, 0, szCustId, 11, &strlen_or_intPtr);

rc = SQLBindParameter(hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR, SQL_VARCHAR,
                                    30, 0, szName, 31, &strlen_or_indPtr2);
strcpy (szCustId,"0000012345");
// Execute the stored procedure
rc = SQLExecute(hstmt);
```

関連資料

615 ページの『ストアード・プロシージャー』
ストアード・プロシージャーは、System i データベース・トランザクション処理でサポートされています。

関連情報

DB2 for i5/OS SQL 解説書

ブロック挿入およびブロック取り出しの C の例:

ブロック挿入およびブロック取り出しを使用して、System i Access の ODBC アプリケーションのパフォーマンスを改善することができます。

これらを使用すると、個別にではなく、ブロック単位で行を挿入したり、取り出したりすることができます。この機能では、クライアントとサーバー間のデータ・フローと回線反転が削減されます。ブロック取り出しは、SQLFetch (順方向専用)、SQLExtendedFetch または SQLFetchScroll API のいずれかを使用して実行することができます。

ブロック取り出しは、以下のとおりです。

- バインド済みの列ごとに配列の形式で、ブロック単位のデータ (1 行のセット) を戻す。
- スクロール・タイプの引数の設定に従って、結果セットをスクロールする (設定には、フォワード、バックワード、または行番号がある)。
- SQLSetStmtAttr API で指定された行セット・サイズを使用する。

以下の C の例では、6 行のデータを 1 回ブロック挿入した後、 2 行のデータを 2 回ブロック取り出しています。

```
#define NUM_ROWS_INSERTED 6
#define NAME_LEN          10

HSTMT hstmt;
SQLINTEGER rowcnt = NUM_ROWS_INSERTED;
SQLCHAR itemNames[NUM_ROWS_INSERTED][NAME_LEN+1] = { "puzzle    ", "candy bar ",
 "gum       ","kite      ", "toy car   ", "crayons   " };
SQLINTEGER itemPrices[NUM_ROWS_INSERTED] = { 5, 2, 1, 10, 3, 4 };
SQLCHAR queryItemNames[NUM_ROWS_INSERTED][NAME_LEN+1];  // Name return array
SQLINTEGER queryItemPrices[NUM_ROWS_INSERTED];  // price return array
SQLINTEGER cbqueryItemNames[NUM_ROWS_INSERTED], cbqueryItemPrices[NUM_ROWS_INSERTED];

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

rc = SQLExecDirect(hstmt, "CREATE TABLE ITEMS (NAME VARCHAR(10), PRICE INT)", SQL_NTS);

// set the paramset size to 6 as we are block inserting 6 rows of data
rc = SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)rowcnt, SQL_IS_INTEGER);

// bind the arrays to the parameters
rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
                                    NAME_LEN, 0, itemNames[0], NAME_LEN + 1, NULL);
rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER,
                                    NUM_ROWS_INSERTED, 0, &itemPrices[0],
                                     sizeof(long), NULL);

// do the block insert
rc = SQLExecDirect(hstmt, "INSERT INTO ITEMS ? ROWS VALUES(?,?)", SQL_NTS);

ROWS VALUES(?,?)", SQL_NTS);

// set up things for the block fetch

// We set the concurrency below to SQL_CONCUR_READ_ONLY, but since SQL_CONCUR_READ_ONLY
// is the default this API call is not necessary.
If update was required then you would use
// SQL_CONCUR_LOCK value as the last parameter.
rc = SQLSetStmtAttr(hstmt, SQL_ATTR_CONCURRENCY, (SQLPOINTER)SQL_CONCUR_READ_ONLY,
                              SQL_IS_INTEGER);

// We set the cursor type to SQL_CURSOR_FORWARD_ONLY, but since SQL_CURSOR_FORWARD_ONLY
// is the default this API call is not necessary.
rc = SQLSetStmtAttr(hstmt, SQL_ATTR_CURSOR_TYPE,
                                 (SQLPOINTER)SQL_CURSOR_FORWARD_ONLY, SQL_IS_INTEGER);

// We want to block fetch 2 rows at a time so we need to set SQL_ATTR_ROW_ARRAY_SIZE to 2.
// If we were going to use SQLExtendedFetch instead of SQLFetchScroll we would instead need
// to set the statement attribute SQL_ROWSET_SIZE to 2.
rc = SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_ARRAY_SIZE, (SQLPOINTER)2, SQL_IS_INTEGER);
```

```
rc = SQLExecDirect(hstmt, "SELECT NAME, PRICE FROM ITEMS WHERE PRICE < 5", SQL_NTS);

// bind arrays to hold the data for each column in the result set
rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, queryItemNames, NAME_LEN + 1, cbqueryItemNames);
rc = SQLBindCol(hstmt, 2, SQL_C_LONG, queryItemPrices, sizeof(long), cbqueryItemPrices);

// We know that there are 4 rows that fit the criteria for the SELECT statement so we call
// two fetches to get all the data
rc = SQLFetchScroll(hstmt, SQL_FETCH_FIRST, 0);
// at this point 2 rows worth of data will have been fetched and put into the buffers
// that were bound by SQLBindCol

rc = SQLFetchScroll(hstmt, SQL_FETCH_NEXT, 0);
// at this point 2 rows worth of data will have been fetched and put into the buffers
// that were bound by SQLBindCol. Note that this second fetch overwrites the data in
// those buffers with the new data
// ...
// Application processes the data in bound columns...
// ...
```

関連資料

597 ページの『ODBC ブロック化 INSERT ステートメント』
このブロック化ステートメントは、System i Access ODBC に複数の行を挿入する場合に使用します。

例: Visual Basic によるブロック挿入:

「パラメーター化された」挿入よりもかなり高速な、System i Access for Windows の Visual Basic によるブロック挿入の例です。

ブロック挿入では、次のことが実行できます。

- 1 回の SQL 呼び出しを使って、ブロック単位のレコードを挿入する。
- クライアントとサーバー間のフローを削減する。

詳細については、 520 ページの『ブロック挿入およびブロック取り出しの C の例』を参照してください。

```
Dim cbNTS(BLOCKSIZE - 1)    As Long           'NTS array
    Dim lCustnum(BLOCKSIZE - 1) As Long           'Customer number array

    '2nd parm passed by actual length for demo purposes
    Dim szLstNam(7, BLOCKSIZE - 1)  As Byte      'NOT USING NULL ON THIS PARM
    Dim cbLenLstNam(BLOCKSIZE - 1)  As Long      'Actual length of string to pass
    Dim cbMaxLenLstNam              As Long      'Size of one array element

    'These will be passed as sz string so size must include room for null
    Dim szInit(3, BLOCKSIZE - 1)    As Byte      'Size for field length + null
    Dim szStreet(13, BLOCKSIZE - 1) As Byte      'Size for field length + null
    Dim szCity(6, BLOCKSIZE - 1)    As Byte      'Size for field length + null
    Dim szState(2, BLOCKSIZE - 1)   As Byte      'Size for field length + null
    Dim szZipCod(5, BLOCKSIZE - 1)  As Byte      'Size for field length + null

    Dim fCdtLmt(BLOCKSIZE - 1) As Single
    Dim fChgCod(BLOCKSIZE - 1) As Single
    Dim fBalDue(BLOCKSIZE - 1) As Single
    Dim fCdtDue(BLOCKSIZE - 1) As Single

    Dim irow        As Long           ' row counter for block errors
    Dim lTotalRows  As Long           '  ************ Total rows to send *************
    Dim lNumRows    As Long           '  Rows to send in one block
    Dim lRowsLeft   As Long           '  Number of rows left to send

    Dim I As Long
```

```
 Dim J As Long
 Dim S As String
 Dim hStmt As Long

' This program needs QCUSTCDT table in your own collection.
' At the System i  command line type:
'===> CRTLIB SAMPCOLL
'===> CRTDUPOBJ OBJ(QCUSTCDT) FROMLIB(QIWS)
'        OBJTYPE(*FILE) TOLIB(SAMPCOLL) NEWOBJ(*SAME)
'===> CHGPF FILE(SAMPCOLL/QCUSTCDT) SIZE(*NOMAX)
'===> CLRPFM FILE(SAMPCOLL/QCUSTCDT)

'************** Start ****************************************************
S = "Number of records to insert into QCUSTCDT.  "
S = S & "Use menu option Table Mgmt, Create QCUSTCDT to "
S = S & "create the table.  Use Misc, System i Cmd and CLRPFM "
S = S & "command if you wish to clear it"
S = InputBox(S, gAppName, "500")
If Len(S) = 0 Then Exit Sub

lTotalRows = Val(S)             'Total number to insert

rc = SQLAllocHandle(SQL_HANDLE_STMT, ghDbc, hStmt)
If (Not (rc = SQL_SUCCESS Or rc = SQL_SUCCESS_WITH_INFO)) Then GoTo errBlockInsert

rc = SQLPrepare(hStmt, _
         "INSERT INTO QCUSTCDT ? ROWS VALUES (?,?,?,?,?,?,?,?,?,?,?)", _
        SQL_NTS)
If (Not (rc = SQL_SUCCESS Or rc = SQL_SUCCESS_WITH_INFO)) Then GoTo errBlockInsert


  rc = SQLBindParameter(hStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, _
                        10, 0, lCustnum(0), 0, ByVal 0)
  If (rc = SQL_ERROR) Then _
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")

  'Pass first parm w/o using a null
  cbMaxLenLstNam = UBound(szLstNam, 1) - LBound(szLstNam, 1) + 1
  rc = SQLBindParameter(hStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                        8, _
                        0, _
                        szLstNam(0, 0), _
                        cbMaxLenLstNam, _
                        cbLenLstNam(0))
  If (rc = SQL_ERROR) Then _
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")

  rc = SQLBindParameter(hStmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                       3, 0, szInit(0, 0), _
                       UBound(szInit, 1) - LBound(szInit, 1) + 1, _
                       cbNTS(0))
  If (rc = SQL_ERROR) Then _
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")

  rc = SQLBindParameter(hStmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                       13, 0, szStreet(0, 0), _
                       UBound(szStreet, 1) - LBound(szStreet, 1) + 1, _
                       cbNTS(0))
  If (rc = SQL_ERROR) Then _
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")

  rc = SQLBindParameter(hStmt, 5, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                       6, 0, szCity(0, 0), _
                       UBound(szCity, 1) - LBound(szCity, 1) + 1, _
                       cbNTS(0))
  If (rc = SQL_ERROR) Then _
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")
```

```
  rc = SQLBindParameter(hStmt, 6, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                        2, 0, szState(0, 0), _
                        UBound(szState, 1) - LBound(szState, 1) + 1, _
                        cbNTS(0))
  If (rc = SQL_ERROR) Then _
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")

  rc = SQLBindParameter(hStmt, 7, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_NUMERIC, _
                          5, 0, szZipCod(0, 0), _
                          UBound(szZipCod, 1) - LBound(szZipCod, 1) + 1, _
                          cbNTS(0))
  If (rc = SQL_ERROR) Then _
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")

  rc = SQLBindParameter(hStmt, 8, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                           4, 0, fCdtLmt(0), 0, ByVal 0)
  If (rc = SQL_ERROR) Then _
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")

  rc = SQLBindParameter(hStmt, 9, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                           1, 0, fChgCod(0), 0, ByVal 0)
  If (rc = SQL_ERROR) Then _
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")
  rc = SQLBindParameter(hStmt, 10, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                           6, 2, fBalDue(0), 0, ByVal 0)
  If (rc = SQL_ERROR) Then _
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")
  rc = SQLBindParameter(hStmt, 11, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                           6, 2, fCdtDue(0), 0, ByVal 0)
  If (rc = SQL_ERROR) Then _
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")


  lRowsLeft = lTotalRows        'Initialize row counter
  For J = 0 To ((lTotalRows - 1) ¥ BLOCKSIZE)
     For I = 0 To BLOCKSIZE - 1
        cbNTS(I) = SQL_NTS                        ' init array to NTS
        lCustnum(I) = I + (J * BLOCKSIZE)         'Customer number = row number
        S = "Nam" & Str(lCustnum(I))              'Last Name
        cbLenLstNam(I) = Len(S)
        rc = String2Byte2D(S, szLstNam(), I)
        'Debug info: Watch address to see layout
        addr = VarPtr(szLstNam(0, 0))
        'addr = CharNext(szLstNam(0, I))           'address of 1,I
        'addr = CharPrev(szLstNam(0, I), szLstNam(1, I))     'address of 0, I)
        'addr = CharNext(szLstNam(1, I))
        'addr = CharNext(szLstNam(6, I))           'should point to null (if used)
        'addr = CharNext(szLstNam(7, I))           'should also point to next row

        rc = String2Byte2D("DXD", szInit, I)
        'Vary the length of the street
        S = Mid("1234567890123", 1, ((I Mod 13) + 1))
        rc = String2Byte2D(S, szStreet, I)

        rc = String2Byte2D("Roches", szCity, I)
        rc = String2Byte2D("MN", szState, I)
        rc = String2Byte2D("55902", szZipCod, I)
        fCdtLmt(I) = I
        fChgCod(I) = 1
        fBalDue(I) = 2 * I
        fCdtDue(I) = I / 2
     Next I

     lNumRows = lTotalRows Mod BLOCKSIZE      ' Number of rows to send in this block
     If (lRowsLeft >= BLOCKSIZE) Then _
                 lNumRows = BLOCKSIZE         ' send remainder or full block
```

```
        irow = 0
        lRowsLeft = lRowsLeft - lNumRows

        rc = SQLSetStmtAttr(hStmt, SQL_ATTR_PARAMSET_SIZE, lNumRows, 0)
        If (rc = SQL_ERROR) Then GoTo errBlockInsert

        rc = SQLSetStmtAttr(hStmt, SQL_ATTR_PARAMS_PROCESSED_PTR, irow, 0)
        If (rc = SQL_ERROR) Then GoTo errBlockInsert

        rc = SQLExecute(hStmt)
        If (rc = SQL_ERROR) Then
           S = "Error on Row: " & Str(irow) & Chr(13) & Chr(10)
           MsgBox S, , gAppName
           GoTo errBlockInsert
        End If
     Next J
     rc = SQLEndTran(SQL_HANDLE_DBC, ghDbc, SQL_COMMIT)
     If (Not (rc = SQL_SUCCESS Or rc = SQL_SUCCESS_WITH_INFO)) Then GoTo errBlockInsert
     rc = SQLFreeHandle(SQL_HANDLE_STMT, hStmt)
     Exit Sub



errBlockInsert:
     rc = SQLEndTran(SQL_HANDLE_DBC, ghDbc, SQL_ROLLBACK)
     rc = SQLFreeHandle(SQL_HANDLE_STMT, hStmt)

Public Function String2Byte2D(InString As String, OutByte() As Byte, RowIdx As Long)
As Boolean
  'VB byte arrays are layed out in memory opposite of C.  The string would
  'be by column instead of by row so must flip flop the string.
  'ASSUMPTIONS:
  '   Byte array is sized before being passed
  '   Byte array is padded with nulls if > size of string

   Dim I As Integer
   Dim SizeOutByte As Integer
   Dim SizeInString As Integer

   SizeInString = Len(InString)
   SizeOutByte = UBound(OutByte, 1)

   'Convert the string
   For I = 0 To SizeInString - 1
      OutByte(I, RowIdx) = AscB(Mid(InString, I + 1, 1))
   Next I
   'If byte array > len of string pad
   If SizeOutByte > SizeInString Then             'Pad with Nulls
      For I = SizeInString To SizeOutByte - 1
         OutByte(I, RowIdx) = 0
      Next I
   End If
   'ViewByteArray OutByte, "String2Byte"
   String2Byte2D = True
End Function
```

Visual Basic: Jet と ODBC API の間の均衡:

データベース・オブジェクトは容易にコーディングできますが、パフォーマンスに悪影響を及ぼすこともあります。また、API およびストアード・プロシージャーのコーディングには、非常に労力を要することがあります。 Windows 95 環境で Visual Basic エンタープライズ版を使用している場合には、追加オプションを使用できます。これらのオプションは、データベース・オブジェクトの使用可能度と API のハイパフォーマンスとの間の妥協点になっています。ここで、API とは、リモート・データ・オブジェクト (RDO) およびリモート・データ・コントロール (RDC) のことを指します。

データベース・オブジェクトは容易にコーディングできますが、パフォーマンスに悪影響を及ぼすこともあります。また、API およびストアード・プロシージャーのコーディングには、非常に労力を要することがあります。 Windows 95 環境で Visual Basic エンタープライズ版を使用している場合には、追加オプションを使用できます。これらのオプションは、データベース・オブジェクトの使用可能度と API のハイパフォーマンスとの間の妥協点になっています。ここで、API とは、リモート・データ・オブジェクト (RDO) およびリモート・データ・コントロール (RDC) のことを指します。

RDO は ODBC API 上の薄いレイヤーです。 RDO によって、API レベルのプログラミングが要求されずに、拡張 ODBC 機能への単純なインターフェースが提供されます。 RDO は、Jet エンジンに制御されているデータ・アクセス・オブジェクト (DAO)、または DAO の SQL 最適化ルーチンのオーバーヘッドのすべてを持っているわけではありません。しかし、RDO には DAO とほぼ等しいプログラミング・インターフェースがあります。 DAO へのプログラミングを理解している場合は、RDO へ移行する方が、API 呼び出しへ移行するよりも簡単です。

DAO と RDO の相違点を以下に示します。

- DAO モデルは ISAM、Access および ODBC データベース用に使用されています。 RDO モデルは、ODBC データベース専用として設計されており、MicrosoftSQL サーバー 6.0 および Oracle 用に最適化されています。
- ローカル・マシン上ではなくサーバー上で処理を行っているため、RDO モデルのほうがパフォーマンスが向上しています。 DAO モデルにおいては、処理の一部はローカルで行われるのでパフォーマンスの面では RDO モデルに劣ります。
- DAO モデルでは Jet エンジンが使用されています。RDO モデルでは Jet エンジンではなく、ODBC バックエンド・エンジンが使用されています。
- RDO モデルでは、同期または非同期の照会を実行することができます。 DAO モデルでは、同期または非同期の照会の実行には制限があります。
- RDO モデルでは複合カーソルを実行できますが、DAO モデルでは複合カーソルの実行は制限されています。

RDC は標準データ・コントロールに類似したデータ・コントロールです。このことは、ユーザーがデータ・コントロールや Jet エンジンを使用した場所であれば、どこでも RDC を使用できることを意味します。フォーム上の ″data aware″ 制御をドラッグしてください。それによって、標準データ・コントロールにバインドするのと同じように RDC にバインドすることができます。

RDO によって可能になる拡張 ODBC 機能のいくつかは、準備済み SQL ステートメント、複数の結果セット、およびストアード・プロシージャーです。Jet が SQL ステートメントを動的に実行する場合、System i プロセスは 2 ステップで実行されます。最初のステップでは、システムがステートメントを確認して、要求されたデータを取り出すための最適なプランを、現行のデータベース・スキーマに基づいて決定します。 2 番目のステップでは、そのプランは実際にデータを取り出すために使用されます。System i では多数の選択肢を評価して、データへの最適なアクセス方法を決定するため、プランの作成に時間がかかる場合があります。SQL ステートメントが実行されるたびにシステムにアクセス・プランの再作成を強制することに代わる別の方法があります。 **rdoConnection** オブジェクトの **CreatePreparedStatement** メソッドを使用すると、SQL ステートメントを実行することなしに、このステートメントの System i データ・アクセス・プランをコンパイルすることができます。準備済みステートメントにパラメーターを含めることもできるため、選択ステートメントを実行するたびに新規の選択基準を渡すこともできます。

以下の Visual Basic サンプル・コードは、パラメーター・マーカーを使用して SQL ステートメントを準備する方法と、異なる値でそのステートメントを複数回実行する方法を示しています。

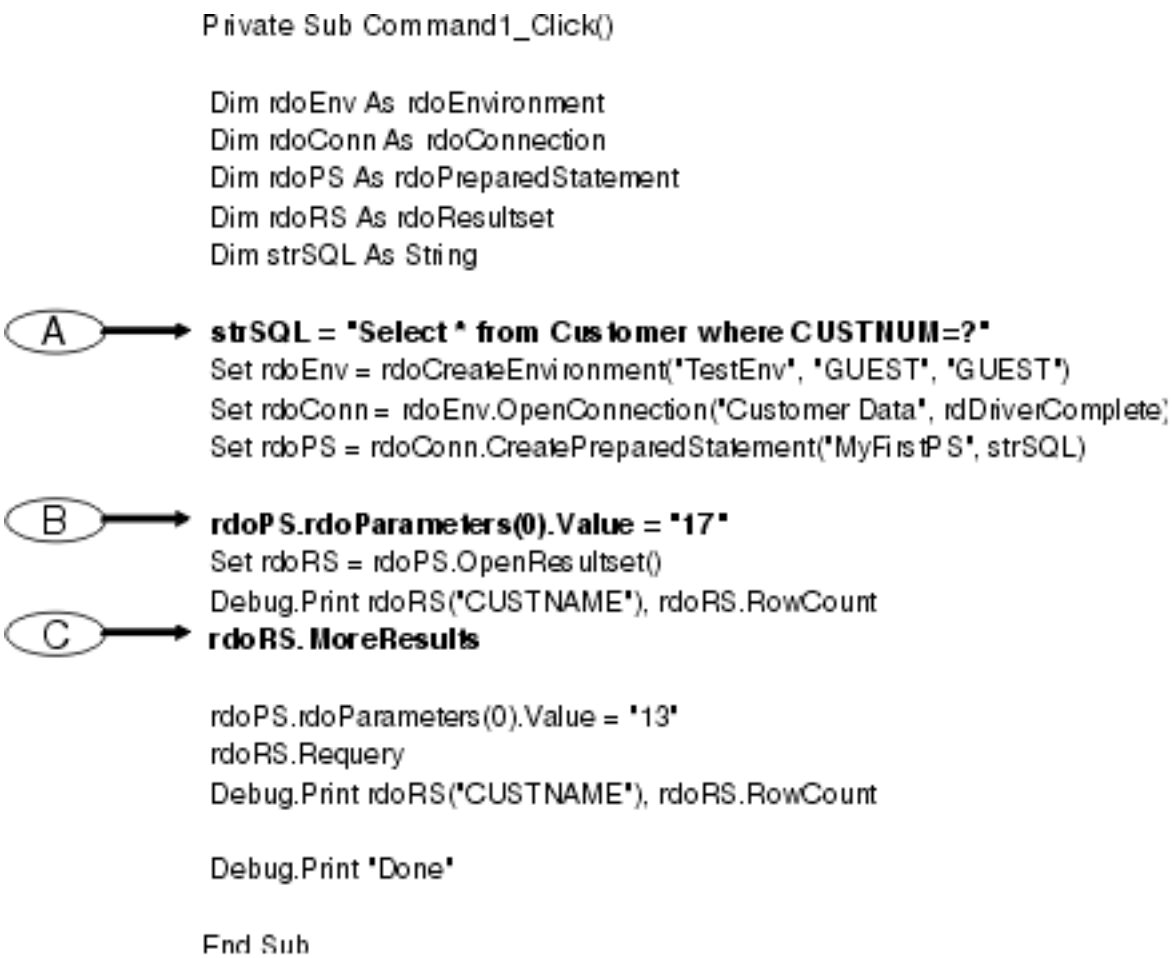


図 1. Visual Basic 4.0 RDO サンプル・コード

ラベル A は、SQL ステートメントが定義されている場所を示します。ステートメントは CUSTNUM に対する特定の値を含んではいませんが、値に対して疑問符 (?) があります。疑問符 (?) はこの値が準備済みステートメントのパラメーターであることを意味します。準備済みステートメントで結果セットを作成する前に、ステートメント内の任意のパラメーターの値を設定する必要があります。

ラベル B は、パラメーターに対する値が定義されている場所を示します。初期のパラメーターは 1 ではなく 0 に定義されています。パラメーターに対する値が設定されれば、 **rdoPreapredStatement** の **OpenResultSet** メソッドを実行して、要求されたデータを戻すことができます。

System i の準備済みステートメントを再照会するには、あらかじめカーソルの処理を終わらせて、クローズしておく必要があります。ラベル C はこの確認をするための **rdoResultSet** の **MoreResults** メソッドを示します。 **MoreResults** メソッドはデータベースを照会し、結果セット内に他に処理データがまだあるかどうか、または結果セットが完全に処理されているのかどうかを判別します。カーソルが完全に処理済みであれば、パラメーターの値を再設定して **rdoResultSet** の **ReQuery** メソッドを実行して、新規の結果セットをオープンできます。

検索結果:

System i Access for Windows 関数を使用するときには、結果セットのすべての行を処理するために、行が戻されなくなるまで **SQLFetch** API を呼び出します。

SQL ステートメントの中には、実行すると、アプリケーション・プログラムに対して結果が戻されるものがあります。例えば、SQL SELECT ステートメントを実行すると、選択されている行が結果セットに戻されます。次に、**SQLFetch** API によって結果セットから選択行が順次取り出され、アプリケーション・プログラムの内部記憶域に入れられます。

戻される列を指定しない SELECT ステートメントを発行することもできます。例えば、SELECT * FROM RWM.DBFIL とすると、すべての列が選択されます。場合によってはどの列が、またはどれだけの数の列が戻されるかがわかりません。

SQLNumResultCols

結果セット内の列数を戻します。

- 情報を受け取る記憶域バッファーがパラメーターとして渡されます。

```
SQLSMALLINT nResultCols;

rc = SQLNumResultCols(hstmt, &nResultCols);
```

SQLDescribeCol

結果セット内の 1 列に対して、結果の記述子を返します。

- **列名**
- **列タイプ**
- **列サイズ**

SQLDescribeCol は **SQLNumResultCols** と共に使用され、戻り列の情報を取り出します。この方法を使用すると、プログラム内での情報のハード・コーディングとは対照的に、柔軟性のあるプログラムを作成できます。

プログラマーはまず **SQLNumResultCols** を使って、SELECT ステートメントによってどれだけの列が結果セットに戻されたか確認します。次に、各列の情報を取り出せるように、**SQLDescribeCol** を使ったループを設定します。

C 言語では、このステートメントは以下のようにコーディングされます。

```
SQLCHAR szColName[51];
SQLSMALLINT lenColName, colSQLtype, scale, nullable;
SQLUSMALLINT colNum = 1;
SQLUINTEGER cbColDef;

rc = SQLDescribeCol(hstmt, colNum, szColName, sizeof(szColName),
                                    &lenColName, &colSQLtype,
&cbColDef, &scale, &nullable);
```

SQLBindCol

結果セット内の列に対して、記憶域とデータ・タイプを割り当てます。

- 情報を受け取る記憶域バッファー
- 記憶域バッファーの長さ
- データ・タイプの変換

C 言語では、このステートメントは以下のようにコーディングされます。

```
SQLUSMALLINT colNum = 1;
SQLUINTEGER cbColDef;
SQLINTEGER idNum, indPtr, strlen_or_indPtr;
SQLCHAR szIDName[51];

colNum = 1;
rc = SQLBindCol(hstmt, colNum, SQL_C_LONG, &idNum,
```

```
sizeof(SQLINTEGER), &indPtr);
colNum = 2;
rc = SQLBindCol(hstmt, colNum, SQL_C_CHAR, szIDName,
sizeof(szIDName), &strlen_or_indPtr);
```

注: これを Visual Basic で使用する場合は、ストリング・データ・タイプではなく、バイト・データ・タイプの配列を使用することをお勧めします。

SQLFetch

SQLFetch が呼び出されるときは必ず、ドライバーによって次の行が取り出されます。バインドされた列は、指定の位置に保管されます。バインドされていない列のデータは、**SQLGetData** を使って取り出すことができます。

C 言語では、このステートメントは以下のようにコーディングされます。

```
rc = SQLFetch(hstmt);
```

Visual Basic は、ポインターまたは固定メモリー位置の ANSI 文字ヌル終了ストリングを直接はサポートしていません。このため、文字パラメーターおよびバイナリー・パラメーターのバインドには別の方法を使用することが最良の方法です。 1 つの方法として、 Visual Basic ストリング・データ・タイプをバイト・データ・タイプの配列との間で相互に変換し、バイトの配列をバインドするという方法があります。別の方法としては、**SQLBindCol** 関数ではなく **SQLGetData** 関数を使う方法があります。

SQLGetData

取り出し後に、バインドされていない列のデータを検索します。この例では、3 列が戻され、**SQLGetData** を使用して、正しい保管場所にそれらを移動しています。

C 言語では、このステートメントは以下のようにコーディングされます。

```
SQLCHAR szTheName[16], szCredit[2];
float iDiscount, iTax;

rc = SQLFetch(hstmt);
rc = SQLGetData(hstmt, 1, SQL_C_CHAR, szTheName, 16, &strlen_or_indPtr);
rc = SQLGetData(hstmt, 2, SQL_C_FLOAT, &iDiscount, sizeof(float), &indPtr);
rc = SQLGetData(hstmt, 3, SQL_C_CHAR, szCredit, 2, &strlen_or_indPtr);
rc = SQLGetData(hstmt, 4, SQL_C_FLOAT, &iTax, sizeof(float), &indPtr);
```

Visual Basic では、このステートメントは以下のようにコーディングされます。

```
 rc = SQLFetch(hStmt)
If rc = SQL_NO_DATA_FOUND Then
    Call DisplayWarning("No record found!")
    rc = SQLCloseCursor(hStmt)
    If rc <> SQL_SUCCESS Then
        Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Close cursor failed.")
    End If
Else
    ' Reset lcbBuffer for the call to SQLGetData
    lcbBuffer = 0
    'Get part ID from the fetched record
    rc = SQLGetData(hStmt, 1, SQL_C_LONG, _
  lPartIDReceived, Len(lPartIDReceived), lcbBuffer)
    If rc <> SQL_SUCCESS And rc <> SQL_SUCCESS_WITH_INFO Then _
        Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, _
    "Problem getting data for PartID column")

    'Get part description from the fetched record
    rc = SQLGetData(hStmt, 2, SQL_C_CHAR, _
  szDescription(0), 257, lcbBuffer)
    If rc <> SQL_SUCCESS And rc <> SQL_SUCCESS_WITH_INFO Then _
        Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, _
```

```
    "Problem getting data for PartDescription column")

     'Get part provider from the fetched record
     rc = SQLGetData(hStmt, 3, SQL_C_CHAR, _
   szProvider(0), 257, lcbBuffer)
     If rc <> SQL_SUCCESS And rc <> SQL_SUCCESS_WITH_INFO Then _
         Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, _
     "Problem getting data for PartProvider column")

     Call DisplayMessage("Record found!")
     rc = SQLCloseCursor(hStmt)
     If rc <> SQL_SUCCESS Then _
         Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Close cursor failed.")
End If
```

ODBC アプリケーションでのデータベース・サーバーへのアクセス:

System i Access ODBC アプリケーションでは、データベースにアクセスするために基本的なステップに従う必要があります。

1. データ・ソースに接続します。
2. 処理する SQL ステートメント・ストリングをバッファーに入れます。これは、テキスト・ストリングです。
3. ステートメントが準備できるように、または即実行されるようにサブミットします。
   - 結果を受け取り、処理します。
   - エラーがある場合は、ドライバーからエラー情報を取り出します。
4. コミットまたはロールバックの操作で、各トランザクションを終了します (必要な場合)。
5. 接続を終了します。

ODBC 接続の確立:

以下のハンドル・タイプを使用して、System i Access の ODBC 接続を確立します。

ハンドル・タイプが SQL_HANDLE_ENV の SQLAllocHandle

- 環境ハンドルにメモリーを割り当てます。
  - グローバル情報の記憶域を識別します。
    - 有効な接続ハンドル
    - 現在®活動状態の接続ハンドル
    - 変数タイプ HENV
- ほかの ODBC 関数を呼び出す前に、アプリケーションから呼び出されている必要があります。
- 変数の型 HENV は、C プログラミング言語コンパイラーまたは SDK (ODBC ソフトウェア開発キット) で用意されている SQL.H ヘッダー・ファイルで、ODBC に定義されています。

  ヘッダー・ファイルには、far ポインターに対する型定義があります。

  ```
  typedef void far * HENV
  ```

- C 言語では、このステートメントは以下のようにコーディングされます。

  ```
  SQLRETURN rc;
  HENV henv;

  rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
  ```

- Visual Basic では、このステートメントは以下のようにコーディングされます。

```
Dim henv As long
SQLAllocEnv(henv)
```

ハンドル・タイプが SQL_HANDLE_DBC の SQLAllocHandle

- 環境の中で接続ハンドルに対してメモリーを割り当てます。
  - 特定の接続に関する情報の記憶域を識別します。
    - 変数型 HDBC
    - アプリケーションは、複数の接続ハンドルを持つことができます。
- アプリケーションは、データ・ソースに接続する前に接続ハンドルを要求する必要があります。
- C 言語では、このステートメントは以下のようにコーディングされます。

```
HDBC hdbc;

rc = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
```

- Visual Basic では、このステートメントは以下のようにコーディングされます。

```
Dim hdbc As long
SQLAllocConnect(henv,hdbc)
```

SQLSetEnvAttr

- アプリケーションにより、環境の属性設定が可能です。
- ODBC 3.x アプリケーションの場合は、接続ハンドルを割り振る前に、SQL_ATTR_ODBC_VERSION を SQL_OV_ODBC3 に設定しなければなりません。
- C 言語では、このステートメントは以下のようにコーディングされます。

```
rc = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
```

SQLConnect

- ドライバーをロードし、接続を確立します。
- 接続ハンドルは、接続情報を参照します。
- データ・ソースは、アプリケーション・プログラムにコーディングされます。

  C 言語では、このステートメントは以下のようにコーディングされます。

```
SQLCHAR source[ ] = "myDSN";
SQLCHAR uid[ ] = "myUID";
SQLCHAR pwd[ ] = "myPWD";

rc = SQLConnect(hdbc, source, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

  注: SQL_NTS は、パラメーター・ストリングがヌル終了ストリングであることを示しています。

SQLDriverConnect

- **SQLConnect** の代替手段
- アプリケーションにより、データ・ソースの設定をオーバーライドすることができます。
- ダイアログ・ボックスを表示します (任意選択)。

ODBC 関数の実行:

以下のハンドル・タイプを使用して、System i Access の ODBC 関数を実行します。

ハンドル・タイプが SQL_HANDLE_STMT の SQLAllocHandle

- SQL ステートメントに関する情報に対してメモリーを割り当てます。

– アプリケーションでは、SQL ステートメントをサブミットする前にステートメント・ハンドルを要求しておく必要があります。
– 変数の型 HSTMT

C 言語では、このステートメントは以下のようにコーディングされます。

```
HSTMT hstmt;

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
```

SQLExecDirect

- 準備可能なステートメントを実行します。
- 最も速く、1 回の実行で SQL ストリングをサブミットする方法です。
- rc が SQL_SUCCESS でない場合は、SQLGetDiagRec API を使用して、エラー条件の原因を突き止めることができます。

