SYBASE®
An SAP® Company

ORCA ガイド

PowerBuilder® Classic

12.5

LAST REVISED: July 2011

目次

第 1 章 ORCA を使用する方法 1
ORCA とは？ 1
ORCA を使用してできること 2
ORCA 呼び出しプログラムの開発者 3
ORCA のインストール 4
ORCA とライブラリ ペインタ 5
PowerBuilder ライブラリ中のオブジェクト 5
オブジェクトのソース コード 5
PowerBuilder コマンドと ORCA 関数 6
ORCA 関数について 7
ORCA セッションの管理に関する関数 7
PowerBuilder ライブラリの管理に関する関数 8
PowerBuilder オブジェクトのインポートとコンパイルに関する関数 9
PowerBuilder オブジェクトのクエリ関数 10
実行ファイルと動的ライブラリの作成に関する関数 11
EAServer にコンポーネントを配布する関数 11
ソース管理の操作を管理する関数 12
ORCA コールバック関数について 13
コールバックを使用する ORCA 関数 13
コールバックの動作 14
コールバック関数の内容 15
ORCA プログラムを書く 18
ORCA プログラムの概要 18
新しいアプリケーションのブートストラップ 20
廃止された ORCA 関数の削除 22

第 2 章 ORCA 関数 23
例について 24
ORCA 戻り値 25
PBORCA_ApplicationRebuild 27
PBORCA_BuildProject 29
PBORCA_BuildProjectEx 30
PBORCA_BuildProjectWithOverrides 31

PBORCA_CompileEntryImport 33
PBORCA_CompileEntryImportList 41
PBORCA_CompileEntryRegenerate 47
PBORCA_ConfigureSession 50
PBORCA_DeployWinFormProject 54
PBORCA_DynamicLibraryCreate 56
PBORCA_ExecutableCreate 58
PBORCA_LibraryCommentModify 64
PBORCA_LibraryCreate 65
PBORCA_LibraryDelete 67
PBORCA_LibraryDirectory 68
PBORCA_LibraryEntryCopy 72
PBORCA_LibraryEntryDelete 74
PBORCA_LibraryEntryExport 76
PBORCA_LibraryEntryExportEx 81
PBORCA_LibraryEntryInformation 83
PBORCA_LibraryEntryMove 86
PBORCA_ObjectQueryHierarchy 88
PBORCA_ObjectQueryReference 90
PBORCA_SccClose 92
PBORCA_SccConnect 93
PBORCA_SccConnectOffline 95
PBORCA_SccExcludeLibraryList 97
PBORCA_SccGetConnectProperties 99
PBORCA_SccGetLatestVersion 101
PBORCA_SccRefreshTarget 102
PBORCA_SccResetRevisionNumber 103
PBORCA_SccSetTarget 105
PBORCA_SessionClose 108
PBORCA_SessionGetError 109
PBORCA_SessionOpen 110
PBORCA_SessionSetCurrentAppl 111
PBORCA_SessionSetLibraryList 113
PBORCA_SetDebug 115
PBORCA_SetExeInfo 117

第 3 章 ORCA コールバック関数と構造体 119
オブジェクトをコンパイルするコールバック関数 120
PBORCA_COMPERR 構造体 121
EAServer にコンポーネントを配布するコールバック関数 123
PBORCA_BLDERR 構造体 124
PBORCA_LibraryDirectory のコールバック関数 125
PBORCA_DIRENTRY 構造体 126
PBORCA_ObjectQueryHierarchy のコールバック関数 127

PBORCA_HIERARCHY 構造体 128
PBORCA_ObjectQueryReference のコールバック関数 129
PBORCA_REFERENCE 構造体 130
PBORCA_ExecutableCreate のコールバック関数 131
PBORCA_LINKERR 構造体 132
PBORCA_SccSetTarget のコールバック関数 133
PBORCA_SCCSETTARGET 構造体 134

本書について

目的

このマニュアルでは、PowerBuilder® の一部である Powersoft OpenLibrary API（ORCA）について説明します。

ORCA プログラムの開発に必要な情報を提供します。以下の内容を説明しています。

- ORCA ソフトウェアのインストール
- ORCA 関数と PowerBuilder 開発者がライブラリ ペインタでできることの比較
- ORCA プログラムを書くこと
- ORCA 関数とコールバック関数

対象とする読者

このマニュアルはツール ベンダを対象にしています。 このマニュアルは、PowerBuilder と一緒に使用して PowerBuilder ライブラリのオブジェクトを操作および管理するツールや関連製品を開発する CODE パートナやツール ベンダを対象としています。

制約

ORCA ユーザは、ORCA の使用に関する制約を知っておく必要があります。これについては、3 ページの「ORCA 呼び出しプログラムの開発者」を参照してください。

このマニュアルは、PowerBuilder アプリケーションの開発者向けではありません。 ORCA は PowerBuilder アプリケーションを構築することを目的としたものではありません。また、このマニュアルは ORCA を使用するプログラムの実行方法については説明していません。そのようなプログラムには専用のマニュアルが提供されているはずです。

第 1 章 ORCA を使用する方法

この章について

この章では、Powersoft Open Library API（ORCA）について説明します。

PowerBuilder 開発者がライブラリ ペインタで実行できるタスクと PowerBuilder ライブラリに対して ORCA でプログラム的に処理したいタスクの対応について説明します。

また、ORCA で使用可能な関数やプログラム中での ORCA セッション管理方法だけでなく、ORCA プログラムの開発に関する制限と、ORCA の使用対象者についても説明しています。

内容

項目	ページ
ORCA とは？	1
ORCA のインストール	4
ORCA とライブラリ ペインタ	5
ORCA 関数について	7
ORCA コールバック関数について	13
ORCA プログラムを書く	18
廃止された ORCA 関数の削除	22

ORCA とは？

ORCA は、PowerBuilder がライブラリ ペインタの中で使用する PowerBuilder Library Manager 関数にアクセスするためのソフトウェアです。プログラム（多くの場合 C プログラム）は、ライブラリ ペインタ インタフェースが提供するオブジェクトおよびライブラリの管理タスクと同様のことを行うために ORCA を使用します。

ORCA の歴史

ORCA は、Powersoft CODE (Client/Server Open Development Environment) プログラムの一部として、CASE ツール ベンダ向けに作成されました。CASE ツールは、アプリケーションの設計に基づいて PowerBuilder オブジェクトを作成および変更するために PowerBuilder ライブラリへのプログラムによるアクセスを必要としました。

一般的な ORCA プログラム

アプリケーションは、ORCA を使用して PowerBuilder オブジェクトを操作します。その内容は以下のとおりです。

- オブジェクトのソース コードを記述し、ORCA 関数を使用して、PBL にそのオブジェクト ソースを置く
- ORCA 関数を使用してライブラリからオブジェクトを抽出し、オブジェクトのソースを変更し、ORCA を再度使用してオブジェクトをライブラリに戻す

サンプル ORCA アプリケーション

ORCA は、PowerBuilder と連動する以下のような各種のツールで使用されています。

- OrcaScript ユーティリティ
- CASE ツール
- クラス ライブラリ
- ドキュメント ツール
- アプリケーション管理ツール
- テキストを検索してライブラリ全体で置換したり、ライブラリ中のオブジェクトをツリー ビューで表示するようなユーティリティ
- PowerBuilder が直接サポートしていないソース管理システムのインタフェース
- ソース管理されているオブジェクトから PowerBuilder ターゲットを再構築するためのユーティリティ

ORCA を使用してできること

ORCA を使用すると、PowerBuilder 開発環境で開発者が行うようなライブラリとオブジェクトの管理を、プログラムを使用してアプリケーションから行うことができます。ORCA では、ライブラリ ペインタの機能の大部分と、アプリケーション ペインタとプロジェクト ペインタの一部の機能をカバーしています。

ライブラリ ペインタでは、以下のことができます。

- PBL 中のオブジェクトのコピー、削除、移動、名前の変更、およびエクスポート
- オブジェクトのインポートとコンパイル
- プロジェクト ペインタで使用可能なすべてのオプションがある実行ファイルまたは PowerBuilder 動的ライブラリ（PBD または DLL）の作成
- オブジェクトの先祖の階層の調査や、参照しているオブジェクトの確認
- 新しいライブラリでのアプリケーション全体の作成（アプリケーションのブートストラップと呼ばれる）
- PowerBuilder のプロジェクト オブジェクト の指示に従った EAServer コンポーネント の構築と 配布
- ソース管理から PowerBuilder のターゲットを開くこととターゲット オブジェクト上のさまざまなソース管理操作の実行

ORCA 呼び出しプログラムの開発者

開発ツールとしての ORCA は、PowerBuilder 開発者向けのツールを提供するツール ベンダ向けに設計されました。ツール ベンダは、以下の制約を認識する必要があります。

開発ツールとしての ORCA は、一般の PowerBuilder 開発者にとって有効ではありません。ORCA を呼び出すプログラムを開発する場合は、この節で説明する制約を理解し、これに従う必要があります。

ORCA 使用の制約について　PowerBuilder と ORCA はどちらも、PowerBuilder のコンパイラを使用します。ただし、コンパイラは再入可能ではなく、また複数のプログラムを同時に使用することはできません。このため、プログラムが ORCA を呼び出すときは PowerBuilder を実行することができません。

ORCA を使用するツール プロバイダは、PowerBuilder 開発者が ORCA をベースにしたモジュールを呼び出すときに、ツールが以下を実行するように注意してプログラムを作成する必要があります。

1　PowerBuilder の終了

2　要求された ORCA 関数の実行

3　PowerBuilder の再起動

注意

ORCA 実行時に PowerBuilder 開発環境をシャットダウンしていないと、PowerBuilder ライブラリが壊れる恐れがあります。このため、気軽に ORCA を使用することはお勧めしません。

ORCA のインストール

ORCA は、PowerBuilder と連携して使用するために PowerBuilder ライブラリ中のオブジェクトを操作および管理する関連製品を開発する CODE パートナ、ツール ベンダ、およびお客様が利用できます。

ORCA プログラムを実行するには

ORCA を使用したプログラムを実行するには、ORCA DLL（PowerBuilder バージョン 12.5 では、PBORC125.DLL）が必要です。PowerBuilder をインストールすると、この DLL はほかの PowerBuilder DLL として同じディレクトリにインストールされます。

ORCA プログラムを開発するには

ORCA を使用した C プログラムを開発するには、いくつかのアイテムが必要です。詳細は、日本コンピュータシステムにお問い合わせください。

- C 開発ファイル
  PBORCA.H
  PBORCA.LIB
- 本書（PDF 形式）

ORCA とライブラリ ペインタ

PowerBuilder ライブラリ（PBL）はバイナリ ファイルです。そこに、PowerBuilder ペインタで定義したオブジェクトが ソースとコンパイル済みの 2 つの形式で格納されます。オブジェクトのソースはテキスト形式です。コンパイル済みのフォームはバイナリ形式であり、判読不可能です。

ライブラリ ペインタを使用して、PowerBuilder 開発者は PBL の内容を表示および保守できます。このペインタは、PBL 中のオブジェクトを更新日時やコメントなどのプロパティとともに一覧表示します。

PowerBuilder 開発者は、ライブラリ ペインタの中でオブジェクトの削除、移動、コンパイル、エクスポート、およびインポートを行えます。また、ソース管理システムを使用して、PowerBuilder 動的ライブラリと DLL を作成できます。

ライブラリ ペインタから、各ペインタ内のオブジェクトを開いて、グラフィカルにオブジェクトを表示し、修正することができます。

PowerBuilder ライブラリ中のオブジェクト

ペインタ中のオブジェクトを開くと、PowerBuilder はライブラリ エントリを解釈した上で、オブジェクトをグラフィカルな形式で表示します。ペインタはソース コードを表示しません。PBL 中にグラフィカルに表示されているオブジェクトを変更して再保存した場合、PowerBuilder は変更を取り込むためにソース コードを書き直してからオブジェクトを再コンパイルします。

オブジェクトのソース コード

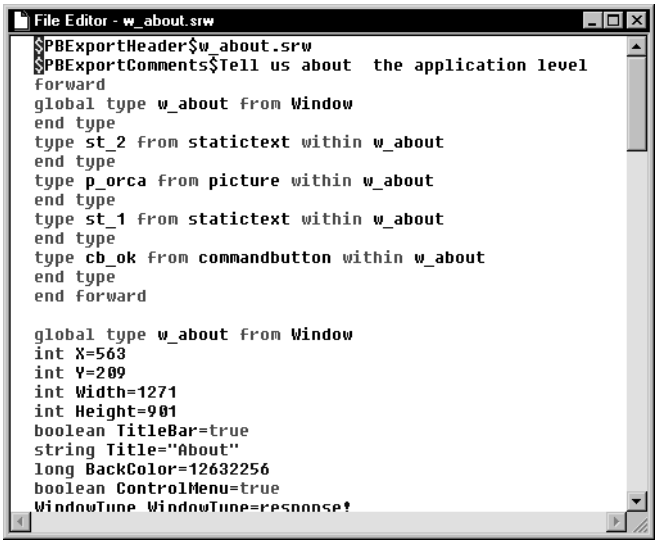
ライブラリ ペインタを使用して、ソース コードをエクスポートし、その内容を任意のテキスト エディタで調べて修正し、それをインポートしてライブラリに戻します。PowerBuilder は、ソース コードの有効性をチェックするためにインポートしたオブジェクトをコンパイルします。有効でない場合は、インポートしたオブジェクトのコンパイルに失敗します。

ファイルにエクスポートされたソース コードは、ソース コードの前に 2 行のヘッダが付いています。

```
$PBExportHeader$w_about.srw
$PBExportComments$Tell us about the application level
```

ORCA 関数は、これらのヘッダ行を無視して、関数に引き渡される lpszEntryName と lpszComments 引数を使用します。

PowerBuilder のファイル エディタで、エクスポートされたソース コードを見ることができます。



ソース コード構文の習得

オブジェクトのソース コードの構文に関するマニュアルはありません。ソース コードに何が含まれるかを知る方法は、オブジェクトをエクスポートしてそのソースを調べることだけです。

ORCA とソース コード

ORCA にはエクスポート関数があり、これにより既存のオブジェクトを調査したり変更したりできます。PowerBuilder 10 以降では、開発者はメモリ バッファまたはファイルにソースをエクスポートするために ORCA セッションを設定することができます。開発者は、4 種類のソース エンコーディング形式のどれを使用するか、2 行のエクスポート ヘッダ行をエクスポートするかどうか、オブジェクトのバイナリ コンポーネントを含めるかどうかを指定することもできます。

PowerBuilder コマンドと ORCA 関数

ほとんどの ORCA 関数には、ライブラリ ペインタやアプリケーション ペインタ、プロジェクト ペインタに対応するものや、PowerBuilder セッションの開始や停止を行うコマンドがあります。

次の節では、ORCA 関数とそれぞれの目的、およびそれらが PowerBuilder 開発環境で何に対応するのかについて確認します。

ORCA 関数について

すべての ORCA 関数は、関数の呼び出し規則を指定する WINAPI マクロを使用する外部 C 関数です。Windows プラットフォームでは、WINAPI は __stdcall と定義されています。

このマニュアル中のコード例について

すべての ORCA 関数は、ANSI クライアント プログラムや Unicode クライアント プログラムから呼び出されます。このマニュアル中のコード例は、PowerBuilder と一緒にインストールされた Shared/Sybase/PowerBuilder/cgen/h ディレクトリにある tchar.h ファイルの中で定義されているマクロを使用しています。/D _UNICODE コンパイラ ディレクティブが設定されると、これらのマクロは Unicode 文字列引数を受け取ります。_UNICODE が定義されていない場合、これらのマクロは ANSI 文字列引数を受け取ります。このコーディング方法を使うことにより、ANSI クライアントと Unicode クライアントのどちらでも正常に実行する ORCA プログラムを作成することができます。

ORCA 関数は、以下の 7 つのグループの関数に分けることができます。

- ORCA セッションの管理
- PowerBuilder ライブラリの管理
- PowerBuilder オブジェクトのコンパイル
- PowerBuilder オブジェクトのクエリ
- 実行ファイルと動的ライブラリの作成
- EAServer へのコンポーネントの配布
- PowerBuilder オブジェクトに関するソース管理の操作の管理

ORCA セッションの管理に関する関数

PowerBuilder を起動することで PowerBuilder セッションを開始し、PowerBuilder を終了することで PowerBuilder セッションを終了するのと同じように、ORCA を使用するときにはセッションを開き、終了するときにはセッションを閉じる必要があります。

ライブラリ リストと現行のアプリケーション

PowerBuilder 開発環境では、最初に現行のアプリケーションが必要です。また、オブジェクトの参照や変更、実行ファイルの作成などの予定がある場合には、ライブラリ リストの探索パスも設定します。ORCA でも要件は同じですが、順序は逆になります。ORCA では、ライブラリ リストを設定してから次に現行のアプリケーションを設定します。

オブジェクトのコンパイルやアプリケーションの構築に関係しない ORCA 関数は、ライブラリ リストと現行のアプリケーションを必要としません。これらは、ライブラリ管理の関数です。ソース管理の関数の場合、PBORCA_SccSetTarget が暗黙的にライブラリ リストと現行のアプリケーションを設定します。

セッション管理

セッション管理の関数（接頭辞はすべて PBORCA_）、それぞれの目的、および PowerBuilder 開発環境でそれらに相当するものは以下のとおりです。

関数（接頭辞 PBORCA_）	目的	PowerBuilder で同等のもの
ConfigureSession	後続の ORCA コマンドの動作に影響を与えるセッション プロパティを設定する	オプション
SessionOpen	ORCA セッションを開いてセッションのハンドルを戻す	PowerBuilder の起動
SessionClose	ORCA セッションを閉じる	PowerBuilder の終了
SessionSetLibraryList	セッションに対してライブラリを指定する	[ファイル｜ライブラリ リスト]
SessionSetCurrentAppl	セッションに対してアプリケーション オブジェクトを指定する	[ファイル｜アプリケーションを選択]
SessionGetError	エラーに関する情報を提供する	なし

PowerBuilder ライブラリの管理に関する関数

ライブラリを管理する関数は、ライブラリ ペインタのコマンドによく似ています。ライブラリを管理するこれらの関数を使用すると、ライブラリの作成や削除、ライブラリ コメントの修正、およびライブラリ中にあるオブジェクトの一覧を見ることができます。また、ライブラリ中のオブジェクトを調べたり、構文をエクスポートしたり、エントリのコピー、移動、削除を行うこともできます。

これらの関数は、ライブリ リストと現行のアプリケーションの外部から呼び出すことができます。

ライブラリ管理の関数（接頭辞はすべて PBORCA_）と、それぞれの目的と、PowerBuilder のライブラリ ペインタでそれらに相当するものは以下のとおりです。

関数（接頭辞 PBORCA_）	目的	PowerBuilder で同等のもの
LibraryCommentModify	ライブラリに対するコメントの変更	[ライブラリ｜プロパティ]
LibraryCreate	新しいライブラリ ファイルの作成	[ライブラリ｜作成]
LibraryDelete	ライブラリ ファイルの削除	[ライブラリ｜削除]
LibraryDirectory	ライブラリのコメントとそのオブジェクトの一覧の取得	リスト ビュー
LibraryEntryCopy	あるライブラリから別のライブラリへオブジェクトをコピー	[エントリ｜コピー]
LibraryEntryDelete	ライブラリからオブジェクトを削除	[エントリ｜削除]
LibraryEntryExport	オブジェクトのソース コードの取得	[エントリ｜エクスポート]
LibraryEntryExportEx	オブジェクトのソース コードの取得	[エントリ｜エクスポート]
LibraryEntryInformation	オブジェクトの詳細取得	リスト ビュー
LibraryEntryMove	あるライブラリから別のライブラリへオブジェクトを移動	[エントリ｜移動]

PowerBuilder オブジェクトのインポートとコンパイルに関する関数

これらの関数を使用して、ソース コードを一覧しているテキストからライブラリへ新しいオブジェクトをインポートしたり、ライブラリ中にすでに存在するエントリをコンパイルすることができます。

ライブラリのエントリには、ソース コード表記とコンパイル済みバージョンの両方があります。新しいオブジェクトをインポートすると、PowerBuilder はそれをコンパイルします。エラーがあると、インポートされません。

これらの関数を呼び出す前に、ライブラリ リストと現行のアプリケーションを設定する必要があります。

コンパイルの関数（接頭辞はすべて PBORCA_）と、それぞれの目的と、PowerBuilder のライブラリ ペインタでそれらに相当するものは以下のとおりです。

関数（接頭辞 PBORCA_）	目的	ライブラリ ペインタでこれと同等のもの
CompileEntryImport	オブジェクトのインポートとコンパイル	[エントリ｜インポート]
CompileEntryImportList	オブジェクト一覧のインポートとそれらのコンパイル	なし
CompileEntryRegenerate	オブジェクトのコンパイル	[エントリ｜再生成]
ApplicationRebuild	アプリケーションに関連するすべてのライブラリ中のすべてのオブジェクトのコンパイル	[実行｜ワークスペースのインクリメンタル再構築] または [実行｜ワークスペースのフル構築]

コンパイルする関数は、ライブラリから実行可能ファイルを作成する関数ではありません。詳細については、次の「実行ファイルと動的ライブラリの作成に関する関数」を参照してください。

PowerBuilder オブジェクトのクエリ関数

オブジェクトのクエリ関数は、オブジェクトの先祖と参照しているオブジェクトに関する情報を取得します。

これらの関数を呼び出す前に、ライブラリ リストと現行のアプリケーションを設定する必要があります。

オブジェクトのクエリ関数（接頭辞はすべて PBORCA_）は以下のとおりです。相当する PowerBuilder のコマンドはありません。

関数（接頭辞 PBORCA_）	目的
ObjectQueryHierarchy	オブジェクトの先祖の一覧を取得
ObjectQueryReference	参照しているオブジェクトの一覧を取得

実行ファイルと動的ライブラリの作成に関する関数

これらの関数を使用して、実行ファイルと PowerBuilder 動的ライブラリ（PBD と DLL）を作成できます。プロジェクト ペインタで指定するのと同じように、P コードとマシン コードとトレースに関するオプションを指定することができます。

ORCA を使用する場合、実行ファイルを作成する前に別のステップの中で PBD や DLL を作成する必要があります。

これらの関数を呼び出す前に、ライブラリ リストと現行のアプリケーションを設定する必要があります。

実行ファイルとライブラリを作成する関数（接頭辞はすべて PBORCA_）、それぞれの目的と、PowerBuilder の開発環境でそれらに相当するものは以下のとおりです。

関数（接頭辞 PBORCA_）	目的	ペインタでこれと同等のもの
ExecutableCreate	ORCA のライブラリ リストと現行のアプリケーション オブジェクトを使用してアプリケーションの実行ファイルを作成	プロジェクト ペインタ
DynamicLibraryCreate	PBL から PowerBuilder 動的ライブラリを作成	プロジェクト ペインタまたはライブラリ ペインタ ［ライブラリ｜動的ライブラリの構築］
SetExeInfo	生成される DLL と EXE に関連する追加ファイルのプロパティを設定	プロジェクト ペインタ

EAServer にコンポーネントを配布する関数

これらの関数は、プロジェクト オブジェクトの仕様を使用または上書きして、EAServer コンポーネントを配布します。

関数（接頭辞 PBORCA_）	目的
BuildProject	プロジェクト オブジェクトの仕様によるコンポーネントの配布
BuildProjectEx	コンポーネント配布時におけるサーバ名とポート番号の上書き

ソース管理の操作を管理する関数

これらの関数を使用して、PowerBuilder のターゲットとオブジェクトに関するソース管理の操作を行うことができます。

関数（接頭辞 PBORCA_）	目的
SccClose	アクティブな SCC プロジェクトを閉じる
SccConnect	ソース管理の初期化とプロジェクトの開始
SccConnectOffline	ソース管理への接続をシミュレートする
SccExcludeLibraryList	ターゲットのライブラリ リストの中で、次の PBORCA_SccRefreshTarget 操作で同期されないライブラリの名前
SccGetConnectProperties	PowerBuilder ワークスペースに関連する SCC 接続のプロパティを戻す
SccGetLatestVersion	SCC リポジトリからローカル プロジェクト パスへのオブジェクトの最新バージョンのコピー
SccRefreshTarget	ターゲット ライブラリ中の各オブジェクトのソースのリフレッシュ
SccSetPassword	先行する SccConnect への password プロパティの設定
SccSetTarget	ソース管理からターゲット ファイルを検索し、アプリケーション オブジェクト名を ORCA へ渡し、ORCA セッションのライブラリ リストを設定

ORCA コールバック関数について

一部の ORCA 関数では、コールバック関数を記述する必要があります。コールバック関数は、呼び出されたプログラム（ORCA DLL またはライブラリ マネージャ）が、呼び出しているプログラム（ORCA プログラム実行ファイル）の中でコードを実行するための方法を提供しています。

ORCA のコールバックの使い方

ORCA は、処理する項目の数が不明な場合にコールバック関数を使用します。コールバック関数の目的は、戻されたそれぞれの項目を処理することであり、多くの場合、ユーザに情報を戻します。

オプションか必須か

一部のコールバック関数は、オブジェクトのコンパイルや実行ファイルの構築などの、主要作業が終了したときに発生するエラーに対処します。エラーの対処に関しては、コールバック関数はオプションです。ほかのコールバック関数は、ディレクトリ リスト中の各項目など、関数を呼び出すときに必要な情報を処理します。情報に関するコールバック関数は必須です。

言語要件

コールバック関数を使用する必要がある ORCA 関数は、C および C++ のようにポインタを使用する言語で書かれたプログラムでのみ使用することが可能です。

新しい ORCA コールバック関数を作成する場合、関数の呼び出し規則を指定する CALLBACK マクロを使用します。Windows プラットフォームでは、CALLBACK は __stdcall として定義されています。

コールバックを使用する ORCA 関数

これらの関数（接頭辞はすべて PBORCA_）は、コールバック関数を使用します。

ORCA 関数呼び出し（接頭辞 PBORCA_）	コールバックの目的
BuildProjectEx BuildProject	各配布エラーにつき 1 度呼び出される
CompileEntryImport CompileEntryImportList CompileEntryRegenerate	各コンパイル エラーにつき 1 度呼び出される
ExecutableCreate	各リンク エラーにつき 1 度呼び出される
LibraryDirectory	各ライブラリ エントリ名につき 1 度呼び出される
ObjectQueryHierarchy	各先祖名につき 1 度呼び出される

ORCA 関数呼び出し（接頭辞 PBORCA_）	コールバックの目的
ObjectQueryReference	エントリ中の参照される各オブジェクトにつき 1 度呼び出される
SccSetTarget	ライブラリ リスト中の各ライブラリにつき 1 度呼び出される

コールバックの動作

ORCA は以下のようにコールバック関数を呼び出します。

1 呼び出しプログラムがデータを保持するためのバッファ（UserData バッファ）を割り当てます。

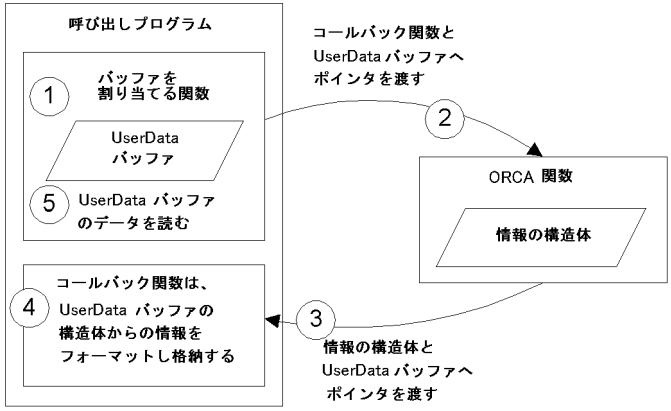
2 呼び出しプログラムは ORCA 関数を呼び出し、コールバック関数と UserData バッファにポインタを渡します。

3 ORCA 関数がレポート情報を必要とする場合は、コールバック関数を呼びます。ポインタを、情報を保持している構造体と UserData バッファへ渡します。

4 コールバック関数は、構造体から情報を読み取り、それを UserData バッファ内で形式設定します。

手順 3 と 4 は、ORCA がレポートするために必要な各情報について繰り返します。1 つの ORCA 関数がコールバックを呼び出す回数は、エラー発生の有無や、レポートに情報が必要かどうかにより変わります。1 度だけの場合もあれば、数回の場合もあり、1 度も呼び出さない場合もあります。

5　ORCA 関数が完了して呼び出しプログラムに制御が戻ると、UserData バッファ中の情報を読み取ります。



コールバック関数の内容

コールバック関数の中で行われる処理は、すべて開発者が決定します。この節では、その扱い方について簡単に説明します。

UserData バッファ

この例では、UserData バッファは、実際のメッセージ バッファを指し示す値のフィールドを持つ構造体です。ほかのフィールドは、メッセージ バッファの内容を管理します。

```
typedef struct ORCA_UserDataInfo {
    LPBYTE lpszBuffer;      // データを格納するバッファ
    DWORD dwCallCount;     // バッファ内のメッセージの数
    DWORD dwBufferSize;    // バッファのサイズ
    DWORD dwBufferOffset; // バッファ内の現在のオフセット
} ORCA_USERDATAINFO, FAR *PORCA_USERDATAINFO;
```

呼び出しプログラム

呼び出しプログラムの中で、UserDataInfo 構造体が初期化されます。

呼び出しプログラムは、メッセージに必要となる容量が不明なので、60000 バイト（任意のサイズ）を割り当てます。リンク エラーを収集する場合は、これで十分です。大きなライブラリのディレクトリ情報が必要な場合は、これでは不十分な場合があります。

```
ORCA_USERDATAINFO UserDataBuffer;
PORCA_USERDATAINFO lpUserDataBuffer;
```

```
lpUserDataBuffer = &UserDataBuffer;
lpUserDataBuffer->dwCallCount = 0;
lpUserDataBuffer->dwBufferOffset = 0;
lpUserDataBuffer->dwBufferSize = 60000;
lpUserDataBuffer->lpszBuffer =
  (LPTSTR)malloc((size_t)lpUserDataBuffer->
     dwBufferSize);
memset(lpUserDataBuffer->lpszBuffer,
       0x00,(size_t)lpUserDataBuffer->dwBufferSize);
```

関数ポインタの定義　呼び出しプログラムは、ORCA 関数へ渡すコールバック関数の関数ポインタを定義します。

```
PBORCA_LINKPROC fpLinkProc;
fpLinkProc = (PBORCA_LINKPROC)LinkErrors;
```

ORCA 呼び出し　呼び出しプログラムは ORCA 関数を呼び出し、コールバック関数のポインタと UderData バッファのポインタを渡します。この例では、PBORCA_ExecutableCreate（コールバックの種類は PBORCA_LNKPROC）を呼び出しています。

```
rtn = PBORCA_ExecutableCreate(..., (PBORCA_LNKPROC)
fpLinkProc, lpUserDataBuffer);
```

処理の結果　最後に、呼び出しプログラムは UserData バッファに格納されたコールバック関数の情報を処理したり、表示したりすることができます。

割り当てたメモリの解放　UserData 構造体がメモリを割り当てている場合には、割り当てたメモリを解放します。

```
free( lpUserDataBuffer->lpszBuffer )
```

コールバック プログラム

コールバック プログラムは、現行のエラーまたは情報と一緒に構造体を受け取り、UserData バッファ中の lpszBuffer が指し示すメッセージバッファ中の情報を格納します。また、UserData バッファ中に格納されているポインタの管理も行います。

単純なコールバック　単純なコールバックは、以下を実行します。

- 呼び出された回数を保持する
- メッセージを保持し、オーバーフローする場合にはバッファの再割り当てを行う

次のコード例は、PBORCA_ExecutableCreate に対して LinkErrors と呼ばれるコールバックを実装しています。

```
void CALLBACK LinkErrors(PPBORCA_LINKERR lpLinkError,
       LPVOID lpUserData)
```

```
{
    PORCA_USERDATAINFO lpData;
    LPBYTE lpCurrByte;
    LPTSTR lpCurrentPtr;
    int iNeededSize;
    lpData = (PORCA_USERDATAINFO) lpUserData;

    // リンク エラーの数を記録する
    lpData->dwCallCount++;

    // バッファがすでに満杯か？
    if (lpData->dwBufferOffset==lpData->dwBufferSize)
    return;

    // 新しいメッセージの長さは？
    // メッセージの長さに CR と LF を加える
    iNeededSize =
        (_tcslen(lpLinkError->lpszMessageText) + 2)*
            sizeof(TCHAR);
     // 必要があればバッファを再度割り当てる
    if ((lpData->dwBufferOffset + iNeededSize) >
        lpData->dwBufferSize)
    {
       LPVOID lpNewBlock;
       DWORD  dwNewSize;
       dwNewSize = lpData->dwBufferSize * 2;
       lpNewBlock = realloc(lpData->lpszBuffer,
          (size_t)dwNewSize);
       if (lpNewBlock)
       {
           lpData->lpszBuffer = (LPTSTR) lpNewBlock;
           lpData->dwBufferSize = dwNewSize;
       }
       else
           return;
    }

     // バッファへメッセージをコピーするためのポインタを設定
       lpCurrentPtr = lpData->lpszBuffer
          + lpData->dwBufferOffset;
       lpCurrString = (LPTSTR) lpCurrByte;

  // リンク エラー メッセージ、CR、LF をバッファへコピーする
  _tcscpy(lpCurrentPtr, lpLinkError->lpszMessageText);
  _tcscat(lpCurrentPtr, _TEXT("¥r¥n"));
  lpData->dwBufferOffset += iNeededSize;
```

```
    return;
}
```

ORCA プログラムを書く

この節では、セッションを開いて開始する ORCA プログラムの概要について説明します。また、アプリケーション オブジェクトを含むライブラリから開始せずに、アプリケーションをゼロから構築するための方法についても説明します。

ORCA プログラムの概要

ORCA インタフェースを使用するために、呼び出しプログラムは以下のことを行います。

1 ORCA セッションを開きます。

2 （オプション。呼び出す ORCA 関数により異なる）ライブラリ リストと現行のアプリケーション オブジェクトを設定します。

3 必要に応じて、ほかの ORCA 関数を呼び出します。

4 ORCA セッションを閉じます。

最初の手順： セッションを開く

ほかの ORCA 関数を呼び出す前に、セッションを開く必要があります。PBORCA_SessionOpen 関数は、このプログラムの ORCA セッションを管理するために ORCA が使用するハンドルを戻します。LPVOID として定義するハンドル タイプの HPBORCA は、どのデータ型のポインタでも扱えることを意味します。これは、ORCA 内では構造体にマップされていて呼び出しプログラムは使用できないためです。

サンプル コード

以下の C 関数のサンプルは、ORCA セッションを開いています。

```
HPBORCA WINAPI SessionOpen()
{
    HPBORCA hORCASession;
    hORCASession = PBORCA_SessionOpen();
```

```
    return hORCASession;
}
```

オプションの手順： ライブラリ リストと現行アプリケーションを設定する

ORCA プログラム作成の次の手順は、プログラムの目的に依存します。選択肢は以下のとおりです。

- プログラムがライブラリの管理、ライブラリ間でのエントリの移動、エントリのソースの検査のみを実行する場合は、ほかに必要な呼び出しはありません。ORCA セッションを続行できます。
- プログラムがほかの ORCA 関数を呼び出す場合は、ライブラリ リストを設定してから現行のアプリケーションを設定します。

PowerBuilder との比較

これは、PowerBuilder 開発環境の要件と似ています。ライブラリ ペインタでは、エントリがライブラリ リストや現行アプリケーションの中になくても、それらをある PBL からほかの PBL にコピーできます。ライブラリ リスト中にないライブラリ エントリの構文をエクスポートすることができます。ただし、現行アプリケーションのライブラリ リスト中のライブラリに対してのみエントリをインポートすることができます。

PowerBuilder の開発環境では、アプリケーション ペインタでアプリケーション オブジェクトを選択し、アプリケーション オブジェクトのプロパティ シート上でライブラリ探索パスを設定します。ORCA の場合は、最初にライブラリ リストを設定し、次にアプリケーション オブジェクトを設定します。

セッションにつき 1 回設定　ORCA セッション中に 1 度だけライブラリ リストと現行アプリケーションの設定を行うことができます。ほかのライブラリ リストとアプリケーションを使用するには、ORCA セッションを閉じてから新しいセッションを開きます。

サンプル コード

以下の C 関数のサンプルは、ライブラリ リストと現行アプリケーションを設定しています。

```
int WINAPI SetUpSession(HPBORCA hORCASession)
{
    TCHAR szApplName[36];
    int nReturnCode;
    LPTSTR lpLibraryNames[2] =
     {_TEXT("c:¥¥pbfiles¥¥demo¥¥master.pbl"),
      _TEXT("c:¥¥pbfiles¥¥demo¥¥work.pbl")};

    // ORCA 関数を呼び出す
    nReturnCode = PBORCA_SessionSetLibraryList(
```

```
        hORCASession, lpLibraryNames, 2);
    if (nReturnCode != 0)
    return nReturnCode;  // 失敗したら戻る

    // アプリケーション名を含む文字列をセットアップする
    _tcscpy(szApplName, _TEXT("demo"));

    // 最初のライブラリ中にあるアプリケーション オブジェクト
    nReturnCode = PBORCA_SessionSetCurrentAppl(
      hORCASession, lpLibraryName[0], szApplName))
    return nReturnCode;
}
```

次の手順： ORCA セッションを続行する

ライブラリ リストとアプリケーションの設定後、PBORCA_SessionOpen 関数から戻るハンドルを使用して ORCA 関数を呼び出せます。関数呼び出しの多くは非常に単純です。コールバックが必要なものなどは、少し複雑です。

コールバック関数の詳細については、13 ページの「ORCA コールバック関数について」を参照してください。

最後の手順： セッションを閉じる

ORCA プログラムの最後の手順では、セッションを閉じます。これにより、ライブラリ マネージャがセッションに関連するすべてのリソースをクリーンアップして解放します。

以下の C 関数のサンプルは、セッションを閉じています。

```
void WINAPI SessionClose(hORCASession)
{
    PBORCA_SessionClose(hORCASession);
    return;
}
```

新しいアプリケーションのブートストラップ

PowerBuilder 5.0 以降、ORCA を使用して、オブジェクトのソース コードからすべてのアプリケーションのライブラリを作成できます。既存の PBL を起動する必要はありません。

オブジェクトをインポートするには、本来、既存のアプリケーション オブジェクトを伴うライブラリが必要です。ブートストラップ処理時にアプリケーション オブジェクトに NULL 値を設定した場合、ORCA は独自のアプリケーション オブジェクトをインポートするために一時的なアプリケーション オブジェクトを使用しますが、セッションを閉じて、新しいセッションを開始し、現行アプリケーションを設定するまで、アプリケーション オブジェクトは現行アプリケーションにはなりません。

❖ 新しいアプリケーションをブートストラップするには

1. PBORCA_SessionOpen を使用して ORCA セッションを開始します。
2. PBORCA_LibraryCreate を使用して新しいライブラリを作成します。
3. PBORCA_SessionSetLibraryList を使用して新しいライブラリへのセッション用のライブラリ リストを設定します。
4. PBORCA_SessionSetCurrentAppl で、ライブラリ名とアプリケーション名として NULL 変数を渡します。
5. PBORCA_CompileEntryImportList を使用して、アプリケーション オブジェクトを新しいライブラリにインポートします。

   ここでほかのオブジェクトをインポートしないでください。

   アプリケーション オブジェクトだけをインポートしなければならない理由

   ライブラリにほかのオブジェクトをインポートすることもできますが、お勧めしません。ブートストラップ セッション中、デフォルトのアプリケーション オブジェクトは現行アプリケーションです。そのオブジェクトがアプリケーション オブジェクトに依存する場合（グローバル変数を参照するなど）、エラーが発生してインポートは失敗します。

6. セッションを閉じます。

ブートストラップしたアプリケーションの終了

ブートストラップ処理で新しいアプリケーションが開始されます。処理を完了するには、残りのオブジェクトを 1 つ以上のライブラリへインポートする必要があります。

セッション中に 1 度だけライブラリ リストと現行アプリケーションの設定をすることができるため、処理を終了するために新しい ORCA セッションを開始する必要があります。この時点で、使用するアプリケーション オブジェクトと一緒にライブラリを開いているので、処理はオブジェクトをインポートする ORCA セッションと同じです。

❖ **ブートストラップしたアプリケーションを終了するには**

1 ほかの ORCA セッションを開きます。

2 アプリケーションに必要な追加ライブラリを作成します。

3 ブートストラップの処理中に作成されたライブラリと作成されたばかりの空のライブラリを、ライブラリ リストに設定します。

4 ブートストラップの処理でインポートしたアプリケーション オブジェクトを現行アプリケーションに設定します。

5 必要に応じて各ライブラリにオブジェクトをインポートします。

ライブラリを作成するタイミング

最初のブートストラップ処理中に追加ライブラリを作成することができます。ただし、正しいアプリケーション オブジェクトが最新のものの場合、2 番目の処理になるまでオブジェクトをインポートしてはいけません。

廃止された ORCA 関数の削除

PowerBuilder 8 で、SCC API を使用した新しいソース管理へのアクセス方法が導入されました。ソース管理を行う ORCA 関数は廃止されましたが、ORCA 8 API では削除されませんでした。

PowerBuilder 9 以降、新しい ORCA ソース管理関数が追加され、古い ORCA ソース管理関数は ORCA API から削除されました。このため、既存の ORCA アプリケーションから以下の関数に対するすべての呼び出しを削除する必要があります。

- PBORCA_CheckOutEntry
- PBORCA_CheckInEntry
- PBORCA_ListCheckOutEntries

新しい ORCA 関数については、第 2 章「ORCA 関数」で説明しています。

第 2 章 ORCA 関数

この章について

この章では、ORCA 関数について説明しています。

内容

項目	ページ
例について	24
ORCA 戻り値	25
ORCA 関数（アルファベット順）	27

例について

この章の例では、ORCA セッション開始時に ORCA に関する情報を格納するための構造体を設定したと仮定しています。例中で、変数 lpORCA_Info はこの構造体のインスタンスへのポインタです。

```
typedef struct ORCA_Info {
    LPTSTR lpszErrorMessage; // メッセージ テキストへのポインタ
    HPBORCA hORCASession; // ORCA セッション ハンドル
    DWORD dwErrorBufferLen; // エラー バッファの長さ
    long lReturnCode; // 戻り値
    HINSTANCE hLibrary; // ORCA ライブラリへのハンドル
    PPBORCA_CONFIG_SESSION pConfig; // ConfigureSession
} ORCA_INFO, FAR *PORCA_INFO;
```

ORCA 戻り値

ヘッダ ファイル PBORCA.H には、以下の戻り値が定義されています。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト
-2　PBORCA_DUPOPERATION	重複した処理
-3　PBORCA_OBJNOTFOUND	オブジェクトが見つからない
-4　PBORCA_BADLIBRARY	ライブラリ名が正しくない
-5　PBORCA_LIBLISTNOTSET	ライブラリ リストが未設定
-6　PBORCA_LIBNOTINLIST	ライブラリ リストにライブラリが見つからない
-7　PBORCA_LIBIOERROR	ライブラリ I/O エラー
-8　PBORCA_OBJEXISTS	既存のオブジェクト
-9　PBORCA_INVALIDNAME	不正な名前
-10　PBORCA_BUFFERTOOSMALL	バッファ サイズが小さすぎる
-11　PBORCA_COMPERROR	コンパイル エラー
-12　PBORCA_LINKERROR	リンク エラー
-13　PBORCA_CURRAPPLNOTSET	現行のアプリケーションが未設定
-14　PBORCA_OBJHASNOANCS	オブジェクトに先祖がない
-15　PBORCA_OBJHASNOREFS	オブジェクトは未参照
-16　PBORCA_PBDCOUNTERROR	無効な PBD 数
-17　PBORCA_PBDCREATERROR	PBD 作成エラー
-18　PBORCA_CHECKOUTERROR	ソース管理エラー（廃止）
-19　PBORCA_CBCREATEERROR	ComponentBuilder クラスのインスタンス化は不可
-20　PBORCA_CBINITERROR	コンポーネント ビルダの Init メソッド失敗
-21　PBORCA_CBBUILDERROR	コンポーネント ビルダの BuildProject メソッド失敗
-22　PBORCA_SCCFAILURE	ソース管理に接続できない
-23　PBORCA_REGREADERROR	レジストリを読み込めない
-24　PBORCA_SCCLOADDLLFAILED	DLL をロードできない
-25　PBORCA_SCCINITFAILED	SCC 接続を初期化できない
-26　PBORCA_OPENPROJFAILED	SCC プロジェクトを開けない
-27　PBORCA_TARGETNOTFOUND	ターゲット ファイルが見つからない
-28　PBORCA_TARGETREADERR	ターゲット ファイルの読み取り不可