C 言語では、このステートメントは以下のようにコーディングされます。

```
SQLCHAR stmt[ ] = "CREATE TABLE NAMEID (ID INTEGER, NAME VARCHAR(50))";

rc = SQLExecDirect(hstmt, stmt, SQL_NTS);
```

- 戻りコード
  – SQL_SUCCESS
  – SQL_SUCCESS_WITH_INFO
  – SQL_ERROR
  – SQL_INVALID_HANDLE

SQLGetDiagRec

ステートメントに関するエラーのエラー情報を検索するには、以下のようにします。

C 言語では、このステートメントは以下のようにコーディングされます。

```
SQLSMALLINT i = 1, cbErrorMsg ;
SQLCHAR    szSQLState[6], szErrorMsg[SQL_MAX_MESSAGE_LENGTH];
SQLINTEGER nativeError;

rc = SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, i, szSQLState, &nativeError, szErrorMsg,
                                   SQL_MAX_MESSAGE_LENGTH, &cbErrorMsg);
```

- **szSQLState**
  – 5 文字のストリング
  – 00000 = 正常終了
  – 01004 = データ切り捨て
  – 07001 = パラメーターの数値が誤り

  注: 上記の項目は、可能性のある数多くの SQL 状態のうちのいくつかに過ぎません。
- **fNativeError** - データ・ソースに特定
- **szErrorMsg** - エラー・メッセージのテキスト

準備済みステートメントの実行:

SQL System i Access ODBC ステートメントが複数回使われている場合は、ステートメントを準備してから実行することをお勧めします。

ステートメントが準備済みであれば、変数情報をパラメーター・マーカーとして渡すことができます。パラメーター・マーカーは疑問符 (?) で示されます。ステートメントが実行されると、パラメーター・マーカーは実際の変数情報に置換されます。

ステートメントの準備は、サーバーで実行されます。SQL ステートメントがコンパイルされ、アクセス・プランが作成されます。これによって、ステートメントの実行効率が向上します。動的 SQL を使ったステートメントの実行と比較すると、静的 SQL に近い結果が得られます。 Extened Dynamic は、複数のジョブ・セッションにわたって準備済みステートメントを保持します。このため、パラメーター・マーカーを持つ準備済みステートメントは、 Extended Dynamic をオンにしなくても、ジョブ・セッション内で複数回実行することができます。データベース・サーバーでステートメントが準備されると、その一部がパッケージ (*SQLPKG) と呼ばれる特別の System i オブジェクトに保存されます。この方法を、**拡張動的 SQL** と呼びます。ドライバーによってパッケージが自動的に作成されますが、パッケージ・サポートをオフにするオプションも提供されています。詳しくは、ドライバーのパフォーマンス・アーキテクチャーに関する、次のトピックを参照してください。

SQLPrepare:

この関数は、System i Access の ODBC SQL ステートメントを実行できるように準備します。

C 言語では、このステートメントは以下のようにコーディングされます。

注: SQL_NTS は、ストリングが NULL で終わることを示しています。

```
SQLCHAR szSQLstr[ ] = "INSERT INTO NAMEID VALUES (?,?)";

rc = SQLPrepare(hstmt, szSQLstr, SQL_NTS);
```

SQLBindParameter:

この関数を使うと、System i Access の ODBC アプリケーションで、SQL ステートメントのパラメーター・マーカーに関連した記憶域、データ・タイプ、および長さを指定できるようになります。

例では、パラメーター 1 が **id** という符号の付いたダブルワード・フィールドにあります。パラメーター 2 は、符号なしの **name** という文字配列の中にあります。最後のパラメーターがヌルなので、ドライバーは、ストリングの長さを計算するときに、**name** がヌル終了であることを前提としています。

C 言語では、このステートメントは以下のようにコーディングされます。

```
SQLCHAR szName[51];
SQLINTEGER id, parmLength = 50, lenParm1 = sizeof(SQLINTEGER) , lenParm2 = SQL_NTS ;

rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER,
                                        sizeof(SQLINTEGER), 0, &id, sizeof(SQLINTEGER), &lenParm1);
rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
                                      parmLength, 0, szName, sizeof(szName), &lenParm2);
```

SQLExecute:

この関数は、パラメーター・マーカーの現行値を使って、準備済みのステートメントを実行します。

C 言語では、このステートメントは以下のようにコーディングされます。

```
id=500;
strcpy(szName, "TEST");
rc = SQLExecute(hstmt);  // Insert a record with id = 500, name = "TEST"
```

```
id=600;
strcpy(szName, "ABCD");
rc = SQLExecute(hstmt);  // Insert a record with id = 600, name = "ABCD"
```

SQLParamData および SQLPutData:

これらのステートメントは、System i Access の ODBC SQL ステートメントの実行時に、アンバインドされた入力パラメーター値を提供します。

Visual Basic では、ポインターや固定位置での ANSI 文字のヌル終了ストリングを直接サポートしていません。このため、文字パラメーターおよびバイナリー・パラメーターのバインドには別の方法を使用することが最良の方法です。例えば、Visual Basic のストリング・データ・タイプとバイト・データ・タイプの配列とを相互に変換して、バイトの配列をバインドするという方法があります。この方法については、以下のトピック『ストリングおよびバイトの配列の変換』で説明します。

別の方法としては、入力パラメーターに対してのみ使用できるものですが、処理時にパラメーターを提供する方法があります。これには、**SQLParamData** API および **SQLPutData** API を使います。

- これらの 2 つのステートメントは一緒に動作し、ステートメントの実行時に、バインドされていないパラメーター値を提供します。
- **SQLParamData** に対するそれぞれの呼び出しによって、**SQLPutData** がデータを提供する次のパラメーターに、内部ポインターを移動します。最後のパラメーターが指定されると、実行するステートメントに対して **SQLParamData** を再度呼び出す必要があります。
- **SQLPutData** がパラメーター・マーカーに対するデータを提供している場合は、そのパラメーターをバインドする必要があります。 **cbValue** パラメーターを使って、ステートメントの実行時に SQL_DATA_AT_EXEC の値を持つ変数を設定します。

```
's_parm is a character buffer to hold the parameters
    's_parm(1) contains the first parameter
    Static s_parm(2) As String
        s_parm(1) = "Rear Bumper"
        s_parm(2) = "ABC Auto Part Store"
    Dim rc As Integer
    Dim cbValue As Long
    Dim s_insert As String
    Dim hStmt As Long
    Dim lPartID As Long

    rc = SQLAllocHandle(SQL_HANDLE_STMT, ghDbc, hStmt)
    If rc <> SQL_SUCCESS Then _
  Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLAllocStmt failed.")

    s_insert = "INSERT INTO ODBCSAMPLE VALUES(?, ?, ?)"

    rc = SQLBindParameter(hStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, _
                          4, 0, lPartID, 4, ByVal 0)
    If rc <> SQL_SUCCESS Then _
  Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLBindParameter failed.")

 '#define SQL_LEN_DATA_AT_EXEC_OFFSET  (-100) the parms will be supplied at run time
    cbValue = -100

 ' Caller set 8th parameter to "ByVal 2" so driver will return
 ' 2 in the token when caller calls SQLParamData
    rc = SQLBindParameter(hStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                          4, 0, ByVal 2, 0, cbValue)
    If rc <> SQL_SUCCESS Then _
  Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLBindParameter failed.")

    ' Caller set 8th parameter to "ByVal 3" so driver will return
 ' 3 in the token when caller calls SQLParamData the second time.
```

```
   rc = SQLBindParameter(hStmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                         4, 0, ByVal 3, 0, cbValue)
   If rc <> SQL_SUCCESS Then _
 Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLBindParameter failed.")

' Prepare the insert statement once.
   rc = SQLPrepare(hStmt, s_insert, SQL_NTS)

   lPartID = 1
   rc = SQLExecute(hStmt) ' Execute multiple times if needed.

' Since parameters 2 and 3 are bound with cbValue set to -100,
' SQLExecute returns SQL_NEED_DATA

   If rc = SQL_NEED_DATA Then

 ' See comment at SQLBindParameter: token receives 2.
       rc = SQLParamData(hStmt, token)

       If rc <> SQL_NEED_DATA Or token <> 2 Then _
  Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLParamData failed.")

  ' Provide data for parameter 2.
       rc = SQLPutData(hStmt, ByVal s_parm(1), Len(s_parm(1)))
       If rc <> SQL_SUCCESS Then _
  Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLPutData failed.")

  ' See comment at SQLBindParameter: token receives 3.
       rc = SQLParamData(hStmt, token)
       If rc <> SQL_NEED_DATA Or token <> 3 Then _
  Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLParamData failed.")

  ' Provide data for parameter 2.
       rc = SQLPutData(hStmt, ByVal s_parm(2), Len(s_parm(2)))
       If rc <> SQL_SUCCESS Then _
  Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLPutData failed.")

  ' Call SQLParamData one more time.
  ' Since all data are provided, driver will execute the request.
       rc = SQLParamData(hStmt, token)
       If rc <> SQL_SUCCESS Then _
  Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLParamData failed.")
   Else
       Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "SQLExecute failed.")
   End If
```

ストリングおよびバイトの配列の変換:

以下の System i Access for Windows の Visual Basic 関数は、バイトのストリングと配列の変換の際に役立ちます。

```
Public Sub Byte2String(InByte() As Byte, OutString As String)
  'Convert array of byte to string
   OutString = StrConv(InByte(), vbUnicode)
End Sub

Public Function String2Byte(InString As String, OutByte() As Byte) As Boolean
    'vb byte-array / string coercion assumes Unicode string
    'so must convert String to Byte one character at a time
 'or by direct memory access

    Dim I As Integer
    Dim SizeOutByte As Integer
    Dim SizeInString As Integer

    SizeOutByte = UBound(OutByte)
```

```
    SizeInString = Len(InString)
    'Verify sizes if desired

    'Convert the string
    For I = 0 To SizeInString - 1
      OutByte(I) = AscB(Mid(InString, I + 1, 1))
    Next I
   'If size byte array > len of string pad with Nulls for szString
   If SizeOutByte > SizeInString Then           'Pad with Nulls
      For I = SizeInString To SizeOutByte - 1
         OutByte(I) = 0
      Next I
   End If

   String2Byte = True
End Function

Public Sub ViewByteArray(Data() As Byte, Title As String)
   'Display message box showing hex values of byte array

   Dim S As String
   Dim I As Integer
   On Error GoTo VBANext

   S = "Length: " & Str(UBound(Data)) & "  Data (in hex):"
   For I = 0 To UBound(Data) - 1
      If (I Mod 8) = 0 Then
         S = S & "  "         'add extra space every 8th byte
      End If
      S = S & Hex(Data(I)) & " "
   VBANext:
   Next I
   MsgBox S, , Title

End Sub
```

System i Access for Windows ODBC ドライバーのパフォーマンス・アーキテクチャー:

System i Access ODBC ドライバーでは、クライアントとサーバーの間でやりとりされる内部データのフローはすべて連鎖しており、必要な場合にのみ送信されます。

通信レイヤーの資源が割り当てられるのは 1 回のみであるため、サーバーの使用率は削減されます。これによって、応答時間は短縮されます。

ユーザーは、こうした機能の拡張を意識することはありません。ただし、System i Access の「ODBC セットアップ (ODBC Setup)」ダイアログに見られるような、いくつかの機能強化が行われています。詳細については、セットアップ GUI の「**パフォーマンス**」タブのオンライン・ヘルプ、または『接続ストリング・キーワード』の「パフォーマンス」オプションの説明を参照してください。これらのパフォーマンス・オプションのうちの一部については、以下のリンク先でさらに詳しく説明されています。

ODBC API 戻りコード:

System i Access の ODBC API 関数は、いずれも SQLRETURN 型の値 (短整数) を戻します。有り得る戻りコードとしては 7 種類のものが存在し、それぞれが 1 つの明示された定数と関連付けられています。

以下のリストには、それぞれの特定のコードの説明が載っています。戻りコードは、関数呼び出しでのエラーと解釈されるものもあれば、正常終了を示していると解釈されるものもあります。また、さらに情報が必要であるか、保留中であることを示している場合もあります。

特定の関数では、使用可能なコードのすべてを戻すとは限らない場合があります。特定の関数で指定できる値、ならびに、それらの正確な解釈については、「*Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference, Version 3.0 ISBN 1-57231-516-4*」を参照してください。

プログラム中の戻りコード、特に SQL ステートメントの処理およびデータ・ソースのデータ・アクセスに関連する戻りコードについては、細心の注意を払ってください。多くの場合、戻りコードは、関数が正常に実行されたかどうかを判別する唯一の信頼できる方法です。

SQL_SUCCESS
関数は正常終了しました。追加情報はありません。

SQL_SUCCESS_WITH_INFO
関数は、正常に終了しました。致命的ではないエラーがある可能性があります。アプリケーションは、SQLGetDiagRec を呼び出して、追加情報を検索することができます。

SQL_NO_DATA_FOUND
結果セットのすべての行は、取り出されました。

SQL_ERROR
関数の実行は失敗しました。アプリケーションは、SQLGetDiagRec を呼び出して、エラー情報を検索することができます。

SQL_INVALID_HANDLE
環境、接続、またはステートメントのハンドルに誤りがあり、関数の実行は失敗しました。プログラミング・エラーです。

SQL_NEED_DATA
ドライバーが、アプリケーションに、パラメーター・データ値を送信するよう要求しています。

ODBC 関数の終了:

System i Access の ODBC アプリケーション終了させる前に行う必要がある最後の手続きは、アプリケーションによって割り振られた資源とメモリーを解放することです。これは、次にアプリケーションが実行されるときに、資源とメモリーが使用可能になるように必ず実行する必要があります。

SQLFreeStmt
特定のステートメント・ハンドルに関連した処理を停止します。

```
rc = SQLFreeStmt(hstmt, option); // option can be SQL_CLOSE, SQL_RESET_PARAMS. or SQL_UNBIND
```

SQL_CLOSE
ステートメント・ハンドルに関連したカーソルをクローズし、保留中の結果をすべて破棄します。代わりに、SQLCloseCursor を使用できます。

SQL_RESET_PARAMS
SQLBindParameter によってバインドされている共通バッファーをすべて解放します。

SQL_UNBIND
SQLBindCol によってバインドされている共通バッファーをすべて解放します。

ハンドル・タイプが SQL_HANDLE_STMT の SQLFreeHandle
このステートメントのための資源をすべて解放します。

```
rc = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
```

SQLDisconnect
特定の接続ハンドルに関連した接続をクローズします。

```
rc = SQLDisconnect(hdbc);
```

ハンドル・タイプが SQL_HANDLE_DBC の SQLFreeHandle
接続ハンドルおよび接続ハンドルに関連したすべてのメモリーを解放します。

```
rc = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
```