戻り値	説明
-29 PBORCA_GETINTERFACEERROR	SCC インタフェースへのアクセス不可
-30 PBORCA_IMPORTONLY_REQ	SCC 接続オフラインが IMPORTONLY リフレッシュ オプションを要求する
-31 PBORCA_GETCONNECT_REQ	SCC 接続オフラインは Exclude_Checkout と一緒に GetConnectProperties を要求する
-32 PBORCA_PBCFILE_REQ	Exclude_Checkout と一緒に使用する SCC 接続オフラインは PBC ファイルを要求する

PBORCA_ApplicationRebuild

機能

ライブラリ リストに含まれているライブラリのすべてのオブジェクトをコンパイルします。必要に応じて、相互依存関係を解決するために複数のパスでコンパイルを行います。

構文

```
INT PBORCA_ApplicationRebuild ( HPBORCA hORCASession,
        PBORCA_REBLD_TYPE eRebldType,
        PBORCA_ERRPROC pCompErrProc,
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
eRebldType	再構築する種類を指定するカタログ データ型 PBORCA_REBLD_TYPE の値。値は以下のとおりです。 PBORCA_FULL_REBUILD PBORCA_INCREMENTAL_REBUILD PBORCA_MIGRATE
pCompErrorProc	コールバック関数 PBORCA_ApplicationRebuild へのポインタ。コールバック関数は、オブジェクトのコンパイルで発生する各エラーに対して呼び出されます。 ORCA がコールバック関数に渡す情報（エラー レベル、メッセージ番号、メッセージ テキスト、行番号、カラム番号）は、PBORCA_COMPERR 構造体に格納されます。オブジェクト名とスクリプト名は、メッセージ テキストに含まれます。 コールバック関数を使用しない場合は、*pCompErrorProc* に 0 を設定します。
pUserData	コールバック関数 PBORCA_CompileEntryImport に渡されるユーザ データへのポインタ 通常、ユーザ データにはコールバック関数が格納するエラー情報とバッファ サイズの情報を持つバッファまたはバッファへのポインタが含まれます。 コールバック関数を使用しない場合は、pUserData に 0 を設定します。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト
-13　PBORCA_CURRAPPLNOTSET	現行のアプリケーションが未設定

解説

この関数を呼び出す前に、ライブラリ リストと現行のアプリケーションを設定する必要があります。

コンパイル関数を使用する場合、オブジェクトがコンパイルされる順序によりエラーが生じることがあります。2 つのオブジェクトが相互に参照している場合、単純なコンパイルは失敗します。オブジェクトの依存関係によるエラーを解決するには、PBORCA_ApplicationRebuild を使用します。PBORCA_ApplicationRebuild は、コンパイル処理の際に複数のパスを持つ循環相互参照の問題を解決します。

オブジェクトが影響を受ける再構築の種類を指定します。選択肢は以下のとおりです。

- **インクリメンタル再構築** 最後のアプリケーション構築以降に変更したオブジェクトが参照しているすべてのオブジェクトとライブラリを更新します。
- **フル再構築** アプリケーション中のすべてのオブジェクトとライブラリを更新します。
- **移行** アプリケーション中のすべてのオブジェクトとライブラリを、現行バージョンへ更新します。オブジェクトが旧バージョンで構築されている場合のみ対象となります。

例

この例は、現行ライブラリ リスト上にあるライブラリ中のすべてのオブジェクトを再コンパイルします。

エラーが発生するたびに、PBORCA_ApplicationRebuild はコールバック関数 CompileEntryErrors を呼び出します。CompileEntryErrors 用に記述するコードでは、lpUserData が指し示すバッファにエラー メッセージを格納します。

```
PBORCA_ERRPROC fpError;
int nReturnCode;

fpError = (PBORCA_ERRPROC) ErrorProc;
nReturnCode = PBORCA_ApplicationRebuild(
    lpORCA_Info->hORCASession,
    PBORCA_FULL_REBUILD,
    fpError, lpUserData);
```

コールバック用データ バッファの設定についての詳細は、15 ページの「コールバック関数の内容」と PBORCA_LibraryDirectory の例を参照してください。

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_CompileEntryRegenerate
PBORCA_CompileEntryImport
PBORCA_CompileEntryImportList

PBORCA_BuildProject

機能

プロジェクト オブジェクトの指定に従い、EAServer コンポーネントを配布します。

構文

```
INT PBORCA_BuildProject ( HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTRlpszProjectName,
        PBORCA_BLDPROC pBuildErrProc,
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	プロジェクト エントリを含むライブラリ のファイル名
lpszProjectName	配布情報を含むプロジェクト オブジェクト
pBuildErrProc	エラーのコールバック関数 PBORCA_BuildProject へのポインタ コールバック関数を使用しない場合は、*pBuildErrProc* に NULL を設定します。
pUserData	コールバック関数に渡されるユーザ データへのポインタ

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト
-19　PBORCA_CBCREATEERROR	コンポーネント ビルダ クラスの未作成
-20　PBORCA_CBINITERROR	EAServer 接続の初期化失敗
-21　PBORCA_CBBUILDERROR	エラーによる配布失敗

解説

戻されるエラー情報について　エラーのコールバック関数 PBORCA_BuildProject は、次の構造体中にエントリに関する情報を格納します。引数 *pBuildErrProc* に構造体へのポインタを渡します。

```
typedef struct PBORCA_blderr
{
    LPTSTR lpszMessageText; // メッセージ テキスト へのポインタ
} PBORCA_BLDERR, FAR *PPBORCA_BLDERR;
```

典型的なコールバック関数　コールバック関数には次のシグネチャがあります。

```
typedef PBCALLBACK (void, *PPBORCA_BLDPROC)
(PBORCA_BLDERR, LPVOID);
```

関連項目

PBORCA_BuildProjectEx

PBORCA_BuildProjectEx

機能

プロジェクト オブジェクトの指示に従い、EAServer コンポーネントを配布します。指定した引数値によりサーバとポートのプロパティが上書きされます。ただし、これらのプロパティがサーバ プロファイルに設定されている場合は、指定した引数値によって上書きされません。サーバ プロファイルおよびプロジェクト オブジェクトのプロパティを上書きする場合は、PBORCA_BuildProjectWithOverrides を使用します。

構文

```
INT PBORCA_BuildProjectEx ( HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTRlpszProjectName,
        PBORCA_BLDPROC pBuildErrProc,
        LPTSTR lpszServerName,
        INT iPort,
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	プロジェクト エントリを含むライブラリのファイル名
lpszProjectName	配布情報を含むプロジェクト オブジェクト
pBuildErrProc	エラーのコールバック関数 PBORCA_BuildProject へのポインタ コールバック関数を使用しない場合は、*pBuildErrProc* に NULL を設定します。
lpszServerName	EAServer 配布のサーバ名。この値は、プロジェクト オブジェクトのサーバ プロパティを上書きします。
iPort	EAServer 配布のポート番号。この値は、プロジェクト オブジェクトのサーバ プロパティを上書きします。
pUserData	コールバック関数に渡されるユーザ データへのポインタ

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-19 PBORCA_CBCREATEERROR	コンポーネント ビルダ クラス未作成
-20 PBORCA_CBINITERROR	EAServer 接続の初期化失敗
-21 PBORCA_CBBUILDERROR	エラーにより配布失敗

関連項目

PBORCA_BuildProject
PBORCA_BuildProjectWithOverrides

PBORCA_BuildProjectWithOverrides

機能

プロジェクト オブジェクトの仕様に従って EAServer コンポーネントを配布しますが、指定した引数値に基づいて強制的に上書きします。このメソッドは、PBORCA_BuildProjectEx に似ていますが、サーバのログイン ID とパスワードが追加の入力値として必要です。また、これらの値は、サーバ プロファイルまたはプロジェクト オブジェクトに設定された値を上書きします。

構文

```
INT PBORCA_BuildProjectWithOverrides ( HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTRlpszProjectName,
        PBORCA_BLDPROC pBuildErrProc,
        LPTSTR lpszServerName,
        INT iPort,
        LPTSTR lpszUserid,
        LPTSTR lpszPassword,
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	プロジェクト エントリを含むライブラリのファイル名
lpszProjectName	配布情報を含むプロジェクト オブジェクト
pBuildErrProc	エラーのコールバック関数 PBORCA_BuildProject へのポインタ コールバック関数を使用しない場合は、*pBuildErrProc* に NULL を設定します。
lpszServerName	EAServer 配布のサーバ名。この値は、プロジェクト オブジェクトのサーバ プロパティを上書きします。
iPort	EAServer 配布のポート番号。この値は、プロジェクト オブジェクトのサーバ プロパティを上書きします。
lpszUserid	サーバのログイン ID。この値は、プロジェクト オブジェクトのログイン ID を上書きします。
lpszPassword	サーバのパスワード。この値は、プロジェクト オブジェクトのログイン パスワードを上書きします。
pUserData	コールバック関数に渡されるユーザ データへのポインタ

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト

戻り値	説明
-19 PBORCA_CBCREATEERROR	コンポーネント ビルダ クラス未作成
-20 PBORCA_CBINITERROR	EAServer 接続の初期化失敗
-21 PBORCA_CBBUILDERROR	エラーにより配布失敗

関連項目

PBORCA_BuildProject
PBORCA_BuildProjectEx

PBORCA_CompileEntryImport

機能

PowerBuilder オブジェクトのソース コードをライブラリへインポートしてコンパイルします。

構文

```
INT PBORCA_CompileEntryImport ( HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTR lpszEntryName,
        PBORCA_TYPE otEntryType,
         lpszComments,
        LPTSTR lpszEntrySyntax,
        LONG lEntrySyntaxBuffSize,
        PBORCA_ERRPROC pCompErrorProc,
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	オブジェクトにインポートするライブラリのファイル名である文字列値へのポインタ
lpszEntryName	インポートされるオブジェクトの名前の文字列値へのポインタ
otEntryType	インポートしたエントリのオブジェクトの種類を指定する PBORCA_TYPE カタログ データ型の値。値は以下のとおりです。 PBORCA_APPLICATION PBORCA_BINARY PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW
lpszComments	オブジェクトに提供しているコメントの文字列値へのポインタ
lpszEntrySyntax	インポートされるオブジェクトのソース コードへのバッファへのポインタ。ソース コード中にエクスポート ヘッダが存在している場合、無視されます。*lpszEntrySyntax* のソース エンコーディングは、PBORCA_CONFIG_SESSION 構造体中の eImportEncoding プロパティで指定します。
lEntrySyntaxBuffSize	*lpszEntrySyntax* バッファの長さ。この長さは、ソース エンコーディングを考慮せずにバイトで指定します。

引数	説明
pCompErrorProc	コールバック関数 PBORCA_CompileEntryImport へのポインタ。インポートしたオブジェクトのコンパイルによりエラーが発生するたびに呼び出されるコールバック関数です。 ORCA がコールバック関数に渡す情報（エラー レベル、メッセージ番号、メッセージ テキスト、行番号、カラム番号）は、PBORCA_COMPERR 構造体に格納されます。オブジェクト名とスクリプト名は、メッセージ テキストに含まれます。 コールバック関数を使用しない場合は、*pCompErrorProc* に 0 を設定します。
pUserData	コールバック関数 PBORCA_CompileEntryImport に渡されるユーザ データへのポインタ 通常、ユーザ データにはコールバック関数が格納するエラー情報とバッファ サイズの情報を持つバッファまたはバッファへのポインタが含まれます。 コールバック関数を使用しない場合は、pUserData に 0 を設定します。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-4 PBORCA_BADLIBRARY	不正なライブラリ名、ライブラリが見つからない、またはオブジェクトをライブラリに保存できない
-6 PBORCA_LIBNOTINLIST	リスト中にないライブラリ
-8 PBORCA_COMPERROR	コンパイル エラー
-9 PBORCA_INVALIDNAME	名前が PowerBuilder の命名規則に違反している
-13 PBORCA_CURRAPPLNOTSET	現行アプリケーションが設定されていない

解説

この関数を呼び出す前に、ライブラリ リストと現行のアプリケーションを設定する必要があります。

PowerBuilder

PowerBuilder 10 以降では、インポートするオブジェクトのソース エンコーディングを指定する必要があります。これを行うためには、PBORCA_CONFIG_SESSION 構造体の中で eImportEncoding プロパティを設定し、PBORCA_ConfigureSession を呼び出します。ANSI クライアントの場合、デフォルトのソース エンコーディングは ANSI/DBCS であり、Unicode クライアントの場合、デフォルトのソース エンコーディングは Unicode です。

埋め込みバイナリ情報と共にオブジェクトをインポート　OLE オブジェクトのような埋め込みバイナリ データを含むオブジェクトをインポートするために、PBORCA_CompileEntryImport に対する 2 つの個別の呼び出しが必要です。最初の呼び出しで、ソース コンポーネントをインポートします。2 番目の呼び出しでは、otEntryType 引数を使用して PBORCA_BINARY を設定し、lpszEntrySyntax 引数を使用してバイナリ ヘッダ レコードの開始位置を指定して、バイナリ コンポーネントをインポートします。

エラーが発生した場合　オブジェクトのインポート処理中にエラーが発生した場合、オブジェクトはライブラリに入りますが編集が必要なこともあります。オブジェクトのエラーが些細な場合は、ペインタで開いて編集できます。深刻なエラーの場合は、ペインタで開こうとすると失敗するので、オブジェクトをエクスポートし、ソース コードを修正してから再度インポートします。エラーの原因がオブジェクトをコンパイルする順番による場合は、すべてのオブジェクトをインポートした後に PBORCA_ApplicationRebuild 関数を呼び出します。

注意

既存のエントリと同じ名前のエントリをインポートすると、インポートの前に、既存のエントリが削除されます。インポートに失敗した場合でも、既存のオブジェクトはすでに削除されています。

エラーのコールバック処理については、PBORCA_CompileEntryImportList を参照してください。

例

この例は、d_labels という 名前のデータウィンドウを DWOBJECTS.PBL ライブラリ にインポートします。ソース コード は、szEntrySource という 名前のバッファに格納されます。

エラーが発生するたびに、PBORCA_CompileEntryImport はコールバック関数 CompileEntryErrors を呼び出します。CompileEntryErrors 用に記述するコードでは、lpUserData が指し示すバッファにエラー メッセージを格納します。

```
PBORCA_ERRPROC fpError;
int nReturnCode;

fpError = (PBORCA_ERRPROC) ErrorProc;
nReturnCode = PBORCA_CompileEntryImport(
    lpORCA_Info->hORCASession,
    _TEXT("c:¥¥app¥¥dwobjects.pbl)",
    _TEXT("d_labels"), PBORCA_DATAWINDOW,
    (LPTSTR) szEntrySource, 60000,
    fpError, lpUserData);
```

24 ページの「例について」で紹介しているように、これらの例のセッション情報は ORCA_Info データ構造体に保存されます。

この例は、ソース ファイルを読み、ソース ファイルのエンコーディング形式を判別し、PBL にインポートします。ファイルに埋め込みバイナリ オブジェクトが含まれている場合、PBORCA_CompileEntryImport の 2 番目の呼び出しを使用してそのデータもインポートします。

```
//    ヘッダ、定義、型定義
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <tchar.h>
extern "C" {
#include "pborca.h"
}

//    グローバル変数
HPBORCA  hPbOrca;
PBORCA_ERRPROC  fpError;
//    関数宣言
void CALLBACK ErrorProc(PBORCA_COMPERR *lpCompErr,
      LPVOID lpUserData);


// 名前 :Impbin.cpp
// 概要 : w_edit_connect.srw（埋め込み OLE オブジェクトを
// 含む）を作業用 PBL にインポートします。
//  この例は、ANSI クライアントとしても Unicode クライアント
//  としてもコンパイルすることができます。Unicode として
//  コンパイルするには、/DUNICODE /D_UNICODE コンパイラ
//  ディレクティブを使用します。
#if defined (UNICODE)
INT wmain ( int argc, wchar_t *argv[])
#else
INT main ( int argc, char *argv[])
#endif
{
```

```
                    LPTSTR        pszLibraryName[5];
                    LPTSTR        pszImportFile;
                    HANDLE        hOpenFile = NULL;
                    INT           iErrCode;
                    BOOL             rc;
                    wchar_t       chMarker;
                    unsignedchar  chMarker3;
                    DWORD         dBytesRead;
                    DWORD         dFileSize;
                    PBORCA_CONFIG_SESSION  Config;
                    LPBYTE        pReadBuffer = NULL;
                    LPBYTE        pEndBuffer;
                    INT           iSourceSize;
                    INT           iBinarySize;

pszLibraryName[0] =
_TEXT("c:¥¥pb12.5¥¥main¥¥pbls¥¥qadb¥¥qadbtest¥¥qadbtest.pbl");
pszLibraryName[1] =
_TEXT("c:¥¥pb12.5¥¥main¥¥pbls¥¥qadb¥¥shared_obj¥¥shared_obj.pbl");
pszLibraryName[2] =
_TEXT("c:¥¥pb12.5¥¥main¥¥pbls¥¥qadb¥¥datatypes¥¥datatype.pbl");
pszLibraryName[3] =
_TEXT("c:¥¥pb12.5¥¥main¥¥pbls¥¥qadb¥¥chgreqs¥¥chgreqs.pbl");
pszLibraryName[4] =
_TEXT("c:¥¥pb12.5¥¥main¥¥orca¥¥testexport¥¥work.pbl");
pszImportFile =
_TEXT("c:¥¥pb12.5¥¥main¥¥pbls¥¥qadb¥¥qadbtest¥¥w_edit_connect.srw");
memset(&Config, 0x00, sizeof(PBORCA_CONFIG_SESSION));

                PbOrca = PBORCA_SessionOpen();
                //  work.pbl を削除して再作成します。
                iErrCode = PBORCA_LibraryDelete(hPbOrca,
                pszLibraryName[4]);
                iErrCode = PBORCA_LibraryCreate(hPbOrca,
                   pszLibraryName[4], _TEXT("work pbl"));
                iErrCode = PBORCA_SessionSetLibraryList(hPbOrca,
                    pszLibraryName, 5);

                if (iErrCode != PBORCA_OK)
                    goto TestExit;
                iErrCode = PBORCA_SessionSetCurrentAppl(hPbOrca,
                   pszLibraryName[0], _TEXT("qadbtest"));
                if (iErrCode != PBORCA_OK)
                   goto TestExit;

                //  PBORCA_CompileEntryImport はエクスポート ヘッダを
                //  無視します。このため、ORCA アプリケーションは、インポート
                //  ファイルのソース エンコーディングをプログラムで判断する必要
```

```
//  があります。これは、ファイルの先頭 2 バイトまたは 3 バイト
//  を読み込むことで判断されます。
hOpenFile = CreateFile(pszImportFile, GENERIC_READ, 0,
    NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
if( hOpenFile == INVALID_HANDLE_VALUE )
    goto TestExit;

rc =  ReadFile(hOpenFile, (LPVOID)&chMarker,
    sizeof(wchar_t), &dBytesRead, NULL);
if( rc )
{
    if (chMarker == 0xfeff)
       Config.eImportEncoding = PBORCA_UNICODE;
    else if (chMarker == 0xbbef)
    {
       rc =  ReadFile(hOpenFile, (LPVOID)&chMarker3,
          sizeof(CHAR),&dBytesRead, NULL);
       if (chMarker3 == 0xbf)
          Config.eImportEncoding = PBORCA_UTF8;
       }
    else if (memcmp((LPBYTE) &chMarker, "HA", 2) == 0)
       Config.eImportEncoding = PBORCA_HEXASCII;
    else
       Config.eImportEncoding = PBORCA_ANSI_DBCS;

//  ここではソース バッファ用にメモリを割り当ててファイル全体を
//  読み込みます。
SetFilePointer( hOpenFile, 0, NULL,FILE_BEGIN);
dFileSize = GetFileSize(hOpenFile, NULL) ;
pReadBuffer = (LPBYTE) malloc((size_t) dFileSize + 2);
rc =  ReadFile(hOpenFile, pReadBuffer, dFileSize,
     &dBytesRead, NULL);

//  strstr() 呼び出しを可能にするために Null 区切り文字を
//  追加します。
pEndBuffer = pReadBuffer + dFileSize;
memset(pEndBuffer, 0x00, 2);  // unicode EOF マーカー
if (!rc)
    goto TestExit;

//  バイナリ コンポーネントを含むオブジェクトかどうかを
//  判断します。含む場合には、PBORCA_CompileEntryImport
//  の 2 つの呼び出しを個別に行います。
if (Config.eImportEncoding == PBORCA_UNICODE)
{
    LPWSTR pszUniBinHeader;
    LPWSTR pUniBinStart;
```

```
        pszUniBinHeader = "PowerBuilder バイナリ データ
        セクション 開始 ";
        pUniBinStart = wcsstr((const wchar_t *)
            pReadBuffer, pszUniBinHeader);

        if (pUniBinStart)
        {
        pEndBuffer = (LPBYTE) pUniBinStart;
        iSourceSize = (INT) (pEndBuffer - pReadBuffer);
        iBinarySize = (INT) (dFileSize - iSourceSize);
        }
        else
        {
        iSourceSize = (INT) dFileSize;
        iBinarySize = 0;
        }
}
else
{
        LPSTR  pszAnsiBinHeader;
        LPSTR  pAnsiBinStart;
        pszAnsiBinHeader = "PowerBuilder バイナリ データ
        セクション 開始 ";
        pAnsiBinStart = (LPSTR) strstr((const char *)
           pReadBuffer, (const char *) pszAnsiBinHeader);
        if (pAnsiBinStart)
        {
           pEndBuffer = (LPBYTE) pAnsiBinStart;
           iSourceSize = (INT) (pEndBuffer - pReadBuffer);
           iBinarySize = (INT) (dFileSize - iSourceSize);
        }
        else
        {
           iSourceSize = (INT) dFileSize;
           iBinarySize = 0;
        }
 }
//  適切なソース エンコーディングを読み取るために ORCA
//  セッションを設定します。
iErrCode = PBORCA_ConfigureSession(hPbOrca, &Config);

//  エントリのソースをインポートします。
fpError = (PBORCA_ERRPROC) ErrorProc;
iErrCode = PBORCA_CompileEntryImport(
    hPbOrca,
    pszLibraryName[4],
```

```
        _TEXT("w_edit_connect"), PBORCA_WINDOW,
        _TEXT("埋め込み OLE オブジェクトをテスト"),
        (LPTSTR) pReadBuffer, iSourceSize,
              fpError, NULL);
    if (iErrCode != PBORCA_OK)
       goto TestExit;
    if (iBinarySize > 0)
    {
       iErrCode = PBORCA_CompileEntryImport(
           hPbOrca,
           pszLibraryName[4],
           _TEXT("w_edit_connect"), PBORCA_BINARY,
           NULL,
          (LPTSTR) pEndBuffer, iBinarySize,
               fpError, NULL);
    }
    }
    TestExit:
    if ( hOpenFile != INVALID_HANDLE_VALUE )
       CloseHandle(hOpenFile);
    if (pReadBuffer)
       free(pReadBuffer);
    PBORCA_SessionClose(hPbOrca);
    return iErrCode;
    }
    // オブジェクトをコンパイルするための呼び出しで使用されたエラー
    // プロシージャをコールバックします。この例では、ORCA クラスの
    // メソッドではなく、プログラムによって指定されます。
    void CALLBACK ErrorProc(PBORCA_COMPERR *lpCompErr,
        LPVOID lpUserData)
    {
     _tprintf(_TEXT("%s ¥n"), lpCompErr->lpszMessageText );
    }
```

関連項目

PBORCA_LibraryEntryExport
PBORCA_CompileEntryImportList
PBORCA_CompileEntryRegenerate
PBORCA_ApplicationRebuild

PBORCA_CompileEntryImportList

機能

PowerBuilder オブジェクトの一覧用のソース コードをライブラリにインポートしてコンパイルします。インポートされる各オブジェクト名は、配列に保持されます。ほかの配列には、送り先となるライブラリ、オブジェクトの種類、コメント、およびソース コードが保持されます。配列には、各オブジェクト用の要素があります。

構文

```
INT PBORCA_CompileEntryImportList ( PBORCA hORCASession,
        LPTSTR far *pLibraryNames,
        LPTSTR far *pEntryNames,
        PBORCA_TYPE far *otEntryTypes,
        LPTSTR far *pComments,
        LPTSTR far *pEntrySyntaxBuffers,
        LONG far *pEntrySyntaxBuffSizes,
        INT iNumberOfEntries,
        PBORCA_ERRPROC pCompErrorProc,
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
**pLibraryNames*	対応するオブジェクトをインポートするライブラリのファイル名を値とする string 型配列へのポインタ
**pEntryNames*	対応するライブラリにインポートするオブジェクト名を値とする string 型配列へのポインタ
**otEntryTypes*	カタログ データ型の PBORCA_TYPE で表示されるライブラリ エントリのオブジェクト型の配列を指し示すポインタ。値は以下のとおりです。 PBORCA_APPLICATION PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT PBORCA_BINARY
**pComments*	対応するオブジェクトのコメントを値とする string 型配列へのポインタ
**pEntrySyntaxBuffers*	対応するオブジェクトのソース コードを値とする string 型配列へのポインタ
**pEntrySyntaxBuffSizes*	*pEntrySyntaxBuffers* が指し示す文字列長の値を持つ long 型配列へのポインタ

引数	説明
iNumberOfEntries	インポートされるエントリの数で、すべての配列引数の配列長と同じ
pCompErrorProc	コールバック関数の PBORCA_CompileEntryImportList へのポインタ。コールバック関数は、インポートされたオブジェクトのコンパイル時にエラーが発生するたびに呼び出されます。 ORCA がコールバック関数に渡す情報（エラー レベル、メッセージ番号、メッセージ テキスト、行番号、カラム番号）は、PBORCA_COMPERR 構造体に格納されます。オブジェクト名とスクリプト名は、メッセージ テキストに含まれます。 コールバック関数を使用しない場合は、*pCompErrorProc* に 0 を設定します。
pUserData	コールバック関数 PBORCA_CompileEntryImportList に渡されるユーザ データへのポインタ 通常、ユーザ データは、ユーザ情報とバッファのサイズに関する情報と一緒にコールバック関数の形式でバッファへのポインタまたはバッファに含まれています。 コールバック関数を使用しない場合は、pUserData に 0 を設定します。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-4 PBORCA_BADLIBRARY	ライブラリ名が正しくない、ライブラリが見つからない、またはオブジェクトがライブラリ中に保存できない
-6 PBORCA_LIBNOTINLIST	リスト中にないライブラリ
-7 PBORCA_LIBIOERROR	ライブラリ I/O エラー
-8 PBORCA_COMPERROR	コンパイル エラー
-9 PBORCA_INVALIDNAME	名前が PowerBuilder の命名規則に違反している
-13 PBORCA_CURRAPPLNOTSET	現行アプリケーションが設定されていない

解説

この関数を呼び出す前に、ライブラリ リストと現行のアプリケーションを設定する必要があります。

PBORCA_CompileEntryImportList は、相互に関連するいくつかのオブジェクト（たとえば、ウィンドウ、メニュー、またはオブジェクトが使用するユーザ オブジェクト）をインポートする際に役立ちます。

インポートされたオブジェクトの処理方法　ORCA は、リスト中のすべてのオブジェクトをインポートし、各オブジェクト型の定義をコンパイルします。エラーがない場合は、ORCA は一覧にあるすべてのライブラリのすべてのオブジェクトをコンパイルします。

オブジェクトの依存性

インポートするオブジェクトの一覧では、先祖オブジェクトを最初にインポートするために、先祖オブジェクトを子孫オブジェクトの前に記述します。

オブジェクトの一覧には、参照されるオブジェクトを最初にインポートするために、ユーザ オブジェクトを参照するオブジェクトの前にそれを記述します。

オブジェクトが相互に参照している場合には、PBORCA_ApplicationRebuild を呼び出し、エラーのないコンパイルを行います。

インポートしたオブジェクトの情報配列の格納　インポートした各オブジェクトの情報は、複数の並列した配列に含まれています。たとえば、d_labels という名前のデータウィンドウがオブジェクト名配列（サブスクリプト 2）の 3 番目のエレメントである場合、その格納先ライブラリ名へのポインタはライブラリ名配列の 3 番目のエレメントにあり、そのオブジェクト型はオブジェクト型配列の 3 番目のエレメントにあり、そのソース コード バッファへのポインタは構文バッファ配列の 3 番目のエレメントにあります。

PBORCA_BINARY を使用したエントリ タイプの指定　OLE オブジェクトのような埋め込みバイナリ情報を含むエントリをインポートまたはエクスポートする場合、この PBORCA_TYPE のカタログ データ型の値を使用します。バイナリ情報は、事前にエクスポートしたバイナリ データを HEXAscii 表示で格納しているバッファからインポートされます。

インポート時に PBORCA_BINARY を使用するサンプル コードについては、35 ページの「例」を参照してください。

エラーが発生した場合　オブジェクトのインポート中にエラーが発生すると、そのオブジェクトはライブラリには入りますが編集が必要な場合があります。オブジェクトのエラーが些細な場合は、ペインタで開いて編集できます。深刻なエラーの場合は、ペインタで開こうとすると失敗するので、オブジェクトをエクスポートし、ソース コードを修正してから再度インポートします。エラーの原因がオブジェクトをコンパイルする順番による場合は、すべてのオブジェクトをインポートした後に PBORCA_ApplicationRebuild 関数を呼び出します。

注意

既存のエントリと同じ名前のエントリをインポートすると、インポートの前に、既存のエントリが削除されます。インポートが失敗した場合、古いオブジェクトはすでに削除されています。

コールバック関数内でのエラー処理について　コンパイル中に発生する各エラーに関して、ORCA は *pCompErrorProc* 内で指し示しているコールバック関数を呼び出します。エラー情報が呼び出しプログラムにどのようにして戻るかは、コールバック関数での記述により異なります。ORCA は PBORCA_COMPERR の構造体を使用してエラー情報をコールバック関数に渡します。コールバック関数はその構造体を調べ、*pUserData* が指し示しているバッファの中に必要な情報を格納します。

エラーの発生件数は予想できないため、*pUserData* バッファのサイズを見積もることは困難です。バッファ中の使用可能なスペースの把握は、コールバック関数に任せます。

例

この例は、3 つのオブジェクトを 2 つのライブラリにインポートするために必要な配列を作成し、オブジェクトをインポートします。例では、オブジェクトのソース コードに変数 szWindow1、szWindow2、szMenu1 を設定済みであると仮定しています。

エラーが発生するたびに、PBORCA_CompileEntryImportList はコールバック関数である CompileEntryErrors を呼び出します。CompileEntryErrors 用に記述したコード中では、lpUserData が指し示すバッファにエラー メッセージを格納します。例では、lpUserData バッファはすでに設定済みです。

```
LPTSTR lpLibraryNames[3];
LPTSTR lpObjectNames[3];
PBORCA_TYPE ObjectTypes[3];
LPTSTR lpObjComments[3];
LPTSTR lpSourceBuffers[3];
long BuffSizes[3];
```

```
PBORCA_ERRPROC fpError;
int nReturnCode;

fpError = (PBORCA_ERRPROC) ErrorProc;
// Unicode のソース エンコーディングを指定
lpORCA_Info->pConfig->eImportEncoding =
   PBORCA_UNICODE;
PBORCA_ConfigureSession(lpORCA_Info->hORCASession,
   lpORCA_Info->pConfig);

// ライブラリ名を指定
lpLibraryNames[0] =
    _TEXT("c:¥¥sybase¥¥pb12.5¥¥demo¥¥windows.pbl");
lpLibraryNames[1] =
    _TEXT("c:¥¥sybase¥¥pb12.5¥¥demo¥¥windows.pbl");
lpLibraryNames[2] =
    _TEXT("c:¥¥sybase¥¥pb12.5¥¥demo¥¥menus.pbl");

// オブジェクト名を指定
lpObjectNames[0] = _TEXT("w_ancestor");
lpObjectNames[1] = _TEXT("w_descendant");
lpObjectNames[2] = _TEXT("m_actionmenu");

// オブジェクト型配列を設定
ObjectTypes[0] = PBORCA_WINDOW;
ObjectTypes[1] = PBORCA_WINDOW;
ObjectTypes[2] = PBORCA_MENU;

// オブジェクトのコメントを指定
lpObjComments[0] = _TEXT("Ancestor window");
lpObjComments[1] = _TEXT("Descendent window");
lpObjComments[2] = _TEXT("Action menu");

// ソース コードへのポインタを設定
lpSourceBuffers[0] = (LPTSTR) szWindow1;
lpSourceBuffers[1] = (LPTSTR) szWindow2;
lpSourceBuffers[2] = (LPTSTR) szMenu1;

// ソース コード長配列を設定
BuffSizes[0] = _tcslen(szWindow1)*2;
    //Unicode ソース バッファ
BuffSizes[1] = _tcslen(szWindow2)*2;
    // サイズは常にバイトで指定
BuffSizes[2] = _tcslen(szMenu1)*2;

nReturnCode = PBORCA_CompileEntryImportList(
```

```
        lpORCA_Info->hORCASession,
        lpLibraryNames, lpObjectNames, ObjectTypes,
        lpObjComments, lpSourceBuffers, BuffSizes, 3,
        fpError, lpUserData );
```

コールバック用データ バッファの設定についての詳細は、15 ページの「コールバック関数の内容」と PBORCA_LibraryDirectory の例を参照してください。

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_LibraryEntryExport
PBORCA_CompileEntryImport
PBORCA_CompileEntryRegenerate
PBORCA_ApplicationRebuild

PBORCA_CompileEntryRegenerate

機能

PowerBuilder ライブラリ中のオブジェクトをコンパイルします。

構文

```
INT PBORCA_CompileEntryRegenerate ( PBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTR lpszEntryName,
        PBORCA_TYPE otEntryType,
        PBORCA_ERRPROC pCompErrorProc,
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	コンパイルされるオブジェクトを含むライブラリのファイル名を値として持つ文字列を指し示すポインタ
lpszEntryName	コンパイルされるオブジェクト名を値として持つ文字列を指し示すポインタ
otEntryType	コンパイルされるエントリのオブジェクト型を指定する PBORCA_TYPE のカタログ データ型の値。値は以下のとおりです。 PBORCA_APPLICATION PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT
pCompErrorProc	PBORCA_CompileEntryRegenerate コールバック関数へのポインタ。オブジェクトをコンパイルする際にエラーが発生すると、各エラーに対してコールバック関数が呼び出されます。 ORCA がコールバック関数に渡す情報（エラー レベル、メッセージ番号、メッセージ テキスト、行番号、カラム番号）は、PBORCA_COMPERR 構造体に格納されます。オブジェクト名とスクリプト名は、メッセージ テキストに含まれます。 コールバック関数を使用しない場合は、*pCompErrorProc* に 0 を設定します。

引数	説明
pUserData	PBORCA_CompileEntryRegenerate コールバック関数に渡されるユーザ データへのポインタ 通常、ユーザ データにはコールバック関数が格納するエラー情報とバッファ サイズの情報を持つバッファまたはバッファへのポインタが含まれます。 コールバック関数を使用しない場合は、pUserData に 0 を設定します。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-3 PBORCA_OBJNOTFOUND	オブジェクトが見つからない
-4 PBORCA_BADLIBRARY	ライブラリ名が正しくない
-5 PBORCA_LIBLISTNOTSET	ライブラリ リストが未設定
-6 PBORCA_LIBNOTINLIST	ライブラリ リストにライブラリが見つからない
-7 PBORCA_LIBIOERROR	ライブラリ I/O エラー
-11 PBORCA_COMPERROR	コンパイル エラー

解説

この関数を呼び出す前に、ライブラリ リストと現行のアプリケーションを設定する必要があります。

エラーが発生した場合　再生成する間に発生したエラーを修正するには、ソース コードをエクスポートし、エラーを修正してそのオブジェクトをインポートするという作業を、正常にコンパイル処理されるまで繰り返します。

些細なエラーがあるオブジェクトは PowerBuilder ペインタで開いて修正することもできますが、大きなエラーがあるオブジェクトの場合はエクスポートして修正する必要があります。

エラーのコールバック処理については、PBORCA_CompileEntryImportList を参照してください。

例

この例は、DWOBJECTS.PBL ライブラリ中の d_labels という名前のデータウィンドウをコンパイルします。

エラーが発生するたびに、PBORCA_CompileEntryRegenerate はコールバック関数の CompileEntryErrors を呼び出します。CompileEntryErrors 用に記述したコード中では、lpUserData が指し示すバッファにエラー メッセージを格納します。例では、lpUserData バッファはすでに設定済みです。

```
PBORCA fpError;
int nReturnCode;

fpError = (PBORCA_ERRPROC) ErrorProc;
nReturnCode = PBORCA_CompileEntryRegenerate(
    lpORCA_Info->hORCASession,
    _TEXT("c:¥¥app¥¥dwobjects.pbl"),
    _TEXT("d_labels"), PBORCA_DATAWINDOW,
    fpError, lpUserData );
```

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_LibraryEntryExport
PBORCA_CompileEntryImport
PBORCA_CompileEntryImportList
PBORCA_ApplicationRebuild

PBORCA_ConfigureSession

機能

PBORCA_ConfigureSession は、PowerBuilder 10 との下位互換性を容易にします。これにより、API の柔軟性が増し、ほかの ORCA 関数シグネチャに必要な変更を最小限に抑えます。

構文

INT **PBORCA_ConfigureSession** (PBORCA *hORCASession*,
PPBORCA_CONFIG_SESSION *pSessionConfig*);

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
pSessionConfig	ORCA クライアントに、後続の要求の動作を指定させる構造体。設定は、セッション中、または再度 PBORCA_ConfigureSession を呼び出すまで保持されます。PBORCA_ConfigureSession を呼び出すたびに必ずすべての設定を指定してください。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	セッションが開かないか、pConfig ポインタが Null

解説

PBORCA_CONFIG_SESSION 構造体のインスタンスを作成し、環境を設定します。SessionOpen の直後に PBORCA_ConfigureSession を呼び出します。また、環境設定プロパティをリセット後、いつでもこの関数を呼び出すことができます。

```
typedef enum pborca_clobber
{
    PBORCA_NOCLOBBER,
    PBORCA_CLOBBER,
    PBORCA_CLOBBER_ALWAYS
    PBORCA_CLOBBER_DECIDED_BY_SYSTEM
} PBORCA_ENUM_FILEWRITE_OPTION;

typedef enum pborca_type
{
    PBORCA_UNICODE,
    PBORCA_UTF8,
    PBORCA_HEXASCII,
    PBORCA_ANSI_DBCS
} PBORCA_ENCODING;

typedef struct  pborca_configsession
```

```
{
    PBORCA_ENUM_FILEWRITE_OPTION
      eClobber; // 既存のファイルを上書きするか？
    PBORCA_ENCODINGeExportEncoding;
      // エクスポートされるソースのエンコーディング
    BOOL  bExportHeaders;
      // エクスポート ヘッダ付きソースのフォーマット
    BOOL  bExportIncludeBinary;// バイナリを含める
    BOOL  bExportCreateFile;// ソースをファイルにエクスポートする
    LPTSTR pExportDirectory;
      // エクスポートされるファイルのディレクトリ
    PBORCA_ENCODING   eImportEncoding;
      // インポートされるソースのエンコーディング
    BOOL      bDebug;//  コンパイラ ディレクティブのデバッグ
    PVOID     filler2;// 将来使用するために予約
    PVOID     filler3;
    PVOID     filler4;
} PBORCA_CONFIG_SESSION,  FAR *PPBORCA_CONFIG_SESSION;
```

メンバー変数	説明
eClobber	ファイル システム上の既存ファイルを上書きする場合に指定します。このプロパティは以下で使用されます。 PBORCA_LibraryEntryExport PBORCA_LibraryEntryExportEx PBORCA_DynamicLibraryCreate PBORCA_ExecutableCreate PBORCA_LibraryDelete 環境設定セッションには、以下の *eClobber* 値のいずれかを設定できます。 • **PBORCA_NOCLOBBER**　既存ファイルを上書きしない • **PBORCA_CLOBBER**　既存ファイルが書き込み不可でない場合は上書き • **PBORCA_CLOBBER_ALWAYS**　既存ファイルが書き込み不可でも上書き • **PBORCA_CLOBBER_DECIDED_BY_SYSTEM**　上記の関数が以前の ORCA リリースと同様に動作するようにする

メンバー変数	説明
eExportEncoding	PBORCA_LibraryEntryExport によって使用されるソースのエンコーディングを指定します。 • **PBORCA_UNICODE** Unicode ORCA クライアントのデフォルト • **PBORCA_ANSI_DBCS** ANSI ORCA クライアントのデフォルト • **PBORCA_UTF8** • **PBORCA_HEXASCII**
bExportHeaders	この変数に TRUE を設定すると、PBORCA_LibraryEntryExport はエクスポート ヘッダを生成します。下位互換性のため、デフォルト値は FALSE です。
bExportIncludeBinary	この変数に TRUE を設定すると、PBORCA_LibraryEntryExport はソース コンポーネントに追加するオブジェクトのバイナリ コンポーネントを生成します。下位互換性のため、デフォルト値は FALSE です。
bExportCreateFile	この変数に TRUE を設定すると、PBORCA_LibraryEntryExport はファイルにソースをエクスポートします。生成されるファイル名は、.sr? ファイル拡張子が付いた PowerBuilder オブジェクト エントリ名です。デフォルト値は FALSE です。
pExportDirectory	bExportCreateFile が TRUE の場合に、PowerBuilder オブジェクトをエクスポートするディレクトリです。
eImportEncoding	ソース エンコーディング。PBORCA_CompileEntryImport と PBORCA_CompileEntryImportList に対する後続の呼び出しは、lpszEntrySyntax 引数にこの情報が含まれていることを想定しています。
bDebug	この値に FALSE を設定すると、DEBUG 条件付きコンパイラ ディレクティブは解除されます。PowerScript コンパイラを起動するその後すべてのメソッドは、DEBUG 条件付きコンパイラ ブロック内のスクリプトを評価するときに、この設定を使用します。PBORCA_DeployWinFormProject は DEBUG ディレクティブを有効にするか無効にするかを決定するために、これらのターゲットのプロジェクト オブジェクト内の設定を使用するため、この設定は、Windows フォーム ターゲット内では使用されません。

例

この例は、PBORCA_CONFIG_SESSION 構造体に環境を設定します。