ハンドル・タイプが SQL_HANDLE_ENV の SQLFreeHandle
環境ハンドルおよび環境ハンドルに関連したすべてのメモリーを解放します。

```
rc = SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

ODBC API のインプリメンテーションに関する事項

System i Access ODBC API を使用したインプリメンテーション時の問題について確認します。

ODBC API のインプリメンテーションに関する情報については、以下のトピックを選択してください。

注: Microsoft の ADO インターフェースで System i Access ODBC ドライバーを使用する際に発生する可能性のある問題について、説明および次善策を確認したい場合には、Software Knowledge Base で検索ストリングを MSDASQL 付きの ADO ストアード・プロシージャー呼び出しとして検索してください。

注: コード例を使用することで、 633 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』の条件に同意します。

関連資料

618 ページの『例: SQL ストアード・プロシージャーと ODBC による CL コマンドの実行』
ストアード・プロシージャー・サポートは、SQL の CALL ステートメントを使用して System i 制御言語 (CL) コマンドを実行する手段を提供します。

関連情報

Software Knowledge Base
Microsoft の ADO インターフェース で System i Access ODBC ドライバー・サポートを使用する際に発生する可能性のある幾つかの問題の説明および次善策については、MSDASQL 付きの ADO ストアード・プロシージャー呼び出しを検索ストリングとして使用して、Software Knowledge Base を検索します。

ODBC 3.x API に関する注意事項:

次の表は、System i Access ODBC 3.x API をその関連タスクごとにリストし、それぞれの API に関する考慮事項を示しています。

注:

- System i Access ODBC Driver は、ユニコード・ドライバーですが、ANSI アプリケーションも、引き続きそれと共に稼働します。ODBC Driver Manager は、System i Access ODBC Driver を呼び出す前に、ANSI ODBC API 呼び出しをワイド・バージョンに変換します。ユニコード・アプリケーションを作成するには、これらの API のいくつかのワイド・バージョンを呼び出さなければなりません。ワイド ODBC インターフェースに対応するアプリケーションを作成する場合、各 API の長さが文字として定義されているのか、バイト単位で定義されているのか、あるいは長さを適用できないのかどうかが分かっている必要があります。この情報については、次の表の「タイプ」列を参照してください。
- これらの API の動作について詳しくは、 Microsoft Web サイトで ODBC を検索してください。

タイプ	API	説明	その他の考慮事項
データ・ソースへの接続 **注:** 接続 API がサインオン・ダイアログのプロンプトを出す方法については、 568 ページの『サインオン・ダイアログの振る舞い』を参照してください。詳しくは、『接続プール』も参照してください。			
N/A	SQLAllocHandle	環境および接続ハンドルを入手します。 1 個の環境ハンドルが、複数の接続に使用されます。ステートメントや記述子ハンドルを割り振ることもできます。	
Char	SQLConnect	特定のユーザー ID およびパスワードを使用して特定のデータ・ソースに接続します。	ユーザー ID およびパスワードが指定されない場合に、この API がサインオン・ダイアログを行うためのプロンプトを出すかどうかを制御するオプションがあります。このオプションは、DSN の「**一般 (General)**」タブの「**接続オプション**」のダイアログから設定できます。
Char	SQLDriverConnect	接続ストリングで特定のドライバーへ接続するか、ユーザーのためのドライバー・マネージャーとドライバー接続のダイアログを表示するように要求します。	すべてのキーワードを使用。DSN のみ必須。その他の値は任意選択。詳しくは、 549 ページの『接続ストリング・キーワード』を参照してください。
Char	SQLBrowseConnect	連続レベルの接続属性と有効属性の値を戻します。接続属性ごとに値が指定されると、データ・ソースに接続します。	接続を試みるためには、SYSTEM キーワード、および DSN または DRIVER のいずれかのキーワードを指定しなければなりません。それ以外のキーワードは、すべてオプションです。セキュリティー上の理由により、PWD キーワードは出力ストリングでは戻されないようになっている点に注意してください。インプリメンテーションの問題については、 549 ページの『接続ストリング・キーワード』を参照してください。
ドライバーまたはデータ・ソースに関する情報の取得			

タイプ	API	説明	その他の考慮事項
Byte	SQLGetInfo	特定のドライバーとデータ・ソースに関する情報を戻します。	属性およびキーワードに基づいて別々に戻される特殊な属性です。SQLGetInfo によって戻される情報は、使用されているキーワードおよび属性に応じて異なります。影響を受ける InfoType オプションは以下のとおりです。 • SQL_CATALOG_NAME_SEPARATOR - デフォルトではピリオドが戻されます。接続ストリング・キーワード NAM を 1 に設定すると、コンマが戻されます。 • SQL_CURSOR_COMMIT_BEHAVIOR、SQL_CURSOR_ROLLBACK_BEHAVIOR - デフォルトでは SQL_CB_PRESERVE が戻されます。接続属性 1204 が設定されている場合、SQL_CB_DELETE が戻されます。 • SQL_DATA_SOURCE_READ_ONLY – デフォルトでは N が戻されます。接続ストリング・キーワード CONNTYPE を 0 に設定すると、Y が戻されます。 • SQL_IDENTIFIER_QUOTE_CHAR - デフォルトでは二重引用符が戻されます。使用されているアプリケーションが MS QUERY (MSQRY32) である場合には、単一のブランクが戻されます。 • SQL_IDENTIFIER_CASE – デフォルトでは SQL_IC_UPPER が戻されます。接続ストリング・キーワード DEBUG でオプション 2 が設定されている場合には、SQL_IC_MIXED が戻されます。 • SQL_MAX_QUALIFIER_NAME_LEN – デフォルトでは 18 が戻されます。接続ストリング・キーワード DEBUG で 8 ビットが設定されている場合には、0 が戻されます。 • SQLDriverVer - ドライバーのバージョンを、VV.RR.SSST の書式で戻します。 – VV は、System i Access for Windows 製品のバージョンを表します。 – RR は、System i Access for Windows 製品のリリース ID を表します。 – SSS は、System i Access for Windows 製品に適用済みの Service Pack の番号です。 – T は、ODBC ドライバーの問題に適用済みのテスト修正のバージョンです。ない場合は 0 になります。

タイプ	API	説明	その他の考慮事項
N/A	SQLGetTypeInfo	サポートするデータ・タイプに関する情報を戻します。	異なる System i バージョンに対して実行すると、異なる結果セットが得られます。例えば、BIGINT データ・タイプは、V4R5 またはそれ以降のサーバーに対して実行したときの結果セットにのみ戻されます。 ″LONG VARCHAR″ データ・タイプは、結果セットには戻されません。この問題は、このタイプの長さを指定することを前提としているアプリケーションが原因となっています。″LONG VARCHAR FOR BIT DATA″ および ″LONG VARGRAPHIC″ も同様の理由のために戻されません。 TYPE_NAME 列で、データ・タイプに括弧で囲んだ値が必要なときは、データ・タイプ名に括弧が含まれます。ただし、括弧がデータ・タイプ・ストリングの最後に来るときは、その括弧は省略されます。次のストリングの例で、″CHAR″ データ・タイプの後には括弧が続いていますが、″DATA″ データ・タイプの後には括弧が続いていません。例: ″CHAR() FOR BIT DATA″。 接続ストリング・キーワード GRAPHIC の設定値は、ドライバーがグラフィック (DBCS) データ・タイプをサポートされるタイプとして戻すかどうかに影響を与えます。詳しくは、 569 ページの『ODBC データ・タイプおよびそれらと DB2 for i5/OS データベース・タイプとの対応』を参照してください。
ドライバー属性の設定および検索 **注:** 次の API に適用可能なドライバー固有の接続およびステートメントの属性について詳しくは、 573 ページの『接続とステートメントの属性』を参照してください。			
Byte	SQLSetConnectAttr	接続オプションを設定します。	
Byte	SQLGetConnectAttr	接続オプションの値を戻します。	
N/A	SQLSetEnvAttr	環境オプションを設定します。	
N/A	SQLGetEnvAttr	環境オプションの値を戻します。	

タイプ	API	説明	その他の考慮事項
Byte	SQLSetStmtAttr	ステートメント・オプションを設定します。	SQL_ATTR_PARAMSET_SIZE、SQL_ATTR_ROW_ARRAY_SIZE、SQL_DESC_ARRAY_SIZE、および SQL_ROWSET_SIZE 属性は、32767 行までサポートします。 LOB ロケーター・フィールドを利用している場合、ドライバーはこれらの値を一度に 1 行に制限します。 MAXFIELDLEN 接続ストリングの値が LOB フィールドのサイズよりも小さい場合、LOB フィールドはロケーターとして扱われます。 FOR FETCH ONLY または FOR UPDATE 文節を含む SELECT ステートメントは、SQL_ATTR_CONCURRENCY 属性の現在の設定値をオーバーライドします。SQL_ATTR_CONCURRENCY の設定値が SQL ステートメント内の文節と競合している場合、SQLExecute または SQLExecDirect の実行中にはエラーが戻されません。 以下はサポートされていません。 • SQL_ATTR_ASYNC_ENABLE • SQL_ATTR_RETRIEVE_DATA • SQL_ATTR_SIMULATE_CURSOR • SQL_ATTR_USE_BOOKMARKS • SQL_ATTR_FETCH_BOOKMARK_PTR • SQL_ATTR_KEYSET_SIZE SQL_ATTR_MAX_ROWS の設定はサポートされますが、静的カーソルのパフォーマンスにしか影響しません。結果セット全体は、このオプションが設定されていても、他のカーソル・タイプを使用して作成されます。 SQL 照会で FETCH FIRST x ROWS ONLY 文節を使用すると、サーバーの作業量が削減されるため、パフォーマンスが向上することがあります。この API は、次の 2 つの結果セット・タイプのカーソル行数を含むようにも拡張されています。 • ストアード・プロシージャーの結果セット • 静的カーソルの結果セット
Byte	SQLGetStmtAttr	ステートメント・オプションの値を戻します。	以下はサポートされていません。 • SQL_ATTR_ASYNC_ENABLE • SQL_ATTR_RETRIEVE_DATA • SQL_ATTR_SIMULATE_CURSOR • SQL_ATTR_USE_BOOKMARKS • SQL_ATTR_FETCH_BOOKMARK_PTR
記述子フィールドの設定および検索			
Byte	SQLGetDescField	記述子から情報を戻します。	

タイプ	API	説明	その他の考慮事項
Char	SQLGetDescRec	記述子から複数の情報を戻します。	
Byte	SQLSetDescField	記述子フィールドを設定します。	SQL_DESC_ARRAY_STATUS_PTR および SQL_DESC_ROWS_PROCESSED_PTR 以外の IRD には、記述子フィールドは設定できません。 名前付きパラメーターはサポートしていません。
Char	SQLSetDescRec	記述子の複数のオプションを設定します。	
N/A	SQLCopyDesc	ある記述子から別の記述子に情報をコピーします。	SQLCopyDesc は名前付きパラメーターをサポートしません。
SQL 要求の作成			
Char	SQLPrepare	後の実行のために、SQL ステートメントを作成します。	パッケージは、その接続用の SQL ステートメントが初めて準備されるときに作成されます。このために、最初の準備には通常よりも完了するまで少し長く時間がかかります。既存パッケージに何か問題がある場合、DSN セットアップ GUI で指定されたパッケージの設定によっては最初の準備でエラーが戻される場合があります。 DSN セットアップ GUI の「パッケージ」タブに、デフォルトのパッケージ設定があります。これらの設定は、パッケージ設定が当該アプリケーションに合わせてまだカスタマイズされていない場合に使用されます。これらは、グローバルな設定ではない点に注意してください。 デフォルトでは、ドライバーは SQL ステートメント・テキストを、ユーザーのジョブに関連付けられている EBCDIC CCSID でホストに送信します。ドライバーが SQL ステートメント・テキストを Unicode でホストに送信できるように、UNICODESQL キーワードを 1 または 2 に設定します。 Unicode の SQL ステートメントを送信する場合は、EBCDIC SQL ステートメントを含む既存のパッケージとの衝突を避けるため、ドライバーは別のパッケージ名を生成することに注意してください。接続ストリング・キーワード UNICODESQL を設定すると、アプリケーションでは、SQL ステートメント内のリテラルとしてユニコード・データを指定できるようになります。 準備および実行することが推奨されていない幾つかの SQL ステートメントについては、『SQL ステートメントの考慮事項』を参照してください。 ドライバーがサポートするエスケープ・シーケンスおよびスカラー関数については、575 ページの『SQLPrepare および SQLNativeSQL エスケープ・シーケンスおよびスカラー関数』を参照してください。

タイプ	API	説明	その他の考慮事項
Byte	SQLBindParameter	SQL ステートメントの中のパラメーター用に記憶域を割り当てます。詳細については、 518 ページの『パラメーター・マーカー』を参照してください。	実際のホスト・パラメーター (列) データ・タイプに指定されている C タイプから直接、データ変換されます。 指定されている SQL データ・タイプおよび列サイズは無視されます。 文字データを含む変換では、クライアント・コード・ページから列 CCSID に直接変換が行われます。 V6R1 ホストの場合: • SQL ステートメント の Insert または Update へのバインディングで有効なのは、SQL_DEFAULT_PARAM および SQL_UNASSIGNED のみです。 • Strlen_or_IndPtr = SQL_DEFAULT_PARAM である場合、ドライバーはその列のデフォルト値を使用します。その列にデフォルト値が指定されておらず、ステートメントが CALL 以外の場合、ドライバーは SQL ステートメントの実行中にエラーを戻します。 • SQL_UNASSIGNED または Strlen_or_IndPtr = -7 である場合、パラメーターは SQL ステートメントから省略されます。 V6R1 より前のホストの場合: • デフォルト・パラメーターと未割り当てのパラメーターはサポートされません。 • Strlen_or_IndPtr パラメーターに SQL_DEFAULT_PARAM または SQL_UNASSIGNED が指定されると、ドライバーは SQL ステートメントの実行中にエラーを戻します。 • SQL_DEFAULT_PARAM または SQL_UNASSIGNED を使用してステートメントをバインディングすると、ドライバーは CALL ステートメントの実行中に SQLSTATE エラー (07S01) を戻します。
Char	SQLGetCursorName	ステートメント・ハンドルに関連したカーソル名を戻します。	ドライバーは、二重引用符で囲まれていないすべてのカーソル名を大文字にします。

タイプ	API	説明	その他の考慮事項
Char	SQLSetCursorName	カーソル名を指定します。	カーソル名は、引用符で囲まれていない場合には、大文字に変換されます。引用符で囲まれたカーソル名は変換されません。例えば、**myCursorName** は **MYCURSORNAME** となりますが、″**myCursorName**″ は長さ 14 の **myCursorName** として扱われます (引用符も長さに含まれるためです)。 ドライバーは、″″、**a から z、A から Z、0 から 9、または** _ の文字のみをカーソル名の中でサポートします。無効な名前が入力されても SQLSetCursorName はエラーを戻しませんが、後で無効な名前を使用しようとした際に、エラーが戻されます。 カーソル名は、最大 128 文字 (前後に二重引用符がある場合はそれらも含めて) までとし、UNICODE から ANSI に変換可能な文字で指定する必要があります。 アプリケーションで ODBC により DRDA® 接続を使用したい場合は、以下の制約事項があります。 • DRDA 接続時にカーソル名は変更できません。 • カーソル名は、ドライバーによって変更され、カーソルがオープンした後、SQLGetCursorName により検査する必要があります (SQLExecute または SQLExecDirect の後)。
要求の投入			
N/A	SQLExecute	作成済みステートメントを実行します。	SQLExecute は、PREFETCH、CONNTYPE、CMT、および LAZYCLOSE などのいくつかの接続ストリング・キーワードの設定値の影響を受けます。これらのキーワードの説明については、549 ページの『接続ストリング・キーワード』を参照してください。
Char	SQLExecDirect	ステートメントを実行します。	SQLPrepare および SQLExecute を参照してください。
Char	SQLNativeSQL	ドライバーにより変換された SQL ステートメントのテキストを戻します。	
Char	SQLDescribeParam	ステートメントの中の特定のパラメーターについての記述を戻します。	
N/A	SQLNumParams	ステートメントの中のパラメーターの数を戻します。	

タイプ	API	説明	その他の考慮事項
N/A	SQLParamData	実行時にデータが送られるパラメーターに割り当てられた記憶域の値を戻します (長いデータ値の場合に有用)。	
Byte	SQLPutData	パラメーター値の一部または全部を送信します (長いデータ値の場合に有用)。	
結果および関連情報の検索			
N/A	SQLRowCount	挿入、更新、または削除要求の影響を受けた行数を戻します。	この API は、V5R1 またはそれ以降のバージョンのサーバーに対して静的カーソルを使用して、結果セットのカーソル行カウントも含めるように拡張されました。
N/A	SQLNumResultCols	結果セットの中の列の番号を戻します。	
Char	SQLDescribeCol	結果セットの中の列を記述します。	
Byte	SQLColAttribute	結果セットの中の列の属性を記述します。	
Byte	SQLBindCol	結果列用に記憶域を割り当て、データ・タイプを指定します。	
N/A	SQLExtendedFetch	結果セットの中の行を戻します。これは、2.x ODBC API としてサポートされています。しかし、新規アプリケーションは、これではなく SQLFetchScroll API を使用する必要があります。	行セット・サイズに、SQL_ATTR_ROW_ARRAY_SIZE ではなく、ステートメント属性 SQL_ROWSET_SIZE の値を使用します。 SQLExtendedFetch は、行サイズが 1 の場合に、SQLSetPos および SQLGetData との組み合わせでのみ使用できます。 SQL_FETCH_BOOKMARK はサポートされていません。 カタログ API の結果セット (SQLTables や SQLColumns など) は順方向専用および読み取り専用です。カタログ API によって生成された結果セットと一緒に SQLExtendedFetch を使用した場合、スクロールは行えません。
N/A	SQLFetch	結果セットの中の行を戻します。	カーソルは、前方にしか移動しないため、SQL_FETCH_FIRST および SQL_FETCH_NEXT でしか使用できません。
N/A	SQLFetchScroll	結果セットの中の行を戻します。スクロール可能カーソルで使用できます。	ドライバーがブックマークをサポートしていないため、SQL_FETCH_BOOKMARK の取り出し方向はサポートされません。

タイプ	API	説明	その他の考慮事項
Byte	SQLGetData	結果セットの 1 行、1 列の一部または全部を戻します (長いデータ値の場合に有用)。詳細については、 518 ページの『SQLFetch および SQLGetData』を参照してください。	SQLGetData は、単一行取り出しでのみ使用することができます。行配列サイズが 1 よりも大きい場合、SQLGetData によってエラーが報告されます。
N/A	SQLSetPos	取り出したデータのブロックの中にカーソルを置きます。	SQL_UPDATE、SQL_DELETE、および SQL_ADD は、Operations パラメーターのオプションとしてはサポートされません。 SQL_LOCK_EXCLUSIVE および SQL_LOCK_UNLOCK は、LockType パラメーターのオプションとしてはサポートされません。
N/A	SQLBulkOperations	更新、削除、およびブックマークによる取り出しなど、大量の挿入および大量のブックマーク操作を実行します。	このドライバーは SQLBulkOperations をサポートしていません。
N/A	SQLMoreResults	結果セットがさらにあるかどうかを判別します。まだある場合は、次の結果セットが処理できるように初期設定を行います。	
Byte	SQLGetDiagField	診断情報を戻します。	SQL_DIAG_CURSOR_ROW_COUNT オプションが正確となるのは、V5R1 以降のバージョンのサーバーで静的カーソルを使用する場合のみです。
Char	SQLGetDiagRec	追加のエラーまたは状況情報を戻します。	
データ・ソース・システム・テーブル情報の取得			
Char	SQLColumnPrivileges	1 つまたは複数のテーブルについての列のリストと関連する特権を戻します。	以下の場合に空の結果セットを戻します。 • V5R1 またはそれ以前のサーバー • V5R2 サーバーで、CATALOGOPTIONS 接続ストリング・キーワードのオプション 2 が設定されていない。 デフォルトでは、V5R2 サーバーにアクセスする際に、列の特権情報が戻されます。
Char	SQLColumns	1 つまたは複数のテーブルの列に関する情報のリストを戻します。	
Char	SQLForeignKeys	外部キーが、指定されたテーブル用に存在する場合は、その外部キーを含む列名のリストを戻します。	
Char	SQLProcedureColumns	指定のプロシージャーについての結果セットを構成する列とともに、入出力パラメーターのリストを戻します。	ドライバーは、プロシージャーによって生成された、結果セットを構成する列に関する情報を戻しません。ドライバーは、プロシージャーに指定されたパラメーターに関する情報のみを戻します。

タイプ	API	説明	その他の考慮事項
Char	SQLProcedures	特定のデータ・ソースに保管されたプロシージャー名のリストを戻します。	
Char	SQLSpecialColumns	指定されたテーブルにある 1 つの行を固有に識別する、最適な列のセットに関する情報を検索します。また、トランザクションでその行の任意の値が更新されるたびに自動的に更新される列についての情報も検索します。	SQL_BEST_ROWID オプションを指定して呼ばれた場合は、そのテーブルのすべての索引付きの列を戻します。
Char	SQLStatistics	単一テーブルと、そのテーブルと関連した索引のリストに関する統計を検索します。	V6R1 から、派生キーの索引を定義できるようになりました。索引に関する情報の検索に SQLStatistics を使用すると、COLUMN_NAME 結果セット列は派生キーの索引を表す式を戻します。 索引の作成時に WHERE 文節を使用した場合、FILTER_CONDITION 結果セット列に Where 式が戻されます。
Char	SQLTables	データ・ソースのスキーマ、テーブル、またはテーブル・タイプのリストを戻します。	582 ページの『SQLTables の説明』を参照してください。
Char	SQLTablePrivileges	テーブルのリストおよびそれぞれのテーブルと関連する特権を戻します。	以下の場合に空の結果セットを戻します。 • V5R1 またはそれ以前のサーバー • V5R2 サーバーで、CATALOGOPTIONS 接続ストリング・キーワードのオプション 2 が設定されていない。 デフォルトでは、V5R2 サーバーにアクセスする際に、テーブルの特権情報が戻されます。
Char	SQLPrimaryKeys	テーブル用の基本キーを構成する列の名前のリストを戻します。	
ステートメントの終結処理			
N/A	SQLFreeStmt	ステートメント処理を終了し、関連するカーソルをクローズし、保留結果を廃棄します。	
N/A	SQLCloseCursor	ステートメント・ハンドルでオープンされているカーソルをクローズします。	
N/A	SQLCancel	SQL ステートメントを取り消します。	すべての照会を取り消せるわけではありません。これは、実行に時間のかかっている照会の場合にのみお勧めします。詳細については、583 ページの『長時間実行照会の処理』 を参照してください。

タイプ	API	説明	その他の考慮事項
N/A	SQLEndTran	トランザクションをコミット、またはロールバックします。	コミットメント制御については、『コミットメント制御の考慮事項』を参照してください。
接続の終了			
N/A	SQLDisconnect	接続をクローズします。	
N/A	SQLFreeHandle	ハンドルに関連した資源をリリースします。	

関連資料

567 ページの『ODBC API の制約事項およびサポートされない関数』
System i Access の ODBC ドライバーにおける関数のインプリメント方法の中には、「Microsoft ODBC Software Development Kit Programmer's Reference」に記述されている仕様と一致しないものがあります。

関連情報

Microsoft Web サイト

SQL ステートメントの考慮事項:

System i Access 関数で ODBC を使用する際に避けるべき SQL ステートメントを確認します。

準備および実行することが推奨されていない SQL ステートメントが幾つかあります。以下はその例です。

- SET TRANSACTION
- SET SCHEMA
- SET PATH
- COMMIT
- ROLLBACK
- CONNECT TO
- DISCONNECT ALL

これらのステートメントでは、同じ振る舞いを、ODBC を介して別の方法で実現できます。例えば、ODBC 接続の自動コミットをオフにする場合は、COMMIT ステートメントまたは ROLLBACK ステートメントを実行する代わりに、 SQLEndTran オプションを使用することができます。

SET SESSION AUTHORIZATION SQL ステートメントは、ODBC 接続プールと組み合わせて使用した場合に、予測不能の振る舞いを生じる接続の制御下にあるユーザーを変更します。 ODBC を介して SET SESSION AUTHORIZATION ステートメントを使用する場合には、実行される SET SESSION AUTHORIZATION 用以外のすべてのオープン・ステートメント・ハンドルを解放することをお勧めします。 SET SESSION AUTHORIZATION の実行が終了したら、そのステートメント・ハンドルを解放してください。

接続ストリング・キーワード:

System i Access の ODBC ドライバー・サポートでは、ODBC 接続の振る舞いを変更するために、複数の接続ストリング・キーワードを使用します。

ODBC データ・ソースがセットアップされる際に、同じキーワードおよびそれらの値が保管されます。ODBC アプリケーションが接続を行う際には、接続ストリングで指定されたキーワードにより、ODBC データ・ソースで指定した値がオーバーライドされます。

System i Access の ODBC ドライバー・サポートが認識する接続ストリング・キーワードについて詳しくは、以下の表にある該当項目を参照してください。

接続ストリング・キーワード - 一般プロパティー:

これらの System i Access ODBC ドライバー接続ストリング・キーワードは、ODBC 接続の一般プロパティーの変更に使用します。

以下の表は、System i Access ODBC ドライバーによって認識される、一般プロパティー用の接続ストリング・キーワードをリストしたものです。

表 3. 一般プロパティー用の System i Access ODBC 接続ストリング・キーワード

キーワード	説明	選択項目	デフォルト
DSN	接続に使用する ODBC データ・ソースの名前を指定します。	データ・ソース (DSN) 名	なし
DRIVER	使用する ODBC ドライバーの名前を指定します。 **注:** DSN プロパティーが指定されている場合には、これは使用しないでください。	″System i Access ODBC Driver″ クライアント・アクセス ODBC ドライバー (32 ビット) **注:** System i Access for Windows V5R2 以降のインストール時に、2 つの ODBC ドライバーが登録されます。**クライアント・アクセス ODBC ドライバー (32 ビット)** と **System i Access ODBC ドライバー**の両方の名前が登録されますが、どちらの登録名も同じ ODBC ドライバーを示します。これら 2 つの登録名は、2 つの異なるインストール済み ODBC ドライバーを示すわけではありません。**クライアント・アクセス ODBC ドライバー (32 ビット)** の古い方の名前は、後方互換性をサポートするために登録されます。	なし
PWD または Password	System i 接続のパスワードを指定します。	System i パスワード	なし
SIGNON	現行のユーザー ID およびパスワード情報で接続を行うことができない場合にどのデフォルトのユーザー ID を使用するのかを指定します。	0 = Windows ユーザー名を使用 1 = デフォルトのユーザー ID を使用 2 = なし 3 = System i ナビゲーターのデフォルトを使用 4 = Kerberos プリンシパルを使用	3

表 3. 一般プロパティー用の System i Access ODBC 接続ストリング・キーワード (続き)

キーワード	説明	選択項目	デフォルト
SSL	サーバーと通信するために Secure Sockets Layer (SSL) 接続を使用するかどうかを指定します。 SSL 接続は、V4R4 またはそれ以降のサーバーに接続する場合にのみ使用可能です。	0 = パスワードのみを暗号化する 1 = すべてのクライアント/サーバー通信を暗号化する	0
SYSTEM	接続する System i の名前を指定します。	System i 名。『ODBC 接続 API のための System i 名の形式』を参照。	なし
UID または UserID	System i 接続のユーザー ID を指定します。	System i ユーザー ID	なし

接続ストリング・キーワード - サーバー・プロパティー:

これらの System i Access ODBC ドライバー接続ストリング・キーワードは、ODBC 接続のサーバー・プロパティーの変更に使用します。

以下の表は、System i Access ODBC ドライバーによって認識される、サーバー・プロパティー用の接続ストリング・キーワードをリストしたものです。

表 4. サーバー・プロパティー用の System i Access ODBC 接続ストリング・キーワード

キーワード	説明	選択項目	デフォルト
CMT または CommitMode	デフォルトのトランザクション分離レベルを指定します。	0 = 即時コミット (*NONE) 1 = コミット読み取り (*CS) 2 = 非コミット読み取り (*CHG) 3 = 反復可能読み取り (*ALL) 4 = シリアライズ可能 (*RR)	2
CONNTYPE または ConnectionType	接続におけるデータベース・アクセスのレベルを指定します。	0 = 読み取り / 書き込み (すべての SQL ステートメントを使用可能) 1 = 読み取り / 呼び出し (SELECT および CALL ステートメントを使用可能) 2 = 読み取り専用 (SELECT ステートメントのみ)	0

表 4. サーバー・プロパティー用の System i Access ODBC 接続ストリング・キーワード (続き)

キーワード	説明	選択項目	デフォルト
DATABASE	接続する System i リレーショナル・データベース (RDB) 名を指定します。このオプションは System i V5R2 バージョンに対してのみ有効である点に注意してください。このオプションは、V5R2 よりも前のサーバーに接続する場合には無視されます。 このオプションの特殊値には、空ストリングまたは *SYSBAS の指定が含まれます。空ストリングは、データベースに関してユーザー・プロファイルのデフォルトの設定値を使用することを表します。 *SYSBAS を指定すると、ユーザーは SYSBAS データベース (RDB 名) に接続されます。	System i リレーショナル・データベース名	空ストリング
DBQ または DefaultLibraries	サーバー・ジョブのライブラリー・リストに追加する System i ライブラリーを指定します。ライブラリーはコンマまたはスペースで区切って指定します。サーバー・ジョブの現行ライブラリー・リストでは、プレースホルダーとして *USRLIBL を使用することができます。このライブラリー・リストは、未修飾のストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しからのライブラリーの検出に使用されます。 *USRLIBL を指定しない場合、サーバー・ジョブの現行ライブラリー・リストは、指定されたライブラリーによって置き換えられます。 **注:** このプロパティーでリストされた最初のライブラリーは、デフォルトのライブラリーでもあり、SQL ステートメント内の未修飾名を解決するために使用されます。デフォルト以外のライブラリーを指定するには、ライブラリーの前にコンマを入力する必要があります。	System i ライブラリー V5R1 よりも前のサーバーへの接続については、ライブラリー・リストで 25 個のライブラリーしかサポートされません。 V5R1 およびそれ以降のサーバーでは、75 項目がサポートされます。 75 を超えた項目は無視されます。	QGPL
MAXDECPREC または Maximum Decimal Precision	戻す 10 進数データの最大精度を指定します。	31 または 63	31
MAXDECSCALE または Maximum Decimal Scale	10 進データを必要とする演算で使用する最大スケールを指定します。この値は MAXDECPREC の値より小さくなければなりません。	0 から 63	31

表 4. サーバー・プロパティー用の System i Access ODBC 接続ストリング・キーワード (続き)

キーワード	説明	選択項目	デフォルト
MINDIVSCALE または Minimum Divide Scale	10 進データを必要とする演算で使用する最小スケールを指定します。	0 から 9	0
NAM または Naming	テーブルの参照時に使用する命名規則を指定します。	0 = ″sql″ (*schema.table* など) 1 = ″system″ (*schema/table* など)	0

接続ストリング・キーワード - データ・タイプ:

これらの System i Access ODBC ドライバー接続ストリング・キーワードは、ODBC 接続のデータ・タイプ・プロパティーの変更に使用します。

以下の表は、System i Access ODBC ドライバーで認識される、データ・タイプ・プロパティー用の接続ストリング・キーワードをリストしたものです。

表 5. System i Access ODBC 接続ストリング・キーワード (データ・タイプ・プロパティー用)

キーワード	説明	選択項目	デフォルト
DFT または DateFormat	SQL ステートメント内の日付リテラルで使用される日付形式を指定します。	0 = yy/dd (*JUL) 1 = mm/dd/yy (*MDY) 2 = dd/mm/yy (*DMY) 3 = yy/mm/dd (*YMD) 4 = mm/dd/yyyy (*USA) 5 = yyyy-mm-dd (*ISO) 6 = dd.mm.yyyy (*EUR) 7 = yyyy-mm-dd (*JIS)	5
DSP または DateSeparator	SQL ステートメント内の日付リテラルで使用される日付区切り文字を指定します。このプロパティーは、DateFormat プロパティーが 0 (*JUL)、1 (*MDY)、2 (*DMY)、または 3 (*YMD) に設定されていない場合には効果がありません。	0 = ″/″ (スラッシュ) 1 = ″-″ (ダッシュ) 2 = ″.″ (ピリオド) 3 = ″,″ (コンマ) 4 = ″ ″ (ブランク)	1
DEC または Decimal	SQL ステートメント内の数値リテラルで使用される小数点を指定します。	0 = ″.″ (ピリオド) 1 = ″,″ (コンマ)	0
DECFLOATERROROPTION	10 進浮動小数点データ・タイプのエラーを検出した場合に、警告として報告するか、またはデータ・マッピング・エラーとして報告するかを指定します。これを指定しない場合、サーバー属性値は変更されません。	0 = 10 進浮動小数点エラーを、データ・マッピング・エラーとして報告します。 1 = 10 進浮動小数点エラーを、警告として報告します。	0

表 5. System i Access ODBC 接続ストリング・キーワード (データ・タイプ・プロパティー用) (続き)

キーワード	説明	選択項目	デフォルト
DECFLOATROUNDMODE	結果に対する丸めが許可されている場合、丸めモードを指定します。	0 = ROUND_HALF_EVEN - 最も近い数字に丸めます。 2 つの数字の中間にある場合は、最も近い偶数に丸めます。これが、デフォルトの丸めモードになります。 1 = ROUND_HALF_UP - 最も近い数字に丸めます。 2 つの数字の中間にある場合は、切り上げます。 2 = ROUND_DOWN - 最も近い数字のうち、小さい方の数字に丸めます。これは、切り捨てと同じです。 3 = ROUND_CEILING - 正の無限大に近づくように丸めます。 4 = ROUND_FLOOR - 負の無限大に近づくように丸めます。 5 = ROUND_HALF_DOWN - 最も近い数字に丸めます。 2 つの数字の中間にある場合は、切り捨てます。 6 = ROUND_UP - 最も近い数字のうち、大きい方の数字に丸めます。	0
MAPDECIMALFLOATDESCRIBE	DECFLOAT 演算の結果のフォーマットを指定します。	1 = SQL_ VARCHAR 3 = SQL_ DOUBLE	1
TFT または TimeFormat	SQL ステートメント内の時刻リテラルで使用される時刻形式を指定します。	0 = hh:mm:ss (*HMS) 1 = hh:mm AM/PM (*USA) 2 = hh.mm.ss (*ISO) 3 = hh.mm.ss (*EUR) 4 = hh:mm:ss (*JIS)	0
TSP または TimeSeparator	SQL ステートメント内の時刻リテラルで使用される時刻区切り文字を指定します。このプロパティーは、「時刻形式」プロパティーが ″hms″ に設定されていない場合には効果がありません。	0 = ″:″ (コロン) 1 = ″.″ (ピリオド) 2 = ″,″ (コンマ) 3 = ″ ″ (ブランク)	0

接続ストリング・キーワード - パッケージ・プロパティー:

これらの System i Access ODBC ドライバー接続ストリング・キーワードは、ODBC 接続のパッケージ・プロパティーの変更に使用します。

以下の表は、System i Access ODBC ドライバーによって認識される、パッケージ・プロパティー用の接続ストリング・キーワードをリストしたものです。

表 6. パッケージ・プロパティー用の System i Access ODBC 接続ストリング・キーワード

キーワード	説明	選択項目	デフォルト
DFTPKGLIB または DefaultPkgLibrary	SQL パッケージ用のライブラリーを指定します。このプロパティーは、XDYNAMIC プロパティーが 1 に設定されていない場合には効果がありません。	SQL パッケージ用のライブラリー	QGPL
PKG または DefaultPackage	拡張動的 (パッケージ) サポートがどのように振る舞うのかを指定します。このプロパティーのストリングは、**A/DEFAULT(IBM),x,0,y,z,0** という形式になっていなければなりません。 x、y、および z は特殊属性であり、パッケージの使用法に応じて値を置き換える必要があります。 • x = 既存の SQL パッケージにステートメントを追加するかどうかを指定します。 • y = SQL パッケージのエラーが発生した場合に取る処置を指定します。SQL パッケージ・エラーが発生した場合、ドライバーは、このプロパティーの値に基づいて戻りコードを戻します。 • z = SQL パッケージをメモリーへキャッシュに入れるかどうかを指定します。 SQL パッケージをローカルでキャッシングすると、サーバーへの通信量を削減できる場合があります。 **注:** このプロパティーは、XDYNAMIC プロパティーが 1 に設定されていない場合には効果がありません。	**A/DEFAULT(IBM),x,0,y,z,0** x オプションの値: • 1 = 使用 (パッケージを使用するが、パッケージにそれ以上 SQL ステートメントを追加しない) • 2 = 使用 / 追加 (パッケージを使用し、新規 SQL ステートメントをパッケージに追加する) y オプションの値: • 0 = アプリケーションにエラー (SQL_ERROR) を戻す • 1 = アプリケーションに警告 (SQL_SUCCESS_WITH_INFO) を戻す • 2 = アプリケーションに成功 (SQL_SUCCESS) を戻す z オプションの値: • 0 = SQL パッケージをローカル・キャッシュに入れない • 1 = SQL パッケージをローカル・キャッシュに入れる	default

表 6. パッケージ・プロパティー用の System i Access ODBC 接続ストリング・キーワード (続き)

キーワード	説明	選択項目	デフォルト
XDYNAMIC または ExtendedDynamic	拡張動的 (パッケージ) サポートを使用するかどうかを指定します。 拡張動的サポートを使用すると、動的 SQL ステートメントをサーバーでキャッシングするためのメカニズムが提供されます。特定の SQL ステートメントを最初に実行するときには、そのステートメントがサーバー上の SQL パッケージに保管されます。その後同じ SQL ステートメントを実行するときには、サーバーは SQL パッケージに保管された情報を使用することにより、かなりの部分の処理をスキップすることができます。 **注:** 詳細については、 587 ページの『拡張動的 SQL の使用』を参照してください。	0 = 拡張動的サポートを使用不可にする 1 = 拡張動的サポートを使用可能にする	1

注: 「**A/DEFAULT(IBM),x,0,y,z,0**」は PKG または DefaultPackage のデフォルト値です。

接続ストリング・キーワード - パフォーマンス・プロパティー:

これらの System i Access ODBC ドライバー接続ストリング・キーワードは、ODBC 接続のパフォーマンス・プロパティーの変更に使用します。

以下の表は、System i Access ODBC ドライバーによって認識される、パフォーマンス・プロパティー用の接続ストリング・キーワードをリストしたものです。

表 7. パフォーマンス・プロパティー用の System i Access ODBC 接続ストリング・キーワード

キーワード	説明	選択項目	デフォルト
BLOCKFETCH	1 行の取り出しで内部ブロックを行うかどうかを指定します。これを設定すると、ドライバーは、あるレコードがアプリケーションによって要求されたときに、レコードの取り出しを最適化しようと試みます。そのアプリケーションが後で検索できるように、ドライバーが複数のレコードを検索し、保管します。アプリケーションが別の行を要求したときに、ドライバーは、あらためてホスト・データベースにフローを送らなくてもその行を獲得することができます。これを設定しない場合、ブロックは、特定のステートメントに関するアプリケーションの ODBC 設定値に基づいて使用されます。 **注:** このオプションの設定に関する詳細については、『レコード・ブロックの調整』トピックを参照してください。	0 = ODBC 設定値を使用してブロック化を行う 1 = 1 行の取り出しでブロックを使用する	1

表 7. パフォーマンス・プロパティー用の System i Access ODBC 接続ストリング・キーワード (続き)

キーワード	説明	選択項目	デフォルト
BLOCKSIZE または BlockSizeKB	System i から取り出されて、クライアントのキャッシュに入れられるブロック・サイズ (キロバイト単位) を指定します。このプロパティーは、BLOCKFETCH プロパティーが 1 に設定されていない場合には効果がありません。ブロック・サイズを大きくするほど、サーバーへの通信頻度が少なくなるため、パフォーマンスが向上する可能性があります。	1 から 8192	256
COMPRESSION または AllowDataCompression	サーバーとの間で送受信されるデータを圧縮するかどうかを指定します。多くの場合には、データ圧縮をおこなうと、ドライバーとサーバーとの間で伝送されるデータが少なくなるため、パフォーマンスが向上します。	0 = 圧縮を使用不可にする 1 = 圧縮を使用可能にする	1
CONCURRENCY	すべてのカーソルを更新可能として開き、ODBC の並行性設定値をオーバーライドするかどうかを指定します。 **注:** 次の 2 つの場合、このオプションを設定しても効果はありません。 1. SELECT SQL ステートメントを作成するときに、FOR FETCH ONLY または FOR UPDATE 文節が追加される可能性があります。これらのいずれかの文節が SQL ステートメントに存在していると、ODBC ドライバーはその文節に関連付けられている並行性を優先します。 2. カタログ結果セットは常に読み取り専用です。	0 = ODBC の並行性設定値を使用する 1 = すべてのカーソルを更新可能として開く	0
CURSORSENSITIVITY	カーソルをオープンする場合に使用するカーソル感度を指定します。このオプションは、同じ接続でオープンされるすべての下方専用の動的カーソルに適用されます。静的カーソルは常にインセンシティブです。	0 - 未指定/アセンシティブ 1 = インセンシティブ 2 = センシティブ	
EXTCOLINFO または ExtendedColInfo	拡張列情報は、SQLGetDescField および SQLColAttribute API がインプリメンテーション行記述子 (IRD) 情報として戻す内容に影響を与えます。拡張列情報は、SQLPrepare API が呼び出された後で使用可能になります。戻される情報は以下のとおりです。 • SQL_DESC_AUTO_UNIQUE_VALUE • SQL_DESC_BASE_COLUMN_NAME • SQL_DESC_BASE_TABLE_NAME および SQL_DESC_TABLE_NAME • SQL_DESC_LABEL • SQL_DESC_SCHEMA_NAME • SQL_DESC_SEARCHABLE • SQL_DESC_UNNAMED • SQL_DESC_UPDATABLE **注:** ドライバーが SQL_DESC_AUTO_UNIQUE_VALUE フラグを設定するのは、ある列が、数値データ・タイプ (整数など) に関する、ALWAYS オプションを指定された識別列である場合のみです。識別列の詳細については、『DB2 for i5/OS SQL 解説書』を参照してください。	0 = 拡張列情報を検索しない 1 = 拡張列情報を検索する	0

表 7. パフォーマンス・プロパティー用の *System i Access ODBC* 接続ストリング・キーワード *(続き)*

キーワード	説明	選択項目	デフォルト
LAZYCLOSE	後続の要求があるまでカーソルのクローズを遅延させるかどうかを指定します。遅延を指定すると、要求の合計数が減少し、全体的なパフォーマンスが向上します。 **注:** このオプションを指定すると、クローズ要求の後もカーソルが結果セット行でロックを引き続き維持するために、問題が生じることがあります。	0 = すべてのカーソルを即時にクローズする 1 = カーソルのクローズを次の要求まで遅延する	0
MAXFIELDLEN または MaxFieldLength	結果セットの一部として検索することのできる最大 LOB (ラージ・オブジェクト) サイズを、K バイト単位で指定します。このしきい値よりも大きな LOB は、サーバーとの追加の通信を使用して、分割して検索されます。 LOB しきい値を大きくすると、サーバーとの通信頻度は減少しますが、使用されないものも含め、ダウンロードされる LOB データが多くなります。 LOB しきい値を小さくすると、サーバーとの通信頻度が増大しますが、必要な LOB データのみがダウンロードされるようになります。 **注:** • このプロパティーを 0 に設定すると、ドライバーは常に LOB 値を追加の通信の流れとともに取得するように強制されます。 • このプロパティーを 15360 KB より大きく設定すると無効になります。 15360 KB よりも大きいものは、サーバーから分割して取得されます。データを分割して取得することで、どの時点でもクライアント上で必要なメモリー量を減らします。	0 から 2097152	32
PREFETCH	SELECT ステートメントの実行時にデータを事前取り出しするかどうかを指定します。事前取り出しを行うと、ResultSet の先頭部分の行にアクセスするときのパフォーマンスが向上します。	0 = データを事前取り出ししない 1 = データを事前取り出しする	1
QRYSTGLMT	照会のストレージ制限を指定します。ストレージ使用量の見積もりが、このパラメーターで指定したストレージ制限を上回る場合、照会は実行されません。	*NOMAX = 照会制限なし 0 から 2147352578	*NOMAX
QUERYOPTIMIZEGOAL	V5R4 以降のシステムで、System i 照会を最適化するために使用する目標を指定します。このパラメーターは、QAQQINI オプションの OPTIMIZATION_GOAL に対応します。詳しくは、『DB2for i5/OS SQL 解説書』の QAQQINI オプションを参照してください。	0 = 拡張動的サポートが有効な場合は、*ALLIO 目標を使用し、それ以外の場合は *FIRSTIO 目標を使用する。 1 = *FIRSTIO - 最初のデータ・ブロックを可能な限りすぐに戻す、という目的を使用する。 2 = *ALLIO - 照会全体を完了するまで実行する、という目的を使用する。	0
QUERYTIMEOUT	ドライバーが照会タイムアウト属性 SQL_ATTR_QUERY_TIMEOUT のサポートを使用不可にするかどうかを指定します。使用不可にすると、SQL 照会は終了するまで実行されます。	0 = 照会タイムアウト属性のサポートを使用不可にする 1 = 照会タイムアウト属性を設定できるようにする	1

関連資料

586 ページの『レコード・ブロックの調整』

レコード・ブロックは、System i Access ODBC を使用した場合に、ネットワーク・フローの数を大幅に削減する技法の 1 つです。

接続ストリング・キーワード - 言語プロパティー:

これらの System i Access ODBC ドライバー接続ストリング・キーワードは、ODBC 接続の言語プロパティーの変更に使用します。

以下の表は、System i Access ODBC ドライバーによって認識される、言語用の接続ストリング・キーワードをリストしたものです。

表 8. System i Access ODBC 接続ストリング・キーワード (言語プロパティー用)

キーワード	説明	選択項目	デフォルト
LANGUAGEID	ソート・シーケンスの選択に使用する 3 文字の言語 ID を指定します。このプロパティーは、SORTTYPE プロパティーが 2 に設定されていない場合には効果がありません。	AFR、ARA、BEL、BGR、CAT、CHS、CHT、CSY、DAN、DES、DEU、ELL、ENA、ENB、ENG、ENP、ENU、ESP、EST、FAR、FIN、FRA、FRB、FRC、FRS、GAE、HEB、HRV、HUN、ISL、ITA、ITS、JPN、KOR、LAO、LVA、LTU、MKD、NLB、NLD、NON、NOR、PLK、PTB、PTG、RMS、ROM、RUS、SKY、SLO、SQI、SRB、SRL、SVE、THA、TRK、UKR、URD、VIE	ENU
SORTTABLE	システムに保管されるソート・シーケンス・テーブルのライブラリーおよびファイル名を指定します。このプロパティーは、SORTTYPE プロパティーが 3 に設定されていない場合には効果がありません。	修飾されたソート・テーブル名	なし
SORTTYPE または SortSequence	レコードをクライアントに送信する前にサーバーがそのレコードをどのようにソートするのかを指定します。	0 または 1 = 16 進値に基づいてソートする 2 = LANGUAGEID プロパティーに設定されている言語に基づいてソートする 3 = SORTTABLE プロパティーに設定されているソート・シーケンス・テーブルに基づいてソートする	0
SORTWEIGHT	レコードをソートする際にサーバーが大文字小文字をどのように扱うのかを指定します。このプロパティーは、SORTTYPE プロパティーが 2 に設定されていない場合には効果がありません。	0 = 共通の重み (大文字と小文字を同じ文字としてソート) 1 = 固有の重み (大文字と小文字を別の文字としてソート)	0

接続ストリング・キーワード - カタログ・プロパティー:

これらの System i Access ODBC ドライバー接続ストリング・キーワードは、ODBC 接続のカタログ・プロパティーの変更に使用します。

以下の表は、System i Access ODBC ドライバーによって認識される、カタログ・プロパティー用の接続ストリング・キーワードをリストしたものです。

表 9. カタログ・プロパティー用の System i Access ODBC 接続ストリング・キーワード

キーワード	説明	選択項目	デフォルト
CATALOGOPTIONS	カタログ API が情報を戻す方法に影響を与える、1 つまたは複数のオプションを指定します。複数のカタログ・オプションを指定するには、必要なオプションに関連した値を追加してください。	このキーワードの値を決定するためには、以下の、必要な各オプションに関連した値を加算してください。 1 = SQLColumns 結果セット内の別名に関する情報を戻す。 2 = SQLTablePrivileges および SQLColumnPrivileges に関する結果セット情報を戻す。これは、V5R2 ホストでのみ使用できるという点に注意してください。以前のホストでは、ドライバーは空の結果セットを戻します。	3
LIBVIEW または LibraryView	カタログ API でワイルドカードを使用する場合に、情報を戻す際に検索するライブラリーのセットを指定します。多くの場合には、サーバー上のすべてのライブラリーを検索すると時間がかかることから、デフォルトのライブラリー・リストまたはデフォルトのライブラリー・オプションを使用してください。	0 = デフォルトのライブラリー・リストを使用する 1 = サーバー上のすべてのライブラリー 2 = デフォルトのライブラリーのみを使用する	0
REMARKS または ODBCRemarks	カタログ API 結果セット内の REMARKS 列のテキストのソースを指定します。	0 = i5/OS オブジェクト記述 1 = SQL オブジェクト・コメント	0
SEARCHPATTERN	ドライバーがライブラリーおよびテーブル名内のストリング検索パターンおよび下線をワイルドカード (検索パターン) として解釈するかどうかを指定します。デフォルトでは、% は「任意の数の文字」のワイルドカードとして処理され、_ は「単一の文字」のワイルドカードとして処理されます。	0 = 検索パターンをワイルドカードとして処理しない。 1 = 検索パターンをワイルドカードとして処理する。	1

注: 「A/DEFAULT(IBM),x,0,y,z,0」は PKG または DefaultPackage のデフォルト値です。

接続ストリング・キーワード - 変換プロパティー:

これらの System i Access ODBC ドライバー接続ストリング・キーワードは、ODBC 接続の変換プロパティーの変更に使用します。

以下の表は、System i Access ODBC ドライバーによって認識される、変換プロパティー用の接続ストリング・キーワードをリストしたものです。

表 10. System i Access ODBC 接続ストリング・キーワード (変換プロパティー用)

キーワード	説明	選択項目	デフォルト
ALLOWUNSCHAR または AllowUnsupportedChar	変換できない文字 (サポートされていない文字) が検出された際に発生するエラー・メッセージを抑制するかどうかを指定します。	0 = 文字を変換できない場合にエラー・メッセージを報告する 1 = 文字を変換できない場合のエラー・メッセージを抑制する	0
CCSID	デフォルトのクライアント・コード・ページ設定値をオーバーライドするコード・ページを指定します。	クライアント・コード・ページ設定値または 0 (デフォルトのクライアント・コード・ページ設定値を使用)	0
GRAPHIC	このプロパティーは、ユニコード以外の CCSID が指定されたグラフィック (DBCS) データ・タイプ GRAPHIC、VARGRAPHIC、LONG VARGRAPHIC、および DBCLOB の処理に影響を与えます。このプロパティーは、2 つの異なる振る舞いに影響を与えます。 1. グラフィックス・フィールドの長さを SQLDescribeCol API が文字カウントで報告するか、バイト・カウントで報告するか。 2. グラフィックス・フィールドを、SQLGetTypeInfo 結果セット内でサポートされるタイプとして報告するかどうか。	0 = 文字カウントを報告、サポートされないタイプとして報告する 1 = 文字カウントを報告、サポートされるタイプとして報告する 2 = バイト・カウントを報告、サポートされないタイプとして報告する 3 = バイト・カウントを報告、サポートされるタイプとして報告する	0
HEXPARSEROPT または Hex Parser Option	SQL 16 進定数を SQL ステートメントでどのように解釈するかを指定します。	0 = 16 進定数を文字データとして扱う 1 = 16 進定数をバイナリー・データとして扱う	0
TRANSLATE または ForceTranslation	バイナリー・データ (CCSID 65535) をテキストに変換するかどうかを指定します。このプロパティーが 1 に設定されると、バイナリー・フィールドは文字フィールドとして扱われます。 **注:** V5R3 以降のサーバーで新しい BINARY および VARBINARY データ・タイプを含むテーブルにアクセスする場合、この設定は適用されません。	0 = バイナリー・データをテキストに変換しない 1 = バイナリー・データをテキストに変換する	0
UNICODESQL	ユニコード SQL ステートメントをサーバーに送信するかどうかを指定します。	0 = EBCDIC SQL ステートメントをサーバーに送信する 1 = UCS-2 ユニコード SQL ステートメントをサーバー UCS-2 に送信する	0

表 10. System i Access ODBC 接続ストリング・キーワード (変換プロパティー用) (続き)

キーワード	説明	選択項目	デフォルト
XLATEDLL または TranslationDLL	ODBC ドライバーとサーバーの間でやり取りされるデータを変換するために ODBC ドライバーが使用する DLL の絶対パス名を指定します。この DLL は、接続が確立された際にロードされます。	変換 DLL の絶対パス名	なし
XLATEOPT または TranslationOption	変換 DLL に渡される 32 ビット整数変換オプションを指定します。このパラメーターはオプションです。このオプションの意味は、使用されている変換 DLL によって異なります。詳細については、変換 DLL とともに提供されている資料を参照してください。このオプションは、XLATEDLL プロパティーが設定されていない場合には使用されません。	32 ビット整数変換オプション	0

接続ストリング・キーワード - 診断プロパティー:

これらの System i Access ODBC ドライバー接続ストリング・キーワードは、ODBC 接続の診断プロパティーの変更に使用します。

以下の表は、System i Access ODBC ドライバーによって認識される、診断プロパティー用の接続ストリング・キーワードをリストしたものです。

表 11. 診断プロパティー用の System i Access ODBC 接続ストリング・キーワード

キーワード	説明	選択項目	デフォルト
QAQQINILIB または QAQQINILibrary	照会オプションのファイル・ライブラリーを指定します。照会オプションのファイル・ライブラリーが指定されていると、ドライバーは、QRYOPTLIB パラメーターにライブラリー名を渡して CHGQRYA コマンドを発行します。このコマンドは、接続が確立された直後に発行されます。このオプションは、使用可能にするとパフォーマンスに悪影響を与えるため、問題のデバッグ時またはサポート提供者によって推奨された場合にのみ使用してください。	照会オプションのファイル・ライブラリー	なし
SQDIAGCODE	設定する DB2 for i5/OS SQL 診断オプションを指定します。技術サポートの提供者によって指示された場合にのみ使用してください。	DB2 for i5/OS SQL 診断オプション	なし

表 11. 診断プロパティー用の System i Access ODBC 接続ストリング・キーワード (続き)

キーワード	説明	選択項目	デフォルト
TRACE	1 つまたは複数のトレース・オプションを指定します。複数のトレース・オプションを指定するためには、必要なオプションの値を合計して指定してください。例えば、データベース・モニターおよびデバッグ開始コマンドをサーバーで活動状態にしたい場合には、値 6 を指定する必要があります。これらのオプションは、パフォーマンスに悪影響を与えるため、問題のデバッグ時またはサポート提供者によって推奨された場合にのみ使用するようにしてください。	このキーワードの値を決定するためには、以下の、必要な各オプションに関連した値を加算してください。 0 = トレースを行わない 2 = データベース・モニターを使用可能にする 4 = デバッグ開始 (STRDBG) コマンドを使用可能にする 8 = 切断時にジョブ・ログを印刷する 16 = ジョブ・トレースを使用可能にする 32 = データベース・ホスト・サーバー・トレースを使用可能にする	0

接続ストリング・キーワード - その他のプロパティー:

これらの System i Access ODBC ドライバー接続ストリング・キーワードは、ODBC 接続のその他のプロパティーの変更に使用します。

以下の表は、System i Access ODBC ドライバーによって認識される、その他の接続ストリング・キーワードをリストしたものです。

表 12. その他のプロパティー用の System i Access ODBC 接続ストリング・キーワード

キーワード	説明	選択項目	デフォルト
ALLOWPROCCALLS	接続属性 SQL_ATTR_ACCESS_MODE が SQL_MODE_READ_ONLY に設定されている場合にストアード・プロシージャーを呼び出すことができるかどうかを指定します。	0 = ストアード・プロシージャーを呼び出すことができないようにする 1 = ストアード・プロシージャーを呼び出すことができるようにする	0

表 12. その他のプロパティー用の System i Access ODBC 接続ストリング・キーワード (続き)

キーワード	説明	選択項目	デフォルト
DB2SQLSTATES	ODBC で定義された SQL 状態または DB2 の SQL 状態を戻すかどうかを指定します。 DB2 の SQL 状態について詳しくは、 『DB2 for i5/OS SQL 解説書』を参照してください。このオプションは、ODBC アプリケーションのソース・コードを変更することができる場合にのみ使用するようにしてください。ほとんどのアプリケーションは、ODBC で定義された SQL 状態のみを処理するようにコーディングされているため、ODBC アプリケーションのソース・コードを変更することができない場合には、このオプションは 0 に設定されたままにしてください。	0 = ODBC 定義の SQL 状態を戻す 1 = DB2 の SQL 状態を戻す	0
DATETIMETOCHAR または ConvertDateTimeToChar	日付、時間、およびタイム・スタンプ・データ・タイプをアプリケーションに報告する方法についての、1 つ以上のオプションを指定します。複数のオプションを指定するためには、オプションの値を合計して指定してください。このオプションは、24:00:00 のような日付値が使用されるケースをサポートします。	このキーワードの値を決定するためには、以下の、必要な各オプションに関連した値を加算してください。 0 = DATE、TIME、および TIMESTAMP データ・タイプを SQL_TYPE_DATE 、SQL_TYPE_TIME、および SQL_TYPE_TIMESTAMP として継続してマップする 1 = DATE データ・タイプを SQL_CHAR として戻す 2 = TIME データ・タイプを SQL_CHAR として戻す 4 = TIMESTAMP データ・タイプを SQL_CHAR として戻す	0
DBCSNoTruncError	DBCS ストリング変換オーバーフロー・エラーを ODBC 切り捨てエラーとして報告するかどうかを指定します。	0 = DBCS ストリング変換オーバーフロー・エラーを ODBC 切り捨てエラーとして報告する 1 = 切り捨てエラーを無視する	0

表 12. その他のプロパティー用の System i Access ODBC 接続ストリング・キーワード (続き)

キーワード	説明	選択項目	デフォルト
DEBUG	1 つまたは複数のデバッグ・オプションを指定します。複数のデバッグ・オプションを指定するためには、必要なオプションの値を合計して指定してください。ほとんどの場合、このオプションを設定する必要はありません。	このキーワードの値を決定するためには、以下の、必要な各オプションに関連した値を加算してください。 2 = SQLGetInfo の SQL_IDENTIFIER_CASE オプションとして SQL_IC_MIXED を戻す 4 = パッケージ内のすべての SELECT ステートメントを保管する 8 = SQLGetInfo の SQL_MAX_QUALIFIER_NAME_LEN オプションとしてゼロを戻す 16 = 配置されている UPDATE / DELETE をパッケージに追加する 32 = 静的カーソルを動的カーソルに変換する 64 = 可変長フィールド (VARCHAR、VARGRAPHIC、BLOB など) のデータに相当する合計の列サイズを送信する。このオプションは、パフォーマンスに悪影響を与える可能性があるため、注意して使用してください。 128 = バッファー内の最後の文字がヌル終止符文字の場合に、SQLBindParameter のソース長から 1 を減算する 256 = データ 10 進数エラーを無視する 512 = 両方向スクロール・カーソルのキャスト警告 (SQL0402) を無視する 1024 = 可変長圧縮を使用不可にする 2048 = SQLGetInfo の SQL_CONVERT_TIMESTAMP オプションを呼び出す場合に、SQL_CVT_DATE のサポートを戻さない 32768 = 照会の結果、列が 0 で除算された場合に、エラーの代わりにヌル値を戻す	0

表 12. その他のプロパティー用の System i Access ODBC 接続ストリング・キーワード (続き)

キーワード	説明	選択項目	デフォルト
TRUEAUTOCOMMIT	自動コミット・サポートの処理方法を指定します。従来の ODBC ドライバーでは、自動コミットをオンにすると、サーバーは *NONE 分離レベルで実行されました。現在、自動コミットはどの分離レベルでも実行できます。 SQL 仕様に厳密に準拠する必要があるアプリケーションでは設定値 1 を使用します。この設定では、すべてのファイルをジャーナル処理する必要があることに注意してください。0 に設定すると、ほとんどのアプリケーションのパフォーマンスは向上します。トランザクション分離レベルについて詳しくは、「SQL 解説書」を参照してください。	0 = 自動コミットを *NONE 分離レベルで実行する 1 = 自動コミットを、その接続に対して設定された分離レベルで実行する。接続の分離レベルは、SQLSetConnectAttr API および SQL_ATTR_TXN_ISOLATION オプションを使用して設定します。	0
NEWPWD	現在のユーザーの System i パスワードをオーバーライドするために使用する、新規パスワードを指定します。このオプションは、アプリケーションによって設定される場合にのみ尊重されます。このオプションを使用する場合は、UID キーワードおよび PWD キーワードも指定する必要があります。	使用する新規パスワード	なし
XALCS または XALooselyCoupledSupport	疎結合分散トランザクション・ブランチ間でロックを共有するかどうかを指定します。	0 = ロックを共有しない 1 = ロックを共有する	1 (Windows の場合) 0 (Linux の場合)
XALOCKTIMEOUT	分散トランザクションがタイムアウトまでロック要求を待機する最大時間 (秒数) を指定します。	0 = デフォルトのシステム設定を使用する 0 – 999999999 = 待機時間数 (秒)	0
XATXNTIMEOUT または XATransactionTimeout	分散トランザクションがタイムアウトまで待機する時間 (秒数) を指定します。	0 = トランザクションが終了するまで無限に待機する 0 – 999999999 = 待機時間数 (秒)	0

バージョンおよびリリースの変更に伴う ODBC ドライバーの振る舞いの変更:

このトピックでは、System i Access for Windows 製品でサポートされる ODBC ドライバーのバージョンおよびリリースの変更点について説明します。

以下のリストは、V6R1 の重要な変更内容の一部について説明しています。

ODBC ドライバーを使用して V6R1 System i データへのアクセスする際に、以下のようなサポートが新たに行われます。

- SQL 照会ストレージの制限
- ODBC アプリケーションと QZDASOINIT システム・ジョブの関連付け
- 128 バイトのカーソル名
- 10 進浮動小数点 (DECFLOAT) データ・タイプ
- ストアード・プロシージャーの日時形式の追加

以下のリストは、V5R4 の重要な変更内容の一部について説明しています。

ODBC ドライバーを使用して V5R4 System i データにアクセスする際に、新しい機能が使用可能になりました。これらの機能は、以下のとおりです。

- 128 バイト列名のサポート
- 長い SQL ステートメントのサポート (コマンドの長さは、2,097,152 バイトまたは 1,048,576 文字まで)
- i5/OS ホストへの IBM Enterprise Workload Manager (eWLM) 相関係数引き渡しのサポート
- 大文字だけではないテーブル名および列名に対する改良済みのサポート
- 疎結合トランザクションに対する拡張分散トランザクション・サポート
- Linux 64 ビット ODBC ドライバー

以下のリストは、V5R3 の重要な変更内容の一部について説明しています。

ODBC ドライバーを使用して V5R3 System i データにアクセスする場合、いくつかの新規機能が使用可能です。これらの機能は、以下のとおりです。

- DB2 for i5/OS のデータベース・タイプ BINARY と VARBINARY。
- UTF-8 および UTF-16 データ
- 10 進数の精度の向上
- スクロール可能カーソルによるストアード・プロシージャー結果セットへのアクセスのサポート
- 自動コミットをすべての分離レベルで実行できる拡張コミットメント制御モデル
- 複数の System i 接続間でトランザクションの調整が可能な、Microsoft Transaction Server (MTS) / XA 拡張サポート

ODBC API の制約事項およびサポートされない関数:

System i Access の ODBC ドライバーにおける関数のインプリメント方法の中には、「Microsoft ODBC Software Development Kit Programmer's Reference」に記述されている仕様と一致しないものがあります。

以下の表は、グローバルな制限事項とサポートされない関数を示しています。個々の API およびそれらに関連する考慮事項については、538 ページの『ODBC 3.x API に関する注意事項』に示したリストを参照してください。

表 13. ODBC API 関数の制限事項

関数	説明
グローバルな考慮事項	非同期処理はサポートされません。ただし、SQLCancel を (マルチスレッド化されたアプリケーションにおいて) 異なるスレッドから呼び出して、実行に時間のかかっている照会を取り消すことができます。 Translation DLL は、バッファーから得られたデータを変換する際にのみ呼び出されます。
SQLSetScrollOptions (2x API)	SQL_CONCUR_ROWVER、SQL_CONCUR_VALUES は Concurrency パラメーターの非サポート・オプションです。 SQL_SCROLL_KEYSET_DRIVEN は、ドライバーによって SQL_SCROLL_DYNAMIC にマップされます。

関連資料

538 ページの『ODBC 3.x API に関する注意事項』

次の表は、System i Access ODBC 3.x API をその関連タスクごとにリストし、それぞれの API に関する考慮事項を示しています。

サインオン・ダイアログの振る舞い:

System i Access for Windows のサインオン・ダイアログ、ユーザー ID、およびパスワード・プロンプトを制御します。

V5R4 以降、サインオン・ダイアログの振る舞いが、従来の ODBC ドライバーのサポートにおける振る舞いよりも単純化されました。サインオン・ダイアログの振る舞いは、データ・ソースのセットアップ方法およびアプリケーションが接続に使用する ODBC API (SQLConnect、SQLDriverConnect、SQLBrowseConnect) によって決まります。

ODBC データ・ソースを構成する際に、サインオン・ダイアログの振る舞いに影響する可能性のあるオプションが 2 つあります。これらのオプションは、いずれも DSN セットアップ GUI の「**一般 (General)**」タブにある「**接続オプション (Connection Options)**」をクリックして表示されるダイアログにあります。

注: DSN セットアップ GUI に、サインオン情報のダイアログ・プロンプトを許可するかどうかを制御するオプションがあります。 3 層環境で SQLConnect を呼び出すアプリケーションは、必ず「SQLConnect のプロンプトを出さない (Never prompt for SQLConnect)」を選択する必要があります。この 3 層アプリケーションは、また、SQLConnect の呼び出し時にユーザー ID およびパスワードを必ず指定する必要があります。

- 「**デフォルト・ユーザー ID (Default user ID)**」セクションでは、使用するデフォルト・ユーザー ID を以下の中から指定することができます。
  - Windows ユーザー名を使用
  - 以下で指定したユーザー ID を使用 (Use the user ID specified below)
  - なし (None)
  - System i ナビゲーターのデフォルトを使用
  - Kerberos プリンシパルを使用 (Use Kerberos principal)
- 「**サインオン・ダイアログ・プロンプト (Signon dialog prompting)**」セクションでは、アプリケーションが SQLConnect ODBC API を使用する場合に、サインオン・ダイアログのプロンプトを出すかどうか指定することができます。

アプリケーションをコーディングする際に、ユーザー ID、パスワード、およびサインオン・ダイアログ・プロンプトの振る舞いを全体的に制御することができます。使用されるユーザー ID およびパスワードは、以下の順序で評価されます。

1. アプリケーションで指定されたユーザー ID / パスワード引数。
   - SQLConnect API はユーザー ID およびパスワード引数を受け入れます。
   - SQLDriverConnect API および SQLBrowseConnect API は UID、PWD、および SIGNON 接続ストリング・キーワードを受け入れます。
2. デフォルト・ユーザー ID の GUI 設定

サインオン・ダイアログ・プロンプトは、アプリケーションが接続のために使用する ODBC API によって決まります。サインオン・ダイアログ・プロンプトの GUI 設定でプロンプトを出さないと指定されていない限り、 SQLConnect は、必要に応じてサインオン・ダイアログを表示します。 SQLDriverConnect は、DriverCompletion の値に従って、サインオン・ダイアログのプロンプトを出します。SQL_DRIVER_NOPROMPT と設定すると、サインオン・ダイアログのプロンプトは全く出されなくなります。 SQL_DRIVER_PROMPT、SQL_DRIVER_COMPLETE または SQL_DRIVER_COMPLETE_REQUIRED と設定すると、必要に応じて、サインオン・ダイアログのプロンプトが出されます。 SQLBrowseConnect は、必要に応じてサインオン・ダイアログを出します。

ActiveX Data Objects (ADO) prompting

ActiveX Data Objects (ADO) を使用して ODBC アプリケーションをコード化する場合、プロンプトのデフォルトの振る舞いは **adPromptNever** です。プロンプトの出し方を変えるには、接続の Open メソッドを呼び出す前に、接続オブジェクトに対してプロンプト・プロパティーを設定します。例えば以下の ADO コードを指定すると、必要なときのみプロンプトが出されます。 SIGNON、UID、または PWD キーワードを追加することで、プロンプトの量をより制御することができます。