```
INT    ConfigureSession(LPTSTR sEncoding)
{
    INT    iErrCode = -1;
```

```
    lpORCA_Info->pConfig = (PPBORCA_CONFIG_SESSION)
       malloc(sizeof(PBORCA_CONFIG_SESSION));
    memset(lpORCA_Info->pConfig, 0,
       sizeof(PBORCA_CONFIG_SESSION));

if (!_tcscmp(sEncoding, _TEXT("ANSI")))
{
   lpORCA_Info->pConfig->eExportEncoding =
     PBORCA_ANSI_DBCS;
   lpORCA_Info->pConfig->eImportEncoding =
     PBORCA_ANSI_DBCS;
}
else if (!_tcscmp(sEncoding, _TEXT("UTF8")))
{
lpORCA_Info->pConfig->eExportEncoding = PBORCA_UTF8;
lpORCA_Info->pConfig->eImportEncoding = PBORCA_UTF8;
}
else if (!_tcscmp(sEncoding, _TEXT("HEXASCII")))
{
   lpORCA_Info->pConfig->eExportEncoding =
       PBORCA_HEXASCII;
   lpORCA_Info->pConfig->eImportEncoding =
       PBORCA_HEXASCII;
}
else
{
    lpORCA_Info->pConfig->eExportEncoding =
      PBORCA_UNICODE;
    lpORCA_Info->pConfig->eImportEncoding =
      PBORCA_UNICODE;
}
lpORCA_Info->pConfig->eClobber = PBORCA_CLOBBER;
lpORCA_Info->pConfig->bExportHeaders  = TRUE;
lpORCA_Info->pConfig->bExportIncludeBinary = FALSE;
lpORCA_Info->pConfig->bExportCreateFile = FALSE;
lpORCA_Info->pConfig->pExportDirectory = NULL;
lpORCA_Info->pConfig->bDebug = FALSE;
iErrCode = PBORCA_ConfigureSession(
        lpORCA_Info->hORCASession,
        lpORCA_Info->pConfig);
        return iErrCode;
}
```

関連項目

PBORCA_ApplicationRebuild
PBORCA_CompileEntryImportList
PBORCA_SetDebug

PBORCA_DeployWinFormProject

機能

Windows フォーム プロジェクトを生成およびコンパイルして、プロジェクト オブジェクトに含まれている指定に従ってアセンブリを配布します。

構文

```
INT PBORCA_DeployWinFormProject (
        HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTRlpszProjectName,
        LPTSTR lpszIconFileName,
        PBORCA_DOTNETPROC  pDotNetProc
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	プロジェクト エントリを含むファイル名を値とする文字列へのポインタ
lpszProjectName	配布情報を含むプロジェクト オブジェクト
lpszIconFileName	アプリケーション アイコン ファイルの名前
pDotNetProc	PBORCA_DOTNETPROC コールバック関数へのポインタ。コールバック関数は、生成される各メッセージに対して呼び出されます。最初にすべての ORCA_ERROR_MESSAGE メッセージが返され、その後、すべての PBORCA_WARNING_MESSAGE メッセージ、そして次にすべての PBORCA_UNSUPPORTED_FEATURE メッセージが返されます。
pUserData	PBORCA_DOTNETPROC コールバック関数に渡されるユーザ データへのポインタ

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-4 PBORCA_BADLIBRARY	必要な DLL のライブラリのロードに失敗
-5 PBORCA_LIBLISTNOTSET	SessionSetLibraryList が必要
-13 PBORCA_CURRAPPLNOTSET	SessionSetCurrentAppl が必要
-19 PBORCA_CBCREATEERROR	コンポーネント ビルダ作成エラー
-20 PBORCA_CBINITERROR	コンポーネント ビルダ初期化エラー
-21 PBORCA_CBBUILDERROR	コンポーネント ビルダ構築エラー

解説

エラー情報は、次の関数シグネチャを使用する PBORCA_DeployWinFormProject に関連するコールバック関数を最初に作成することにより返されます。

```
void  MyDotNetMessageProc (
    PPBORCA_DOTNET_MESSAGE pMsg,
        LPVOID pMyUserData)
```

pMsg 引数は、次の構造体へのポインタです。

```
typedef  struct  pborca_dotnetmsg
  {
  PBROCA_DOTNET_MSGTYPE   eMessageType;
  LPTSTR  lpszMessageText;
   }
  PBORCA_DOTNET_MESSAGE  FAR *PPBORCA_DOTNET_MESSAGE;
```

eMessageType 引数は、次の列挙型を使用します。

```
typedef enum pborca_dotnet_msgtype
  {
  PBORCA_ERROR_MESSAGE,
  PBORCA_WARNING_MESSAGE,
  PBORCA_UNSUPPORTED_FEATURE
  } PBORCA_DOTNET_MSGTYPE;
```

メッセージは、次の順番で 1 度に 1 つずつ返されます。
PBORCA_ERROR_MESSAGE メッセージ、
PBORCA_WARNING_MESSAGE メッセージ、
PBORCA_UNSUPPORTED_FEATURE メッセージ

PBORCA_DynamicLibraryCreate

機能

PowerBuilder 動的ライブラリ（PBD）または PowerBuilder DLL を生成します。

構文

```
INT PBORCA_DynamicLibraryCreate (
        HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTR lpszPBRName,
        LONG lFlags );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	PBD または DLL を構築するライブラリ ファイル名を値として持つ文字列へのポインタ
lpszPBRName	PBD や DLL に取り込むオブジェクトの PowerBuilder リソース ファイル名を値として持つ文字列へのポインタ。アプリケーションのリソース ファイルがない場合には、ポインタに 0 を指定します。
lFlags	ライブラリ構築時に適用するコード生成のオプションを指定する long 型の値 *lFlags* に 0 を設定すると、ネイティブの Pcode 形式の実行ファイルを生成します。 マシン コードを生成するオプションの設定については、PBORCA_ExecutableCreate を参照してください。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-4 PBORCA_BADLIBRARY	ライブラリ名が正しくない
-17 PBORCA_PBDCREATERROR	PBD 作成エラー

解説

この関数を呼び出す前に、ライブラリ リストと現行のアプリケーションを設定する必要があります。

実行ファイルの中にいくつかの動的ライブラリを含める計画がある場合は、実行ファイルを構築する前に動的ライブラリを構築する必要があります。

ファイルの場所と名前　PBD や DLL は PBL と同じディレクトリに同じファイル名で作成され、拡張子だけが異なります。たとえば、ライブラリが C:¥DIR1¥DIR2¥PROG.PBL の場合、以下のようになります。

- Pcode の出力は C:¥DIR1¥DIR2¥PROG.PBD

- マシン コードの出力は C:¥DIR1¥DIR2¥PROG.DLL

eClobber の設定　ファイル システム中に PBD や DLL がすでに存在する場合、ORCA 環境設定ブロック中の eClobber プロパティの現行の設定（PBORCA_ConfigureSession 呼び出しで設定）で、PBORCA_DynamicLibraryCreate が成功したか失敗したかを判断します。

現行の eClobber 設定	PBORCA_DynamicLibraryCreate
PBORCA_NOCLOBBER	ファイル システム中に実行ファイルがすでに存在している場合、ファイルの属性設定に関わらず、処理は失敗します。
PBORCA_CLOBBER または PBORCA_CLOBBER_DECIDED_BY_SYSTEM	既存の実行ファイルが読み書き属性を持つ場合は処理は成功し、読み取り属性のみを持つ場合は処理は失敗します。
PBORCA_CLOBBER_ALWAYS	既存の実行ファイルの属性設定に関係なく、処理は成功します。

例

この例は、ライブラリ PROCESS.PBL からマシン コードの DLL を構築します。トレース情報およびエラー コンテキスト情報のスピードを最適化します。

```
LPTSTR pszLibFile;
LPTSTR pszResourceFile;
long lBuildOptions;
int rtn;

// ファイル名をコピー
pszLibFile = _TEXT("c:¥¥app¥¥process.pbl");
pszResourceFile = _TEXT("c:¥¥app¥¥process.pbr");

lBuildOptions = PBORCA_MACHINE_CODE_NATIVE |
    PBORCA_MACHINE_CODE_OPT_SPEED |
    PBORCA_TRACE_INFO | PBORCA_ERROR_CONTEXT;

// ライブラリから DLL を作成
rtn = PBORCA_DynamicLibraryCreate(
    lpORCA_Info->hORCASession,
    pszLibFile, pszResourceFile, lBuildOptions );
```

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_ConfigureSession
PBORCA_ExecutableCreate

PBORCA_ExecutableCreate

機能

Pcode 形式またはマシン コード形式の PowerBuilder 実行ファイルを作成します。マシン コード形式の実行ファイルには、いくつかのデバッグと最適化のオプションを付けることができます。

アプリケーションを作成するために ORCA ライブラリ リストを使用します。すでに構築されている PBD や DLL のうちのどのライブラリをどの実行ファイルへ取り入れるのかを指定することができます。

構文

```
INT PBORCA_ExecutableCreate ( HPBORCA hORCASession,
        LPTSTR lpszExeName,
        LPTSTR lpszIconName,
        LPTSTR lpszPBRName,
        PBORCA_LNKPROC pLinkErrProc,
        LPVOID pUserData,
        INT FAR *iPBDFlags,
        INT iNumberOfPBDFlags,
        LONG lFlags );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszExeName	作成する実行ファイル名を値に持つ文字列へのポインタ
lpszIconName	アイコン ファイル名を値の持つ文字列へのポインタ。アイコン ファイルはすでに存在していなければなりません。
lpszPBRName	PowerBuilder リソース ファイル名を値に持つ文字列へのポインタ。名前を付けるリソース ファイルはすでに存在していなければなりません。アプリケーションのリソース ファイルがない場合には、ポインタに 0 を指定します。
pLinkErrProc	コールバック関数 PBORCA_ExecutableCreate へのポインタ。コールバック関数は、リンク エラーが発生するたびに呼び出されます。 ORCA がコールバック関数に渡す情報は、PBORCA_LINKERR 構造体に格納されるメッセージ テキストです。 コールバック関数を使用しない場合は、*pLinkErrProc* に 0 を設定します。
pUserData	コールバック関数 PBORCA_ExecutableCreate に渡すユーザ データへのポインタ ユーザ データは通常、ディレクトリの情報とバッファ サイズの情報をフォーマットするコールバック関数のバッファまたはバッファへのポインタを含んでいます。 コールバック関数を使用しない場合は、pUserData に 0 を設定します。

引数	説明
iPBDFlags	ORCA セッションのライブラリ リスト上にあるライブラリのうち、どのライブラリを PowerBuilder 動的ライブラリ（PBD）に組み込む必要があるのかを指し示す整数配列へのポインタ。各配列要素は、ライブラリ リスト中のライブラリに対応しています。フラグの値は以下のとおりです。 • 0 – 実行ファイルにライブラリのオブジェクトを含める • 1 – ライブラリは、すでに PBD または PowerBuilder DLL であり、そのオブジェクトは実行ファイルに含めない
iNumberOfPBDFlags	*iPBDFlags* 配列中のエレメント数。これは ORCA ライブラリ リスト上のライブラリ数と同じです。
lFlags	実行ファイル構築時に適用するコード生成のオプションを指定する long 型の値 *lFlags* に 0 を設定すると、ネイティブの Pcode 形式の実行ファイルを生成します。マシン コードのその他の設定については、以下の解説を参照してください。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト
-5　PBORCA_LIBLISTNOTSET	ライブラリ リストが未設定
-12　PBORCA_LINKERROR	リンク エラー
-13　PBORCA_CURRAPPLNOTSET	現行のアプリケーションが未設定

解説

この関数を呼び出す前に、ライブラリ リストと現行のアプリケーションを設定する必要があります。

実行ファイル構築時に指定するさまざまなオプションについての詳細は、『PowerBuilder ユーザーズ ガイド』マニュアルを参照してください。

実行ファイル内で使用されるライブラリ　構築される実行ファイルは、ORCA のライブラリ リスト上のライブラリにオブジェクトを組み入れます。ライブラリ リストは、実行ファイル作成の前に PBORCA_SessionSetLibraryList を呼び出して設定する必要があります。

iPBDFlags 引数を使用して、どのライブラリが PBD か、どれが実行ファイルの中に取り込まれるのかを指定できます。*iPBDFlags* 配列の各整数は、ORCA のライブラリ リスト上のライブラリに関連付けられています。1 を設定すると、対応するライブラリ中のオブジェクトは PBD ファイル（Pcode 生成時）、または、PowerBuilder DLL（マシン コード生成時）に組み込まれます。整数フラグに 0 を設定すると、ライブラリ中のオブジェクトはメインの実行ファイルの中に組み込まれます。

PBORCA_ExecutableCreate を呼び出す前に、PBD または DLL を生成するための PBORCA_DynamicLibraryCreate を呼び出す必要があります（*iPBDFlags* 配列中で明示しています）。

コード生成オプションの設定　*lFlags* 引数に、個別のビットを設定することで、さまざまなマシン コード生成オプションを設定できます。次の表は、long 値にビットを定義した場合のそれぞれの意味と、オプションを設定するためにビット単位での OR 演算式を使用する定数について説明しています。表中にないビットは、予約済みです。

ビット	値と意味	OR 演算式に含む定数
0	0 = Pcode 1 = マシン コード	マシン コードを取得するには、PBORCA_MACHINE_CODE または PBORCA_MACHINE_CODE_NATIVE を使用します。
1	0 = ネイティブ コード 1 = 16 ビット コード	16 ビット マシン コードを取得するには、PBORCA_MACHINE_CODE および PBORCA_MACHINE_CODE_16 を使用します。 16 ビット Pcode を取得するには、PBORCA_P_CODE_16 を使用します。 **PowerBuilder 7 以降はサポート対象外** PowerBuilder は、今後 Windows 3.x 16 ビット プラットフォームをサポートしません。
2	0 = Open Server なし 1 = Open Server あり	Open Server 実行ファイルを構築するには、PBORCA_OPEN_SERVER を使用します。 **PowerBuilder 5 以降はサポート対象外** OpenClientServer ドライバは、PowerBuilder 5 以降はサポート対象外です。このため、Open Server 実行ファイルのオプションは今後サポートされません。

ビット	値と意味	OR 演算式に含む定数
4	0 = トレース情報なし 1 = トレース情報あり	トレース情報を取得するには、PBORCA_TRACE_INFO を使用します。
5	0 = エラー コンテキストなし 1 = エラー コンテキストあり	エラー コンテキスト情報を取得するには、PBORCA_ERROR_CONTEXT を使用します。 エラー コンテキストは、エラーのあるスクリプト名と行番号の情報を提供します。
8	0 = 最適化なし 1 = 最適化あり	ビット 9 を参照
9	0 = スピードを最適化する 1 = スペースを最適化する	実行ファイルをスピード面で最適化するには、PBORCA_MACHINE_CODE_OPT または PBORCA_MACHINE_CODE_OPT_SPEED を使用します。 実行ファイルをスペース面で最適化するには、PBORCA_MACHINE_CODE_OPT および PBORCA_MACHINE_CODE_OPT_SPACE を使用します。
10	0 = 以前からの表示スタイル コントロール 1 = 新しい表示スタイル コントロール (XP)	PBORCA_NEW_VISUAL_STYLE_CONTROLS

Pcode を生成するには、lFlags を 0 にします。ほかのビットは関連しません。

```
lFlags = PBORCA_P_CODE;
```

さまざまなマシン コード オプションに lFlags 引数を設定するために、ビット フラグの定数が OR 条件で判断され、希望する組み合わせを実現します。

```
lFlags = PBORCA_MACHINE_CODE |
       PBORCA_MACHINE_CODE_OPT |
       PBORCA_MACHINE_CODE_OPT_SPACE;
```

定数は一般的なオプションの組み合わせとして PBORCA.H に定義してあります。その内容は、以下のとおりです。

- **PBORCA_MACHINE_DEFAULT**　スピードを最適化したネイティブ マシン コードを意味します。

  以下は同じ意味になります。

  ```
  PBORCA_MACHINE_CODE |
        PBORCA_MACHINE_CODE_OPT_SPEED
  ```

- **PBORCA_MACHINE_DEBUG** トレース情報とエラー コンテキスト情報を伴うネイティブ マシン コードを意味します。

  以下は同じ意味になります。

  ```
  PBORCA_MACHINE_CODE | PBORCA_TRACE_INFO |
        PBORCA_ERROR_CONTEXT
  ```

eClobber 設定　ファイル システム中に実行ファイルがすでに存在する場合、ORCA 環境設定ブロック中の eClobber プロパティの現行の設定（PBORCA_ConfigureSession 呼び出しで設定）で、PBORCA_ExecutableCreate が成功したか失敗したかを判断します。

現行の eClobber 設定	PBORCA_ExecutableCreate
PBORCA_NOCLOBBER または PBORCA_CLOBBER_DECIDED_BY_SYSTEM	ファイル システム中に実行ファイルがすでに存在している場合、ファイルの属性設定に関わらず、処理は失敗します。
PBORCA_CLOBBER	既存の実行ファイルが読み書き属性を持つ場合は処理は成功し、読み取り属性のみを持つ場合は処理は失敗します。
PBORCA_CLOBBER_ALWAYS	既存の実行ファイルの属性設定に関係なく、処理は成功します。

例

この例は、ORCA のライブラリ リストと現行アプリケーションを使用してスピードを最適化したネイティブのマシン コードを構築します。現行の ORCA セッションのライブラリ リストには、4 つのエントリがあると想定しています。例では、最後の 2 つのライブラリ用の DLL を生成します。

コールバック関数は LinkErrors を呼び出し、lpUserData はコールバック関数が使用する空のバッファを指し示します。

```
LPTSTR pszExecFile;
LPTSTR pszIconFile;
LPTSTR pszResourceFile;
int iPBDFlags[4];
long lBuildOptions;
int rtn;

fpLinkProc = (PBORCA_LNKPROC) LinkProc;
// ファイル名を指定
pszExecFile     = _TEXT("c:¥¥app¥¥process.exe");
pszIconFile     = _TEXT("c:¥¥app¥¥process.ico");
pszResourceFile = _TEXT("c:¥¥app¥¥process.pbr");
```

```
iPBDFlags[0] = 0;
iPBDFlags[1] = 0;
iPBDFlags[2] = 1;
iPBDFlags[3] = 1;

lBuildOptions = PBORCA_MACHINE_CODE_NATIVE |
    PBORCA_MACHINE_CODE_OPT_SPEED;

// 実行ファイル作成
rtn = PBORCA_ExecutableCreate(
    lpORCA_Info->hORCASession,
    pszExecFile, pszIconFile, pszResourceFile,
    fpLinkProc, lpUserData,
    (INT FAR *) iPBDFlags, 4, lBuildOptions );
```

コールバック用データ バッファの設定についての詳細は、15 ページの「コールバック関数の内容」と PBORCA_LibraryDirectory の例を参照してください。

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_ConfigureSession
PBORCA_DynamicLibraryCreate

PBORCA_LibraryCommentModify

機能

PowerBuilder ライブラリ用のコメントを変更します。

構文

INT **PBORCA_LibraryCommentModify** (HPBORCA *hORCASession*,
LPTSTR *lpszLibName*,
LPTSTR *lpszLibComments*);

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibName	コメントを変更するライブラリ名を値に持つ文字列へのポインタ
lpszLibComments	新しいライブラリ コメントを値に持つ文字列へのポインタ

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-3 PBORCA_OBJNOTFOUND	ライブラリが見つからない
-4 PBORCA_BADLIBRARY	ライブラリ名が正しくない
-7 PBORCA_LIBIOERROR	ライブラリ I/O エラー

解説

この関数を呼び出す前に、ライブラリ リストや現行のアプリケーションを設定する必要はありません。

例

この例は、MASTER.PBL ライブラリのコメントを変更します。

```
LPTSTR pszLibraryName;
LPTSTR pszLibraryComments;
// ライブラリ名とコメントの文字列を指定
pszLibraryName =
    _TEXT("c:¥¥sybase¥¥pb12.5¥¥demo¥¥master.pbl");
pszLibraryComments =
    _TEXT("PBL contains ancestor objects for XYZ app.");
// ライブラリにコメントを挿入
lpORCA_Info->lReturnCode =
    PBORCA_LibraryCommentModify(
    lpORCA_Info->hORCASession,
    pszLibraryName, pszLibraryComments);
```

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_LibraryCreate

PBORCA_LibraryCreate

機能

新しい PowerBuilder ライブラリを作成します。

構文

```
INT PBORCA_LibraryCreate ( HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTR lpszLibraryComments );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	作成するライブラリのファイル名を値に持つ文字列へのポインタ
lpszLibraryComments	新しいライブラリを説明するコメントを値に持つ文字列へのポインタ

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト
-4　PBORCA_BADLIBRARY	ライブラリ名が正しくない
-7　PBORCA_LIBIOERROR	ライブラリ I/O エラー
-8　PBORCA_OBJEXISTS	オブジェクトがすでに存在する
-9　PBORCA_INVALIDNAME	ライブラリ名が不正

解説

この関数を呼び出す前に、ライブラリ リストや現行のアプリケーションを設定する必要はありません。

オブジェクトの追加　PBORCA_LibraryCreate は、ディスク上に空のライブラリ ファイルを作成します。PBORCA_LibraryEntryCopy や PBORCA_CheckOutEntry のような関数を使用して、ほかのライブラリからオブジェクトを追加することができます。新しいライブラリを取り込むためにライブラリ リストを設定してから現行のアプリケーションを設定した場合、PBORCA_CompileEntryImport と PBORCA_CompileEntryImportList を使用してオブジェクトのソース コードをインポートできます。

例

この例は NEWLIB.PBL というライブラリを作成し、説明のコメントを付けます。

```
LPTSTR pszLibraryName;
LPTSTR pszLibraryComments;
// ライブラリ名とコメントの文字列を指定
pszLibraryName =
    _TEXT("c:¥¥sybase¥¥pb12.5¥¥demo¥¥newlib.pbl");
pszLibraryComments =
```

```
        _TEXT("PBL contains ancestor objects for XYZ app.");
    // ライブラリを作成
    lpORCA_Info->lReturnCode =
        PBORCA_LibraryCreate(lpORCA_Info->hORCASession,
        pszLibraryName, pszLibraryComments);
```

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目 PBORCA_LibraryDelete

PBORCA_LibraryDelete

機能

PowerBuilder のライブラリ ファイルをディスクから削除します。

構文

```
INT PBORCA_LibraryDelete ( HPBORCA hORCASession,
        LPTSTR   lpszLibraryName );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	削除するライブラリ ファイル名を値に持つ文字列へのポインタ

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト
-4　PBORCA_BADLIBRARY	ライブラリ名が正しくない
-7　PBORCA_LIBIOERROR	ライブラリ I/O エラー

解説

この関数を呼び出す前に、ライブラリ リストや現行のアプリケーションを設定する必要はありません。読み取り専用の属性を持つ PowerBuilder ライブラリを削除する場合は、eClobber 環境設定プロパティに PBORCA_CLOBBER_ALWAYS を設定する必要があります。

例

この例は、ライブラリ名 EXTRA.PBL を削除します。

```
LPTSTR pszLibraryName;
// ライブラリ名を指定
pszLibraryName =
   _TEXT("c:¥¥sybase¥¥pb12.5¥¥demo¥¥extra.pbl");

// ライブラリを削除
lpORCA_Info->lReturnCode =
    PBORCA_LibraryDelete(lpORCA_Info->hORCASession,
    pszLibraryName);
```

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_ConfigureSession
PBORCA_LibraryCreate

PBORCA_LibraryDirectory

機能

ディレクトリ中のオブジェクト リストなど、PowerBuilder ライブラリのディレクトリに関する情報をレポートします。

構文

```
INT PBORCA_LibraryDirectory ( HPBORCA hORCASession,
        LPTSTR lpszLibName,
        LPTSTR lpszLibComments,
        INT iCmntsBuffLen,
        PBORCA_LISTPROC pListProc,
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibName	ディレクトリ情報が必要なライブラリのファイル名を値に持つ文字列へのポインタ
lpszLibComments	ライブラリと共に格納されるコメントを ORCA が出力するバッファへのポインタ
iCmntsBuffLen	lpszLibComments が指し示すバッファ長 (TCHAR で指定)。推奨される長さは、PBORCA_MAXCOMMENTS + 1 です。
pListProc	コールバック関数 PBORCA_LibraryDirectory へのポインタ。コールバック関数は、ライブラリ中の各エントリに対して呼び出されます。 ORCA がコールバック関数に渡す情報は、エントリ名、コメント、エントリのサイズ、変更時間で、これらは、PBORCA_DIRENTRY 型の構造体に格納されています。
pUserData	コールバック関数 PBORCA_LibraryDirectory に渡すユーザ データへのポインタ ユーザ データは通常、ディレクトリの情報とバッファ サイズの情報をフォーマットするコールバック関数のバッファまたはバッファへのポインタを含んでいます。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-4 PBORCA_BADLIBRARY	ライブラリ名が正しくない
-7 PBORCA_LIBIOERROR	ライブラリ I/O エラー

解説

この関数を呼び出す前に、ライブラリ リストや現行のアプリケーションを設定する必要はありません。

ライブラリのコメント　PBORCA_LibraryDirectory は、lpszLibComments が指し示す文字列にライブラリ のコメント を入れます。コールバック関数は、UserData バッファ に各オブジェクト のコメント を格納できます。

ライブラリ エントリに関する情報　ライブラリ中の個別のエントリに関して戻される情報は、コールバック関数で指定する処理に依存します。ORCA は、PBORCA_DIRENTRY 構造体のライブラリ エントリに関する情報をコールバック関数に渡します。コールバック関数は、その構造体を調べて、必要なすべての情報を pUserData が指し示すバッファに格納できます。

PBORCA_LibraryDirectory を呼び出すとき、ライブラリ中のエントリ数は不明です。この場合の対処法は以下の 2 通りです。

- 適度なサイズのメモリ ブロックを割り当て、オーバーフローした場合はバッファを再度割り当てる（13 ページの「ORCA コールバック関数について」を参照）。
- lpUserDataBuffer をリンク リストの先頭に位置付ける。各 PBORCA_DIRENTRY の戻りに対して、要求された情報を得るために新しいリストのエントリを動的に割り当てます（以下の例で説明しています）。

例

この例は、リンク リストのヘッダを定義します。

```
typedef struct libinfo_head
{
    TCHAR    szLibName[PBORCA_SCC_PATH_LEN];
    TCHAR    szComments[PBORCA_MAXCOMMENT+1];
    INT      iNumEntries;
    PLIBINFO_ENTRY    pEntryAnchor;
    PLIBINFO_ENTRY    pLast;
} LIBINFO_HEAD, FAR *PLIBINFO_HEAD;
```

各 DirectoryProc コールバック関数の呼び出しは、以下で定義されるように、新しいリンク リスト エントリを割り当てます。

```
typedef struct libinfo_entry
{
    TCHAR  szEntryName[41];
    LONG   lEntrySize;
    LONG   lObjectSize;
    LONG   lSourceSize;
    PBORCA_TYPE  otEntryType;
    libinfo_entry  *pNext;
} LIBINFO_ENTRY, FAR *PLIBINFO_ENTRY;
```

```
PBORCA_LISTPROC   fpDirectoryProc;
PLIBINFO_HEAD  pHead;
fpDirectoryProc = (PBORCA_LISTPROC) DirectoryProc;
pHead = new LIBINFO_HEAD;
_tcscpy(pHead->szLibName, _TEXT("c:¥¥myapp¥test.pbl");
memset(pHead->szComments, 0x00,
   sizeof(pHead->szComments));
pHead->iNumEntries = 0;
pHead->pEntryAnchor = NULL;
pHead->pLast = NULL;

lpORCA_Info->lReturnCode = PBORCA_LibraryDirectory(
   lpORCA_Info->hORCASession,
   pHead->szLibName,
   pHead->szComments,
   (PBORCA_MAXCOMMENT+1), // TCHAR で長さを指定
   fpDirectoryProc,
   pHead);
// PBORCA_LibraryEntryInformation の例を参照
if (lpORCA_Info->lReturnCode == PBORCA_OK)
   GetEntryInfo(pHead);
CleanUp(pHead);

// CleanUp -  割り当てたメモリを解放
INT CleanUp(PLIBINFO_HEAD  pHead)
{
    INT    iErrCode = PBORCA_OK;
    PLIBINFO_ENTRY   pCurrEntry;
    PLIBINFO_ENTRY   pNext;
    INT    idx;
for (idx = 0, pCurrEntry = pHead->pEntryAnchor;
   (idx < pHead->iNumEntries) && pCurrEntry; idx++)
{
   pNext = pCurrEntry->pNext;
   delete pCurrEntry;
   if (pNext)
       pCurrEntry = pNext;
   else pCurrEntry = NULL;
}
delete pHead;
return iErrCode;
}

// PBORCA_LibraryDirectory で使用されるコールバック手順
void __stdcall DirectoryProc(PBORCA_DIRENTRY
    *pDirEntry, LPVOID lpUserData)
{
   PLIBINFO_HEAD   pHead;
```

```
PLIBINFO_ENTRY   pNewEntry;
PLIBINFO_ENTRY   pTemp;

pHead = (PLIBINFO_HEAD) lpUserData;
pNewEntry = (PLIBINFO_ENTRY) new LIBINFO_ENTRY;
memset(pNewEntry, 0x00, sizeof(LIBINFO_ENTRY));
if (pHead->iNumEntries == 0)
{
   pHead->pEntryAnchor = pNewEntry;
   pHead->pLast = pNewEntry;
}
else
{
   pTemp = pHead->pLast;
   pTemp->pNext = pNewEntry;
   pHead->pLast = pNewEntry;
}
pHead->iNumEntries++;
_tcscpy(pNewEntry->szEntryName,
   pDirEntry->lpszEntryName);
pNewEntry->lEntrySize = pDirEntry->lEntrySize;
pNewEntry->otEntryType = pDirEntry->otEntryType;
```

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目　PBORCA_LibraryEntryInformation

PBORCA_LibraryEntryCopy

機能

PowerBuilder ライブラリ エントリをあるライブラリから別のライブラリにコピーします。

構文

```
INT PBORCA_LibraryEntryCopy ( HPBORCA hORCASession,
        LPTSTR lpszSourceLibName,
        LPTSTR lpszDestLibName,
        LPTSTR lpszEntryName,
        PBORCA_TYPE otEntryType );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszSourceLibName	オブジェクトを含むソース ライブラリのファイル名を値に持つ文字列へのポインタ
lpszDestLibName	コピーするオブジェクトの格納先ライブラリのファイル名を値に持つ文字列へのポインタ
lpszEntryName	コピーされるオブジェクトの名前を値に持つ文字列へのポインタ
otEntryType	コピーされるエントリのオブジェクト型を指定する PBORCA_TYPE のカタログ データ型の値。値は以下のとおりです。 PBORCA_APPLICATION PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-3 PBORCA_OBJNOTFOUND	オブジェクトが見つからない
-4 PBORCA_BADLIBRARY	ライブラリ名が正しくない
-7 PBORCA_LIBIOERROR	ライブラリ I/O エラー

解説

この関数を呼び出す前に、ライブラリ リストや現行のアプリケーションを設定する必要はありません。

2 つの個別の API 呼び出しを必要とする PBORCA_CompileEntryImport とは異なり、PBORCA_LibraryEntryCopy は自動的にソース コンポーネントをコピーし、オブジェクトのバイナリ コンポーネントが存在する場合はそれもコピーします。

例

この例は、d_labels という名前のデータウィンドウを SOURCE.PBL から DESTIN.PBL へコピーします。

```
lpORCA_Info->lReturnCode = PBORCA_LibraryEntryCopy(
    lpORCA_Info->hORCASession,
    _TEXT("c:¥¥app¥¥source.pbl"),
    _TEXT("c:¥¥app¥¥destin.pbl"),
    _TEXT("d_labels"), PBORCA_DATAWINDOW);
```

この例では、lpszSourceLibraryName、lpszDestinationLibraryName および lpszEntryName のポインタが有効なライブラリとオブジェクト名を指し示しており、otEntryType は有効なオブジェクト型であると仮定しています。

```
lpORCA_Info->lReturnCode = PBORCA_LibraryEntryCopy(
    lpORCA_Info->hORCASession,
    lpszSourceLibraryName,
    lpszDestinationLibraryName,
    lpszEntryName, otEntryType );
```

関連項目

PBORCA_LibraryDelete
PBORCA_LibraryEntryMove

PBORCA_LibraryEntryDelete

機能

PowerBuilder ライブラリ エントリを削除します。

構文

```
INT PBORCA_LibraryEntryDelete (HPBORCA hORCASession,
        LPTSTR lpszLibName,
        LPTSTR lpszEntryName,
        PBORCA_TYPE otEntryType );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibName	オブジェクトを含むライブラリのファイル名を値に持つ文字列へのポインタ
lpszEntryName	削除されるオブジェクトの名前を値に持つ文字列へのポインタ
otEntryType	削除されるエントリのオブジェクト型を指定する PBORCA_TYPE のカタログ データ型の値。値は以下のとおりです。 PBORCA_APPLICATION PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-3 PBORCA_OBJNOTFOUND	オブジェクトが見つからない
-4 PBORCA_BADLIBRARY	ライブラリ名が正しくない
-7 PBORCA_LIBIOERROR	ライブラリ I/O エラー

解説

この関数を呼び出す前に、ライブラリ リストや現行のアプリケーションを設定する必要はありません。

例

この例は、SOURCE.PBL ライブラリから d_labels という名前のデータウィンドウを削除します。

```
lrtn = PBORCA_LibraryEntryDelete(
    lpORCA_Info->hORCASession,
    _TEXT("c:¥¥app¥¥source.pbl"),
    _TEXT("d_labels"), PBORCA_DATAWINDOW);
```

この例では、ポインタ lpszLibraryName と lpszEntryName は有効なライブラリとオブジェクト名を指し示しており、otEntryType は有効なオブジェクト型であると仮定しています。

```
lpORCA_Info->lReturnCode = PBORCA_LibraryEntryDelete(
    lpORCA_Info->hORCASession,
    lpszLibraryName,
    lpszEntryName,
    otEntryType);
```

関連項目

PBORCA_LibraryEntryCopy
PBORCA_LibraryEntryMove

PBORCA_LibraryEntryExport

機能

PowerBuilder ライブラリ エントリのソース コードをソース バッファまたはファイルにエクスポートします。

構文

```
INT PBORCA_LibraryEntryExport ( HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTR lpszEntryName,
        PBORCA_TYPE otEntryType,
        LPTSTR lpszExportBuffer,
        LONG lExportBufferSize );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	エクスポートするオブジェクトを含むライブラリのファイル名を値に持つ文字列へのポインタ
lpszEntryName	エクスポートされるオブジェクトの名前を値に持つ文字列へのポインタ
otEntryType	エクスポートされるエントリのオブジェクト型を指定する PBORCA_TYPE カタログ データ型の値。値は以下のとおりです。 PBORCA_APPLICATION PBORCA_BINARY PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW
lpszExportBuffer	PBORCA_CONFIG_SESSION プロパティの bExportCreateFile が FALSE の場合に、ORCA がエクスポートされるソースのコードを格納するデータ バッファへのポインタ。bExportCreateFile が TRUE の場合、この引数は NULL にすることができます。
lExportBufferSize	lpszExportBuffer のサイズ（バイト）。この引数は、PBORCA_CONFIG_SESSION プロパティの bExportCreateFile が TRUE の場合は必要ありません。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト
-3　PBORCA_OBJNOTFOUND	オブジェクトが見つからない
-4　PBORCA_BADLIBRARY	ライブラリ名が正しくない
-7　PBORCA_LIBIOERROR	ライブラリ I/O エラー
-10　PBORCA_BUFFERTOOSMALL	バッファ サイズが小さすぎる
-33　PBORCA_DBCSERROR	Unicode を ANSI_DBCS に変換する際のロケールの設定エラー

解説

この関数を呼び出す前に、ライブラリ リストや現行のアプリケーションを設定する必要はありません。

PowerBuilder 10 以降の変更

PowerBuilder 10 以降では、PBORCA_CONFIG_SESSION 変数を使用して、この関数の動作をカスタマイズできます。ただし、下位互換性なので、デフォルトでは動作は変更されません。

ソース コードの戻り方　pConfigSession->bExportCreateFile が FALSE の場合、オブジェクトのソース コードはエクスポート バッファに戻されます。bExportCreateFile プロパティが TRUE の場合、ソースは pConfigSession->pExportDirectory が指し示すディレクトリの中のファイルに書き込まれます。

pConfigSession->bExportHeaders が TRUE の場合、ORCA はエクスポート バッファまたはファイルの先頭に 2 行のエクスポート ヘッダを書き込みます。バッファ内で、エクスポートされたソース コードには、各表示行の最後に改行（Hex 0D）および新規行（Hex 0A）キャラクタが含まれます。

ソース コード エンコーディング　PowerBuilder は 4 つの異なるエンコーディング形式でソースをエクスポートします。デフォルトでは、ANSI/DBCS クライアントはソースを PBORCA_ANSI_DBCS 形式でエクスポートし、Unicode クライアントは PBORCA_UNICODE 形式でエクスポートします。pConfigSession->eExportEncoding を設定してエンコーディング形式を明示的に要求することができます。

バイナリ コンポーネント　PowerBuilder では、pConfigSession->eExportIncludeBinary = TRUE を設定して、エクスポート バッファまたはファイルにオブジェクトのバイナリ コンポーネントが自動的に含められるように明示的に要求することができます。

新規開発では、推奨される設定です。以前のリリースの ORCA ではこの機能をサポートしていなかったため、以前の方法もまだサポートしています。

下位互換テクニック

以前のバージョンのように、各 PBORCA_LibraryEntryExport 要求の後、PBORCA_BINARY が *otEntryType* の PBORCA_LibraryEntryInformation を呼び出すことができます。この関数は、バイナリ データが存在すると PBORCA_OK を返し、PBORCA_BINARY が *otEntryType* に設定された 2 回目の PBORCA_LibraryEntryExport 呼び出しを実行できます。下位互換のため、PBORCA_BINARY に *otEntryType* を設定すると、pConfigSession->bExportHeaders =TRUE および pConfigSession->bExportIncludeBinary = TRUE. の環境設定プロパティが無視されます。

ソース コードのサイズ　エクスポート関数を呼び出す前にオブジェクトのソース サイズを知るには、最初に PBORCA_LibraryEntryInformation 関数を呼び出し、適切な lExportBufferSize 値を算出するために pEntryInfo->lSourceSize 情報を使用します。lExportBufferSize は lpszExportBuffer のバイト 単位のサイズです。

ORCA エクスポート処理は、エクスポート ソースを収めるのに十分な容量のバッファが割り当てられているかどうかを判断する前に、必要な全データの変換を行います。十分なバッファ容量がない場合は、戻り値 PBORCA_BUFFERTOOSMALL が返されます。lExportBufferSize が必要な長さに等しい場合 PBORCA_LibraryEntryExport は成功しますが、エクスポートされたソースに Null 区切り文字を追加しません。lExportBufferSize が十分に大きい場合、ORCA は Null 区切り文字を追加します。Sybase 社は、データ変換と Null 区切り文字に対応するために十分な大きさのバッファを割り当てることを推奨しています。pConfigSession->bExportCreateFile = TRUE の場合、lExportBufferSize は無視されます。

データ変換とエクスポート後のソース サイズの判断　実際に返されるバッファやファイルのサイズを確認する必要がある場合は、PBORCA_LibraryEntryExport の代わりに PBORCA_LibraryEntryExportEx を呼び出すことができます。これらの関数は、PBORCA_LibraryEntryExportEx 関数のシグネチャが追加の *plReturnSize 引数を含んでいるという点以外はまったく同じ動作をします。

既存のエクスポート ファイルの上書き　pConfigSession->eClobber の値は、既存のエクスポート ファイルを上書きするかどうかを指定します。エクスポート ファイルが存在しない場合、eClobber の設定に関わらず PBORCA_LibraryEntryExport は PBORCA_OK を返します。次の表は、エクスポート ファイルがすでに存在する場合に eClobber の設定で PBORCA_LibraryEntryExport の動作がどのように変更されるかを説明しています。戻り値 PBORCA_OBJEXISTS は、既存ファイルが上書きされなかったことを意味します。

PConfigSession->eClobber 設定	読み書き可能ファイルが存在する場合の戻り値	読み取り専用ファイルが存在する場合の戻り値
PBORCA_NOCLOBBER	PBORCA_OBJEXISTS	PBORCA_OBJEXISTS
PBORCA_CLOBBER	PBORCA_OK	PBORCA_OBJEXISTS
PBORCA_CLOBBER_ALWAYS	PBORCA_OK	PBORCA_OK
PBORCA_CLOBBER_DECIDED_BY_SYSTEM	PBORCA_OBJEXISTS	PBORCA_OBJEXISTS

例

この例は、SOURCE.PBL ライブラリから d_labels という名前のデータウィンドウをエクスポートし、szEntrySource　というバッファに PBORCA_UTF8 ソース コードを配置します。エクスポート ヘッダが含まれます。

```
TCHAR szEntrySource[60000];
// UTF8 ソース エンコーディングを示す
lpORCA_Info->pConfig->eExportEncoding = PBORCA_UTF8;
// エクスポート ヘッダを要求する
lpORCA_Info->pConfig->bExportHeaders  = TRUE;
// 出力をメモリ バッファに書き込む
lpORCA_Info->pConfig->bExportCreateFile = FALSE;
// 既存のセッション環境設定を上書きする
PBORCA_ConfigureSession(lpORCA_Info->hORCASession,
lpORCA_Info->pConfig);
lpORCA_Info->lReturnCode = PBORCA_LibraryEntryExport(
lpORCA_Info->hORCASession,
   _TEXT("c:¥¥app¥¥source.pbl"),
   _TEXT("d_labels"), PBORCA_DATAWINDOW,
   (LPTSTR) szEntrySource, 60000);
```

この例は、SOURCE.PBL ライブラリから d_labels という名前のデータウィンドウをエクスポートし、PBORCA_UNICODE ソース コードを c:¥app¥d_labels.srd に書き込みます。エクスポート ヘッダが含まれます。

```
// UNICODE ソース エンコーディングを示す
lpORCA_Info->pConfig->eExportEncoding =
   PBORCA_UNICODE;
// ファイルに書き込む
```

```
lpORCA_Info->pConfig->bExportCreateFile = TRUE;
// 出力ディレクトリを指定する
lpORCA_Info->pConfig->pExportDirectory =
    _TEXT("c:¥¥app");
// エクスポート ヘッダを要求する
lpORCA_Info->pConfig->bExportHeaders  = TRUE;
// 既存のセッション環境設定を上書きする
PBORCA_ConfigureSession(lpORCA_Info->hORCASession,
lpORCA_Info->pConfig);
// 実際のエクスポートを実行する
lpORCA_Info->lReturnCode = PBORCA_LibraryEntryExport(
   lpORCA_Info->hORCASession,
   _TEXT("c:¥¥app¥¥source.pbl"),
   _TEXT("d_labels"), PBORCA_DATAWINDOW,
   NULL, 0);
```

この例は、SOURCE.PBL ライブラリから w_connect という名前のウィンドウをエクスポートします。ウィンドウには、埋め込み OLE オブジェクトが含まれます。ソース コードとバイナリ オブジェクトは、どちらも c:¥app¥w_connect.srw にエクスポートされます。エクスポート ヘッダが含まれ、ソースは PBORCA_ANSI_DBCS 形式で書き込まれます。

```
// ANSI_DBCS ソース エンコーディングを示す
lpORCA_Info->pConfig->eExportEncoding =
   PBORCA_ANSI_DBCS;
// ファイルにエクスポートする
lpORCA_Info->pConfig->bExportCreateFile = TRUE;
// 出力ディレクトリを指定する
lpORCA_Info->pConfig->pExportDirectory =
    _TEXT("c:¥¥app");
// エクスポート ヘッダを要求する
lpORCA_Info->pConfig->bExportHeaders  = TRUE;
// バイナリ コンポーネントを含める
lpORCA_Info->pConfig->bExportIncludeBinary = TRUE;
// 既存のセッション環境設定を上書きする
PBORCA_ConfigureSession(lpORCA_Info->hORCASession,
lpORCA_Info->pConfig);
// 実際のエクスポートを実行する
lpORCA_Info->lReturnCode = PBORCA_LibraryEntryExport(
    lpORCA_Info->hORCASession,
    _TEXT("c:¥¥app¥¥source.pbl"),
    _TEXT("w_connect"), PBORCA_WINDOW,
    NULL, 0);
```