```
Dim conn As New ADODB.Connection
conn.Properties("Prompt") = adPromptComplete
conn.Open "Provider = MSDASQL;DSN=myODBCDSN;
```

ODBC データ・タイプおよびそれらと DB2 for i5/OS データベース・タイプとの対応:

System i Access の ODBC ドライバーのサポートによって、ODBC タイプと DB2 for i5/OS タイプの間でデータ・タイプがマップされます。

サポートされるデータ・タイプのマッピングについて、次の表で説明します。データ・タイプについて詳しくは、DB2 for i5/OS データベース・タイプへの関連リンク (以下参照) を選択してください。

表 14.

3.x ODBC データ・タイプ	DB2 for i5/OS データベース・タイプ
SQL_BIGINT	BIGINT
SQL_BINARY	BINARY または CHAR FOR BIT DATA
SQL_CHAR	CHAR または GRAPHIC
SQL_DECIMAL	DECIMAL
SQL_DOUBLE	DOUBLE
SQL_FLOAT	FLOAT
SQL_INTEGER	INTEGER
SQL_LONGVARBINARY	BLOB

表 14. (続き)

3.x ODBC データ・タイプ	DB2 for i5/OS データベース・タイプ
SQL_LONGVARCHAR	CLOB または DBCLOB
SQL_NUMERIC	NUMERIC
SQL_REAL	REAL
SQL_SMALLINT	SMALLINT
SQL_TYPE_DATE	DATE
SQL_TYPE_TIME	TIME
SQL_TYPE_TIMESTAMP	TIMESTAMP
SQL_VARBINARY	VARBINARY VARCHAR FOR BIT DATA LONG VARCHAR FOR BIT DATA ROWID
SQL_VARCHAR	VARCHAR VARGRAPHIC LONG VARCHAR LONG VARGRAPHIC DATALINK DECFLOAT
SQL_WCHAR	GRAPHIC CCSID 1200 GRAPHIC CCSID 13488
SQL_WLONGVARCHAR	DBCLOB CCSID 1200 DBCLOB CCSID 13488
SQL_WVARCHAR	VARGRAPHIC CCSID 1200 VARGRAPHIC CCSID 13488 LONGVARGRAPHIC CCSID 1200 LONG VARGRAPHIC CCSID 13488

インプリメンテーションに関する注意:

- 「Microsoft ODBC Software Development Kit Programmer's Reference バージョン 3.5」に記載されている変換はすべて、これらの ODBC SQL データ・タイプでサポートされています。
- 上記のデータ・タイプについて、個々に詳しく知りたい場合は、ODBC API SQLGetTypeInfo を呼び出してください。
- データベース・タイプ VARCHAR は、指定されている列サイズが 255 よりも大きい場合に、データベースにより LONG VARCHAR に変更されます。
- ODBC ドライバーは、インターバル SQL データ・タイプをサポートしていません。

- 2.x ODBC アプリケーションでは、SQL_TYPE_DATE、SQL_TYPE_TIME、および SQL_TYPE_TIMESTAMP 定義に代わって、SQL_DATE、SQL_TIME、および SQL_TIMESTAMP 定義を使用します。
- CCSID が 1200 (UTF-16) または 13488 (UCS-2) のデータ・タイプである Unicode フィールドは、SQL_WCHAR、SQL_WVARCHAR、および SQL_WLONGVARCHAR の代わりに、SQL_CHAR、SQL_VARCHAR、および SQL_LONGVARCHAR として ODBC 2.x アプリケーションに報告します。
- V5R2 (またはそれ以降) の System i バージョンでは、サイズが 2 GB までの LOB (BLOB、CLOB、および DBCLOB) がサポートされます。それ以前のリリースでは、15 MB までサポートされます。 LOB およびデータ・リンクの詳細については、ラージ・オブジェクト (LOB) の考慮事項に関するトピック集への関連リンク (以下参照) を選択してください。
- V5R3 (およびそれ以降) の System i バージョンでは、データ・タイプに対して精度の向上した 10 進数を使用できます。精度の高い 10 進数フィールドを正しく取得するためには、列を SQL_C_CHAR としてバインドする必要があることに注意してください。 SQL_C_NUMERIC データを保管する構造では 38 桁を保持することができます。

関連資料

『ラージ・オブジェクト (LOB) の考慮事項』
大容量のテキスト文書を保管し、それらにアクセスするためには、System i Access の ODBC で LOB を使用します。

関連情報

DB2 for i5/OS データベース・タイプ

ラージ・オブジェクト (LOB) の考慮事項:

大容量のテキスト文書を保管し、それらにアクセスするためには、System i Access の ODBC で LOB を使用します。

ラージ・オブジェクト (LOBs):

ラージ・オブジェクト (LOB) データ・タイプを使用すると、アプリケーションでは、大量のデータ・オブジェクトをストリングとして保管できます。 V5R1 以前のサーバーの ODBC ドライバーは、サイズが 15 MB 以下の LOB フィールドにアクセスすることができました。 V5R2 以降のサーバーの ODBC ドライバーは 2 GB の LOB にアクセスすることができます。

大容量の LOB データ・フィールドをサーバーにアップロードする場合には、SQLParamData および SQLPutData API を使用することをお勧めします。SQLPutData API は、受信した LOB データをサーバーに送信し、クライアントで必要なメモリーの量を減らします。

LOB データ・タイプ

BLOB バイナリー・ラージ・データ・オブジェクト

CLOB シングルバイト文字のラージ・データ・オブジェクト

DBCLOB
2 バイト文字のラージ・データ・オブジェクト

BLOB データ・タイプの使用例については、
以下のトピック『例: BLOB データ・タイプの使用』を参照してください。

LOB の詳細については、
i5/OS Information Center のトピック『SQL プログラミング概念』で、**『オブジェクト・リレーショナル機能の使用』**という見出しの下にあるトピック『ラージ・オブジェクトの使用』を参照してください。

データ・リンク

DataLink データ・タイプを使用すると、さまざまな種類のデータをデータベースに保管することができます。データは、URL として保管されます。 URL によって、オブジェクトが指定されます。オブジェクトは、イメージ・ファイル、音声ファイル、テキスト・ファイルなどの場合があります。

データ・リンクの詳細については、

i5/OS Information Center のトピック『SQL プログラミング概念』で、**『特別なデータ・タイプの処理』**という見出しの下にあるトピック『データ・リンクの使用』を参照してください。

関連資料

569 ページの『ODBC データ・タイプおよびそれらと DB2 for i5/OS データベース・タイプとの対応』

System i Access の ODBC ドライバーのサポートによって、ODBC タイプと DB2 for i5/OS タイプの間でデータ・タイプがマップされます。

関連情報

SQL プログラミングの一般概念

例: BLOB データ・タイプの使用:

System i Access for Windows の BLOB データ・タイプの使用例です。

以下は、C 言語で BLOB データ・タイプを使ったプログラムの一部です。

```
BOOL params = TRUE; // TRUE if you want to use parameter markers
SQLINTEGER char_len = 10, blob_len = 400;
SQLCHAR szCol1[21], szCol2[400], szRecCol1[21], szRecCol2[400];
SQLINTEGER cbCol1, cbCol2;
SQLCHAR stmt[2048];

// Create a table with a CHAR field and a BLOB field
rc = SQLExecDirect(hstmt, "CREATE TABLE TABBLOB(COL1 CHAR(10), COL2 BLOB(400))", SQL_NTS);

strcpy(szCol1, "1234567890");
if (!params)  // no parameter markers
{
  strcpy(szCol2, "414243444546"); // 0x41 = 'A', 0x42 = 'B', 0x43 = 'C', ...
  wsprintf(stmt, "INSERT INTO TABBLOB VALUES('%s', BLOB(x'%s'))", szCol1, szCol2);
}
else
{
  strcpy(szCol2, "ABCDEF"); // 'A' = 0x41, 'B' = 0x42, 'C' = 0x43, ...
  strcpy(stmt, "INSERT INTO TABBLOB VALUES(?,?)");
}

// Prepare the 'Insert' statement
rc = SQLPrepare(hstmt, stmt, SQL_NTS);

// Bind the parameter markers
if (params)  // using parameter markers
{
  cbCol1 = char_len;
  rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
                                      char_len, 0, szCol1, char_len + 1, &cbCol1);

  cbCol2 = 6;
  rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_LONGVARBINARY,
                                      blob_len, 0, szCol2, blob_len, &cbCol2);
}
```

```
// Execute the 'Insert' statement to put a row of data into the table
rc = SQLExecute(hstmt);

// Prepare and Execute a 'Select' statement
rc = SQLExecDirect(hstmt, "SELECT * FROM TABBLOB", SQL_NTS);

// Bind the columns
rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, szRecCol1, char_len + 1, &cbCol1);
rc = SQLBindCol(hstmt, 2, SQL_C_BINARY, szRecCol2, blob_len, &cbCol2);

// Fetch the first row
rc = SQLFetch(hstmt);
szRecCol2[cbCol2] = '¥0';