関連項目

PBORCA_ConfigureSession
PBORCA_CompileEntryImport
PBORCA_LibraryEntryExportEx

PBORCA_LibraryEntryExportEx

機能

PowerBuilder ライブラリ エントリのソース コードをテキスト バッファにエクスポートします。

構文

```
INT PBORCA_LibraryEntryExportEx ( HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTR lpszEntryName,
        PBORCA_TYPE otEntryType,
        LPTSTR lpszExportBuffer,
        LONG lExportBufferSize
        LONG *plReturnSize);
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	エクスポートするオブジェクトを含むライブラリのファイル名を値に持つ文字列へのポインタ
lpszEntryName	エクスポートされるオブジェクトの名前を値に持つ文字列へのポインタ
otEntryType	エクスポートされるエントリのオブジェクト型を指定する PBORCA_TYPE カタログ データ型の値。値は以下のとおりです。 PBORCA_APPLICATION PBORCA_BINARY PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW
lpszExportBuffer	PBORCA_CONFIG_SESSION プロパティの bExportCreateFile が FALSE の場合に、ORCA がエクスポートされるソースのコードを格納するデータ バッファへのポインタ。bExportCreateFile が TRUE の場合、この引数は NULL にすることができます。
lExportBufferSize	lpszExportBuffer のサイズ（バイト）。この引数は、PBORCA_CONFIG_SESSION プロパティの bExportCreateFile が TRUE の場合は必要ありません。
**plReturnSize*	エクスポートされるソース バッファまたはファイルのサイズ（バイト）

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-3 PBORCA_OBJNOTFOUND	オブジェクトが見つからない
-4 PBORCA_BADLIBRARY	ライブラリ名が正しくない
-7 PBORCA_LIBIOERROR	ライブラリ I/O エラー
-10 PBORCA_BUFFERTOOSMALL	バッファ サイズが小さすぎる
-33 PBORCA_DBCSERROR	Unicode を ANSI_DBCS に変換する際のロケールの設定エラー

解説

この関数は、エクスポート されるソースのサイズを追加の *plReturnSize 引数で呼び出し 元へ返すという 点以外は、PBORCA_LibraryEntryExport とまったく 同じ 動作をします。

関連項目

PBORCA_ConfigureSession
PBORCA_CompileEntryImport
PBORCA_LibraryEntryExport

PBORCA_LibraryEntryInformation

機能

PowerBuilder ライブラリ中のオブジェクトに関する情報を返します。情報には、コメント、ソースのサイズ、オブジェクトのサイズ、および修正時間が含まれています。

構文

```
INT PBORCA_LibraryEntryInformation ( HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTR lpszEntryName,
        PBORCA_TYPE otEntryType,
        PPBORCA_ENTRYINFO pEntryInformationBlock );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	情報が必要なオブジェクトを含むライブラリのファイル名を値に持つ文字列へのポインタ
lpszEntryName	情報が必要なオブジェクトの名前を値に持つ文字列へのポインタ
otEntryType	エントリのオブジェクト型を指定する PBORCA_TYPE カタログ データ型の値。値は以下のとおりです。 PBORCA_APPLICATION PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT PBORCA_BINARY
pEntryInformationBlock	ORCA が要求された情報を格納する PBORCA_ENTRYINFO 構造体へのポインタ（以下の解説を参照）

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト
-3　PBORCA_OBJNOTFOUND	オブジェクトが見つからない
-4　PBORCA_BADLIBRARY	ライブラリ名が正しくない
-7　PBORCA_LIBIOERROR	ライブラリ I/O エラー

解説

この関数を呼び出す前に、ライブラリ リストや現行のアプリケーションを設定する必要はありません。

エントリ情報の戻り方　PBORCA_LibraryEntryInformation は、次の構造体にエントリに関する情報を格納します。*pEntryInformationBlock* 引数に、構造体へのポインタを渡します。

```
typedef struct PBORCA_EntryInfo
{
    TCHAR szComments[PBORCA_MAXCOMMENT + 1];
    LONG lCreateTime; // エントリの作成 - 変更時間
    LONG lObjectSize; // オブジェクトのサイズ（バイト）
    LONG lSourceSize; // ソースのサイズ（バイト）
} PBORCA_ENTRYINFO, FAR *PPBORCA_ENTRYINFO;
```

ソース コード サイズの使用法　PBORCA_LibraryEntryInformation は、オブジェクトのエクスポート ソースを取得するために必要なソース バッファのサイズ（バイト）を見積もるためによく使用されます。エクスポートされるソースのサイズは、事実上、ConfigureSession の設定によって変化します。次の表では、LibraryEntryInformation が返す lSourceSize 値に ConfigureSession 変数が与える影響について説明しています。

ConfigureSession 変数	lSourceSize の影響
ANSI/DBCS ORCA クライアント	影響なし。ユーザは、この表以降で記述する使用方法のヒントに基づいて必要なバッファ サイズを計算します。
eExportEncoding	影響なし。PBORCA_LibraryEntryInformation は常に Unicode ソースに必要なバイト 数を返します。
bExportHeaders=TRUE	*otEntryType* が PBORCA_BINARY でない場合、lSourceSize は Unicode エクスポート ヘッダを生成するために必要なバイト数ずつ増分されます。
bExportIncludeBinary=TRUE	*otEntryType* が PBORCA_BINARY でない場合、lSourceSize は Unicode 表記のバイナリ オブジェクトを生成するために必要なバイト数ずつ増分されます。

非 Unicode エンコーディングに必要なバッファ サイズの計算方法　非 Unicode のエクスポート エンコーディングに必要なバッファ サイズは、実際のデータ変換を実行する前には計算できません。開発者は、割り当てるための適切なバッファ サイズを独自に見積もる必要があります。たとえば、エントリ のソースがすべて ANSI の場合は、単純に lSourceSize の値を 2 で割り 、Null 区切り 文字が必要であれば 1 バイト を足します。Unicode ソースの場合は、Null 区切り 文字用に 2 バイト を足します。

エントリ タイプとして PBORCA_BINARY を使用する方法　ORCA の以前のリリースでは、エントリに埋め込み OLE コントロールが含まれているかを判断するために、PBORCA_BINARY を *otEntryType* に設定して 2 回目の PBORCA_LibraryEntryInformation 呼び出しを行う必要がありました。この呼び出しはエクスポートするバイナリ データの表記を保持するために必要なバッファ サイズを決定します。PowerBuilder では下位互換性のために依然としてこの機能をサポートしますが、エントリのソース コンポーネントおよびバイナリ コンポーネントの両方に十分なバッファ サイズを取得するには、pConfigSession->bExportIncludBinary = TRUE を設定するほうがより効果的です。

例

この例は、PBL 中の各オブジェクトの情報を取得します。これは、68 ページの「PBORCA_LibraryDirectory」の例を拡張したものです。

```
INT EntryInfo(PLIBINFO_HEAD  pHead)
{
INT    iErrCode;
INT    idx;
PLIBINFO_ENTRY    pCurrEntry;
PBORCA_ENTRYINFO  InfoBlock;
INT    iErrCount = 0;
for (idx = 0, pCurrEntry = pHead->pEntryAnchor;
      (idx < pHead->iNumEntries) && pCurrEntry;
      idx++, pCurrEntry = pCurrEntry->pNext)
    {
iErrCode = PBORCA_LibraryEntryInformation(
    lpORCA_Info->hORCASession pHead->szLibName,
    pCurrEntry->szEntryName,
    pCurrEntry->otEntryType, &InfoBlock);


    if (iErrCode == PBORCA_OK)
    {
     pCurrEntry->lSourceSize = InfoBlock.lSourceSize;
     pCurrEntry->lObjectSize = InfoBlock.lObjectSize;
    }
    else
    {
       ErrorMsg();
        iErrCount++;
    }
  }
  if (iErrCount)
    iErrCode = -1;
  return iErrCode;
}
```

関連項目

PBORCA_LibraryDirectory
PBORCA_LibraryEntryExport

PBORCA_LibraryEntryMove

機能

あるライブラリから別のライブラリに、PowerBuilder ライブラリ エントリを移動します。

構文

```
INT PBORCA_LibraryEntryMove ( PBORCA hORCASession,
        LPTSTR lpszSourceLibName,
        LPTSTR lpszDestLibName,
        LPTSTR lpszEntryName,
        PBORCA_TYPE otEntryType );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszSourceLibName	オブジェクトを含むソース ライブラリのファイル名を値に持つ文字列へのポインタ
lpszDestLibName	移動するオブジェクトの格納先ライブラリのファイル名を値に持つ文字列へのポインタ
lpszEntryName	移動されるオブジェクトの名前を値に持つ文字列へのポインタ
otEntryType	移動されるエントリのオブジェクト型を指定する PBORCA_TYPE のカタログ データ型の値。値は以下のとおりです。 PBORCA_APPLICATION PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-3 PBORCA_OBJNOTFOUND	オブジェクトが見つからない
-4 PBORCA_BADLIBRARY	ライブラリ名が正しくない
-7 PBORCA_LIBIOERROR	ライブラリ I/O エラー

解説

この関数を呼び出す前に、ライブラリ リストや現行のアプリケーションを設定する必要はありません。

PBORCA_LibraryEntryCopy と同様に、最初の呼び出しで PBORCA_LibraryEntryMove は自動的にソース コンポーネントを移動し、存在すればオブジェクトのバイナリ コンポーネントを移動します。

例

この例は、d_labels という名前のデータウィンドウを SOURCE.PBL ライブラリから DESTIN.PBL ライブラリに移動します。

```
lpORCA_Info->lReturnCode = PBORCA_LibraryEntryMove(
    lpORCA_Info->hORCASession,
    _TEXT("c:¥¥app¥¥source.pbl"),
     _TEXT ("c:¥¥app¥¥destin.pbl"),
      _TEXT("d_labels"), PBORCA_DATAWINDOW);
```

この例では、lpszSourceLibraryName、lpszDestinationLibraryName および lpszEntryName のポインタが有効なライブラリとオブジェクト名を指し示しており、otEntryType は有効なオブジェクト型であると仮定しています。

```
lpORCA_Info->lReturnCode = PBORCA_LibraryEntryMove(
    lpORCA_Info->hORCASession,
    lpszSourceLibraryName, lpszDestinationLibraryName,
    lpszEntryName, otEntryType );
```

関連項目

PBORCA_LibraryEntryCopy
PBORCA_LibraryEntryDelete

PBORCA_ObjectQueryHierarchy

機能

PowerBuilder オブジェクトのクエリを実行し、先祖階層内のオブジェクトの一覧を取得します。ウィンドウ、メニュー、ユーザ オブジェクトのみに先祖階層があり、これらに対してクエリを実行できます。

構文

```
INT PBORCA_ObjectQueryHierarchy ( HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTR lpszEntryName,
        PBORCA_TYPE otEntryType,
        PBORCA_HIERPROC pHierarchyProc,
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	クエリを実行されるオブジェクトを含むライブラリのファイル名を値に持つ文字列へのポインタ
lpszEntryName	クエリを実行されるオブジェクトの名前を値に持つ文字列へのポインタ
otEntryType	クエリを実行されるエントリのオブジェクト型を指定する PBORCA_TYPE カタログ データ型の値。指定できるのは、以下の値のみです。 PBORCA_WINDOW PBORCA_MENU PBORCA_USEROBJECT
pHierarchyProc	PBORCA_ObjectQueryHierarchy コールバック関数へのポインタ。コールバック関数は、先祖オブジェクトごとに呼び出されます。 ORCA がコールバック関数に渡す情報は先祖オブジェクト名で、PBORCA_HIERARCHY 型の構造体に格納されています。
pUserData	コールバック関数 PBORCA_ObjectQueryHierarchy に渡されるユーザ データへのポインタ 通常ユーザ データには、コールバック関数がバッファ サイズの情報と先祖名を格納するバッファまたはバッファへのポインタが含まれます。

戻り値

INT 型。戻り値は以下のとおりです。

戻り値	説明
0 PBORCA_OK	処理成功
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-3 PBORCA_OBJNOTFOUND	オブジェクトが見つからない
-4 PBORCA_BADLIBRARY	ライブラリ名が正しくない

戻り値	説明
-5　PBORCA_LIBLISTNOTSET	ライブラリ リストが未設定
-6　PBORCA_LIBNOTINLIST	ライブラリ リストにライブラリが見つからない
-7　PBORCA_LIBIOERROR	ライブラリ I/O エラー
-9　PBORCA_INVALIDNAME	名前が PowerBuilder の命名規則に違反している

解説

この関数を呼び出す前に、ライブラリ リストと現行のアプリケーションを設定する必要があります。

例

この例は、WINDOWS.PBL ライブラリ中のウィンドウ オブジェクト w_processdata の先祖の一覧を取得します。lpUserData バッファは、名前の一覧を格納するスペースへのポインタを事前に設定しています。

オブジェクト の階層中の各先祖に対して、PBORCA_ObjectQueryHierarchy はコールバック関数 ObjectQueryHierarchy を呼び出します。ObjectQueryHierarchy 用に作成したコード では、lpUserData が指し示すバッファに先祖名を格納します。例では、lpUserData バッファはすでに設定済みです。

```
PBORCA_HIERPROC fpHierarchyProc;
fpHierarchyProc = (PBORCA_HIERPROC)GetHierarchy;
lpORCA_Info->lReturnCode =
    PBORCA_ObjectQueryHierarchy(
   _TEXT("c:¥¥app¥¥windows.pbl"),
   _TEXT("w_processdata"),
  PBORCA_WINDOW,
  fpHierarchyProc,
  lpUserData );
```

コールバック用データ バッファの設定についての詳細は、15 ページの「コールバック関数の内容」と PBORCA_LibraryDirectory の例を参照してください。

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_ObjectQueryReference

PBORCA_ObjectQueryReference

機能

PowerBuilder オブジェクトに対してクエリを実行し、ほかのオブジェクトに対する参照の一覧を取得します。

構文

```
INT PBORCA_ObjectQueryReference ( HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTR lpszEntryName,
        PBORCA_TYPE otEntryType,
        PBORCA_REFPROC pRefProc,
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	クエリを実行されるオブジェクトを含むライブラリのファイル名を値に持つ文字列へのポインタ
lpszEntryName	クエリを実行されるオブジェクトの名前を値に持つ文字列へのポインタ
otEntryType	クエリを実行されるエントリのオブジェクト型を指定する PBORCA_TYPE カタログ データ型の値。値は以下のとおりです。 PBORCA_APPLICATION PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT
pRefProc	コールバック関数 PBORCA_ObjectQueryReference へのポインタ。コールバック関数は、各参照オブジェクトごとに呼び出されます。 ORCA がコールバック関数に渡す情報は、PBORCA_REFERENCE 型の構造体に格納される参照オブジェクト名、そのライブラリ、およびそのオブジェクト型です。
pUserData	コールバック関数 PBORCA_ObjectQueryReference に渡されるユーザ データへのポインタ 通常ユーザ データには、コールバック関数がバッファのサイズに関する情報とオブジェクトの情報を格納するバッファまたはそのバッファへのポインタが含まれます。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト
-3　PBORCA_OBJNOTFOUND	オブジェクトが見つからない
-4　PBORCA_BADLIBRARY	ライブラリ名が正しくない
-5　PBORCA_LIBLISTNOTSET	ライブラリ リストが未設定
-6　PBORCA_LIBNOTINLIST	ライブラリ リストにライブラリが見つからない
-9　PBORCA_INVALIDNAME	名前が PowerBuilder の命名規則に違反している

解説

この関数を呼び出す前に、ライブラリ リストと現行のアプリケーションを設定する必要があります。

例

この例は、WINDOWS.PBL ライブラリ 中のウィンドウ オブジェクト w_processdata の参照オブジェクト の一覧を取得します。w_processdata が参照するオブジェクト ごとに PBORCA_ObjectQueryReference はコールバック関数 ObjectQueryReference を呼び出します。ObjectQueryReference のコード の記述では、lpUserData が指し示すバッファにオブジェクト 名を格納します。例では、lpUserData バッファはすでに設定済みです。

```
PBORCA_REFPROC    fpRefProc;
fpRefProc = (PBORCA_REFPROC) GetReferences;
lpORCA_Info->lReturnCode =
PBORCA_ObjectQueryReference(
   lpORCA_Info->hORCASession,
   _TEXT("c:¥¥app¥¥windows.pbl"),
   _TEXT("w_processdata"),
PBORCA_WINDOW,
fpRefProc,
lpUserData );
```

コールバック用データ バッファの設定についての詳細は、15 ページの「コールバック関数の内容」と PBORCA_LibraryDirectory の例を参照してください。

24 ページの「例について」で示されるように、これらの例のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_ObjectQueryHierarchy

PBORCA_SccClose

機能

アクティブな SCC プロジェクトを閉じます。

構文

INT **PBORCA_SccClose** (HPBORCA *hORCASession*);

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル

戻り値

INT 型。

解説

このメソッドは、ソース管理プロバイダから切断するために SCCUninitialize を呼び出します。PBORCA_SessionClose を呼び出す前に PBORCA_SccClose を呼び出します。

関連項目

PBORCA_SccConnect

PBORCA_SccConnect

機能

ソース管理を初期化してプロジェクトを開きます。

構文

```
INT PBORCA_SccConnect ( HPBORCA hORCASession,
        PBORCA_SCC *pConfig );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
**pConfig*	事前に割り当てた構造体へのポインタ。通常はゼロに初期化されます。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-22　PBORCA_SCCFAILURE	ソース管理に接続できない
-23　PBORCA_REGREADERROR	レジストリを読み込めない
-24　PBORCA_LOADDLLFAILED	DLL をロードできない
-25　PBORCA_SCCINITFAILED	SCC 接続を初期化できない
-26　PBORCA_OPENPROJFAILED	プロジェクトを開けない

解説

このメソッドは、PBORCA_SCC 構造体で指定される接続情報に基づいてソース管理のセッションを初期化します。PBORCA_SCC 構造体は以下のように定義されています。

```
typedef struct pborca_scc
{
   HWND hWnd;
   TCHAR szProviderName [PBORCA_SCC_NAME_LEN + 1];
   LONG *plCapabilities;
   TCHAR szUserID [PBORCA_SCC_USER_LEN + 1];
   TCHAR szProject [PBORCA_SCC_PATH_LEN + 1];
   TCHAR szLocalProjPath [PBORCA_SCC_PATH_LEN + 1];
   TCHAR szAuxPath [PBORCA_SCC_PATH_LEN + 1];
   TCHAR szLogFile [PBORCA_SCC_PATH_LEN + 1];
   LPTEXTOUTPROC pMsgHandler;
   LONG *pCommentLen;
   LONG lAppend;
   LPVOID pCommBlk;
} PBORCA_SCC;
```

構造体を手動で用意するか、PBORCA_SccGetConnectProperties を呼び出して、指定されたワークスペース ファイルに関連する接続情報を取得することができます。この関数は以下のことを行います。

- 要求されたソース管理プロジェクトを開く
- PBORCA_SCC メソッドを実装する CPB_OrcaSourceControl クラスを作成する
- PBORCA_SccClose　が呼び出されるまで保持されるランタイム環境を定義する

ランタイム環境には、ランタイム エンジン（rt）、オブジェクト マネージャ（ob）、PowerScript コンパイラ（cm）、およびストレージ マネージャ（stg）という 4 つのサブシステムがあります。ランタイム環境は、後続の PBORCA_SccSetTarget 呼び出しによって認識されるターゲットを処理するために使用されます。複数のターゲットを処理するには、SCC 接続を終了し、ORCA セッションを閉じて、新しい ORCA セッションを開く必要があります。

例

次の例は、PBNative ソース管理に接続しています。

```
PBORCA_SCC  sccConfig;
memset(&sccConfig, 0x00, sizeof(PBORCA_SCC));
//  PBNative に接続プロパティを手動で設定する
_tcscpy(sccConfig.szProviderName, _TEXT("PB Native"));
_tcscpy(sccConfig.szProject,
  _TEXT("c:¥¥PBNative_Archive¥¥qadb"));
_tcscpy(sccConfig.szUserID, _TEXT("Joe"));
_tcscpy(sccConfig.szLogFile,
  _TEXT("c:¥¥qadb¥¥orcascc.log"));
_tcscpy(sccConfig.szLocalProjPath, _TEXT("c:¥¥qadb"));
sccConfig.lAppend = 0;
lpORCA_Info->lReturnCode = PBORCA_SccConnect(
   lpORCA_Info->hORCASession,
   &sccConfig);
```

関連項目

PBORCA_SccClose
PBORCA_SccConnectOffline
PBORCA_SccGetConnectProperties
PBORCA_SccSetTarget

PBORCA_SccConnectOffline

機能

ソース管理されるプロジェクトを開き、オフラインをリフレッシュして再構築します。

構文

```
INT PBORCA_SccConnectOffline ( HPBORCA hORCASession,
        PBORCA_SCC *pConfig );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
**pConfig*	事前に割り当てた構造体へのポインタ。通常はゼロに初期化されます。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-22　PBORCA_SCCFAILURE	ソース管理に接続できない
-23　PBORCA_REGREADERROR	レジストリを読み込めない
-24　PBORCA_LOADDLLFAILED	DLL をロードできない
-25　PBORCA_SCCINITFAILED	SCC 接続を初期化できない
-26　PBORCA_OPENPROJFAILED	プロジェクトを開けない

解説

この関数は、PBORCA_SCC_IMPRORTONLY が後続の PBORCA_SccSetTarget コマンドで指定されている場合にのみ使用可能です。

インポートのみの処理では、ソース管理されているターゲットをリフレッシュするために必要なオブジェクトは、すべてローカル プロジェクト　パス上に存在していると想定しています。このため、PBORCA_SccConnectOffline は ORCA ソース管理クラスをインスタンス化しますが、実際には SCC プロバイダに接続しません。

この関数は、ラップトップ コンピュータを使用する開発者にとって特に便利です。ネットワークに接続しながら、SCC クライアント ビューをリフレッシュすることが可能です。その後、時間外に、ネットワーク接続を必要としない時間のかかるリフレッシュやアプリケーションの再構築などを実行することができます。

例

この例は、現行の作業ディレクトリにある PocketBuilder qadb.pkw ワークスペース ファイルから、接続情報と共に PBORCA_SCC 構造体を取り込みます。次に、オフライン モードで接続し、現行の作業ディレクトリの下にある qadbtest サブディレクトリに置かれている qadbtest.pbt ターゲット ファイルをリフレッシュします。同期されていないオブジェクトのみがリフレッシュされます。現行ユーザによってチェックアウトされたオブジェクトは上書きされません。