// At this point szRecCol1 should contain the data "1234567890"
// szRecCol2 should contain the data 0x414243444546 or "ABCDEF"
```

接続とステートメントの属性:

System i Access ODBC 仕様では、接続属性およびステートメント属性が複数定義されています。

この ODBC 仕様は、System i Access for Windows のカスタマイズ属性 (次の 2 つの表を参照) で拡張されています。

表 15. カスタマイズ接続属性

属性	Get/Set	説明
1204	両方	カーソル・コミットの振る舞いおよびカーソル・ロールバックの振る舞いを制御する無符号の値。使用できる値は以下のとおりです。 • 0 - SQL_CB_DELETE は、SQLGetInfo の SQL_CURSOR_COMMIT_BEHAVIOR および SQL_CURSOR_ROLLBACK_BEHAVIOR オプションの場合に戻されます。 • 1 - (デフォルト) SQL_CB_PRESERVE は、SQLGetInfo の SQL _CURSOR_COMMIT_BEHAVIOR および SQL_CURSOR_ROLLBACK_BEHAVIOR オプションの場合に戻されます。
1281	両方	ホスト・データベースに送信されるクライアント・ユーザー ID ストリングを指定します。この属性は、データベースへの接続後に設定されます。最大長は 255 文字です。この属性の代わりに、接続ストリング・キーワード CLIENTUSERID を使用できます。
1282	両方	ホスト・データベースに送信されるワークステーション名ストリングを指定します。最大長は 255 文字です。この属性は、データベースへの接続後に設定されます。この属性の代わりに、接続ストリング・キーワード CLIENTWRKSTNNAME を使用できます。
1283	両方	ODBC ドライバーを使用するアプリケーション名ストリングを指定します。この属性の最大長は 255 文字です。この属性は、データベースへの接続後に設定されます。この属性の代わりに、接続ストリング・キーワード CLIENTAPPLNAME を使用できます。
1284	両方	ホスト・データベースに送信されるアカウンティング ID ストリングを指定します。最大長は 255 文字です。この属性は、ホスト・データベースへの接続後に設定されます。この属性の代わりに、接続ストリング・キーワード CLIENTACCTSTR を使用できます。
2100	両方	DFTPKGLIB 接続ストリング・キーワードの代わりに使用できます。これは、使用するデフォルトのパッケージ・ライブラリーを指定する文字ストリングです。この属性は、この接続でステートメントを作成する前に設定しておく必要があります。
2101	両方	これは、使用するパッケージ名を指定する文字ストリングです。この属性は、この接続でステートメントを作成する前に設定しておく必要があります。

表 15. カスタマイズ接続属性 (続き)

属性	Get/Set	説明
2103	get	ODBC 接続が処理するサーバー CCSID 値 (CCSID ジョブ) である、無符号整数値を戻します。デフォルトでは、SQL ステートメントはこの CCSID のホストに送信されます。
2104	両方	DEBUG 接続ストリング・キーワードの「0 除算」オプションの代わりに使用することができます。これは、値をゼロで割った場合に結果セットの特定のセルのデータに対してエラーを戻すかどうかを示す、無符号の値です。使用できる値は以下のとおりです。 • 0 - (デフォルト) 0 除算の計算結果の値が入っている結果セットのセルがエラーとして戻されます。 • 1 - (デフォルト) 0 除算の計算結果の値が入っている結果セットのセルがヌル値として戻されます。エラーは戻されません。
2106	両方	COMPRESSION 接続ストリング・キーワードの代わりに使用することができます。これは無符号整数値です。使用できる値は以下のとおりです。 • 0 = 圧縮しない • 1 = 圧縮する
2109	set	CHAR フィールドから戻されたデータの末尾スペースを除去するかどうかを指定する無符号の値。これにより、VARCHAR フィールドが末尾スペースを必ず除去するように、 CHAR フィールドも VARCHAR フィールドのように表示されます。使用できる値は以下のとおりです。 • 0 - (デフォルト) - CHAR フィールドの末尾スペースを除去しない • 1 - CHAR フィールドの末尾スペースを除去する
2110	get	ODBC 接続が使用している事前開始ジョブに関する情報を含む文字ストリングを戻します。この情報は、以下の形式のストリングとして戻されます。 • 10 文字のジョブ名 • 10 文字のユーザー • 6 文字のジョブ
2116	set	IBM Enterprise Workload Manager (eWLM) 相関係数が含まれているバッファーへのポインター。この属性を指定すると、アプリケーションと eWLM サポート (Enterprise Workload Manager) を結合することができます。
2140	両方	分散トランザクションがタイムアウトまで待機する時間 (秒数) を指定する符号なし整数値。値 0 は、トランザクションが終了するまで無限に待機することを示します。この設定は、XATIMEOUT 接続ストリング・キーワードに設定された値に優先します。この属性のデフォルト値は 0 です。
2141	両方	分散トランザクションがタイムアウトまでロック要求を待機する最大時間 (秒数) を指定する符号なし整数値。値 0 は、デフォルトのシステム設定を使用することを示します。この設定は、XALOCKTIMEOUT 接続ストリング・キーワードに設定された値に優先します。この属性のデフォルト値は 0 です。
2142	両方	XA トランザクション作業に使用する RMID を指定する整数値。これは随時設定可能です。設定する RMID はプロセスで固有のものでなければなりません。この値が 0 の場合は、この接続に対する現行のすべての XA トランザクション作業が完了していることを示します。この属性のデフォルト値は 0 です。
2143	get	XA 呼び出しで呼び出す System i Access ドライバーを識別する文字ストリング。このストリングは 2142 接続属性が設定されている場合のみ有効です。このストリングは、接続の確立後に設定されます。この属性のデフォルト値は空ストリングです。

表 15. カスタマイズ接続属性 (続き)

属性	Get/Set	説明
2511	両方	ホスト・データベースに送信されるプログラム ID ストリングを指定します。最大サイズは 255 文字です。この属性は、データベースへの接続後に設定されます。この属性の代わりに、接続ストリング・キーワード CLIENTPROGRAMID を使用できます。

表 16. カスタマイズ・ステートメント属性

属性	Get/Set	説明
1014	get	取り出し可能な結果セット数を示す無符号整数値を戻します。これは、ストアード・プロシージャーが呼び出され、アプリケーションが、ストアード・プロシージャーにより生成された結果セット数を知りたい場合に役立ちます。
2106	両方	ステートメントの段階で圧縮をオンまたはオフにすることができます。指定できる値は以下のとおりです。 • 0 = 圧縮しない • 1 = 圧縮する
2114	get	SQL 構文エラーが発生した SQL ステートメントに対するオフセットを示す符号なし整数値を戻します。これは、SQLExecute または SQLExecDirect が SQL_ERROR 戻りコードを戻したときに設定されます。

接続プール:

System i Access ODBC 接続では接続プールがサポートされています。

接続プールとは、アプリケーションが System i Access ODBC 接続の切断を要求した後も ODBC 接続を開いたままにしておく振る舞いのことです。プール内にある接続は、同じアプリケーションで再使用でき、新しい接続を確立する場合の作業にかかる時間を節約することができます。

アプリケーションが System i Access ODBC ドライバーの接続プール・サポートを使用できるようにするには、 2 つの基本ステップを実行する必要があります。

1. ドライバーの接続プール・サポートを使用可能にしなければなりません。このサポートを使用可能にするには、ODBC Administrator を開き、「接続プール (Connection Pooling)」タブをクリックし、「System i Access ODBC ドライバー (32 ビット) (iSeries Access ODBC Driver (32-bit))」をダブルクリックしてチェック・ボックスを、このドライバーへの接続をプールするように切り替えます。このウィンドウには、プール内に未使用の接続を保持しておく時間を指定するフィールドもあります。

   注: V5R3 System i Access 製品で開始した場合、ドライバーの接続プール・サポートは自動的に使用可能になります。プール内に未使用の接続を保持しておくデフォルト時間 (60 秒) を指定変更しない限り、追加ステップは必要ありません。
2. 接続プール・サポートをアプリケーションで使用可能にしなければなりません。アプリケーションでこれを行うには、SQL_ATTR_CONNECTION_POOLING 環境属性を接続プロセスの一部として設定します。

接続プール・サポートについて詳しくは、 Microsoft Web サイトで ODBC を検索してください。

関連情報

Microsoft Web サイト

SQLPrepare および SQLNativeSQL エスケープ・シーケンスおよびスカラー関数:

System i Access ODBC サポートによって、エスケープ・シーケンスとスカラー関数が使用できます。

ODBC のエスケープ・シーケンスとスカラー関数を使用すると、特定のバージョンの DBMS の SQL 構文に直接コーディングする必要がなくなります。

エスケープ・シーケンスの使用方法については、Microsoft の ODBC 仕様を参照してください。以下の ODBC エスケープ・シーケンスが、System i Access for Windows ODBC ドライバーでサポートされています。

エスケープ・シーケンス

- d
- t
- ts
- escape
- oj
- call
- ?=call – このエスケープ・シーケンスは、DB2 for i5/OS において、ストアード・プロシージャーからの戻り値のサポートを利用する場合に使用します。パラメーター・マーカーは、SQLBindParameter API を使用して、出力パラメーターとして結合する必要があります。このとき、ストアード・プロシージャーは、整数タイプの値しか戻せないことに注意してください。
- fn – このエスケープ・シーケンスは、以下のスカラー関数を使用している場合に使用します。構文は { fn scalar_function } です。

ODBC ドライバーにより DB2 for i5/OS の SQL 構文にマップされるスカラー関数は、以下のとおりです。

- length
- log
- database (V5R3 より前のサーバーでのみマップされる)
- insert (V5R3 より前のサーバーでのみマップされる)
- right (V5R3 より前のサーバーでのみマップされる)

注: その他のスカラー関数はすべて、DB2 for i5/OS の SQL 構文で最初からサポートされているため、マッピングは必要ありません。

分散トランザクションのサポート:

分散トランザクションでは、1 つの System i Access ODBC アプリケーションが複数のデータベース間で作業単位を調整することができます。

ODBC ドライバーには、分散トランザクションの実行を可能にする 2 つの異なるインターフェースが組み込まれています。この 2 つのインターフェースは MTS (Microsoft Transaction Server) と XA API サポートです。どちらのインターフェースも、XALOCKTIMEOUT と XATXNTIMEOUT の接続ストリング設定の設定の影響を受けます。

MTS

MTS の詳細については、『Microsoft Transaction Server (MTS) の使用』を参照してください。

XA API サポート

XA サポートの動作させるための関連オプションの一部についての説明は、**『接続およびステートメントの属性』**ページの 2140、2141、2142、および 2143 の接続属性を参照してください。 2141 および 2142 の接続属性の動作は、XALOCKTIMEOUT および XATXNTIMEOUT 接続ストリング設定と同じであることに注意してください。

注:

- 分散トランザクションに複数の System i 接続を組み込むことができるのは V5R3 以降のサーバーのみです。
- xa_open がアプリケーションで呼び出されるのは、リカバリーの場合だけです。 ODBC API の SQLConnect または SQLDriverConnect を介して接続するときに、RMID が 2142 接続属性を介して設定されていると、 xa_open は自動的に実行されます。
- 接続属性 SQL_ATTR_AUTOCOMMIT は SQL_AUTOCOMMIT_ON として設定しなければなりません。
- アプリケーションで XA トランザクションを開始した後に別の XA 以外のトランザクション作業を開始したい場合は、 RMID を 0 に設定し、XA 作業が完了したことをドライバーに示さなければなりません。
- XA リカバリーを実行する場合、アプリケーションはストリング *SYSTEM=mySystem;UID=myUserID;PWD="myPassword";DATABASE=myDatabase;* を指定して xa_open を呼び出します。 - ここで、mySystem にはシステム名、myUserID にはそのシステムのユーザー ID、 myPassword にはそのユーザー ID のパスワードを指定します。ストリングは示されているとおりに正確に入力するようにしてください。または、単に *SYSTEM=mySystem;* と指定することもできます。

カーソルの動作に関する注意事項:

System i Access ODBC ドライバーを使用している場合、カーソルの動作がデータを取り出す方法に影響することがあります。

SQLSetStmtAttr に SQL_ATTR_CURSOR_TYPE オプションを指定して、カーソル・タイプを設定することができます。

カーソル・タイプ

- SQL_CURSOR_FORWARD_ONLY - すべてのカタログおよびストアード・プロシージャー結果セットは、このタイプのカーソルを使用します。カタログまたはストアード・プロシージャー結果セットが生成されている場合には、カーソル・タイプは自動的にこれに変更されます。
- SQL_CURSOR_KEYSET_DRIVEN - ホストがサポートしている場合は SQL_CURSOR_STATIC にマップされ、そうでない場合は、SQL_CURSOR_DYNAMIC にマップされます。
- SQL_CURSOR_DYNAMIC - サポートされています。
- SQL_CURSOR_STATIC - System i V5R1 以降のバージョンでは、静的カーソルがサポートされています。このカーソル・タイプは、以前のバージョンの System i の SQL_CURSOR_DYNAMIC にマップされます。

注: カーソル・タイプおよびストアード・プロシージャーの結果セットについて詳しくは、『ストアード・プロシージャーの結果セット』を参照してください。

以下のファクターは、カーソルの並行性に影響を与える可能性があります。

- SQL ステートメントに ″FOR UPDATE″ 文節が含まれている場合、SQL_ATTR_CONCURRENCY の値は SQL_CONCUR_LOCK に設定されます。
- CONCURRENCY キーワードの DSN 設定が 1 (チェック) に設定されている場合、SQL ステートメントに ″FOR FETCH ONLY″ 文節がない場合、ODBC ドライバーは結果セットのレコードをロックします。

行セット・サイズ

ODBC ドライバーは、SQLExtendedFetch を処理する際に SQL_ROWSET_SIZE という値を使用します。このドライバーは、SQLFetch および SQLFetchScroll を処理する際に SQL_ATTR_ROW_ARRAY_SIZE という値を使用します。

結果セットに LOB がある場合、ロケーターがドライバーによって使用される可能性があります。ロケーターは LOB フィールドに対する内部ハンドルです。ロケーターは、MAXFIELDLEN 接続オプションの設定値が結果セットの LOB 列のサイズよりも小さい値である場合に使用されます。ロケーターを使用すると、ドライバーがアプリケーションによって要求されるデータのみを取得するようになるため、パフォーマンスが向上する場合があります。ロケーターの欠点は、サーバーとの間で余分な通信が必要になるという点です。ロケーターを使用しない場合、ドライバーは、使用されないものを含め、より多くの LOB データをダウンロードします。ロケーターを使用していない場合には、COMPRESSION 接続オプションを使用可能にすることを強くお勧めします。 MAXFIELDLEN キーワードの詳細については、『接続ストリング・キーワード』の説明を参照してください。

SQLGetData は、単一行取り出しで得られたデータにアクセスする場合にのみ使用することができます。複数行取り出しでは、SQLGetData の呼び出しはサポートされません。

結果セットの行数

データを取り出す前にアプリケーションで行数を判別するために使用する方法はいくつかあります。

- カーソル・タイプを SQL_CURSOR_STATIC に設定することができます。
- アプリケーションが ADO を使用している場合は、クライアント・サイドのカーソルを使用することができます。
- アプリケーションは、実際の照会を実行する前に SELECT COUNT(*) FROM MYTABLE を呼び出すことで COUNT() 関数を使用することができます。
- 同じ照会を 2 度実行することができます。最初に照会を実行するときに、すべてのデータを取り出して行数をカウントします。

拡張動的使用不可エラー:

System i Access ODBC サポートでは、SQL パッケージが使用できない場合に*拡張動的サポート使用不可*メッセージが表示されます。

以前のサーバーでは、パッケージの作成者ではないユーザーが異なるデフォルト・ライブラリーに接続すると、このメッセージが表示されます。このメッセージへの対処方法は、次のいずれかになります。

1. アプリケーションを実行したときに、パッケージがデフォルトのパッケージ設定で作成されるように、システム上の SQL パッケージを削除します。
2. SQL パッケージと一緒に保管されている設定と一致するように *SQL デフォルト・ライブラリー* の接続ストリング設定を変更します。

3. 「*使用できないパッケージの戻りコード (Return code for unusable package)*」という ODBC DSN 設定を「*無視する (Ignore)*」または「*警告 (Warning)*」に切り替えます。また、この同じ振る舞いは、PKG 接続ストリング設定を設定することでも実現できます。
4. XDYNAMIC 接続ストリング設定を使用不可にします。

ODBC 64 ビット Windows および Linux に関する考慮事項:

System i Access for Windows または System i Access for Linux の環境で ODBC ドライバーを使用する際の、ヘッダー・ファイルおよびデータ型を識別します。

System i Access の ODBC ドライバーでは、64 ビット ODBC API のサポートを実装しています。通常は、Microsoft for Windows 環境および unixODBC for Linux 環境で提供される ODBC ヘッダー・ファイルの定義に合わせて、このサポートが実装されます。 ODBC API を呼び出すコードを作成する場合は、その関数プロトタイプに該当する ODBC ヘッダー・ファイルを参照してください。該当するヘッダー・ファイルは、以下のとおりです。

- sql.h
- sqlext.h
- sqltypes.h
- sqlucode.h

SQLExtendedFetch の場合、Linux では sqlext.h の定義とは異なる処理になります。 sqlext.h における pcrow パラメーターの定義は、SQLROWSETSIZE ポインターとして定義されています。
SQLROWSETSIZE は、64 ビット Linux 実装環境では、4 バイト値になります。ただし ODBC ドライバーは、64 ビット Windows ODBC 実装環境との整合性を保つために、pcrow ポインターのデータを 8 バイト (64 ビット) 値として戻します。

Windows、Linux、および 64 ビット間においては、以下のような特有の相違が生じます。

- 64 ビット Linux 環境では、long C/C++ 型のサイズは 8 バイトになります。その他の環境 (64 ビット Windows など) では、long 型のサイズは 4 バイトになります。以下の表を参照してください。
- 32 ビット環境では、ポインターのサイズは 4 バイトになります。 64 ビット環境では、ポインターのサイズは 8 バイトになります。
- ODBC API の中には、ポインターをパラメーターとして持つものがあります。それらのポインターを使用して、アプリケーションとドライバーの間で、サイズの異なるデータを受け渡す場合があります。 64 ビット実装環境において、この方法で受け渡されたデータのサイズが 4 バイト値から 8 バイト値に変更された場合には、若干の変更が生じます。

共通の C/C++ 型の一部とそれぞれのサイズについて、以下の表で説明します。

表 17. 共通の C/C++ 型および各サイズ

C/C++ 型	Linux 64 ビット	Windows 64 ビット	Linux 32 ビット	Windows 32 ビット
int	4	4	4	4
long	8	4	4	4
long long	8	未定義	8	未定義
LONG LONG	未定義	8	未定義	未定義
ポインター・サイズ	8	8	4	4
INT32	未定義 4[1]	4	未定義 4[1]	4
INT64	未定義 8[1]	8	未定義 8[1]	8

表 17. 共通の C/C++ 型および各サイズ (続き)

C/C++ 型	Linux 64 ビット	Windows 64 ビット	Linux 32 ビット	Windows 32 ビット
SQLSMALLINT	2	2	2	2
SQLINTEGER	4	4	4	4
SQLLEN	8	8	4	4
SQLSETPOSIROW	8	8	2	2
SQLROWCOUNT	8	未定義	4	4
SQLROWSETSIZE	4	未定義	4	4
SQLROWOFFSET	8	未定義	4	4
SQLPOINTER	8	8	4	4
UINT_PTR	未定義 8[1]	8	未定義 4[1]	4
DWORD	未定義 4[1]	4	未定義 4[1]	4
SDWORD	未定義 4[1]	4	未定義 4[1]	4
ULONG_PTR	未定義 8[1]	8	未定義 4[1]	4
SQLHANDLE	8	8	4	4
SQLHDESC	8	8	4	4

注: 1. この型は、標準ヘッダー・ファイルに定義されていません。 System i Access for Linux 製品に付属するツールキットで、定義を行います。

以下の表にある ODBC API のオプションの、パラメーター・ポインター・データに対する振る舞いは、ODBC ドライバーが 32 ビットの場合と 64 ビットの場合とで異なります。通常、特に断りがなければ、64 ビットの ODBC ドライバーが、パラメーター・ポインター・データを 8 バイト (64 ビット) 値として処理します。

SQLColAttribute

SQL_DESC_DISPLAY_SIZE

SQL_DESC_LENGTH

SQL_DESC_OCTET_LENGTH

SQL_DESC_COUNT

SQLColAttributes

SQL_COLUMN_DISPLAY_SIZE

SQL_COLUMN_LENGTH

SQL_COLUMN_COUNT

SQLGetConnectAttr

SQL_ATTR_QUIET_MODE

SQLGetConnectOption (ODBC ドライバー・マネージャーによって、SQLGetConnectAttr にマップされます。) SQL_ATTR_QUIET_MODE

SQLGetDescField

SQL_DESC_ARRAY_SIZE

SQLGetDiagField

SQL_DIAG_CURSOR_ROW_COUNT

SQL_DIAG_ROW_COUNT

SQL_DIAG_ROW_NUMBER

SQLGetInfo (ODBC ドライバー・マネージャーによって、すべて処理されます。)

SQL_DRIVER_HENV

SQL_DRIVER_HDBC

SQL_DRIVER_HLIB

SQL_DRIVER_HSTMT

SQL_DRIVER_HDESC

SQLGetStmtAttr

SQL_ATTR_APP_PARAM_DESC

SQL_ATTR_APP_ROW_DESC

SQL_ATTR_IMP_PARAM_DESC

SQL_ATTR_IMP_ROW_DESC

SQL_ATTR_MAX_LENGTH

SQL_ATTR_MAX_ROWS

SQL_ATTR_PARAM_BIND_OFFSET_PTR

SQL_ATTR_ROW_ARRAY_SIZE

SQL_ATTR_ROW_BIND_OFFSET_PTR

SQL_ATTR_ROW_NUMBER

SQL_ATTR_ROWS_FETCHED_PTR

SQL_ATTR_KEYSET_SIZE

SQLGetStmtOption (ODBC ドライバー・マネージャーによって、SQLGetStmtAttr にマップされます。)

SQL_MAX_LENGTH

SQL_MAX_ROWS

SQL_ROWSET_SIZE

SQL_KEYSET_SIZE

SQLSetConnectAttr

SQL_ATTR_QUIET_MODE

SQLSetConnectOption (ODBC ドライバー・マネージャーによって、SQLSetConnectAttr にマップされます。) SQL_ATTR_QUIET_MODE

SQLSetDescField

SQL_DESC_ARRAY_SIZE

SQLSetStmtAttr

SQL_ATTR_APP_PARAM_DESC

SQL_ATTR_APP_ROW_DESC

SQL_ATTR_IMP_PARAM_DESC

SQL_ATTR_IMP_ROW_DESC

SQL_ATTR_MAX_LENGTH

SQL_ATTR_MAX_ROWS

SQL_ATTR_PARAM_BIND_OFFSET_PTR

SQL_ATTR_ROW_ARRAY_SIZE

SQL_ATTR_ROW_BIND_OFFSET_PTR

SQL_ATTR_ROW_NUMBER

SQL_ATTR_ROWS_FETCHED_PTR

SQL_ATTR_KEYSET_SIZE

SQLSetConnectAttr

SQL_MAX_LENGTH

SQL_MAX_ROWS

SQL_ROWSET_SIZE

SQL_KEYSET_SIZE

64 ビット System i Access for Windows ODBC ドライバーの制約事項:

64 ビット System i Access for Windows ODBC ドライバーでは、MTS はサポートされていません。

MTS について詳しくは、『Microsoft Transaction Server (MTS) の使用』を参照してください。

SQLTables の説明:

System i Access SQL テーブルを使用する際には、いくつかの考慮事項があります。

- CatalogName パラメーターは、ワイルドカードを使用しているかどうかにかかわらず、無視されます。これは、カタログ名が常にリレーショナル・データベース名であるためです。カタログ名の値が問題となるのは、サーバーのライブラリーのリストを生成するために空ストリングにしなければならない場合のみです。

  SQL ステートメントの作成時に指定したとおり正確に、TableName パラメーターにテーブル名を指定しなければなりません。つまり、テーブル名は、二重引用符で囲んで作成していない限り、大文字にしなければなりません。二重引用符で囲んだテーブル名でテーブルを作成している場合は、TableName パラメーターも、引用符で囲まれる場合と同じように、大文字小文字を区別して指定する必要があります。
- DSN セットアップ GUI の「**カタログ**」タブの「ライブラリー表示」オプションは、当該サーバーのライブラリー・リストを検索しようとする組み合わせを選択するときにのみ、この API に影響を与えます。この場合、特定のテーブルの複数のライブラリーの検索に基づいて、結果セットを生成することはできません。
- DSN セットアップ GUI の「**カタログ**」タブの「オブジェクト記述タイプ (Object description type)」オプションは、テーブルのリストを取得する際に結果セットの「結果」列に得られる出力に影響を与えます。
- '¥_' と '_' が混合しているストリングの場合、SQL_ATTR_METADATA_ID が SQL_FALSE であれば、最初の '¥_' は実際には '_' として処理されますが、 '_' はワイルドカードとして処理されます。SQL_ATTR_METADATA_ID が SQL_TRUE である場合、最初の '¥_' は実際には '_' として処理され、 '_' も実際には '_' のように処理されます。ドライバーが、2 番目の '_' を '¥_' に内部変換します。

- ワイルドカード文字、下線 (_) をリテラルとして使用するためには、その前に円記号 (¥) を付けます。例えば、MY_TABLE (MYATABLE でも MYBTABLE でもない) のみを検索するには、検索ストリングを、MY¥_TABLE と指定する必要があります。

  名前に '¥%' を指定しても、無効になります。 System i オペレーティング・システムでは、ライブラリー名またはテーブル名内に、実際の '%' を使用することが許可されていないためです。

  ライブラリーのリストに照会があると、ドライバーは、意味のあるデータとして TABLE_CAT および REMARKS フィールドを戻します。

  ODBC 仕様では、ヌルとしての TABLE_SCHEM を除いて、すべてを戻すようになっています。

長時間実行照会の処理:

System i Access ODBC を介して長時間実行照会を処理する方法は少なくとも 2 つあります。

1. アプリケーションで SQL_ATTR_QUERY_TIMEOUT 接続属性を設定し、照会が実行できる最大時間を指定することができます。照会の処理に必要な時間が SQL_ATTR_QUERY_TIMEOUT 値を超えると SQL Optimizer が判断した場合、その照会は開始されないことに注意してください。SQL_ATTR_QUERY_TIMEOUT のデフォルト値は 0 で、照会は完了するまで実行されることを示します。
2. アプリケーションは SQLCancel API を呼び出すことができます。このためには、アプリケーションをマルチスレッド化する必要があります。長時間実行照会を 1 つのスレッドで実行しながら、別のスレッドで同じステートメント・ハンドルを使用して SQLCancel を呼び出します。

コミットメント制御の考慮事項:

異なるコミット・レベルに対する System i Access ODBC 自動コミット・サポートを実行します。

V5R3 から導入された、System i のコミットメント制御モデルにより、ODBC 自動コミット・サポートを実行して、*NONE 以外のコミット・レベルも使用できるようになりました。 V5R3 より前は、自動コミット・サポートは常に *NONE コミット・レベルで実行されていました。デフォルトは、今後も続けて *NONE です。

*NONE 以外を指定すれば、異なるコミット・レベルで自動コミットを実行することができます。 *NONE 以外の自動コミットのコミットメント・レベルを使用する場合は、追加でその他の変更を行う必要があり、一部の機能の振る舞いが変更される場合があることに注意してください。たとえば、ジャーナル記録されていないファイルの更新機能の除去などです。詳しくは、『SQL 解説書 分離レベル』の一連のトピックを参照してください。

TRUEAUTOCOMMIT という SQLDriverConnect キーワードは、*NONE コミット・レベルまたは SQL_ATTR_TXN_ISOLATION 設定で自動コミットを実行するかどうかをアプリケーションが制御できるようにします。SQLDriverConnect 接続ストリングで TRUEAUTOCOMMIT を 1 に設定すると、アプリケーションは SQL_ATTR_TXN_ISOLATION 設定を使用して自動コミットを実行します。TRUEAUTOCOMMIT を設定しない場合は、デフォルト値の 0 が使用されます。デフォルトの振る舞いでは、自動コミットは *NONE コミット・レベルで実行されます。

関連情報

SQL 解説書 分離レベル

System i Access for Windows ODBC のパフォーマンス

System i Access の ODBC パフォーマンスに関しては、以下のトピックを参照してください。

注: コード例を使用することで、 633 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』の条件に同意します。

System i Access for Windows ODBC のパフォーマンス調整:

System i Access ODBC アプリケーション開発者にとって重要なのは、クライアント/サーバー・アプリケーションの**パフォーマンス**を最大限引き出すことです。

以下のトピックでは、クライアント/サーバーのパフォーマンス上の問題を概説し、一般的な照会ツールおよび開発環境で ODBC を使った場合のパフォーマンスについて記述しています。

サーバー・パフォーマンスの概要:

すべてのコンピューティング環境でのパフォーマンスの特性について、以下の項目を使用して説明します。

応答時間
要求が処理されるまでにかかる時間

使用率 要求の処理時に使用されている資源の割合

スループット
単位時間当たり処理される要求のボリューム

容量 最大限可能なスループット量

通常、サーバーの**ユーザー**にとって、応答時間はパフォーマンスにおける重大な問題です。使用率は、サーバー**管理者**にとって重要であることが多いと言えます。最大スループットはパフォーマンスのボトルネックを示しますが、それほど重要ではない場合があります。これらの特性はすべて相互に関連していますが、サーバーのパフォーマンスについて要約すると以下のようになります。

- いかなるコンピューティング・サーバーにも、パフォーマンス、つまり**スループット**を左右するボトルネックがある
- サーバーの使用率が高くなると、応答時間が低下する

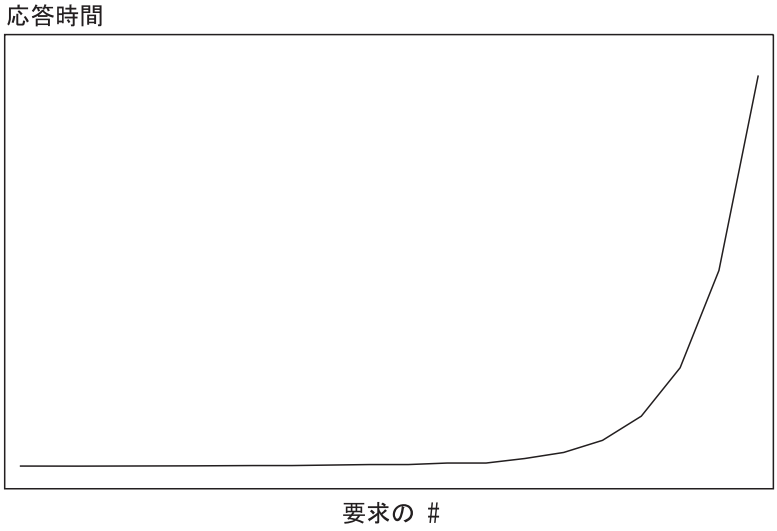
多くのサーバーでは、キャパシティーは無視できませんが、ユーザーにとっては重要ではありません。一方、キャパシティーがパフォーマンス上の最重要問題になっているシステムもあります。応答時間は常にクリティカルです。管理者にとっての最重要課題の 1 つは、*(ユーザーの増加や使用率の増大によって) サーバーのパフォーマンスが低下しても、ユーザーから苦情が出ないのはどの程度までか* ということです。

クライアント/サーバーのパフォーマンスの概要:

クライアント/サーバー環境のパフォーマンスにみられる特性は、集中型の環境の場合とは異なります。

その理由は、クライアント/サーバー・アプリケーションがクライアントとサーバーの間で分割されているためです。クライアントとサーバーは、要求とメッセージを送ったり、受信したりすることによってコミュニケーションしています。このモデルが、集中型の環境と大きく違う点です。集中型の環境では、プログラムが CPU を呼び出し、メモリーとディスク・ドライブはすべて専用に割り当てられています。

それに対して、クライアントがサーバーの処理時間とデータを要求する場合は、クライアントがネットワークにその要求を送信します。送信された要求はサーバーに届くと、サーバーで処理可能になるまで待ち行列に入って待機します。このタイプのアーキテクチャーでのパフォーマンス特性としては、要求の数が増加するにつれて指数関数的に低下するという点があります。言い換えれば、要求が増えるごとにそれだけ応答時間も長くかかってしまいます。その値は、徐々に増えますが、ある時点で飛躍的に増大します。これは、グラフの曲線の急な折れ曲がりとして知られているポイントです。この概念を図で表すと、以下のグラフのようになります。



パフォーマンスの著しい低下が始まるポイントを判別しておくことが重要です。このポイントは、クライアント/サーバーのインストール先によって異なります。

クライアント/サーバーの運用に推奨されるガイドラインは、*サーバーとの通信は必要な場合にのみ行い、できる限りデータ転送を少なくする* ことです。ファイルを開いてレコードを 1 行ずつ読み取る作業は、多くの場合クライアント/サーバーのプロジェクトとツールに問題を発生させます。

System i Access for Windows ODBC ドライバーのパフォーマンス・アーキテクチャー:

System i Access ODBC ドライバーでは、クライアントとサーバーの間でやりとりされる内部データのフローはすべて連鎖しており、必要な場合にのみ送信されます。

通信レイヤーの資源が割り当てられるのは 1 回のみであるため、サーバーの使用率は削減されます。これによって、応答時間は短縮されます。

ユーザーは、こうした機能の拡張を意識することはありません。ただし、System i Access の「ODBC セットアップ (ODBC Setup)」ダイアログに見られるような、いくつかの機能強化が行われています。詳細については、セットアップ GUI の「**パフォーマンス**」タブのオンライン・ヘルプ、または『接続ストリング・キーワード』の「パフォーマンス」オプションの説明を参照してください。これらのパフォーマンス・オプションのうちの一部については、以下のリンク先でさらに詳しく説明されています。

コミットメント制御の強制レベルの選択:

System i Access ODBC コミットメント制御を使用するにあたって、いくつかの重要な考慮事項があります。

不必要に、コミットメント制御を使用しないでください。ロックに伴うオーバーヘッドによって使用率が増大するほか、並行性が低下します。ただし、アプリケーションが読み取り専用でない場合は、コミットメント制御が必要な場合があります。

一般的な方法としては、**最適ロック**を使用します。最適ロックには、特定のレコードを一意に決定する WHERE 文節を使った明示的な UPDATE を発行する必要があります。最適ロックにより、レコードが検索後に変更されないことが確実になります。

多くの第三者ツールではこの方法が使われているため、更新可能なテーブルに対して固有索引を定義する必要があります。これによって、レコードの内容全体が完全に限定され、レコードの更新が可能になります。次の例を参照してください。

```
UPDATE table SET C1=new_val1, C2=new_val2, C2=new_val3
   WHERE C1=old_val1 AND C2=old_val2 AND C3=old_val3
```

このステートメントでは、目的の行が正確に更新されることが保証されるのは、テーブルに含まれている列が 3 列のみであり、各行に固有の値がある場合のみです。以下のようにした方が効率がよくなります。

```
UPDATE table SET C1=new_val1, C2=new_val2, C3=CURRENT_TIMESTAMP
   WHERE C3=old_timestamp
```

ただし、これが有効なのは、レコードが最後に更新された日時の情報を示すタイム・スタンプ列がテーブルに含まれている場合に限られます。この列の新しい値を CURRENT_TIMESTAMP に設定すると、行の一意性が保証されます。

注: この手法は、自動化データ・タイプが使用されるオブジェクト・モデル (Visual Basic、Delphi、スクリプト言語など) では利用できません。バリアント DATE データ・タイプでは、タイム・スタンプの精度はおよそ 1 ミリ秒単位です。System i タイム・スタンプは切り捨てられるか丸められて、WHERE 文節はエラーになります。

コミットメント制御が必要な場合は、使用可能な最低位レベルのレコード・ロックを使用します。例えば、可能なときは ***CS** より ***CHG** を使い、***CS** で事足りる場合は絶対に ***ALL** を使わないでください。

関連情報

データベースのコミットメント制御

DB2 for i5/OS SQL 解説書

レコード・ブロックの調整:

レコード・ブロックは、System i Access ODBC を使用した場合に、ネットワーク・フローの数を大幅に削減する技法の 1 つです。

これは、カーソルに対する最初の FETCH 要求のときにサーバーから行のブロック を戻すことによって行います。後に続く FETCH 要求は、毎回サーバーに送られるのではなく、ローカルにある行のブロックから取り出されます。この方法を適切に使用することで、パフォーマンスが目覚ましく向上します。たいていの場合は、デフォルトの設定で十分です。

レコード・ブロックのパラメーターを変更すると、使用環境のパフォーマンスが 584 ページの『クライアント/サーバーのパフォーマンスの概要』に示されている指数関数的なしきい値に近づいたときに、大きな変化が生じる可能性があります。例えば、ある環境で、通常、1MB のデータを返すような大きな照会を処理している意思決定支援クライアントが *n* 個あるとします。

まったく逆の仮定としては、常時ユーザーが大量のデータを要求しているが、通常は数行しか調べないといった場合です。数行しか必要でないときに 32 KB の行を戻すことで生じるオーバーヘッドによって、パフォーマンスが低下する可能性があります。 *BLOCKSIZE* または *BlockSizeKB* 接続ストリング・キーワードを低い値に設定するか、*BLOCKFETCH* 接続ストリング・キーワードを 0 に設定するか (ODBC ブロック化を使用)、またはレコード・ブロックを完全に使用不可にすると、パフォーマンスが実際に向上します。

クライアント/サーバーでは常にパフォーマンスの結果が異なっています。これらのパラメーターを変更しても、はっきりとした変化が見られない可能性もあります。これはつまり、パフォーマンス上のボトルネックがサーバー上のクライアント要求待ち行列にあるのではないと考えられます。ユーザーから苦情が出た場合のもう 1 つのツールとして、このパラメーターを使用できます。

関連資料

556 ページの『接続ストリング・キーワード - パフォーマンス・プロパティー』
これらの System i Access ODBC ドライバー接続ストリング・キーワードは、ODBC 接続のパフォーマンス・プロパティーの変更に使用します。

拡張動的 SQL の使用:

System i Access ODBC コール・レベル・インターフェースを使用して、動的に SQL ステートメントを実行します。

従来の SQL インターフェースでは、組み込み SQL の方法を使用していました。 SQL ステートメントは、C、COBOL、RPG およびその他のプログラミング言語で記述された高水準言語ステートメントと並んで、アプリケーションのソース・コード中に直接配置されました。次に、ソース・コードがプリコンパイルされ、それによって SQL ステートメントはコンパイルの次の段階で処理できるコードに変換されていました。この方式は、**静的 SQL** と呼ばれました。この方法に対するパフォーマンス上の利点は、SQL ステートメントの最適化が、実行時にユーザーが待機している間ではなく前もって行われることにありました。

しかし ODBC は、別の方法を使用する**呼び出しレベル・インターフェース** (CLI) です。CLI を使用すると、SQL ステートメントは実行時 API のパラメーター内で DBMS (データベース管理システム) に渡されます。SQL ステートメントのテキストは実行時までわからないため、SQL ステートメントが実行されるたびに最適化処理を行う必要があります。通常、この方法は**動的 SQL** と呼ばれます。

この機能 (デフォルトで使用可能に設定されています) を使用すると、応答時間が短縮されるだけでなく、サーバーの使用効率も飛躍的に向上します。これは、SQL 照会の最適化にはコストがかかるものの、この処理を 1 回実行すれば常に効果があるためです。 DB2 for i5/OS の固有の機能においては、非常に効果的です。他の DBMS とは異なり、管理者が介入しなくても、パッケージ内に保管されているステートメントが、常に最新の最適化状態に保たれます。ステートメントが最初に準備されたのが数週間または数カ月前であったとしても、データベースを適切に変更するために再最適化が必要があると判断された場合には、DB2 for i5/OS で自動的にアクセス・プランが再生成されます。

パッケージおよびパッケージに保管されている SQL ステートメントのタイプについて詳しくは、i5/OS Information Center のトピック『SQL パッケージ』を参照してください。

関連情報

SQL パッケージ

一般的なエンド・ユーザー用ツールでのパフォーマンスの考慮事項:

ご使用の System i Access の ODBC ドライバー環境のチューニングに役立つ、各種ツールがあります。

最適の状態に調整された ODBC ドライバーを用意することは、パフォーマンスのバランスをとるために必要なことの一部に過ぎません。その他に、使用ツールについて、データを照会するためにのみ使用するのか、または複雑なプログラムを作成するために使用するのかといったことを確認する必要があります。

一般的に使用されているツールには、以下のものがあります。

- Crystal Services Crystal Reports Professional
- Cognos Impromptu
- Gupta SQL Windows
- IBM Visualizer for Windows
- Lotus® Approach®

- Lotus Notes
- Notes® Pump
- Microsoft Access
- Microsoft Internet Information Server
- Microsoft SQL Server
- Microsoft Visual Basic
- Powersoft PowerBuilder

このリスト以外にも使用できるツールが多数あります。市販されているツールにはそれぞれ長所も短所も、パフォーマンス特性もあります。たいていのツールに共通しているのは、ODBC データベース・サーバーへのサポートです。 ODBC はさまざまなデータベース管理システムに共通の標準として機能しますが、各 ODBC ドライバー間に微妙な違いが存在するため、ツール・プロバイダーの多くは、ごく一般的な ODBC と SQL インターフェースに合わせたツールを作成しています。これでは、特定のデータベース・サーバーのユニークな特性を生かすことができません。プログラミング作業が軽減されることもありますが、全体のパフォーマンスの低下につながることがよくあります。

例: ODBC パフォーマンスを低下させる一般的なツールの動作:

特定の ODBC ドライバーまたはサーバー・データベース管理システムの固有の機能を利用しない、SQL 呼び出しおよび System i Access ODBC 呼び出しの作成に関するパフォーマンス上の問題について、以下の例で説明します。

例: 照会ツール A:

この例は、System i Access ODBC 列バインドを使用して情報を素早く検索します。

照会ツール A では、以下の ODBC 呼び出しを行って SELECT ステートメントを処理します。

```
SQLExecDirect("SELECT * FROM table_name")

WHILE there_are_rows_to_fetch DO

   SQLFetch()
   FOR every_column DO
      SQLGetData( COLn )
   END FOR
   ...process the data

END WHILE
```

ODBC 列バインドはパフォーマンスの維持に役立ちますが、このツールでは使用されません。この処理を速くする方法は、以下のとおりです。

```
SQLExecDirect("SELECT * FROM table_name")
FOR every_column DO
   SQLBindColumn( COLn )
END FOR

WHILE there_are_rows_to_fetch DO
   SQLFetch()
   ...process the data
END WHILE
```

テーブルに含まれている列が 1 列の場合、2 つの方法に大きな違いはありません。しかし、100 列あるテーブルの場合は、最初の例では*取り出す行ごとに* 100 回もの ODBC 呼び出しが行われることになってし

まいます。ツールによって指定されているターゲット・データ・タイプは FETCH ごとに変更されないため、**SQLGetData** 呼び出しごとに変更できるのと同じように、2 番目のシナリオを最適化することもできます。

例: 照会ツール B:

この例では、System i Access ODBC 呼び出し全体に対して 1 つの割り振りステートメントを使用します。

照会ツール B を使用すると、複数行から構成されるスプレッドシートを更新し、その更新情報をデータベースに送ることができます。これは、以下の ODBC 呼び出しを行います。

```
FOR every_row_updated DO

   SQLAllocHandle(SQL_HANDLE_STMT)
   SQLExecDirect("UPDATE...SET COLn='literal'...WHERE COLn='oldval'...")
   SQLFreeHandle( SQL_HANDLE_STMT )

END LOOP
```

初めに注意する点は、このツールでは、行ごとにステートメントの割り当てとドロップが実行されるという点です。必要な割り当てステートメントは 1 つのみです。この変更を行うと、操作ごとにステートメントのハンドルを作成および破棄するときのオーバーヘッドを節減できます。パフォーマンス上のもう 1 つの問題は、パラメーター・マーカーではなくリテラルで SQL を使用している点です。**SQLExecDirect()** 呼び出しによって、毎回 **SQLPrepare** および **SQLExecute** が発生します。この操作を迅速に行う方法は、以下のとおりです。

```
SQLAllocHandle(SQL_HANDLE_STMT)
SQLPrepare("UPDATE...SET COL1=?...WHERE COL1=?...")
SQLBindParameter( new_column_buffers )
SQLBindParameter( old_column_buffers )
FOR every_row_updated DO

   ...move each rows data into the SQLBindParameter buffers
   SQLExecute()
   SQLFreeHandle( SQL_HANDLE_STMT )

END LOOP
```

System i Access for Windows ODBC ドライバーを使用している場合は、これらの一連の ODBC 呼び出しによって、元の呼び出しよりはるかに効率がよくなります。サーバーの CPU 使用率はそれまでの 10% に縮小され、基準化のためのしきい値を考慮する必要がなくなります。

例: 照会ツール C:

この例では、複雑な意思決定支援タイプの照会により System i Access ODBC 照会の実行が長くなります。

照会ツール C を使用すると、ポイント・アンド・クリック・インターフェースで高度な照会基準を定義することで、複雑な意思決定支援タイプの照会ができます。最終的に、次のような SQL を照会に使用する場合があります。

```
SELECT A.COL1, B.COL2, C.COL3 , etc...
   FROM A, B, C, etc...
      WHERE many complex inner and outer joins are specified
```

複雑な照会を記述しなくてもよいのは便利ですが、実際にツールがこのステートメントを処理しない場合がある点に注意してください。例えば、このステートメントを直接 ODBC ドライバーに渡すツールもあれば、以下のように多数の照会に分割し、その結果をクライアントで処理するツールもあります。

```
SQLExecDirect("SELECT * FROM A")
SQLFetch() all rows from A
SQLExecDirect("SELECT * FROM B")
SQLFetch() all rows from B

Process the first join at the client

SQLExecDirect("SELECT * FROM C")
SQLFetch() all rows from C

Process the next join at the client
.
.
.
And so on...
```

この方法では、クライアントに渡されるデータの量が過大になってしまい、パフォーマンスが低下します。実例として、あるプログラマーは 10 とおりの内部結合と外部結合が ODBC に渡されて、4 行が戻されると考えました。しかし、実際に渡されたものは、10 個の簡単な SELECT ステートメントとそれらに関連したすべての FETCH です。最終結果の 4 行が得られたのは、ツールによって *81,000* 回の ODBC 呼び出しが行われた後です。プログラマーは最初、低速パフォーマンスの原因が ODBC にあると考えたわけですが、ODBC のトレースを調べたところ、そうではないことがわかりました。

SQL パフォーマンス:

優れたアプリケーション設計には、マシンの資源の有効利用も含まれます。 System i Access ODBC 環境で、エンド・ユーザーが使いやすい方法で実行されるアプリケーション・プログラムとは、動作効率がよく、適切な応答時間で実行されるものでなければなりません。

SQL パフォーマンスに関する一般的な考慮事項:

System i Access ODBC 環境を設計する際の時期、対象、方法などに関する質問への回答が得られます。

アプリケーション・プログラムにおける SQL のパフォーマンスはすべてのサーバー・ユーザーに重要です。SQL の使用が非効率であると、サーバー資源を浪費してしまう可能性があるためです。

SQL を使用する第 1 の目的は、データベース要求に対して、正確な結果を適切なタイミングで取得することです。

パフォーマンスを考慮した設計を始める前に、以下の考慮事項について検討してください。

パフォーマンスについて考察する必要がある場合

- 10,000 行を超えるデータベース - パフォーマンスへの影響: **要注意**
- 100,000 行を超えるデータベース - パフォーマンスへの影響: **重大**
- 複雑な照会を繰り返し使用する場合
- トランザクションの多いワークステーションを複数使っている場合

最適化する資源

- I/O 使用率
- CPU 使用率
- 索引の効果的な使用

- OPEN/CLOSE のパフォーマンス
- 並行性 (COMMIT)

パフォーマンスを考慮した設計方法

- **データベース設計**
  - テーブル構造
  - 索引
  - テーブル・データ管理
  - ジャーナル管理
- **アプリケーション設計**
  - 関連プログラムの構造
- **プログラム設計**
  - コーディング方法
  - パフォーマンス・モニター

「*SQL 解説書*」ブックには、その他の情報が記載されています。 i5/OS Information Center にある『DB2 for i5/OS SQL 解説書』のトピックから、上記ブックの HTML オンライン版を表示するか、PDF 版を印刷することができます。

関連情報

DB2 for i5/OS SQL 解説書

データベース設計:

DB2 for i5/OS データベースに必要なテーブルを判別し、各テーブル間の関係を理解するために、以下のトピックを役立ててください。

正規化:

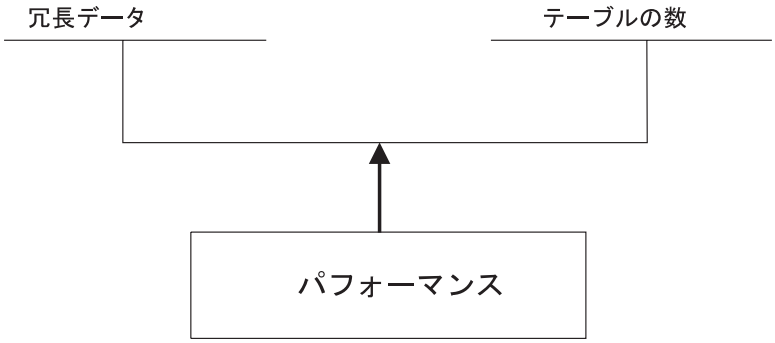
正規化は、System i Access for Windows のデータベース・テーブルの設計時に使用できる方式の 1 つです。

いくつかの有効な設計方式を使用すると、技術的に正しいデータベース、および効率のよいリレーショナル・データベース構造を設計できます。これらの方式には、正規化と呼ばれる設計方法をベースにしているものがあります。正規化とは、冗長データの保存を少なくしたり、除去したりすることです。

正規化の第 1 の目的は、冗長データの更新に関連する問題を回避することです。

ただし、正規化という設計方法 (例えば、3NF–3rd Normal Form) を使用することで、テーブルの数が膨大に増えてしまうことがあります。テーブルの結合操作が多い場合は、SQL パフォーマンスの低下が予想されます。データベースを設計するときは、全体の SQL パフォーマンスについても考えてください。冗長データの量と、完全には正規化されていないテーブルの数とのバランスを考慮してください。

以下の図は、パフォーマンスに影響するテーブルの数と冗長データの割合を示しています。



コード・テーブルを使用してもあまり役に立たない場合は、その使用を最小限にしてください。例えば、EMPLOYEE (従業員) テーブルに 054、057 などのデータ値を持つ JOBCODE (職種コード) 列があるとします。このテーブルを別のテーブルと組み合わせて、コードをプログラマー、技術者などの職種に変換する必要があります。この結合では、節減された記憶域に比較して、コストの方がかなり大きくなり、冗長データによって更新エラーが発生する可能性もあります。

例えば、以下のとおりです。

EMPLOYEE テーブル

従業員番号	ジョブ・コード
00010	057
00020	054
00030	057
...	...

JOBCODE テーブル

ジョブ・コード	ジョブ名称
054	プログラマー
057	技術者
...	...
...	...

図 2. 正規化されたデータ形式

EMPLOYEE テーブル

従業員番号	ジョブ名称
00010	技術者
00020	プログラマー
00030	技術者
...	...

図 3. 冗長データ形式

SQL のセット・レベル (大量操作) の特性によって、特定の冗長データ形式の危険性が著しく小さくなります。例えば、1 つの SQL ステートメントで複数行のセットを更新できる機能によって、このリスクを非常に低くすることができます。以下の例では、条件に合うすべての行に対して、**Engineer** という職種を **Technician** に変更する必要があります。

SQL を使用して JOBTITLE を更新します。

```
UPDATE EMPLOYEE
  SET JOBTITLE = "Technician"
  WHERE JOBTITLE = "Engineer"
```

テーブル・サイズ:

アプリケーション・プログラムがアクセスするテーブルのサイズは、System i Access の ODBC アプリケーション・プログラムのパフォーマンスに大きな影響を与えます。

以下のことを考慮してください。

行が長い場合

列が多い (100 以上ある) ために行が長くなった順次アクセス・テーブルの場合は、テーブルを小さく分割するか、ビューを作成するとパフォーマンスを向上させることができます。これは、アプリケーションがすべての列にはアクセスしていないことを前提としています。パフォーマンスがよくなる主な理由は、ページ当たりの取得行が増えることによって I/O が軽減されるためです。テーブルを分割した場合、すべての列にアクセスするアプリケーションでは、テーブルを再度結合することでオーバーヘッドがかかるため影響が出ます。アプリケーションの特性と多くの列に対するアクセス頻度に基づいて、テーブルの分割位置を決定する必要があります。

行数が多い場合

テーブルの行数が多い場合は、最適化ルーチンで索引を使用してテーブルにアクセスするように、SQL ステートメントを構成してください。最高のパフォーマンスを実現する上で、索引の使用は非常に重要です。

関連資料

594 ページの『最適化ルーチン』
最適化ルーチンは、データベースのパフォーマンス向上において重要な役割を担うため、i5/OS 照会構成要素の重要なモジュールの 1 つといえます。 System i Access ODBC データにおける最も効率のよいアクセス・パスを検出することが、主な目的となります。

『索引の使用』
索引を使用すると、System i Access ODBC アプリケーションのパフォーマンスが著しく向上します。

索引の使用:

索引を使用すると、System i Access ODBC アプリケーションのパフォーマンスが著しく向上します。

この原因は、これらの索引をオプティマイザーが使用して、パフォーマンスを最適化するからです。オプティマイザーの詳細については、関連リンクを参照してください。

索引の作成には、以下の 5 つの方法があります。

- CREATE INDEX (SQL の中で)
- CRTPF (キー使用)
- CRTLF (キー使用)
- CRTLF (結合論理ファイルとして)
- CRTLF (選択 / 除外を指定、キー不使用、動的選択 (DYNSLT) なし)

索引を使用すると、索引対テーブルのスキャン操作によって行を選択することができますが、通常これは処理が遅くなります。テーブルのスキャンでは、テーブルのすべての行が順次に処理されます。永続索引が使用できる場合は、一時索引を作成しなくても済みます。索引は、以下に対して必要です。

- テーブルの結合
- ORDER BY
- GROUP BY

永続索引がない場合は、索引が作成されます。

索引の数を管理して、更新操作中の索引の維持管理作業にかかる追加のサーバー・コストを最小限に抑えます。以下は、特定のタイプのテーブルに対する汎用規則です。

主として読み取り専用のテーブルの場合
必要に応じて、列の索引を作成します。テーブルが 1,000 行分よりも大きく、ORDER BY、GROUP BY または結合処理で使用される場合にのみ索引を作成するようにしてください。索引の維持管理作業には、随時テーブル全体をスキャンするよりもコストがかかる可能性があります。

主として読み取り専用で、更新頻度は低いテーブルの場合
必要に応じて、列の索引を作成します。頻繁に更新される列の索引は作成しません。INSERT、UPDATE、および DELETE によって、テーブルに関連するすべての索引に対して維持管理作業が行われます。

更新頻度の高いテーブルの場合
索引を多く作り過ぎないようにします。更新頻度の高いテーブルには、ログや履歴テーブルがあります。

関連資料

『最適化ルーチン』
最適化ルーチンは、データベースのパフォーマンス向上において重要な役割を担うため、i5/OS 照会構成要素の重要なモジュールの 1 つといえます。 System i Access ODBC データにおける最も効率のよいアクセス・パスを検出することが、主な目的となります。

593 ページの『テーブル・サイズ』
アプリケーション・プログラムがアクセスするテーブルのサイズは、System i Access の ODBC アプリケーション・プログラムのパフォーマンスに大きな影響を与えます。

結合フィールドの属性の突き合わせ:

System i Access ODBC 定義では、結合されるテーブルの属性は同じでなければなりません。

列の長さやデータ・タイプ (文字、数値) など、結合されるテーブルの列の属性は、同一でなければなりません。同一でない属性があると、対応する列の索引が既にある場合でも一時索引が作成されることになります。

次の例では、**結合**によって一時索引が作成され、既存の索引は無視されます。

```
SELECT EMPNO, LASTNAME, DEPTNAME
  FROM TEMPL, TDEPT
  WHERE TEMPL.DEPTNO = TDEPT.DEPTNO
```

最適化ルーチン:

最適化ルーチンは、データベースのパフォーマンス向上において重要な役割を担うため、i5/OS 照会構成要素の重要なモジュールの 1 つといえます。 System i Access ODBC データにおける最も効率のよいアクセス・パスを検出することが、主な目的となります。

照会の最適化は、照会実施方法の選択に要する時間と、その実行にかかる時間との間の妥協です。照会の最適化では、次のようなユーザーの明確なニーズに応える必要があります。

- 高速で対話式の応答
- マシン資源全体の有効利用

データへのアクセス方法を決定するために、最適化ルーチンでは以下が行われます。

- 可能な実施方法の判別
- i5/OS 照会構成要素での実行に最適な実施方法の選択

関連資料

593 ページの『索引の使用』
索引を使用すると、System i Access ODBC アプリケーションのパフォーマンスが著しく向上します。

593 ページの『テーブル・サイズ』
アプリケーション・プログラムがアクセスするテーブルのサイズは、System i Access の ODBC アプリケーション・プログラムのパフォーマンスに大きな影響を与えます。

コストの見積もり:

実行時に、最適化ルーチンは、System i Access ODBC データベースの現在の状態に基づいて実施方法のコストを計算して、照会に対する最適なアクセス方式を選択します。

最適化ルーチンは、以下のそれぞれについてのアクセス・コストをモデル化します。

- テーブルからの直接の行読み取り (データ・スペース・スキャン処理)
- アクセス・パスを介した行読み取り (キー選択またはキー位置を使用)
- データ・スペースからの直接のアクセス・パス作成
- 既存のアクセス・パスからの新しいアクセス・パスの作成 (索引から索引)
- 照会ソート・ルーチンの使用 (条件が満たされている場合)

特定の方式によるコストは、以下の合計です。

- 開始時のコスト
- 該当の最適化モードに関連するコスト。OPTIMIZE FOR n ROWS 文節は、照会の最適化ルーチンに対して、達成すべき最適化目標を示しています。最適化ルーチンでは、以下の 2 つの目標のいずれかで SQL 照会を最適化することができます。
  1. テーブルから、行の最初のバッファーを取り出すのに要する時間を最小限にします。この目標では、最適化が索引の作成をしないように仕向けます。

     注: これは、OPTIMIZE FOR n ROWS を使わない場合のデフォルトの設定です。

     データ・スキャンまたは既存の索引が選択されます。このモードは、以下によって指定できます。
     - ユーザーが照会で検索したい行数を指定できる OPTIMIZE FOR n ROWS。

       最適化ルーチンはこの値を使って、戻される行の割合を判別し、それに応じて最適化します。小さい値を指定すると、最適化ルーチンは、最初の n 行の取り出しに必要な時間を最小限にするように指示されたことになります。

2. 選択されたすべての行がアプリケーションに戻されていると想定して、照会全体の処理時間を最小限にします。これによって、最適化ルーチンが特定のアクセス方式に偏ることはありません。
   OPTIMIZE FOR n ROWS を使って、このモードを指定します。OPTIMIZE FOR n ROWS を使用すると、ユーザーは照会で取り出したい行数を指定できます。

   最適化ルーチンはこの値を使って、戻される行の割合を判別し、それに応じて最適化します。結果の行数が予想していた以上の値であれば、最適化ルーチンは照会全体の実行に要する時間を最小限にするように指示されます。

- アクセス・パス作成のコスト。
- 行読み取りに予想されるページ不在数によるコスト、および予想される行数の処理によるコスト。

  ページ不在数および処理される行数は、最適化ルーチンがデータベース・オブジェクトから取得する以下の統計値によって予測されることがあります。
  - テーブル・サイズ
  - 行サイズ
  - 索引サイズ
  - キー・サイズ

  予想される処理行数の重みの基準。これは、行の選択述部 (デフォルトのフィルター係数) の関係演算子で取り出すと考えられるものに基づいています。
  - 10% 「等しい」
  - 33% 「未満」「大きい」「以下」「以上」
  - 90% 「等しくない」
  - 25% BETWEEN 範囲
  - 10% 各 IN リスト値

  キー範囲の見積もりは、1 つまたは複数の選択述部から選択されている予想行数について、より正確な見積もりを得るために最適化ルーチンで使用される方式です。最適化ルーチンは、既存の索引の左端のキーに対して選択述部を適用することで、見積もります。**デフォルトのフィルター係数**は、キー範囲に基づく見積もりによって、さらに精度が上がる可能性があります。索引の左端のキーが行選択述部で使われている列に一致している場合は、その索引を使って選択基準に合うキーの数を見積もります。キーの数の見積もりは、ページの数とマシン索引のキー密度に基づいています。この見積もりは、実際にキーにアクセスしないで実行されます。選択述部で使用される列の完全な索引があれば、最適化に非常に役立ちます。

最適化ルーチンの意思決定規則:

最適化ルーチンでは、実行時に一般的なガイドラインを使って、System i Access ODBC データへのアクセスに最適な方法を選択します。

最適化ルーチンによって、以下が実行されます。
- 選択文節のそれぞれの述部に対して、デフォルトのフィルター係数を判別します。
- 内部に保管された情報からテーブルの属性を抽出します。
- 選択述部が索引の左端のキーに一致している場合に、見積キー範囲を実行して、述部の実際のフィルター係数を判別します。
- 索引が必要な場合は、テーブルに索引を作成するコストを判別します。
- 選択基準が適用でき、索引が必要な場合は、ソート・ルーチンの使用によるコストを判別します。

- 索引が必要でない場合は、データ・スペースのスキャン処理によるコストを判別します。
- 最適化ルーチンでは、使用できるそれぞれの索引について、最近作成されたものから古い順に、時間制限を超えるまで以下の処理が行われます。
  - 内部に保管されている統計データから索引の属性を抽出します。
  - 索引が選択基準を満たしているかどうか判別します。
  - 見積ページ不在と述部フィルター係数を使って、索引使用によるコストを判別します。
  - この索引の使用によるコストと前のコスト (現在最適のもの) を比較します。
  - 最も低い値を選択します。
  - タイムアウトになるか、または索引がなくなるまで、続けて最適な索引を検索します。

時間制限係数によって、実施方法の選択に要する時間が制御されます。この時間は、所要時間と現在の最適な実施方法に基づいています。動的 SQL 照会は、最適化ルーチンの時間制限に左右されます。静的 SQL 照会には時間制限がありません。

テーブルが小さい場合は、照会の最適化ルーチンで照会の最適化にあまり時間がかかりません。テーブルが大きい場合は、照会の最適化ルーチンで扱う索引が多くなります。一般に、最適化ルーチンは最適化時間が切れるまでに (結合する各テーブルに対して) 5、6 個の索引を検討します。

ODBC ブロック化 INSERT ステートメント:

このブロック化ステートメントは、System i Access ODBC に複数の行を挿入する場合に使用します。

ブロック化された **INSERT** ステートメントによって、単一の **SQLExecute** 要求で複数の行を挿入することができます。パフォーマンスの面では、テーブルにデータを入れるための最適の方法を提供し、他の方法よりも効率がはるかに良いことも多々あります。

ODBC から実行することのできる INSERT ステートメントの形式は、以下の 3 つです。
- VALUES に定数を使用した INSERT ステートメント
- VALUES にパラメーター・マーカーを使用した INSERT ステートメント
- ブロック化 INSERT ステートメント

VALUES に定数を使用した INSERT ステートメントは、挿入を実行するメソッドで最も効率の悪いものです。それぞれの要求に対して、単一の INSERT ステートメントがサーバーに送られます。サーバーでは INSERT ステートメントの準備、基礎テーブルのオープン、レコードの書き込みが行われます。

例:

```
INSERT INTO TEST.TABLE1 VALUES('ENGINEERING',10,'JONES','BOB')
```

VALUES にパラメーター・マーカーを使用した INSERT ステートメントは、定数を使用したステートメントよりも効率よく実行されます。この形式の INSERT ステートメントでは、そのステートメントを 1 回のみ準備して、次の実行で再利用することができます。また、サーバー上のテーブルをオープンしたままにしておけるので、挿入のたびにファイルをオープンしたりクローズしたりする手間を省くことができます。

例:

```
INSERT INTO TEST.TABLE1 VALUES (?, ?, ?, ?)
```

複数のレコードがクライアントにキャッシュされ、同時に送信される場合、ブロック化された INSERT ステートメントはテーブルの挿入を最も効率よく実行します。ブロック化された INSERT ステートメントの利点は以下のとおりです。

- 複数の行にあるデータが、行ごとに 1 つずつの要求ではなく、1 つの通信要求にまとめて送信される。
- サーバーが、ブロック化 INSERT ステートメントのサポート用にデータベース内に組み込まれた最適化パスをもつ。

例:

```
INSERT INTO TEST.TABLE1 ? ROWS VALUES (?, ?, ?, ?)
```

INSERT ステートメントにはブロック化 INSERT ステートメントを識別する構文も加えられています。 ″? ROWS″ 文節は、追加されたパラメーターがこの INSERT ステートメント用に指定されることを示し、また、そのパラメーターがそのステートメントの実行時に送信される行数を含むことを示します。行数は **SQLSetStmtAttr** API によって指定する必要があります。

注: V5R1 ドライバーでは、システムに ″? ROWS″ 文節を指定する必要はありません。 V4R5 では、PTF (SF64146 および SF64149) を使用して、このサポートを追加していました。

ブロック化 INSERT を C から呼び出す場合の例については、トピック『ブロック挿入およびブロック取り出しの C の例』を参照してください。

関連資料

520 ページの『ブロック挿入およびブロック取り出しの C の例』
ブロック挿入およびブロック取り出しを使用して、System i Access の ODBC アプリケーションのパフォーマンスを改善することができます。

カタログ関数:

カタログ関数は、操作中の DB2 for i5/OS のデータベースに関する情報を戻します。

ODBC の **SQLTables** 要求を処理するために、ライブラリー QSYS のサーバー相互参照ファイル QADBXREF についての論理ファイルが作成されます。 QADBXREF は、データベースによって保持される相互参照情報 (サーバーのディクショナリー機能の一部) 用のデータベース・ファイルです。

以下に、**TableType** の設定ごとに、**SQLTables** に対応する処置を示します。

NULL すべての論理ファイル、物理ファイル、SQL テーブルおよびビューを選択します。

TABLE
すべての物理ファイルと、サーバー・ファイル (相互参照、またはデータ・ディクショナリー) ではない SQL テーブルを選択します。

VIEW すべての論理ファイルと、サーバー・ファイル (相互参照、またはデータ・ディクショナリー) ではない SQL ビューを選択します。

SYSTEM TABLE
すべての物理ファイルと論理ファイル、およびサーバー・ファイルまたはデータ・ディクショナリー・ファイルである SQL ビューを選択します。

TABLE, VIEW
すべての論理ファイルと物理ファイル、およびサーバー・ファイルまたはデータ・ディクショナリー・ファイルではないすべての SQL テーブルとビューを選択します。

非リレーショナル・ファイル (複数のファイル形式を持ったファイル) は選択されません。また、索引ファイル、フラット・ファイル、そして IDDU 定義済みファイルも選択されません。

カタログ関数によって戻される結果セットは、テーブル・タイプ順になっています。システムでは、テーブルとビューというタイプに加え、論理ファイルと物理ファイルという、データ・ソース特有のタイプ識別コードを使用しています。物理タイプはテーブルとして取り扱われ、論理タイプはビューとして取り扱われます。

ODBC の **SQLColumns** 要求を処理するために、QSYS ライブラリーのサーバー相互参照ファイル QADBIFLD についての論理ファイルが作成されます。この論理ファイルは、インデックス以外のすべてのリレーショナル・データベース・ファイルを選択します。QADBIFLD は、データベースによって保持される相互参照情報 (サーバーのディクショナリー機能の一部) 用のデータベース・ファイルです。特に、このファイルには、データベース・ファイルの列とフィールドに関する情報が含まれています。

詳細については、以下を参照してください。

「*SQL 解説書*」の付録に追加情報が記載されています。 i5/OS Information Center にある『DB2 for i5/OS SQL 解説書』のトピックから、上記ブックの HTML オンライン版を表示するか、PDF 版を印刷してください。

関連情報

DB2 for i5/OS SQL 解説書

出口プログラム:

System i Access の ODBC 出口プログラムを呼び出すための要件があります。

出口プログラムは、呼び出し側プログラムから制御が渡されるプログラムです。**出口プログラム**を指定すると、サーバーはその要求を実行する前に、以下の 2 つのパラメーターを出口プログラムに渡します。

- 1 バイトの戻りコード値。
- ユーザー要求に関する情報を含んだ構造。この構造は、それぞれの出口点ごとに異なります。

出口プログラムはこの 2 つのパラメーターによって、要求が許可されたかどうかを判別できます。出口プログラムで戻りコードが X'F0' に設定されている場合は、サーバーは要求を拒否します。戻りコードがそれ以外の値に設定されている場合は、サーバーは要求を許可します。

複数の出口点で同じプログラムを使用できます。プログラムは 2 番目のパラメーター構造の中のデータを見ることによって、現在どの関数が呼び出されているのかを判別できます。

出口プログラムをデータベースの出口点に追加するためには、「登録情報処理」(WRKREGINF) コマンドを使用します。

データベース・サーバーには、以下の 5 つの異なる出口点が定義されています。

QIBM_QZDA_INIT
サーバーの開始時に呼び出されます。

QIBM_QZDA_NDB1
ネイティブ・データベース要求に対して呼び出されます。

QIBM_QZDA_SQL1
SQL 要求に対して呼び出されます。

QIBM_QZDA_SQL2
SQL 要求に対して呼び出されます。

QIBM_QZDA_ROI1

オブジェクト情報の取り出し要求や SQL カタログ関数に対して呼び出されます。

注: この出口点は、V5R1 およびそれ以前のクライアント・アクセス ODBC ドライバーの場合に比べると、呼び出される頻度は低くなります。この出口点を使用する出口プログラムがある場合には、引き続き意図されたとおりに機能するかどうかを検査してください。

例: ユーザー出口プログラム:

以下の例では、プログラミングに関する考慮事項または技法のすべてが示されているわけではありません。これらの例をよく検討してから、System i Access ODBC アプリケーションの設計やコーディングを開始してください。

例: 出口点 QIBM_QZDA_INIT のための ILE C/400 ユーザー出口プログラム:

以下の ILE C/400® プログラムは、特定のユーザーからの要求をリジェクトすることで、System i Access の ODBC セキュリティーを処理します。これをシェルとして使用して、ご使用の稼働環境に合わせた出口プログラムを開発することができます。

```
/*--------------------------------------------------------------------------
 *                @@ss1s@@ Servers - Sample Exit Program
 *
 *    Exit Point Name         : QIBM_QZDA_INIT
 *
 *    Description             : The following ILE C/400 program handles
 *                              ODBC security by rejecting requests from
 *                              certain users.
 *                              It can be used as a shell for developing
 *                              exit programs tailored for your
 *                              operating environment.
 *
 *    Input                   : A 1-byte return code value
 *                                 X'F0' server rejects the request
 *                                 anything else server allows the request
 *                              Structure containing information about the
 *                                request.  The format used by this program
 *                                is ZDAI0100.
 *--------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------
 *     Includes
 *--------------------------------------------------------------------------*/
#include <string.h>                /* string functions                      */
/*--------------------------------------------------------------------------
 *    User Types
 *--------------------------------------------------------------------------*/
typedef struct {             /* Exit Point QIBM_QZDA_INIT format ZDAI0100 */
    char User_profile_name[10];     /* Name of user profile calling server*/
    char Server_identifier[10];     /* database server value (*SQL)       */
    char Exit_format_name[8];       /* User exit format name (ZDAI0100)   */
    long Requested_function;        /* function being preformed (0)       */
} ZDAI0100_fmt_t;

 /*--------------------------------------------------------------------------
   --------------------------------------------------------------------------*/

/*==========================================================================
 *    Start of mainline executable code
 *==========================================================================*/
int main (int argc, char *argv[])
{
    ZDAI0100_fmt_t input;           /* input format record                   */

    /* copy input parm into structure                                       */
```

```
    memcpy(&input, (ZDAI0100_fmt_t *)argv[2], 32);

    if /* if user name is GUEST                                          */
        ( memcmp(input.User_profile_name, "GUEST     ", 10)==0 )
    {
        /* set return code to reject the request.                       */
        memcpy( argv[1], "0", 1);
    }
    else /* else user is someone else                                   */
    {
        /* set return code to allow the request.                        */
        memcpy( argv[1], "1", 1);
    }
}  /* End of mainline executable code                                  */
```

例: 出口点 QIBM_QZDA_INIT のための CL ユーザー出口プログラム:

以下の CL プログラムは、特定のユーザーからの要求をリジェクトすることで、System i Access の ODBC セキュリティーを処理します。これをシェルとして使用して、ご使用の稼働環境に合わせた出口プログラムを開発することができます。

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*               @@ss1s@@ Servers - Sample Exit Program                 */
/*                                                                     */
/*    Exit Point Name         : QIBM_QZDA_INIT                         */
/*                                                                     */
/*    Description             : The following Control Language program */
/*                              handles ODBC security by rejecting     */
/*                              requests from certain users.           */
/*                              It can be used as a shell for developing */
/*                              exit programs tailored for your        */
/*                              operating environment.                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
PGM PARM(&STATUS &REQUEST)

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  Program call parameter declarations                                */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  DCL VAR(&STATUS) TYPE(*CHAR) LEN(1)  /* Accept/Reject indicator       */
  DCL VAR(&REQUEST) TYPE(*CHAR) LEN(34) /* Parameter structure          */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  Parameter declares                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  DCL VAR(&USER) TYPE(*CHAR) LEN(10)  /* User profile name calling server*/
  DCL VAR(&SRVID) TYPE(*CHAR) LEN(10) /* database server value (*SQL)   */
  DCL VAR(&FORMAT) TYPE(*CHAR) LEN(8) /* Format name (ZDAI0100)         */
  DCL VAR(&FUNC) TYPE(*CHAR) LEN(4)   /* function being preformed (0)   */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  Extract the various parameters from the structure                  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  CHGVAR VAR(&USER)   VALUE(%SST(&REQUEST  1 10))
  CHGVAR VAR(&SRVID)  VALUE(%SST(&REQUEST 11 10))
  CHGVAR VAR(&FORMAT) VALUE(%SST(&REQUEST 21  8))
  CHGVAR VAR(&FUNC)   VALUE(%SST(&REQUEST 28  4))

/*--------------------------------------------------------------------
   --------------------------------------------------------------------*/

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  Begin main program                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* set return code to allow the request.                             */
  CHGVAR VAR(&STATUS) VALUE('1')
```

```
  /* if user name is GUEST set return code to reject the request.        */
  IF (&USER *EQ 'GUEST') THEN( +
      CHGVAR VAR(&STATUS) VALUE('0') )

EXIT:
ENDPGM
```

例: 出口点 QIBM_QZDA_SQL1 のための ILE C/400 プログラム:

以下の ILE C/400 プログラムは、GUEST ユーザーのすべての UPDATE 要求をリジェクトします。これをシェルとして使用して、ご使用の稼働環境に合わせた System i Access ODBC 出口プログラムを開発することができます。

```
/*--------------------------------------------------------------------------
 *                @@ss1s@@ Servers - Sample Exit Program
 *
 *    Exit Point Name         : QIBM_QZDA_SQL1
 *
 *    Description             : The following ILE C/400 program will
 *                              reject any UPDATE request for user GUEST.
 *                              It can be used as a shell for developing
 *                              exit programs tailored for your
 *                              operating environment.
 *
 *    Input                   : A 1-byte return code value
 *                                  X'F0' server rejects the request
 *                                  anything else server allows the request
 *                              Structure containing information about the
 *                                request.  The format used by this program
 *                                is ZDAQ0100.
 *--------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------
 *     Includes
 *--------------------------------------------------------------------------*/
#include <string.h>                 /* string functions                     */
#include <stdio.h>                  /* standard IO functions                */
#include <ctype.h>                  /* type conversion functions            */
/*==========================================================================
 *    Start of mainline executable code
 *==========================================================================*/
main(int argc, char *argv[])
{
    long i;
    _Packed struct zdaq0100 {
        char name[10];
        char servid[10];
        char fmtid[8];
        long funcid;
        char stmtname[18];
        char cursname[18];
        char prepopt[2];
        char opnattr[2];
        char pkgname[10];
        char pkglib[10];
        short drdaind;
        char commitf;
        char stmttxt[512];
    } *sptr, stx;

/*--------------------------------------------------------------------------
   ------------------------------------------------------------------------*/
    /* initialize return variable to indicate ok status                    */
    strncpy(argv[1],"1",1);

    /*************************************************************************/
```

```
    /*  Address parameter structure for @@sqll@@ exit program and move local   */
    /*  parameters into local variables.                                       */
    /* (note : this is not necessary to evaluate the arguments passed in). */
    /*************************************************************************/
    sptr = (_Packed struct zdaq0100 *) argv[2];

    strncpy(stx.name, sptr->name, 10);
    strncpy(stx.servid, sptr->servid, 10);
    strncpy(stx.fmtid, sptr->fmtid, 8);
    stx.funcid = sptr->funcid;
    strncpy(stx.stmtname, sptr->stmtname, 18);
    strncpy(stx.cursname, sptr->cursname, 18);
    strncpy(stx.opnattr, sptr->opnattr, 2);
    strncpy(stx.prepopt, sptr->prepopt, 2);
    strncpy(stx.pkglib, sptr->pkglib, 10);
    strncpy(stx.pkgname, sptr->pkgname, 10);
    stx.drdaind = sptr->drdaind;
    stx.commitf = sptr->commitf;
    strncpy(stx.stmttxt, sptr->stmttxt, 512);

    /*************************************************************************/
    /* check for user GUEST and an UPDATE statement                          */
    /* if found return an error                                              */
    /*************************************************************************/
    if (! (strncmp(stx.name, "GUEST     ", 10)) )
    {
        for (i=0; i<6; i++)
            stx.stmttxt[i] = toupper(stx.stmttxt[i]);

        if (! strncmp(stx.stmttxt, "UPDATE", 6) )
            /* Force error out of  @@sqll@@ user exit pgm                    */
            strncpy(argv[1], "0", 1);
        else;
    }
    return;
}  /* End of mainline executable code                                        */

/*--------------------------------------------------------------------------
  --------------------------------------------------------------------------*/

    /* initialize return variable to indicate ok status                      */
    strncpy(argv[1],"1",1);

    /*************************************************************************/
    /*  Address parameter structure for @@sqll@@ exit program and move local   */
    /*  parameters into local variables.                                       */
    /* (note : this is not necessary to evaluate the arguments passed in). */
    /*************************************************************************/
    sptr = (_Packed struct zdaq0100 *) argv[2];

    strncpy(stx.name, sptr->name, 10);
    strncpy(stx.servid, sptr->servid, 10);
    strncpy(stx.fmtid, sptr->fmtid, 8);
    stx.funcid = sptr->funcid;
    strncpy(stx.stmtname, sptr->stmtname, 18);
    strncpy(stx.cursname, sptr->cursname, 18);
    strncpy(stx.opnattr, sptr->opnattr, 2);
    strncpy(stx.prepopt, sptr->prepopt, 2);
    strncpy(stx.pkglib, sptr->pkglib, 10);
    strncpy(stx.pkgname, sptr->pkgname, 10);
    stx.drdaind = sptr->drdaind;
    stx.commitf = sptr->commitf;
    strncpy(stx.stmttxt, sptr->stmttxt, 512);

    /*************************************************************************/
    /* check for user GUEST and an UPDATE statement                          */
    /* if found return an error                                              */
```

```
    /*********************************************************************/
    if (! (strncmp(stx.name, "GUEST     ", 10)) )
    {
        for (i=0; i<6; i++)
            stx.stmttxt[i] = toupper(stx.stmttxt[i]);

        if (! strncmp(stx.stmttxt, "UPDATE", 6) )
            /* Force error out of  @@sqll@@ user exit pgm                    */
            strncpy(argv[1], "0", 1);
        else;
    }
    return;
}  /* End of mainline executable code                                      */
```

例: 出口点 QIBM_QZDA_ROI1 のための ILE C/400 プログラム:

以下の ILE C/400 プログラムは、カタログ関数に対するすべての要求を、QGPL 内の ZDALOG ファイルに記録します。これをシェルとして使用して、ご使用の稼働環境に合わせた System i Access ODBC 出口プログラムを開発することができます。

```
/*--------------------------------------------------------------------------
 *                @@ss1s@@ Servers - Sample Exit Program
 *
 *    Exit Point Name         : QIBM_QZDA_ROI1
 *
 *    Description             : The following ILE C/400 program logs all
 *                              requests for catalog functions to the
 *                              ZDALOG file in QGPL.
 *                              It can be used as a shell for developing
 *                              exit programs tailored for your
 *                              operating environment.
 *
 *    Input                   : A 1-byte return code value
 *                                 X'F0' server rejects the request
 *                                 anything else server allows the request
 *                              Structure containing information about the
 *                                request.  The format used by this program
 *                                is ZDAR0100.
 *
 *    Dependencies            : The log file must be created using the
 *                              following command:
 *                              CRTPF FILE(QGPL/ZDALOG) RCDLEN(132)
 *-------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------
 *     Includes
 *-------------------------------------------------------------------------*/
#include <recio.h>                  /* record IO functions                 */
#include <string.h>                 /* string functions                    */
/*--------------------------------------------------------------------------
 *    User Types
 *-------------------------------------------------------------------------*/
typedef struct {             /* Exit Point QIBM_QZDA_ROI1 format ZDAR0100 */
    char User_profile_name[10];     /* Name of user profile calling server*/
    char Server_identifier[10];     /* database server value (*RTVOBJINF) */
    char Exit_format_name[8];       /* User exit format name (ZDAR0100)   */
    long Requested_function;        /* function being preformed           */
    char Library_name[20];          /* Name of library                    */
    char Database_name[36];         /* Name of relational database        */
    char Package_name[20];          /* Name of package                    */
    char File_name[256];            /* Name of file                       */
    char Member_name[20];           /* Name of member                     */
    char Format_name[20];           /* Name of format                     */
} ZDAR0100_fmt_t;

/*--------------------------------------------------------------------------
   -------------------------------------------------------------------------*/
```

```
/*====================================================================
 *    Start of mainline executable code
 *====================================================================*/
int main (int argc, char *argv[])
{
    _RFILE *file_ptr;               /* pointer to log file               */
    char output_record[132];        /* output log file record            */
    ZDAR0100_fmt_t input;           /* input format record               */
    /* set return code to allow the request.                             */
    memcpy( argv[1], "1", 1);

    /* open the log file for writing to the end of the file              */
    if (( file_ptr = _Ropen("QGPL/ZDALOG", "ar") ) == NULL)
    {
        /* open failed                                                   */
        return;
    }

    /* copy input parm into structure                                    */
    memcpy(&input, (ZDAR0100_fmt_t *)argv[2], 404);

    switch /* Create the output record based on requested function       */
        (input.Requested_function)
    {
        case 0X1800:  /* Retrieve library information                    */
            sprintf(output_record,
              "%10.10s retrieved library %20.20s",
              input.User_profile_name, input.Library_name);
            break;
        case 0X1801:  /* Retrieve relational database information        */
            sprintf(output_record,
              "%10.10s retrieved database %36.36s",
              input.User_profile_name, input.Database_name);
            break;
        case 0X1802:  /* Retrieve @@sqll@@ package information               */
            sprintf(output_record,
              "%10.10s retrieved library %20.20s package %20.20s",
              input.User_profile_name, input.Library_name,
              input.Package_name);
            break;
        case 0X1803:  /* Retrieve @@sqll@@ package statement information      */
            sprintf(output_record,
             "%10.10s retrieved library %20.20s package %20.20s statement info",
              input.User_profile_name, input.Library_name,
              input.Package_name);
            break;
/*--------------------------------------------------------------------
  --------------------------------------------------------------------*/

        case 0X1804:  /* Retrieve file information                      */
            sprintf(output_record,
              "%10.10s retrieved library %20.20s file %40.40s",
              input.User_profile_name, input.Library_name, input.File_name);
            break;
        case 0X1805:  /* Retrieve file member information               */
            sprintf(output_record,
            "%10.10s retrieved library %20.20s member %20.20s file %40.40s",
              input.User_profile_name, input.Library_name,
              input.Member_name, input.File_name);
            break;
        case 0X1806:  /* Retrieve record format information             */
            sprintf(output_record,
            "%10.10s retrieved library %20.20s format %20.20s file %40.40s",
              input.User_profile_name, input.Library_name,
              input.Format_name, input.File_name);
            break;
```

```
        case 0X1807:  /* Retrieve field information                      */
            sprintf(output_record,
              "%10.10s retrieved field info library %20.20s file %40.40s",
              input.User_profile_name, input.Library_name, input.File_name);
            break;
        case 0X1808:  /* Retrieve index information                      */
            sprintf(output_record,
              "%10.10s retrieved index info library %20.20s file %40.40s",
              input.User_profile_name, input.Library_name, input.File_name);
            break;
        case 0X180B:  /* Retrieve special column information             */
            sprintf(output_record,
              "%10.10s retrieved column info library %20.20s file %40.40s",
              input.User_profile_name, input.Library_name, input.File_name);
            break;
        default : /* Unknown requested function                          */
            sprintf(output_record, "Unknown requested function");
            break;
    }  /* end switch statement                                          */

    /* write the output record to the file                             */
    _Rwrite(file_ptr, &output_record, 132);

    /* close the log file                                              */
    _Rclose ( file_ptr );

}  /* End of mainline executable code                                  */
```

出口プログラムのパラメーター・フォーマット:

ネイティブ・データベース用の出口点、およびオブジェクト情報取り出し用の出口点には、QIBM_QZDA_SQL1、QIBM_QZDA_SQL2 の 2 つの形式が定義されています。要求された System i データベース関数のタイプに応じて、これらの形式のうちの 1 つが使用されます。

QIBM_QZDA_SQL2 出口点は、データベース・サーバーに対する特定の SQL 要求時に、出口プログラムを実行するように定義されています。この出口点は、QIBM_QZDA_SQL1 出口点よりも優先されます。QIBM_QZDA_SQL2 出口点にプログラムが登録されると、そのプログラムが呼び出され、QIBM_QZDA_SQL1 に登録されたプログラムは呼び出されません。

出口プログラムを呼び出す関数

- 準備
- オープン
- 実行
- 接続
- パッケージの作成
- パッケージのクリア
- パッケージの削除
- パッケージ情報の戻し
- ストリーム取り出し
- 即時実行
- 準備および記述
- 準備および実行、または準備およびオープン
- オープンおよび取り出し
- 実行またはオープン

ZDAQ0200 形式の出口点 QIBM_QZDA_SQL2 のパラメーター・フィールド:

ZDAQ0200 形式を使用する出口点 QIBM_QZDA_SQL2 で呼び出された System i データベース出口プログラムのパラメーター・フィールドとその説明を以下の表に示します。

表 18. ZDAQ0200 形式の出口点 QIBM_QZDA_SQL2

オフセット		タイプ	フィールド	説明
10 進	16 進			
0	0	CHAR(10)	ユーザー・プロファイル名	サーバーを呼び出すユーザー・プロファイルの名前。
10	A	CHAR(10)	サーバー識別コード	この出口点に対する値は *SQLSRV です。
20	14	CHAR(8)	形式名	使用されるユーザー出口の形式名。QIBM_QZDA_SQL1 の形式名は ZDAQ0100 です。
28	1C	BINARY(4)	要求された関数	実行される関数。 このフィールドの内容は、次のいずれかです。 • X'1800' - 準備 • X'1803' - 準備および記述 • X'1804' - オープン / 記述 • X'1805' - 実行 • X'1806' - 即時実行 • X'1809' - 接続 • X'180C' - ストリーム取り出し • X'180D' - 準備および実行 • X'180E' - オープンおよび取り出し • X'180F' - パッケージの作成 • X'1810' - パッケージのクリア • X'1811' - パッケージの削除 • X'1812' - 実行またはオープン • X'1815' - パッケージ情報の戻し
32	20	CHAR(18)	ステートメント名	準備関数や実行関数に使用するステートメントの名前。
50	32	CHAR(18)	カーソル名	オープン関数に使用されるカーソル名。
68	44	CHAR(2)	準備オプション	準備関数に使用されるオプション。
70	46	CHAR(2)	オープン属性	オープン関数に使用されるオプション。
72	48	CHAR(10)	拡張動的パッケージ名	拡張動的パッケージの名前。
82	52	CHAR(10)	パッケージ・ライブラリー名	拡張動的 SQL パッケージ 用ライブラリーの名前。
92	5C	BINARY(2)	DRDA インディケーター	• 0 - ローカル RDB に接続 • 1 - リモート RDB に接続

表 18. ZDAQ0200 形式の出口点 QIBM_QZDA_SQL2 (続き)

オフセット		タイプ	フィールド	説明
10 進	16 進			
94	5E	CHAR(1)	コミットメント制御レベル	• 'A' - コミット *ALL • 'C' - コミット *CHANGE • 'N' - コミット *NONE • 'S' - コミット *CS (カーソル固定性)
95	5F	CHAR(10)	デフォルト SQL コレクション	System i データベース・サーバーによって使用される、デフォルト SQL コレクションの名前。
105	69	CHAR(129)	予約済み	将来使用されるパラメーターのための予約フィールド
234	EA	BINARY(4)	SQL ステートメントのテキストの長さ	後に続くフィールドに入る SQL ステートメント・テキストの長さ。最大で 32K の長さ。
238	EE	CHAR(*)	SQL ステートメントのテキスト	SQL ステートメント全文。
注: この形式は、ライブラリー QSYSINC にある、ファイル H のメンバー EZDAEP、QRPGSRC、QRPGLESRC、QCBLSRC および QCBLLESRC によって定義されています。				

出口点 QIBM_QZDA_INIT は、サーバー開始時に出口プログラムを実行するように定義されています。プログラムがこの出口点を使用するように定義されている場合は、データベース・サーバーが開始されるたびに、呼び出されます。

ZDAI0100 形式の出口点 QIBM_QZDA_INIT のパラメーター・フィールド:

ZDAI0100 形式を使用する出口点 QIBM_QZDA_INIT で呼び出される System i データベース出口プログラムのパラメーター・フィールドとその説明を以下の表に示します。

表 19. ZDAI0100 形式の出口点 QIBM_QZDA_INIT

オフセット		タイプ	フィールド	説明
10 進	16 進			
0	0	CHAR(10)	ユーザー・プロファイル名	サーバーを呼び出すユーザー・プロファイルの名前。
10	A	CHAR(10)	サーバー識別コード	この出口点に対する値は *SQL です。
20	14	CHAR(8)	形式名	使用されるユーザー出口の形式名。QIBM_QZDA_INIT の形式名は ZDAI0100 です。
28	1C	BINARY(4)	要求された関数	実行される関数。 この出口点に対する有効な値は 0 のみです。
注: この形式は、ライブラリー QSYSINC にある、ファイル H のメンバー EZDAEP、QRPGSRC、QRPGLESRC、QCBLSRC および QCBLLESRC によって定義されています。				

QIBM_QZDA_NDB1 出口点は、データベース・サーバーに対するネイティブ・データベース要求時に、出口プログラムを実行するように定義されています。この出口点に対して、2 つの形式が定義されています。

ZDAD0100 形式を使用する関数

- ソース物理ファイルの作成
- 既存ファイルに基づいた、データベース・ファイルの作成
- データベース・ファイル・メンバーの追加、クリア、削除
- データベース・ファイル一時変更
- データベース・ファイル一時変更削除
- ファイルの削除

注: ZDAD0200 形式は、ライブラリー・リストに対するライブラリーの追加要求が受け取られた時に使用されます。

ZDAD0100 形式の出口点 QIBM_QZDA_NDB1 のパラメーター・フィールド:

ZDAD0100 形式を使用する出口点 QIBM_QZDA_NDB1 で呼び出された System i データベース出口プログラムの、パラメーター・フィールドとその説明を以下の表に示します。

表 20. ZDAD0100 形式の出口点 QIBM_QZDA_NDB1

オフセット		タイプ	フィールド	説明
10 進	**16 進**			
0	0	CHAR(10)	ユーザー・プロファイル名	サーバーを呼び出すユーザー・プロファイルの名前。
10	A	CHAR(10)	サーバー識別コード	この出口点に対する値は *NDB です。
20	14	CHAR(8)	形式名	使用されるユーザー出口の形式名。 これ以降の関数についての形式名は ZDAD0100 です。
28	1C	BINARY(4)	要求された関数	実行される関数。 このフィールドの内容は、次のいずれかです。 • **X'1800'** - ソース物理ファイルの作成 • **X'1801'** - 既存ファイルに基づいた、データベース・ファイルの作成 • **X'1802'** - データベース・ファイル・メンバーの追加 • **X'1803'** - データベース・ファイル・メンバーの消去 • **X'1804'** - データベース・ファイル・メンバーの削除 • **X'1805'** - データベース・ファイル一時変更 • **X'1806'** - データベース・ファイル一時変更削除 • **X'1807'** - 保管ファイルの作成 • **X'1808'** - 保管ファイルの消去 • **X'1809'** - ファイルの削除

表 20. ZDAD0100 形式の出口点 QIBM_QZDA_NDB1 (続き)

オフセット		タイプ	フィールド	説明
10 進	16 進			
32	20	CHAR(128)	ファイル名	要求された関数に使用されるファイルの名前。
160	A0	CHAR(10)	ライブラリー名	ファイルが含まれているライブラリーの名前。
170	AA	CHAR(10)	メンバー名	追加、消去、削除するメンバーの名前。
180	B4	CHAR(10)	権限	作成されたファイルに対する権限。
190	BE	CHAR(128)	ファイル名による	既存ファイルに基づいてファイルを作成するときに使用されるファイルの名前。
318	13E	CHAR(10)	ライブラリー名による	基本となるファイルを含むライブラリー名。
328	148	CHAR(10)	変更ファイル名	一時変更されるファイルの名前。
338	152	CHAR(10)	一時変更ライブラリー名	一時変更されるファイルを含むライブラリーの名前。
348	15C	CHAR(10)	一時変更メンバー名	一時変更されるメンバーの名前。
注: この形式は、ライブラリー QSYSINC にある、ファイル H のメンバー EZDAEP、QRPGSRC、QRPGLESRC、QCBLSRC および QCBLLESRC によって定義されています。				

ZDAD0200 形式の出口点 QIBM_QZDA_NDB1 のパラメーター・フィールド:

ZDAD0200 形式を使用して出口点 QIBM_QZDA_NDB1 で呼び出される System i データベース出口プログラムのパラメーター・フィールドとその説明を、以下の表で紹介します。

表 21. ZDAD0200 形式の出口点 QIBM_QZDA_NDB1

オフセット		タイプ	フィールド	説明
10 進	16 進			
0	0	CHAR(10)	ユーザー・プロファイル名	サーバーを呼び出すユーザー・プロファイルの名前。
10	A	CHAR(10)	サーバー識別コード	この出口点に対する値は *NDB です。
20	14	CHAR(8)	形式名	使用されるユーザー出口の形式名。ライブラリー・リストへの追加の関数の形式の名前は、ZDAD0200 です。
28	1C	BINARY(4)	要求された関数	実行される関数。 • **X'180C'** - ライブラリー・リストの追加
32	20	BINARY(4)	ライブラリー数	(次のフィールドの) ライブラリーの数。
36	24	CHAR(10)	ライブラリー名	それぞれのライブラリーの名前。
注: この形式は、ライブラリー QSYSINC にある、ファイル H のメンバー EZDAEP、QRPGSRC、QRPGLESRC、QCBLSRC および QCBLLESRC によって定義されています。				

QIBM_QZDA_SQL1 出口点は、データベース・サーバーに対する、特定の SQL 要求時に出口プログラムを実行するために定義されています。この出口点に対しては、1 つの形式のみ定義されています。

ZDAD0200 形式を使用する関数

- 準備
- オープン
- 実行
- 接続
- パッケージの作成
- パッケージのクリア
- パッケージの削除
- 即時実行
- 準備および記述
- 準備および実行、または準備およびオープン
- オープンおよび取り出し
- 実行またはオープン

ZDAQ0100 形式の出口点 QIBM_QZDA_SQL1 のパラメーター・フィールド:

ZDAQ0100 形式を使用する出口点 QIBM_QZDA_SQL1 で呼び出される System i データベース出口プログラムのパラメーター・フィールドとその説明を以下の表に示します。

表 22. ZDAQ0100 形式の出口点 QIBM_QZDA_SQL1

オフセット		タイプ	フィールド	説明
10 進	16 進			
0	0	CHAR(10)	ユーザー・プロファイル名	サーバーを呼び出すユーザー・プロファイルの名前。
10	A	CHAR(10)	サーバー識別コード	この出口点に対する値は、*SQLSRV です。
20	14	CHAR(8)	形式名	使用されるユーザー出口の形式名。QIBM_QZDA_SQL1 の形式名は ZDAQ0100 です。

表 22. ZDAQ0100 形式の出口点 QIBM_QZDA_SQL1 (続き)

オフセット		タイプ	フィールド	説明
10 進	16 進			
28	1C	BINARY(4)	要求された関数	実行される関数。 このフィールドの内容は、次のいずれかです。 • **X'1800'** - 準備 • **X'1803'** - 準備および記述 • **X'1804'** - オープン / 記述 • **X'1805'** - 実行 • **X'1806'** - 即時実行 • **X'1809'** - 接続 • **X'180D'** - 準備および実行、または準備およびオープン • **X'180E'** - オープンおよび取り出し • **X'180F'** - パッケージの作成 • **X'1810'** - パッケージのクリア • **X'1811'** - パッケージの削除 • **X'1812'** - 実行またはオープン • **X'1815'** - パッケージ情報の戻し
32	20	CHAR(18)	ステートメント名	準備関数や実行関数に使用するステートメントの名前。
50	32	CHAR(18)	カーソル名	オープン関数に使用されるカーソル名。
68	44	CHAR(2)	準備オプション	準備関数に使用されるオプション。
70	46	CHAR(2)	オープン属性	オープン関数に使用されるオプション。
72	48	CHAR(10)	拡張動的パッケージ名	拡張動的 SQL パッケージの名前。
82	52	CHAR(10)	パッケージ・ライブラリー名	拡張動的 SQL パッケージ 用ライブラリーの名前。
92	5C	BINARY(2)	DRDA インディケーター	• **0** - ローカル RDB に接続 • **1** - リモート RDB に接続
94	5E	CHAR(1)	コミットメント制御レベル	• **'A'** - コミット *ALL • **'C'** - コミット *CHANGE • **'N'** - コミット *NONE • **'S'** - コミット *CS (カーソル固定性)
95	5F	CHAR(512)	SQL ステートメントのテキストの最初の 512 バイト	SQL ステートメントの最初の 512 バイト。
注: この形式は、ライブラリー QSYSINC にある、ファイル H のメンバー EZDAEP、QRPGSRC、QRPGLESRC、QCBLSRC および QCBLLESRC によって定義されています。				

QIBM_QZDA_ROI1 出口点は、データベース・サーバーに対する特定のオブジェクト情報の取り出し要求時に出口プログラムを実行するように定義されています。また、この出口点は SQL カタログ関数にも使用されています。

この出口点には、2 つの形式が定義されています。

ZDAR0100 形式は、次のオブジェクトの情報を取り出す際に使用されます。

- フィールド (または、列)
- ファイル (または、テーブル)
- ファイル・メンバー
- インデックス
- ライブラリー (または、コレクション)
- レコード様式
- リレーショナル・データベース (RDB)
- 特殊列
- SQL パッケージ
- SQL パッケージ・ステートメント

ZDAR0200 形式は、次のオブジェクトの情報を取り出す際に使用されます。

- 外部キー
- 基本キー

ZDAR0100 形式の出口点 QIBM_QZDA_ROI1 のパラメーター・フィールド:

ZDAR0100 形式を使用する出口点 QIBM_QZDA_ROI1 で呼び出された System i データベース出口プログラムの、パラメーター・フィールドとその説明を以下の表に示します。

表 23. ZDAR0100 形式の出口点 QIBM_QZDA_ROI1

オフセット		タイプ	フィールド	説明
10 進	16 進			
0	0	CHAR(10)	ユーザー・プロファイル名	サーバーを呼び出すユーザー・プロファイルの名前。
10	A	CHAR(10)	サーバー識別コード	データベース・サーバーに対する値は、*RTVOBJINF です。
20	14	CHAR(8)	形式名	使用されるユーザー出口の形式名。これ以降の関数についての形式名は ZDAR0100 です。

表 23. ZDAR0100 形式の出口点 QIBM_QZDA_ROI1 (続き)

オフセット		タイプ	フィールド	説明
10 進	16 進			
28	1C	BINARY(4)	要求された関数	実行される関数。 このフィールドの内容は、次のいずれかです。 • **X'1800'** - ライブラリー情報の取り出し • **X'1801'** - リレーショナル・データベース情報の取り出し • **X'1802'** - SQL パッケージ情報の取り出し • **X'1803'** - SQL パッケージのステートメント情報の取り出し • **X'1804'** - ファイル情報の取り出し • **X'1805'** - ファイル・メンバー情報の取り出し • **X'1806'** - レコード様式情報の取り出し • **X'1807'** - フィールド情報の取り出し • **X'1808'** - インデックス情報の取り出し • **X'180B'** - 特殊列情報の取り出し
32	20	CHAR(20)	ライブラリー名	ライブラリー、パッケージ、パッケージ・ステートメント、ファイル、メンバー、レコード様式、フィールド、インデックス、および特殊列の情報を取り出すときに使用されるライブラリーや検索パターン。
52	34	CHAR(36)	リレーショナル・データベース名	RDB の情報を取り出すのに使用されるリレーショナル・データベース名や検索パターン。
88	58	CHAR(20)	パッケージ名	パッケージまたはパッケージ・ステートメント情報を取り出すために使用されるパッケージ名や検索パターン。
108	6C	CHAR(256)	ファイル名 (SQL エイリアス名)	ファイル、メンバー、レコード様式、フィールド、インデックス、または特殊列情報を取り出すために使用されるファイル名やサーチ検索パターン。
364	16C	CHAR(20)	メンバー名	ファイル・メンバー情報を取り出すために使用される、メンバー名や検索パターン。
384	180	CHAR(20)	形式名	レコード様式情報を取り出すために使用されるフォーマット名や検索パターン。
注: この形式は、ライブラリー QSYSINC にある、ファイル H のメンバー EZDAEP、QRPGSRC、QRPGLESRC、QCBLSRC および QCBLLESRC によって定義されています。				

ZDAR0200 形式の出口点 QIBM_QZDA_ROI1 のパラメーター・フィールド:

ZDAR0200 形式を使用する出口点 QIBM_QZDA_ROI1 で呼び出された System i データベース出口プログラムの、パラメーター・フィールドとその説明を以下の表に示します。

表 24. ZDAR0200 形式の出口点 QIBM_QZDA_ROI1

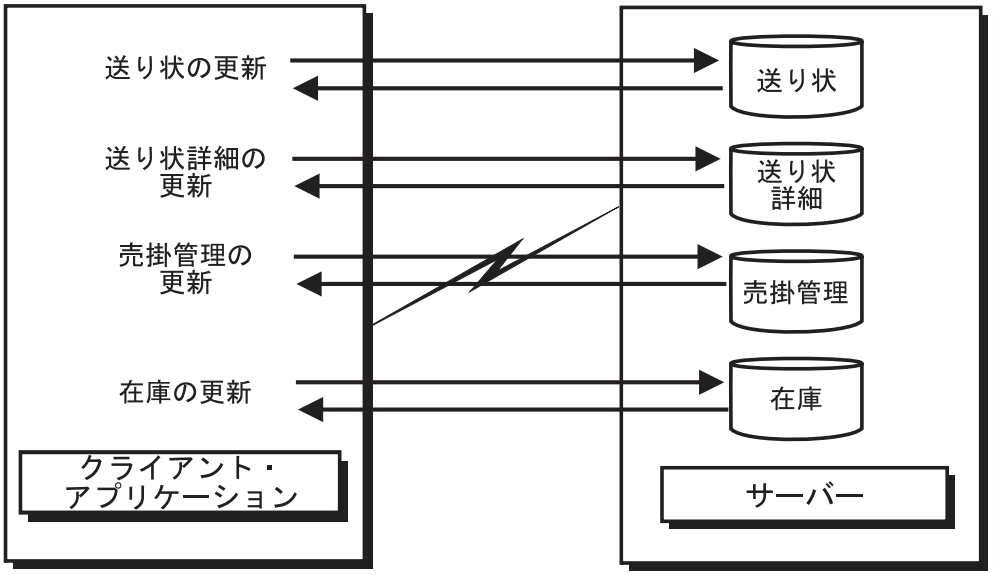
オフセット		タイプ	フィールド	説明
10 進	16 進			
0	0	CHAR(10)	ユーザー・プロファイル名	サーバーを呼び出すユーザー・プロファイルの名前。
10	A	CHAR(10)	サーバー識別コード	データベース・サーバーに対する値は、*RTVOBJINF です。
20	14	CHAR(8)	形式名	使用されるユーザー出口の形式名。これ以降の関数についての形式名は ZDAR0200 です。
28	1C	BINARY(4)	要求された関数	実行される関数。 このフィールドの内容は、次のいずれかです。 • **X'1809'** - 外部キー情報の取り出し • **X'180A'** - 基本キー情報の取り出し
32	20	CHAR(10)	基本キー・テーブル・ライブラリー名	基本キーや外部キーの情報を取り出す際に使用する基本キー・テーブルが含まれているライブラリー名。
42	2A	CHAR(128)	基本キー・テーブル名 (エイリアス名)	基本キー、または外部キーの情報を取り出す際に使用する基本キーが含まれているテーブル名。
170	AA	CHAR(10)	外部キー・テーブル・ライブラリー名	外部キーの情報を取り出す際に使用する外部キー・テーブルが含まれるライブラリー名。
180	64	CHAR(128)	外部キー・テーブル名 (エイリアス名)	外部キーの情報を取り出す際に使用する外部キーが含まれているテーブル名。
注: この形式は、ライブラリー QSYSINC にある、ファイル H のメンバー EZDAEP、QRPGSRC、QRPGLESRC、QCBLSRC および QCBLLESRC によって定義されています。				

ストアード・プロシージャー:

ストアード・プロシージャーは、System i データベース・トランザクション処理でサポートされています。

ストアード・プロシージャーは、パフォーマンス、トランザクションの保全性、およびセキュリティーを高めることから、クライアント/サーバー・アプリケーション、特にオンライン・トランザクション処理 (OLTP) の分野で、ごく一般的に使用されています。ストアード・プロシージャーの例で使用されている特定の SQL コマンドについての情報は、i5/OS Information Center にある『DB2 for i5/OS の SQL 解説書』のトピック集を参照してください。

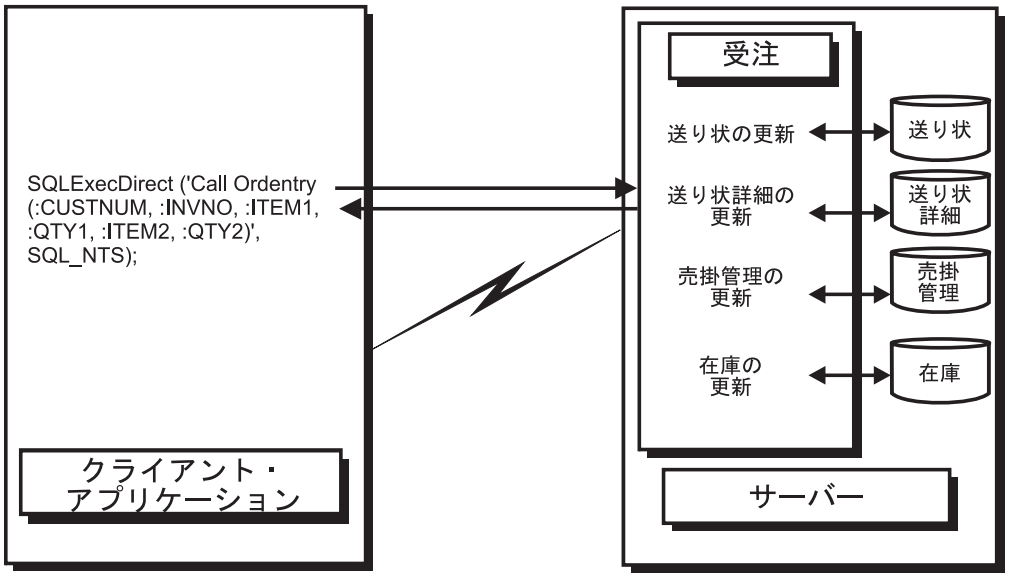
次の図では、1 つのトランザクションが 4 つの別々の I/O オペレーションから構成され、そのそれぞれが SQL ステートメントの処理を要求しているアプリケーションを示しています。この図で示しているように、このアプリケーションでは、サーバーとクライアントとの間で少なくとも 8 つのメッセージのやりとりが必要とされます。これによって、特に、通信速度が遅い場合 (例えば、ダイヤル呼び出し回線を介する場合) あるいは、接続のターンアラウンド・タイムが遅い場合 (例えば、サテライト・リンクを介する場合) に、かなりのオーバーヘッドが発生する可能性があります。



ストアード・プロシージャーを使用しない
クライアント/サーバー・アプリケーション

RV3W347-0

次の図は、同じトランザクションをサーバー上のストアード・プロシージャーによって実行したものです。この図で示されているように、通信量は、一対のメッセージにまで減少されています。このほかにも、利点はあります。例えば、プロシージャーでは、絶対に必要なデータ (長い列からの数個の文字) のみを送り返すように調整することも可能です。ストアード・プロシージャー用の DB2 は、任意の System i プログラムにすることが可能で、データ・アクセスに SQL を使用する必要はありません。



ストアード・プロシージャーを使用した
クライアント/サーバー・アプリケーション

RV3W348-0

関連タスク

519 ページの『ストアード・プロシージャーの呼び出し』
System i Access ODBC アプリケーションのパフォーマンスと機能を高めるためにはストアード・プロシージャーを使用します。

関連資料

624 ページの『ODBC プログラム例』
System i Access の照会およびストアード・プロシージャーについて、ODBC のプログラミング例を使って説明します。

関連情報

DB2 for i5/OS SQL 解説書

ストアード・プロシージャーの結果セット:

System i SQL ストアード・プロシージャーの結果セットをスクロールできます。

V5R3 System i バージョンでの実行時に、スクロール可能な SQL ストアード・プロシージャーの結果セットを、アプリケーションで使用できるようになりました。このサポートを活用するには、以下の 2 つの変更を行ってください。

1. スクロール可能として定義されたカーソルを持つストアード・プロシージャーを作成します。
   a. これは、CREATE PROCEDURE に SCROLL キーワードを追加することで実行できます。 以下の 2 つの例では、1 つ目のストアード・プロシージャーはスクロール可能結果セットを戻しますが、2 つ目は戻しません。
      - ```
        CREATE PROCEDURE MYLIB.SCROLLSP ( ) RESULT SETS 1
        LANGUAGE SQL
        sqlproc: begin
        DECLARE CUR1 SCROLL CURSOR FOR
        SELECT * FROM QIWS.QCUSTCDT;
        OPEN CUR1;
        SET RESULT SETS CURSOR CUR1;
        end
        ```
      - ```
        CREATE PROCEDURE MYLIB.NOSCROLLSP ( ) RESULT SETS 1
        LANGUAGE SQL
        sqlproc: begin
        DECLARE CUR1 CURSOR FOR
        SELECT * FROM QIWS.QCUSTCDT;
        OPEN CUR1;
        SET RESULT SETS CURSOR CUR1;
        end
        ```
2. ODBC を使用してアプリケーションをコード化し、両方向スクロール・カーソル・タイプを要求します。
   a. SQLSetStmtAttr API を呼び出します。
   b. SQL_ATTR_CURSOR_TYPE オプションを SQL_CURSOR_DYNAMIC に設定します。

   両方向スクロール・カーソルを指定していないストアード・プロシージャーを使用して逆方向へのスクロールが試行された場合、複数の問題が発生する可能性があります。ほとんどの場合では、スクロールが無効であることを示すエラーがサーバーから戻されますが、誤ったデータが戻される場合もあります。

   ストアード・プロシージャーが複数の結果セットを戻す場合でも、単一のカーソル・タイプのみを使用することができます。 2 つ目の結果セット用に異なるカーソル・タイプが指定されている場合、ODBC はエラーを戻すか、そのカーソル・タイプを無視します。スクロール可能結果セットを結果セットの 1 つとして使用するには、上記のように、アプリケーションでそのカーソル・タイプをスクロール可能に設定する必要があります。

ストアード・プロシージャーで更新可能カーソルの使用を試行しても無視されます。ストアード・プロシージャーの結果セットは、読み取り専用です。

カーソル・センシティビティーは、ストアード・プロシージャーの結果セットでは保持されない場合があります。カーソル・センシティビティーは、プロシージャーの作成時にサーバー・カーソルが定義された方法で制御されます。

例: ストアード・プロシージャー:

System i データベース・ストアード・プロシージャーの例は、以下を参照してください。

例: SQL ストアード・プロシージャーと ODBC による CL コマンドの実行:

ストアード・プロシージャー・サポートは、SQL の CALL ステートメントを使用して System i 制御言語 (CL) コマンドを実行する手段を提供します。

以下の状況で、CL コマンドを使用することができます。

- ファイルに対する一時変更を行う場合
- デバッグを開始するとき
- 他のコマンドを使用することによって、それに続く SQL ステートメントのパフォーマンスに影響を与えることができる場合

CL コマンドを処理するプログラムを呼び出す CALL ステートメントを使用して、System i CL コマンドを実行するケースを、以下の例で説明します。このプログラム (ライブラリー QSYS の QCMDEXC) には以下の 2 つのパラメーターがあります。

1. 実行するコマンド・テキストを含むストリング
2. コマンド・テキスト長を示す、10 進数 (15,5) フィールド

コマンドを正確に解釈させるために、必ずこれらの属性をパラメーターに含めます。CALL ステートメントの 2 番目のパラメーターには、10 進数 (15,5) のフィールドのすべての桁に明示的に文字を指定する必要があります。

以下の例は、PC 上の C のプログラムが 65 文字 (組み込みブランクを含む) の OVRDBF コマンドを実行しているものです。 OVRDBF コマンドのテキストは以下のとおりです。

```
OVRDBF FILE(TESTER) TOFILE(JMBLIB/TESTER) MBR(NO2) OVRSCOPE(*JOB)
```

ODBC の API を使用して、このコマンドを実行する場合のコードは、以下のとおりです。

```
 HSTMT hstmt;
SQLCHAR stmt[301];

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
strcpy(stmt,"CALL QSYS.QCMDEXC('OVRDBF FILE(TESTER) TOFILE(MYLIB/");
strcat(stmt,"TESTER) MBR(NO2) OVRSCOPE(*JOB)',0000000064.00000)");
rc = SQLExecDirect(hstmt, stmt, SQL_NTS);
```

これで、MYLIB/TESTER ファイルに対して実行されるステートメントは、最初のメンバーではなく、2 番目のメンバーを参照するようになります。

データベース・サーバー・ジョブに対して実行できる有用な CL コマンドには他に STRDBG コマンドがあります。ただし、このコマンドを呼び出すためにストアード・プロシージャーを呼び出す必要はありません。 DSN セットアップ GUI の「診断」タブには、接続試行時に STRDBG コマンドを自動的に実行するオプションがあります。

関連概念

538 ページの『ODBC API のインプリメンテーションに関する事項』
System i Access ODBC API を使用したインプリメンテーション時の問題について確認します。

例: Visual Basic からの、戻り値を伴うストアード・プロシージャーの呼び出し:

System i のストアード・プロシージャーを呼び出して、Visual Basic 変数に戻り値を取り込む方法について、以下の Visual Basic ソース・コードの例で説明します。

Visual Basic は、DLL の中にある外部関数を呼び出すことができます。すべての ODBC ドライバーは、DLL であるため、Visual Basic を使用して、ODBC API を直接コーディングすることができます。 ODBC API に直接コーディングすることによって、Visual Basic アプリケーションで、 System i のストアード・プロシージャーを呼び出し、結果値を戻すことができます。詳しくは、 519 ページの『ODBC API への直接コーディング』を参照してください。

```
'************************************************************************
'*                                                                      *
'* Because of the way Visual Basic stores and manages the String data  *
'* type, it is recommended that you use an array of Byte data type     *
'* instead of a String variable on the SQLBindParameter API.           *
'*                                                                      *
'************************************************************************

Dim sTemp As String
Custnum As Integer
Dim abCustname(34) As Byte
Dim abAddress(34) As Byte
Dim abCity(24) As Byte
Dim abState(1) As Byte
Dim abPhone(14) As Byte
Dim abStatus As Byte
Dim RC As Integer
Dim nullx As Long        'Used to pass null pointer, not pointer to null
Dim lpSQL_NTS As Long    'Used to pass far pointer to SQL_NTS
Static link(7) As Long   'Used as an array of long pointers to the size
                         'each parameter which will be bound


 '************************************************************************
 '*                                                                      *
 '* Initialize the variables needed on the API calls                     *
 '*                                                                      *
 '************************************************************************

link(1) = 6
link(2) = Ubound(abCustname) +1
link(3) = Ubound(abAddress) +1
link(4) = Ubound(abCity) +1
link(5) = Ubound(abState) +1
link(6) = Ubound(abPhone) +1
link(7) = 1

RC = 0
nullx = 0
lpSQL_NTS = SQL_NTS      ' -3 means passed as sz string

'************************************************************************
'*                                                                      *
'* Create a System i procedure. This will define the                    *
'* procedure's name, parameters, and how each parameter is passed.     *
'* Note: This information is stored in the server catalog tables and   *
'* and only needs to be executed one time for the life of the stored   *
'* procedure.  It normally would not be run in the client application. *
```

```
'*                                                                     *
'***********************************************************************

sTemp = "Create Procedure Storedp2 (:Custnum in integer, "
sTemp = sTemp & ":Custname out char(35), :Address out char(35),"
sTemp = sTemp & ":City out char(25), :State out char(2),"
sTemp = sTemp & ":Phone out char(15), :Status out char(1))
sTemp = sTemp & "(External name rastest.storedp2 language cobol General)"

RC = SQLExecDirect(Connection.hstmt, sTemp, Len(sTemp))

'Ignore error assuming that any error would be from procedure already
'created.



'***********************************************************************
'*                                                                     *
'* Prepare the call of the procedure to the system.                   *
'* For best performance, prepare the statement only one time and
'* execute many times.
'*                                                                     *
'***********************************************************************

sTemp = "Call storedp2(?, ?, ?, ?, ?, ?, ?)"
RC = SQLPrepare(Connection.hstmt, sTemp, Len(sTemp))

If (RC <> SQL_SUCCESS) Then
   DescribeError Connection.hdbc, Connection.hstmt
   frmMain.Status.Caption = "Error on SQL_Prepare " & RTrim$(Tag)
End If

'***********************************************************************
'*                                                                     *
'* Bind all of the columns passed to the stored procedure. This will   *
'* set up the variable's data type, input/output characteristics,      *
'* length, and initial value.                                          *
'* The SQLDescribeParam API can optionally be used to retrieve the
'* parameter types.
'*                                                                     *
'* To properly pass an array of byte to a stored procedure and receive *
'* an output value back, you must pass the first byte ByRef.           *
'*                                                                     *
'***********************************************************************

RC = SQLBindParameter(Connection.hstmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT, _
 SQL_NUMERIC, 6, 0, Custnum, 6, link(1))

RC = SQLBindParameter(Connection.hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
 SQL_CHAR, 35, 0, abCustname(0), UBound(abCustname)+1, link(2))
RC = SQLBindParameter(Connection.hstmt, 3, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
 SQL_CHAR, 35, 0, abAddress(0), UBound(abAddress)+1, link(3))
RC = SQLBindParameter(Connection.hstmt, 4, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
 SQL_CHAR, 25, 0, abCity(0), UBound(abCity)+1, link(4))
RC = SQLBindParameter(Connection.hstmt, 5, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
 SQL_CHAR, 2, 0, abState(0), UBound(abState)+1, link(5))
RC = SQLBindParameter(Connection.hstmt, 6, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
 SQL_CHAR, 15, 0, abPhone(0), UBound(abPhone)+1, link(6))
RC = SQLBindParameter(Connection.hstmt, 7, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
 SQL_CHAR, 1, 0, abStatus, 1, link(7))


'***********************************************************************
'*                                                                     *
'* The Prepare and Bind only needs to be execute once.  The Stored
'* procedure can now be called multiple times by just changing the data
'*                                                                     *
```

```
'************************************************************************
Do While

'************************************************************************
'* Read in a customer number                                            *
'*                                                                      *
'************************************************************************

Custnum = Val(input.text)


'************************************************************************
'*                                                                      *
'* Execute the call of the procedure to the system.                    *
'*                                                                      *
'************************************************************************

RC = SQLExecute(Connection.hstmt)
frmMain.Status.Caption = "Ran Stored Proc" & RTrim$(Tag)

If (RC <> SQL_SUCCESS) Then
   DescribeError Connection.hdbc, Connection.hstmt
   frmMain.Status.Caption = "Error on Stored Proc Execute " & RTrim$(Tag
End If

'************************************************************************
'*                                                                      *
'* Set text labels to display the output data                           *
'* You must convert the array of Byte back to a String
'*                                                                      *
'************************************************************************


lblCustname = StrConv(abCustname(), vbUnicode)
lblAddress = StrConv(abAddress(), vbUnicode)
lblCity = StrConv(abCity(), vbUnicode)
lblState = StrConv(abState(), vbUnicode)
lblPhone = StrConv(abPhone(), vbUnicode)
lblStatus = StrConv(abStatus(), vbUnicode)

Loop
```

例: Visual Basic を使用した System i ストアード・プロシージャーの呼び出し:

以下の Visual Basic プログラミングは、準備される System i ストアード・プロシージャー呼び出しの例です。

以下の 2 つのステートメントが示されています。

1. ストアード・プロシージャー作成のためのステートメント
2. 呼び出しの準備のためのステートメント

ストアード・プロシージャーを、1 回のみ作成します。統合された i5/OS アプリケーションのほか、ODBC アプリケーションでも、そのストアード・プロシージャーが提供する定義を利用することができます。

Visual Basic が String データ・タイプを保管して、管理する方法を考えると、次のパラメーター・タイプに対しては、String 変数ではなく Byte データ・タイプの配列を使用することをお勧めします。

- 入出力パラメーター
- 出力パラメーター
- 2 進データを含む (標準 ANSI 文字ではなく) 任意のパラメーター

- 設定は 1 回であるが、複数回参照される可変のアドレスを持つ任意の入力パラメーター

最後のケースは、アプリケーションが、それぞれの呼び出しの間に **Parm1** を変更しながら **SQLExecute** への呼び出しを何度も行う場合に該当します。以下の Visual Basic 関数は、バイトのストリングと配列の変換の際に役立ちます。

```
Public Sub Byte2String(InByte() As Byte, OutString As String)
  'Convert array of byte to string
   OutString = StrConv(InByte(), vbUnicode)
End Sub

Public Function String2Byte(InString As String, OutByte() As Byte) As Boolean
    'vb byte-array / string coercion assumes Unicode string
    'so must convert String to Byte one character at a time
    'or by direct memory access
    'This function assumes Lower Bound of array is 0

    Dim I As Integer
    Dim SizeOutByte As Integer
    Dim SizeInString As Integer

    SizeOutByte = UBound(OutByte) + 1
    SizeInString = Len(InString)

    'Verify sizes if desired

    'Convert the string
    For I = 0 To SizeInString - 1
      OutByte(I) = AscB(Mid(InString, I + 1, 1))
    Next I
    'If size byte array > len of string pad with Nulls for szString
    If SizeOutByte > SizeInString Then           'Pad with Nulls
       For I = SizeInString To UBound(OutByte)
          OutByte(I) = 0
       Next I
    End If

   String2Byte = True
End Function

Public Sub ViewByteArray(Data() As Byte, Title As String)
   'Display message box showing hex values of byte array

   Dim S As String
   Dim I As Integer
   On Error GoTo VBANext

   S = "Length: " & Str(UBound(Data) - LBound(Data) + 1) & "  Data (in hex):"
   For I = LBound(Data) To UBound(Data)
      If (I Mod 8) = 0 Then
         S = S & "  "         'add extra space every 8th byte
      End If
      S = S & Hex(Data(I)) & " "
   VBANext:
   Next I
   MsgBox S, , Title

End Sub
```

例: CL コマンド・ストアード・プロシージャーの呼び出し:

ストアード・プロシージャーを使用して、System i コマンドを実行することができます。ここに記されている 2 つの例は、ODBC プログラムに適用されます。

コマンドを実行するには、コマンド実行 (QCMDEXC) を呼び出すだけです。この処理は比較的シンプルですが、長さパラメーターにゼロをセットすることを忘れないでください。リモート・コマンド API も代替として使用することができます。

最初の例では、SQL を実行しているジョブ (この例の場合は、サーバー・ジョブ) のジョブ・ログにデータを書き込む、強力な SQL トレース機能を使えるようにします。

2 番目の例では、マルチ・メンバー・ファイル処理に関する SQL の機能の制限を広げます。CREATE TABLE コマンドでは、通常マルチ・メンバー・ファイルを作成することはできません。しかし次の例では、DDS で作成したファイルの最初のメンバー以外に対する、ODBC を使用したアクセス方法が示されています。

```
Dim hStmt                   As Long

rc = SQLAllocHandle(SQL_HANDLE_STMT, ghDbc, hStmt)
If rc <> SQL_SUCCESS Then
 Call DspSQLError(SQL_HANDLE_DBC, ghDbc, "Problem: Allocating Debug Statement Handle")
End If

' Note that the string within single quotes 'STRDBG UPDPROD(*YES)' is exactly 20 bytes
cmd = "call qsys.qcmdexc('STRDBG UPDPROD(*YES)',0000000020.00000)"

' Put the system job in debug mode
rc = SQLExecDirect(hStmt, cmd, SQL_NTS)
If rc <> SQL_SUCCESS Then
 Call DspSQLError(SQL_HANDLE_STMT, hStmt, "Problem: Start Debug")
End If

rc = SQLAllocHandle(SQL_HANDLE_STMT, ghDbc, ovrhstmt)
If rc <> SQL_SUCCESS Then
 Call DspSQLError(SQL_HANDLE_DBC, ghDbc, "Problem: Allocating Override Statement Handle")
End If

' Note that the string within single quotes 'OVRDBF FILE(BRANCH)... OVRSCOPE(*JOB)'
   is exactly 68 bytes
  cmd = "call qsys.qcmdexc('OVRDBF FILE(BRANCH) TOFILE(HOALIB/BRANCH) MBR(FRANCE)
                                                OVRSCOPE(*JOB)',0000000068.00000)"

' Override the System i file to point to the 'france' member
rc = SQLExecDirect(hStmt, cmd, SQL_NTS)
If rc <> SQL_SUCCESS Then
 Call DspSQLError(SQL_HANDLE_STMT, hStmt, "File Override")
End If
```

ヒント: System i ストアード・プロシージャーの実行と呼び出し:

System i ストアード・プロシージャーの実行と呼び出しに関するヒントです。

System i ストアード・プロシージャーの実行

ODBC は、ストアード・プロシージャーを呼び出す標準インターフェースを提供しています。ストアード・プロシージャーの実施方法は、データベースによって大きく異なってきます。 System i ストアード・プロシージャーの実行方法における推奨例を、この簡単な例で説明します。

1. **プロシージャー作成**ステートメントをストアード・プロシージャー用に設定し、ストアード・プロシージャーを作成します。ストアード・プロシージャーは、1 回作成するのみであり、ODBC を介して作成する必要はありません。統合された i5/OS アプリケーションのほか、すべての ODBC アプリケーションで、ストアード・プロシージャーが提供する定義を利用することができます。
2. ストアード・プロシージャー呼び出しを準備します。

3. それぞれのパラメーターが、プロシージャーへの入力に使用されるのか、プロシージャーからの出力に使用されるのか、または入出力のいずれに使用されるのかを示して、プロシージャーのパラメーターをバインドします。
4. ストアード・プロシージャーを呼び出します。

Visual Basic を使用した System i ストアード・プロシージャーの呼び出し

SQLBindParameter 関数をコーディングする際には注意してください。列 (**SQLBindCol**) またはパラメーター (**SQLBindParameter**) をバインドしている場合は、Visual Basic ストリングをバッファーとして使用しないでください。代わりに、バイト配列を使用してください。バイト配列は、ストリングとは異なり、メモリー内を移動しません。詳しくは、621 ページの『例: Visual Basic を使用した System i ストアード・プロシージャーの呼び出し』を参照してください。

使用するデータ・タイプには、特に注意してください。使用されるデータ・タイプ、例えばユーザーが選択ステートメントに使用するデータ・タイプには微妙な相違がある場合があります。また、出力と入出力パラメーター用に適当なサイズのバッファーが確保できていることを確認する必要があります。System i ストアード・プロシージャーのコーディング方法によっては、パフォーマンスに大きな影響を与える可能性があります。可能な限り、C 言語の exit()、RPG の SETON LR を使用してのプログラムのクローズは行わないようにしてください。可能であれば、RETRN または return を使用してください。ただし、これを行うと、呼び出しのたびに変数を再度初期設定して、ファイル・オープンをバイパスしなければならなくなる場合もあります。

ODBC プログラム例

System i Access の照会およびストアード・プロシージャーについて、ODBC のプログラミング例を使って説明します。

以下の System i Access ODBC プログラミング例では、簡単な照会を実行する方法と、ストアード・プロシージャーを呼び出して、データへのアクセスやデータの戻しを行う方法を説明しています。
C/C++、Visual Basic、および RPG の各プログラム言語のバージョンが提供されています。

C/C++ サンプルの多くは完全なプログラムではありません。詳しい説明とプログラミング例については、以下の情報を確認してください。

- ODBC プログラミング例 (Visual Basic、C++、および Lotus Script のプログラミング環境用) にアクセスするには、Web 上にある IBM ftp サイトへの関連リンク (以下参照) を選択してください。使用可能なプログラミング例を調べ、PC にダウンロードするには、index.txt を選択してください。
- ストアード・プロシージャーの説明、およびその呼び出し方法の例については、i5/OS Information Center にある『ストアード・プロシージャー』のトピック集を参照してください。
- Visual Basic、ADO、および C/C++ の例については、Microsoft の MSDN ライブラリーまたは ODBC の Web ページで、ODBC サンプルを検索してください。
- Programmer's Toolkit の C プログラミングの例も参照してください。

注: コード例を使用することで、633 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』の条件に同意します。

関連資料

615 ページの『ストアード・プロシージャー』
ストアード・プロシージャーは、System i データベース・トランザクション処理でサポートされています。

関連情報

IBM FTP サイト

MSDN ライブラリー

例: Visual C++ - ストアード・プロシージャーの呼び出しによるデータへのアクセスと戻し:

Visual C++ を使用して System i のストアード・プロシージャーを呼び出し、データにアクセスして、そのデータを戻す方法を、この例で説明します。

この例では、ストアード・プロシージャー呼び出しに関連したコードのみが含まれています。このコードでは、接続が既に確立されていることを前提としています。ストアード・プロシージャーのソース・コードについては、トピック『例: RPG- ODBC ストアード・プロシージャーのホスト・コード』を参照してください。

ストアード・プロシージャーの作成

```
 //* Drop the old Procedure
 strcpy(szDropProc,"drop procedure apilib.partqry2");

 rc = SQLExecDirect(m_hstmt, (unsigned char *)szDropProc, SQL_NTS);

 // This statement is used to create a  stored procedure
 // Unless the
 // procedure is destroyed, this statement need never be re-created
 strcpy(szCreateProc,"CREATE PROCEDURE APILIB.PARTQRY2 (INOUT P1 INTEGER," );
 strcat(szCreateProc,"INOUT P2 INTEGER)");
 strcat(szCreateProc,"EXTERNAL NAME APILIB.SPROC2 LANGUAGE RPG GENERAL")

 //' Create the new Procedure
 rc = SQLExecDirect(m_hstmt, (unsigned char *)szCreateProc, SQL_NTS);
 if (rc != SQL_SUCCESS &&; rc != SQL_SUCCESS_WITH_INFO) {
  DspSQLError(m_henv, m_hdbc, SQL_NULL_HSTMT);
  return APIS_INIT_ERROR;
 }
 if(rc != SQL_SUCCESS) {
  DspSQLError(m_henv, m_hdbc, SQL_NULL_HSTMT);
        return APIS_INIT_ERROR;
    }
```

ステートメントの作成

```
// Prepare the procedure call
  strcpy(szStoredProc, "call partqry2(?, ?)");
 // Prepare the stored procedure statement
    rc = SQLPrepare(m_hstmt, (unsigned char *) szStoredProc, strlen(szStoredProc));
    if(rc != SQL_SUCCESS &&; rc != SQL_SUCCESS_WITH_INFO) {
  DspSQLError(m_henv, m_hdbc, m_hstmt);
        return APIS_INIT_ERROR;
    }
```

パラメーターのバインド

```
 // Bind the parameters for the stored procedure

rc = SQLBindParameter(m_hstmt, 1, SQL_PARAM_INPUT_OUTPUT, SQL_C_LONG,
   SQL_INTEGER, sizeof(m_lOption), 0, &m_lOption, sizeof(m_lOption), &lcbon),
   &lcbOption);
rc |= SQLBindParameter(m_hstmt, 2, SQL_PARAM_INPUT_OUTPUT, SQL_C_LONG,
   SQL_INTEGER, sizeof(m_lPartNo), 0, &m_lPartNo, sizeof(m_lPartNo), &lcbon),
   &lcbOption);

    // Bind the Columns
 rc = SQLBindCol(m_hstmt, 1, SQL_C_SLONG, &m_lSPartNo,
   sizeof(m_lSPartNo), &lcbBuffer);
```

```
rc |= SQLBindCol(m_hstmt, 2, SQL_C_CHAR, &m_szSPartDesc,
  26, &lcbBuffer);
rc |= SQLBindCol(m_hstmt, 3, SQL_C_SLONG, &m_lSPartQty,
  sizeof(m_lSPartQty), &lcbBuffer);
rc |= SQLBindCol(m_hstmt, 4, SQL_C_DOUBLE, &m_dSPartPrice,
  sizeof(m_dSPartPrice), &lcbBuffer);
rc |= SQLBindCol(m_hstmt, 5, SQL_C_DATE, &m_dsSPartDate,
  10, &lcbBuffer);
```

ストアード・プロシージャーの呼び出し

```
// Request a single record
m_lOption = ONE_RECORD;
m_lPartNo = PartNo;

// Run the stored procedure
rc = SQLExecute(m_hstmt);
   if (rc != SQL_SUCCESS) {
       DspSQLError(m_henv, m_hdbc, m_hstmt);
       return APIS_SEND_ERROR;
   }

// (Try to) fetch a record
rc = SQLFetch(m_hstmt);
if (rc == SQL_NO_DATA_FOUND) {
 // Close the cursor for repeated processing
     rc = SQLCloseCursor(m_hstmt);
     return APIS_PART_NOT_FOUND;
}
else if (rc != SQL_SUCCESS) {
       DspSQLError(m_henv, m_hdbc, m_hstmt);
       return APIS_RECEIVE_ERROR;
   }

// If we are still here we have some data, so map it back
// Format and display the data
     .
     .
     .
```

例: Visual Basic - ストアード・プロシージャーの呼び出しによるデータへのアクセスと戻し:

System i のストアード・プロシージャーの作成、準備、バインド、および呼び出しについて、Visual Basic の例を使って説明します。

Visual Basic は、DLL の中にある外部関数を呼び出すことができます。 ODBC ドライバーは すべて DLL であるため、System i のストアード・プロシージャーを呼び出して結果値を戻すように、Visual Basic アプリケーションで ODBC API に直接コーディングすることができます。詳しくは、トピック『ODBC API への直接コーディング』を参照してください。ストアード・プロシージャーのソース・コードについては、トピック『例: RPG- ODBC ストアード・プロシージャーのホスト・コード』を参照してください。

ストアード・プロシージャーの作成

```
  ' This statement will drop an existing stored procedure
szDropProc = "drop procedure apilib.partqry2"


   '* This statement is used to create a  stored procedure
   '* Unless the
   '* procedure is destroyed, this statement need never be re-created
szCreateProc = "CREATE PROCEDURE APILIB.PARTQRY2 (INOUT P1 INTEGER,"
szCreateProc = szCreateProc & "INOUT P2 INTEGER)"
szCreateProc = szCreateProc & "EXTERNAL NAME APILIB.SPROC2 LANGUAGE RPG GENERAL"
```

```
   '*  Allocate statement handle
rc = SQLAllocHandle(SQL_HANDLE_STMT, ghDbc, hStmt)
If rc <> SQL_SUCCESS Then
     Call DisplayError(rc, "SQLAllocStmt failed.")
     Call DspSQLError(henv, SQL_NULL_HDBC, SQL_NULL_HSTMT)
End If
  '* Drop the old Procedure
rc = SQLExecDirect(hstmt, szDropProc, SQL_NTS)

   ' Create the new Procedure
rc = SQLExecDirect(hstmt, szCreateProc, SQL_NTS)
If rc <> SQL_SUCCESS And rc <> SQL_SUCCESS_WITH_INFO Then
     Call DisplayError(rc, "SQLCreate failed.")
     Call DspSQLError(henv, hdbc, hstmt)
End If
```

ステートメントの作成

```
   '* This statement will be used to call the stored procedure
   szStoredProc = "call partqry2(?, ?)"
  '* Prepare the stored procedure call statement

   rc = SQLPrepare(hstmt, szStoredProc, Len(szStoredProc))
   If rc <> SQL_SUCCESS And rc <> SQL_SUCCESS_WITH_INFO Then
       Call DisplayError(rc, "SQLPrepare failed.")
       Call DspSQLError(henv, hdbc, hstmt)
   End If
```

パラメーターのバインド

```
 'Bind the parameters for the stored procedure
    rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, _
               SQL_INTEGER, lLen1, 0, sFlag, lLen1, lCbValue)

    If rc <> SQL_SUCCESS Then
       Call DisplayError(rc, "Problem binding parameter ")
    End If

    rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_SLONG, _
                      SQL_INTEGER, 4, 0, lPartNumber, lLen2, lCbValue)

    If rc <> SQL_SUCCESS Then
       Call DisplayError(rc, "Problem binding parameter ")
    End If
```

ストアード・プロシージャーの呼び出し

```
  rc = SQLExecute(hstmt)
  If lRc <> SQL_SUCCESS Then
         ' Free the statement handle for repeated processing
             rc = SQLFreeHandle(
          Call DspSQLError(henv, hdbc, hstmt)
  End If
  rc = SQLFetch(hstmt)
  If rc = SQL_NO_DATA_FOUND Then
         mnuClear_Click               'Clear screen
         txtPartNumber = lPartNumber 'Show the part number not found
         Call DisplayMessage("RECORD NOT FOUND")
         .
         .
       Else
     'Get Description
          rc = SQLGetData(hstmt, 2, SQL_C_CHAR, sSDescription, _
                         25, lcbBuffer)
          'Get Quantity. SQLGetLongData uses alias SQLGetData
```

```
        rc = SQLGetLongData(hstmt, 3, SQL_C_SLONG, lSQuantity, _
                            Len(lSQuantity), lcbBuffer)
        'Get Price. SQLGetDoubleData uses alias SQLGetData
        rc = SQLGetDoubleData(hstmt, 4, SQL_C_DOUBLE, dSPrice, _
                              Len(dSPrice), lcbBuffer)
        'Get Received date
        rc = SQLGetData(hstmt, 5, SQL_C_CHAR, sSReceivedDate, _
                        10, lcbBuffer)
        txtDescription = sSDescription  'Show description
        txtQuantity = lSQuantity          'Show quantity
        txtPrice = Format(dSPrice, "currency")   'Convert dSPrice to
        txtReceivedDate = CDate(sSReceivedDate)  'Convert string to d
        Call DisplayMessage("Record found")
    End If
```

関連資料

519 ページの『ODBC API への直接コーディング』
PC アプリケーションの多くで、ユーザーが異種プラットフォーム上のデータにシームレスにアクセスできる ODBC 呼び出しが行われます。 ODBC API で独自の System i Access アプリケーションの開発を始める前に、ODBC アプリケーションをデータベース・サーバーに接続して、サーバーと情報を交換する方法について、理解しておいてください。

『例: RPG - ODBC ストアード・プロシージャーのホスト・コード』
この例ではプログラム、**SPROC2** は、System i Access ODBC を介して、ストアード・プロシージャーとしてクライアントから呼び出されます。このプログラムは、データを、PARTS (パーツ) データベース・ファイルからクライアントへ戻します。

例: RPG - ODBC ストアード・プロシージャーのホスト・コード:

この例ではプログラム、**SPROC2** は、System i Access ODBC を介して、ストアード・プロシージャーとしてクライアントから呼び出されます。このプログラムは、データを、PARTS (パーツ) データベース・ファイルからクライアントへ戻します。

RPG/400® (非 ILE) 例:

```
     * THIS EXAMPLE IS WRITTEN IN RPG/400 (NON-ILE)
     *
     *  DEFINES PART AS AN INTEGER (BINARY 4.0)
     *
    I#OPTDS      DS
    I                                    B   1   40#OPT
    I#PRTDS      DS
    I                                    B   1   40#PART
    C           *ENTRY    PLIST
    C                     PARM           #OPTDS
    C                     PARM           #PRTDS
     *  COPY PART NUMBER TO RPG NATIVE VARIABLE WITH SAME
     *  ATTRIBUTES OF FIELD IN PARTS MASTER FILE (PACKED DECIMAL 5,0)
    C                     Z-ADD#PART     PART    50
    C           #OPT      CASEQ1         ONEREC
    C           #OPT      CASEQ2         ALLREC
    C                     ENDCS
    C                     SETON                     LR
    C                     RETRN
     *
     ****************************
    C           ONEREC    BEGSR
     ****************************
     * PROCESS REQUEST FOR A SINGLE RECORD.
    C/EXEC SQL DECLARE C1 CURSOR FOR
    C+   SELECT
    C+   PARTNO,
    C+   PARTDS,
```

```
C+   PARTQY,
C+   PARTPR,
C+   PARTDT
C+
C+   FROM PARTS       -- FROM PART MASTER FILE
C+
C+   WHERE PARTNO  = :PART
C+
C+
C+  FOR FETCH ONLY    -- READ ONLY CURSOR
C/END-EXEC
C*
C/EXEC SQL
C+  OPEN C1
C/END-EXEC
C*
C/EXEC SQL
C+  SET RESULT SETS CURSOR C1
C/END-EXEC
C                     ENDSR
 ****************************
C           ALLREC    BEGSR
 ****************************
 * PROCESS REQUEST TO RETURN ALL RECORDS
C/EXEC SQL DECLARE C2 CURSOR FOR
C+   SELECT
C+   PARTNO,
C+   PARTDS,
C+   PARTQY,
C+   PARTPR,
C+   PARTDT
C+
C+   FROM PARTS       -- FROM PART MASTER FILE
C+
C+
C+  ORDER BY PARTNO   -- SORT BY PARTNO
C+
C+  FOR FETCH ONLY    -- READ ONLY CURSOR
C/END-EXEC
C*
C/EXEC SQL
C+  OPEN C2
C/END-EXEC
C*
C/EXEC SQL
C+  SET RESULT SETS CURSOR C2
C/END-EXEC
C                     ENDSR
```

ILE-RPG の例

```
 * This example is written in ILE-RPG
 *
 * Define option and part as integer
D#opt             s             10i 0
D#part            s             10i 0
 * Define part as packed 5/0
Dpart             s              5p 0

C     *entry        plist
C                   parm                    #opt
C     part          parm                    #part

C     #opt          caseq     1             onerec
C     #opt          caseq     2             allrec
C                   endcs
```

```
C                   eval      *inlr = *on
C                   return
 *
 ****************************
C     onerec        begsr
 ****************************
 * Process request for a single record.
C/EXEC SQL DECLARE C1 CURSOR FOR
C+   SELECT
C+   PARTNO,
C+   PARTDS,
C+   PARTQY,
C+   PARTPR,
C+   PARTDT
C+
C+   FROM PARTS       -- FROM PART MASTER FILE
C+
C+   WHERE PARTNO  = :PART
C+
C+
C+  FOR FETCH ONLY    -- READ ONLY CURSOR
C/END-EXEC
C*
C/EXEC SQL
C+  OPEN C1
C/END-EXEC
C*
C/EXEC SQL
C+  SET RESULT SETS CURSOR C1
C/END-EXEC
C                   endsr
 ****************************
C     allrec        begsr
 ****************************
 * Process request to return all records
C/EXEC SQL DECLARE C2 CURSOR FOR
C+   SELECT
C+   PARTNO,
C+   PARTDS,
C+   PARTQY,
C+   PARTPR,
C+   PARTDT
C+
C+   FROM PARTS       -- FROM PART MASTER FILE
C+
C+
C+  ORDER BY PARTNO   -- SORT BY PARTNO
C+
C+  FOR FETCH ONLY    -- READ ONLY CURSOR
C/END-EXEC
C*
C/EXEC SQL
C+  OPEN C2
C/END-EXEC
C*
C/EXEC SQL
C+  SET RESULT SETS CURSOR C2
C/END-EXEC
C                   endsr
```

関連資料

626 ページの『例: Visual Basic - ストアード・プロシージャーの呼び出しによるデータへのアクセスと戻し』

System i のストアード・プロシージャーの作成、準備、バインド、および呼び出しについて、Visual Basic の例を使って説明します。

System i Access データベースの API

System i Access for Windows の専有 C/C++ データベース API で提供されていた機能のうち、現在はサポート対象外となっているテクノロジーを使用します。

System i Access for Windows の専有 C/C++ データベース API では、System i データベース・ファイルへの SQL アクセスのほかに、System i のデータベース機能およびカタログ機能へのサポートも提供していました。これらの C/C++ API (最適化 SQL API とも呼ばれます) は、現在サポートされていません。

以下のテクノロジーでは、これらの非推奨 API の機能を引き続き提供しています。これらの詳細については、他のトピック集を参照してください。

- NET フレームワーク・クラス
- ADO/OLE DB
- ODBC
- JDBC
- データベース転送
- ActiveX オートメーション・オブジェクト

関連資料

24 ページの『データベース API の戻りコード』
System i Access for Windows のデータベース API の戻りコードを示します。

Java プログラミング

System i Access for Windows プロダクトに同梱されている IBM Toolbox for Java は、単独でも使用できます。

Sun によって定義された **Java** プログラミング言語を使用すると、移植性の高い Web ベース・アプリケーションを開発することができます。

System i Access for Windows プロダクトに同梱される IBM Toolbox for Java は、System i 資源にアクセスするための Java クラスを提供します。IBM Toolbox for Java は、System i Access for Windows の i5/OS ホスト・サーバーを、システムへのアクセス・ポイントとして使用します。ただし、IBM Toolbox for Java を使用する際に、System i Access for Windows 製品は必要ありません。 Toolbox を使用すると、本製品から独立して実行されるアプリケーションを作成することができます。

IBM Toolbox for Java インターフェースの動作 (セキュリティー、トレースなど) は、他の System i Access for Windows インターフェースの動作と異なることがあります。

注: コード例を使用することで、 633 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』の条件に同意します。

ActiveX プログラミング

ActiveX オートメーションは、Microsoft によって定義されたプログラミング・テクノロジーであり、System i Access for Windows 製品でサポートされています。

注: コード例を使用することで、 633 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』の条件に同意します。

System i Access for Windows では、ActiveX オートメーションを使用して System i 資源にアクセスする方法として、以下の方法を備えています。

オートメーション・オブジェクト

これらのオブジェクトは、以下のサポートを提供します。

- System i データ待ち行列へのアクセス
- System i アプリケーション・プログラミング・インターフェースとユーザー・プログラムの呼び出し
- System i 接続の管理とセキュリティーの検証
- System i CL コマンドの実行
- データ・タイプ変換とコード・ページ変換の実行
- データベース転送の実行
- ホスト・エミュレーション・セッションとのインターフェース

System i Access for Windows OLE DB Provider:

Microsoft の ActiveX Data Objects (ADO) を使用して、System i Access for Windows OLE DB Provider を呼び出すと、以下の System i 資源にアクセスできます。

- レコード・レベルのアクセスを介した System i データベース
- SQL を介した System i データベース
- SQL ストアード・プロシージャー
- データ待ち行列
- プログラム
- CL コマンド

カスタム・コントロール

以下のための ActiveX カスタム・コントロールが提供されます。

- Ssytem i データ待ち行列
- System i CL コマンド
- 以前に接続されていたシステムの System i 名
- System i ナビゲーター

Programmer's Toolkit:

System i Access for Windows ActiveX の詳細については、製品の **Programmer's Toolkit** 構成要素のトピック『**ActiveX**』を参照してください。このツールキットには、ADO と ActiveX オートメーション・オブジェクトに関するすべての資料、および ActiveX の情報源へのリンクが含まれています。

ActiveX のトピックにアクセスする方法

1. **Programmer's Toolkit** がインストールされていることを確認します (『Programmer's Toolkit のインストール』を参照してください)。
2. **Programmer's Toolkit** の立ち上げ (『Programmer's Toolkit の立ち上げ』を参照してください)。
3. 「**概要**」トピックを選択します。
4. 「**プログラミング・テクノロジー**」を選択します。
5. 「**ActiveX**」を選択します。

関連タスク

6 ページの『Programmer’s Toolkit のインストール』
Programmer’s Toolkit は、System i Access for Windows 製品のフィーチャーの 1 つとしてインストールされます。

6 ページの『Programmer’s Toolkit の立ち上げ』
Programmer’s Toolkit は、System i Access for Windows 製品のフィーチャーの 1 つとして立ち上げられます。

関連資料

513 ページの『System i Access for Windows OLE DB Provider』
System i データベース・ファイルへのレコード・レベル・アクセスおよび SQL アクセスをサポートします。このサポートを活用するためには、ActiveX データ・オブジェクト (ADO) および OLE DB インターフェースを使用します。

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。

〒106-8711
東京都港区六本木 3-2-12
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。

表示されている IBM の価格は IBM が小売り価格として提示しているもので、現行価格であり、通知なしに変更されるものです。卸価格は、異なる場合があります。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

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。

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_. All rights reserved.

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

この「System i Access for Windows」資料には、プログラムを作成するユーザーが、IBM i5/OS 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

以下は、International Business Machines Corporation の米国およびその他の国における商標です。

Advanced Function Presentation
Advanced Function Printing
AFP
Approach
AS/400
C/400
DB2
DB2 Universal Database
DRDA
i5/OS
IBM
IBM (ロゴ)
Intelligent Printer Data Stream
iSeries
Lotus
Lotus Notes
MVS
NetServer
RPG/400
S/390System i

| Intel、Intel (ロゴ)、Intel Inside、Intel Inside (ロゴ)、Intel Centrino、Intel Centrino (ロゴ)、Celeron、Intel Xeon、Intel SpeedStep、Itanium、Pentium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。

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。

| ITIL は英国 Office of Government Commerce の登録商標および共同体登録商標であって、米国特許商標庁にて登録されています。

| 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。

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

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。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

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
〒106-8711　東京都港区六本木3-2-12