```
PBORCA_SCC    sccConfig;
TCHAR      szWorkSpace[PBORCA_SCC_PATH_LEN];
TCHAR      szTarget[PBORCA_SCC_PATH_LEN];
LONG          lFlags;
memset(&sccConfig, 0x00, sizeof(PBORCA_SCC));
_tcscpy(szWorkSpace,_TEXT("qadb.pkw"));
lpORCA_Info->lReturnCode =
PBORCA_SccGetConnectProperties(
    lpORCA_Info->hORCASession,
    szWorkspace,
    &sccConfig);
if (lpORCA_Info->lReturnCode == PBORCA_OK)
    {
  // 構築処理用に別のログ ファイルを指定
  _tcscpy(sccConfig.szLogFile, _TEXT("bldqadb.log"));
  sccConfig.lAppend = 0;
  lpORCA_Info->lReturnCode = PBORCA_SccConnectOffline(
      lpORCA_Info->hORCASession, &sccConfig);
  if (lpORCA_Info->lReturnCode == PBORCA_OK)
  {
    _tcscpy(szTarget, _TEXT("qadbtest¥¥qadbtest.pkt"));
    lFlags = PBORCA_SCC_IMPORTONLY |
       PBORCA_SCC_OUTOFDATE |
       PBORCA_SCC_EXCLUDE_CHECKOUT;
     lpORCA_Info->lReturnCode = PBORCA_SccSetTarget(
       lpORCA_Info->hORCASession,
       szTarget,
       lFlags,
       NULL,
       NULL);
 if (lpORCA_Info->lReturnCode == PBORCA_OK)
    {
     lpORCA_Info->lReturnCode =
       PBORCA_SccRefreshTarget(
       lpORCA_Info->hORCASession, PBORCA_FULL_REBUILD);
     }
  }
}
```

関連項目

PBORCA_SccClose
PBORCA_SccConnect
PBORCA_SccGetConnectProperties
PBORCA_SccSetTarget

PBORCA_SccExcludeLibraryList

機能

次の PBORCA_SccRefreshTarget 処理で同期する必要のないターゲットのライブラリ リスト中のライブラリ名です。

構文

```
INT PBORCA_SccExcludeLibraryList ( HPBORCA hORCASession,
        LPTSTR *pLibNames,
        INT iNumberofLibs );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
**pLibNames*	リフレッシュされないライブラリの名前
iNumberofLibs	リフレッシュされないライブラリ数

戻り値

INT 型。

解説

複数のターゲット で PBL が共有されており 、一覧にしたライブラリ が前の PBORCA_SccRefreshTarget 処理で正常にリ フレッシュされたと 確信する場合、このメソッド は便利です。ターゲット をリ フレッシュする処理では、除外されるライブラリ はリ フレッシュされませんが、除外されたライブラリもアプリ ケーション のフル再構築では使用されます。

例

前の PBORCA_SccRefreshTarget 処理では、このターゲット ライブラリ リスト中の 4 つの PocketBuilder ライブラリのうち 3 つが正常にリフレッシュされています。

```
LPTSTR pExcludeArray[3];
INT    lExcludeCount = 3;
TCHAR  szTarget[PBORCA_SCC_PATH_LEN];
LONG          lFlags;
pExcludeArray[0] = new TCHAR[PBORCA_SCC_PATH_LEN];
pExcludeArray[1] = new TCHAR[PBORCA_SCC_PATH_LEN];
pExcludeArray[2] = new TCHAR[PBORCA_SCC_PATH_LEN];
_tcscpy(pExcludeArray[0],
_TEXT("..¥¥shared_obj¥¥shared_obj.pkl"));
_tcscpy(pExcludeArray[1],
_TEXT("..¥¥datatypes¥¥datatypes.pkl"));
_tcscpy(pExcludeArray[2],
_TEXT("..¥¥chgreqs¥¥chgreqs.pkl"));
//  ORCA セッションを開き、SCC に接続
// …
_tcscpy(szTarget, _TEXT("dbauto¥¥dbauto.pkt"));
lFlags = PBORCA_SCC_IMPORTONLY | PBORCA_SCC_OUTOFDATE |
    PBORCA_SCC_EXCLUDE_CHECKOUT;
lpORCA_Info->lReturnCode = PBORCA_SccSetTarget(
lpORCA_Info->hORCASession, szTarget, lFlags, NULL,
    NULL);
```

```
if (lpORCA_Info->lReturnCode == PBORCA_OK)
   {
    lpORCA_Info->lReturnCode =
         PBORCA_SccExcludeLibraryList(
lpORCA_Info->hORCASession, pExcludeArray,
lExcludeCount);

if (lpORCA_Info->lReturnCode == PBORCA_OK)
{
lpORCA_Info->lReturnCode = PBORCA_SccRefreshTarget(
lpORCA_Info->hORCASession, PBORCA_FULL_REBUILD );
     }
 }
for (int i = 0; i < lExcludeCount; i++)
delete [] pExcludeArrary[i];
```

関連項目

PBORCA_SccRefreshTarget
PBORCA_SccSetTarget

PBORCA_SccGetConnectProperties

機能

PowerBuilder ワークスペースに関連した SCC 接続プロパティを返します。

構文

```
INT PBORCA_SccGetConnectProperties ( HPBORCA hORCASession,
        LPTSTR pWorkspaceFile,
        PBORCA_SCC *pConfig );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
pWorkspaceFile	PowerBuilder のワークスペース ファイル（PBW）の完全ファイル名または相対ファイル名。
**pConfig*	事前に割り当てた構造体へのポインタ。通常はゼロに初期化されます。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-3　PBORCA_OBJNOTFOUND	ワークスペース ファイルが見つからない

解説

このメソッドは、SCC 接続プロセスを簡素化します。PBORCA_SccGetConnectProperties 呼び出しの引数として含めたワークスペースから戻されるプロパティ値は、事前に割り当てられた構造体 PBORCA_SCC に格納されます。これらのプロパティにより、指定の SCC プロバイダとプロジェクトに正常に接続できますが、これらのプロパティはどれも上書きできます。

PBORCA_SCC 構造体は以下のように定義されています。

```
typedef struct pborca_scc {
    HWND hWnd;
    TCHAR szProviderName [PBORCA_SCC_NAME_LEN + 1];
    LONG *plCapabilities;
    TCHAR szUserID [PBORCA_SCC_USER_LEN + 1];
    TCHAR szProject [PBORCA_SCC_PATH_LEN + 1];
    TCHAR szLocalProjPath [PBORCA_SCC_PATH_LEN + 1];
    TCHAR szAuxPath [PBORCA_SCC_PATH_LEN + 1];
    TCHAR szLogFile [PBORCA_SCC_PATH_LEN + 1];
    LPTEXTOUTPROC pMsgHandler;
    LONG *pCommentLen;
    LONG lAppend;
    LPVOID pCommBlk;
} PBORCA_SCC;
```

PBORCA_SCC 構造体中の変数については、次の表で説明します。

メンバー	説明
hWnd	親ウィンドウのハンドルで、通常、その値は NULL
szProviderName	SCC プロバイダの名前
**plCapabilities*	PBORCA_SccConnect からの戻り値へのポインタ。SCC プロバイダがサポートする機能を判断するために内部で使用します。
szUserID	ソース管理プロジェクト用のユーザ ID
szProject	ソース管理プロジェクトの名前
szLocalProjPath	プロジェクトのローカル ルート ディレクトリ
szAuxPath	補助的なプロジェクト パス（Auxiliary Project Path）には、各 SCC ベンダごとに異なる意味があります。SCC プロバイダがオブジェクトに関連付けるすべての文字列を含めることができます。PBORCA_SccGetConnectProperties はこの値を戻して、SCC プロバイダのダイアログ ボックスを開かずに、暗黙的な接続を実現します。
szLogFile	SCC 接続用のログ ファイル名
pMsgHandler	SCC メッセージ用のコールバック関数
**pCommentLen*	PBORCA_SccConnect からの戻り値へのポインタ。SCC プロバイダが受け取るコメントの長さです。
lAppend	SCC ログ ファイルを追加する（*lAppend*=1）か、上書きする（*lAppend*=0）かを決める
pCommBlk	内部使用のために保持

PBORCA_SccGetConnectProperties 関数を呼び出した後に PBORCA_SCC 構造体に追加されるプロパティ値は、szProviderName、szUserID、szProject、szLocalProjPath、szAuxPath、szLogFile、および lAppend です。PBORCA_SCC 構造体へこれらの値を手動で追加する場合は、ソース管理へ接続するために PBORCA_SccGetConnectProperties を呼び出す必要はありません。

関連項目

PBORCA_SccConnect
PBORCA_SccSetTarget

PBORCA_SccGetLatestVersion

機能

SCC プロバイダから最新バージョンのファイルを取り出します。

構文

```
INT PBORCA_SccGetLatestVer ( HPBORCA hORCASession,
        Long nFiles,
        LPTSTR *ppFileNames );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
nFiles	取得されるファイル数
**ppFileNames*	取得されるファイルの名前

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-22　PBORCA_SCCFAILURE	処理失敗

解説

このメソッドを呼び出して、ソース管理からファイルを取得します。通常、オブジェクトは PowerBuilder ライブラリの外部に存在していますが、それでもアプリケーションに属しています。例には、BMP、JPG、ICO、DOC、HLP、HTM、JSP、および PBR ファイルが含まれます。

例

例は以下の通りです。

```
LPTSTRpOtherFiles[3];
pOtherFiles[0] =
   _TEXT("c:¥¥qadb¥¥qadbtest¥¥qadbtest.hlp");
pOtherFiles[1] =
   _TEXT("c:¥¥qadb¥¥datatypes¥¥datatypes.pbr");
pOtherFiles[2] = _TEXT("c:¥¥qadb¥¥qadbtest.bmp");

lpORCA_Info->lReturnCode = PBORCA_SccGetLatestVer
   (lpORCA_Info->hORCASession, 3, pOtherFiles);
```

関連項目

PBORCA_SccConnect
PBORCA_SccSetTarget

PBORCA_SccRefreshTarget

機能

ターゲット ライブラリ中の各オブジェクトのソースをリフレッシュするために SccGetLatestVersion を呼び出します。

構文

```
INT PBORCA_SccRefreshTarget ( HPBORCA hORCASession,
        PBORCA_REBLD_TYPE eRebldType );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
eRebldType	アプリケーションの再構築の方法を指定することができる（解説の項を参照）

戻り値

INT 型。

解説

このメソッドは、ソース管理からターゲット ライブラリ中のオブジェクトの最新バージョンを取得するために呼び出します。また、リフレッシュ処理により、オブジェクトが、対応する PowerBuilder ライブラリにインポートされ、コンパイルされます。

PBORCA_SccExcludeLibraryList 呼び出しで指定するターゲット ライブラリ中のオブジェクトは、リフレッシュ処理に含まれません。

PBORCA_REBLD_TYPE 引数は、PBORCA_SccRefreshTarget を呼び出したときにアプリケーションをどのように再構築するかを決定します。

PBORCA_REBLD_TYPE	説明
PBORCA_FULL_REBUILD	アプリケーションをフル再構築する
PBORCA_INCREMENTAL_REBUILD	アプリケーションをインクリメンタル再構築する
PBORCA_MIGRATE	アプリケーションを移行して、フル再構築する

関連項目

PBORCA_SccClose
PBORCA_SccConnect
PBORCA_SccExcludeLibraryList
PBORCA_SccSetTarget

PBORCA_SccResetRevisionNumber

機能

オブジェクトのリビジョン番号をリセットするためにこの関数を呼び出します。この関数は、SCC API に SccQueryInfoEx エクステンションを実装した SCC プロバイダを使用しているアプリケーションでのみ有用です。

構文

```
INT PBORCA_SccResetRevisionNumber ( HPBORCA hORCASession,
        LPTSTR lpszLibraryName,
        LPTSTR lpszEntryName,
        PBORCA_TYPE otEntryType,
        LPTSTR lpszRevisionNum );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszLibraryName	リビジョン番号をリセットするオブジェクトを含む PBL ファイルの絶対パスまたは相対パスの指定
lpszEntryName	.sr? 拡張子を含めないオブジェクトの名前を値に持つ文字列へのポインタ
otEntryType	インポートしたエントリのオブジェクトの種類を指定する PBORCA_TYPE カタログ データ型の値。値は以下のとおりです。 PBORCA_APPLICATION PBORCA_BINARY PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW
lpszRevisionNum	文字列値または Null。Null の場合は PBL 中の現行リビジョン番号が削除されます。

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト（*lpszLibraryName* または *lpszEntryName* が Null の場合）
-7　PBORCA_LIBIOERROR	読み書きアクセス用に PBL を開けない

解説

ソース管理に接続しているかどうかに関らず、この関数を呼び出すことができます。PBORCA_SccResetRevisionNumber 関数は、*lpszLibraryName* 引数で割り当てた PowerBuilderライブラリにメタデータとして格納されているオブジェクトのリビジョン番号を変更します。変更されるリビジョン番号はデスクトップ マシンのオブジェクト ソース中のものであり、ソース管理リポジトリ中のものではありません。オブジェクトが存在するライブラリは、現行ライブラリ リスト中に存在する必要はありません。

ORCA プログラムが外部的に PBL のオブジェクト ソースを変更し、以下のいずれかが true の場合、通常は PBORCA_SccResetRevisionNumber を呼び出します。

- ORCA プログラムは、PBORCA_CompileEntryImport 呼び出しによってオブジェクトの特定のリビジョンを PBL にインポートします。ORCA プログラムがインポートされた正確なリビジョン番号を把握している場合は、*lpszRevisionNum* 引数にそのリビジョン番号を指定する必要があります。正確なリビジョン番号がわからない場合、ORCA プログラムは PBORCA_SccResetRevisionNum を呼び出して、*lpszRevisionNum* に NULL を設定します。
- ORCA プログラムは、PBORCA_LibraryEntryExport 呼び出しによって PBL から既存のオブジェクト ソースをエクスポートし、オブジェクト ソースを SCC リポジトリにチェックインすることで SCC にチェックインするのと同様の処理を外部的に行います。ジョブを完了するには、ORCA プログラムは SCC リポジトリから新しいリビジョン番号を取得して PBORCA_SccResetRevisionNumber を呼び出す必要があります。これを行った後、PBL 中に存在するオブジェクト ソースは SCC リポジトリ中の正しいリビジョン番号と関連付けられます。

関連項目

PBORCA_CompileEntryImport
PBORCA_LibraryEntryExport

PBORCA_SccSetTarget

機能

ソース管理からターゲット ファイルを検索し、アプリケーション オブジェクト名を ORCA に渡して、ORCA セッションのライブラリ リストを設定します。

構文

```
INT PBORCA_SccSetTarget ( HPBORCA hORCASession,
        LPTSTR pTargetFile,
        LONG   lFlags,
        PBORCA_SETTGTPROC pSetTgtProc,
        LPVOID pUserData );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
pTargetFile	ターゲット ファイル名
lFlags	ターゲット操作の動作をコントロールすることが可能（解説の項を参照）
pSetTgtProc	ユーザ定義のコールバック関数へのポインタ
pUserData	事前に割り当てられたデータ バッファへのポインタ

戻り値

INT 型。

解説

このメソッドは、従来の ORCA アプリケーションの PBORCA_SetLibraryList と PBORCA_SetCurrentAppl の代わりとなるものです。

ソース管理からのターゲット ファイルの検索とアプリケーション オブジェクトとライブラリ リストの設定に加えて、PBORCA_SccSetTarget はライブラリ リスト中の各ライブラリに対して 1 度だけユーザ定義のコールバック関数を呼び出します。これにより、デフォルトでどのライブラリがリフレッシュされるかがわかります。また、特定の共有ライブラリが事前のタスクですでにリフレッシュ済みの場合には、PBORCA_SccExcludeLibraryList を呼び出す機会が与えられます。

ソース管理から取得するターゲット ライブラリ上でのリフレッシュ動作を設定をするために、lFlags 引数を割り当てます。

フラグ	説明
PBORCA_SCC_OUTOFDATE	PBL 中に存在するオブジェクトが同期されていないかを判断するために比較を行います。PBORCA_SCC_IMPORTONLY を使用している場合、ローカル プロジェクト パス上に存在するソースとは異なるオブジェクトだけがリフレッシュされます。PBORCA_SCC_IMPORTONLY が設定されていない場合、SCC リポジトリよりも古い日付のオブジェクトのみがリフレッシュされます。PBORCA_SCC_OUTOFDATE と PBORCA_SCC_REFRESH_ALL は相互に排他的です。
PBORCA_SCC_REFRESH_ALL	ターゲット ライブラリは完全にリフレッシュされます。PBORCA_SCC_IMPORTONLY を使用している場合、ソース コードはローカル プロジェクト パスから直接インポートされます。PBORCA_SCC_IMPORTONLY が設定されていない場合、最初に全オブジェクトの最新バージョンが SCC プロバイダから取得されてから、ターゲット ライブラリにインポートされます。
PBORCA_SCC_IMPORTONLY	ローカルのプロジェクト パスにすでに存在するターゲット アプリケーションを再構築するために必要なすべてのオブジェクトを示します。SCC ベンダの管理ツールを使用して事前にローカルのパスをリフレッシュした場合は、このフラグを設定します。この ORCA セッション中に事前に PBORCA_SccConnectOffline を呼び出した場合は、PBORCA_SCC_IMPORTONLY が必要です。PBORCA_SCC_IMPORTONLY は、特定の SCC バージョン ラベルやプロモーション グループのターゲットを再構築する場合に特に有用です。

フラグ	説明
PBORCA_SCC_EXCLUDE_CHECKOUT	ユーザの介在を必要としないバッチ ジョブでローカルのターゲットをリフレッシュする仕組みを提供します。その時点でチェックアウトされているオブジェクトが上書きされないようにします。PBORCA_SccConnect と共に使用する場合、チェックアウト ステータスは SCC プロバイダから直接取得されます。PBORCA_SccConnectOffline と共に使用する場合、チェックアウト ステータスは workspace_name.PBC ファイルから取得されます。オフライン処理の場合、ワークスペース名は事前に呼び出した PBORCA_SccGetConnectProperties から取得されます。

PBORCA_SccConnect で指定したローカル プロジェクト パス中にターゲット ライブラリおよびディレクトリが存在しない場合、PBORCA_SccSetTarget 呼び出しにより、これらのディレクトリおよび PBL ファイルは動的に作成されます。

SccSetTarget は、暗黙の PBORCA_SessionSetLibraryList と PBORCA_SessionSetCurrentAppl を行います。PBORCA_SccSetTarget（おそらく PBORCA_SccRefreshTarget も）を呼び出した後に、PBD や EXE の作成など、現行アプリケーションや初期化されたライブラリ リストなどが必要なほかの作業を実行できます。これは PBORCA_SccClose を呼び出して、PBD や EXE を作成するためにライブラリ リストや現行アプリケーションを初期化するより効率的です。

関連項目

PBORCA_SccConnect
PBORCA_SccConnectOffline
PBORCA_SccGetConnectProperties
PBORCA_SccRefreshTarget

PBORCA_SessionClose

機能

ORCA セッションを終了します。

構文

void **PBORCA_SessionClose** (HPBORCA *hORCASession*);

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル

戻り値

なし。

解説

PBORCA_SessionClose は、ORCA セッションに関連して割り当てられている現行のリソースを解放します。セッションを閉じないと、PowerBuilder DLL が割り当てたメモリが解放されず、その結果、メモリ リークが発生します。セッションを閉じることに失敗しても、ORCA セッションは何にも接続していないため、データに影響はありません。

例

この例は、ORCA セッションを閉じます。

```
PBORCA_SessionClose(lpORCA_Info->hORCASession);
lpORCA_Info->hORCASession = 0;
```

24 ページの「例について」で示されるように、これらの例のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_SessionOpen

PBORCA_SessionGetError

機能

ORCA セッションに関する現行のエラーを取得します。

構文

void **PBORCA_SessionGetError** (HPBORCA *hORCASession*, LPTSTR *lpszErrorBuffer*, INT *iErrorBufferSize*);

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszErrorBuffer	ORCA が現行のエラー文字列を入れるバッファへのポインタ
iErrorBufferSize	lpszErrorBuffer が指し示すバッファのサイズ。定数 PBORCA_MSGBUFFER は、推奨されるバッファ サイズである 256 を指定します。これは、ORCA のヘッダ ファイル PBORCA.H で定義されます。

戻り値

なし。

解説

ほかの ORCA 関数呼び出しでエラーが発生した場合は、いつでも PBORCA_SessionGetError を呼び出せます。エラーが発生すると、関数は必ず有用なエラー コードを返します。コードの一覧については、25 ページの「ORCA 戻り値」を参照してください。ただし、PBORCA_SessionGetError を呼び出すことで、ORCA の完全なエラー メッセージを取得することができます。

その時点でエラーがない場合、関数はエラー バッファに空の文字列 ("") をセットします。

例

この例は、現在のエラー メッセージを lpszErrorMessage が指し示す文字列バッファに格納します。バッファ サイズは、事前に設定して dwErrorBufferLen に格納済みです。

```
PBORCA_SessionGetError(lpORCA_Info->hORCASession,
    lpORCA_Info->lpszErrorMessage,
    (int) lpORCA_Info->dwErrorBufferLen);
```

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

PBORCA_SessionOpen

機能

ORCA セッションを確立し、後続の ORCA 呼び出しで使用するハンドルを返します。

構文

HPBORCA **PBORCA_SessionOpen** (void);

戻り値

HPBORCA。成功の場合は ORCA セッションへのハンドルを返し、失敗の場合は 0 を返します。十分なメモリが無い場合のみ、セッションの開始に失敗します。

解説

ほかの ORCA 関数呼び出しの前に、セッションを開く必要があります。

ORCA セッションを開いたままにしても、それに関するオーバーヘッドやリソースの問題はありません。このため、一度セッションを確立したら、必要なだけセッションを開いておくことができます。

オブジェクトのインポートやクエリ、実行ファイルの構築など、一部の ORCA のタスクでは、セッションを開いた後にアプリケーションのコンテキストを指定するために PBORCA_SessionSetLibraryList と PBORCA_SessionSetCurrentAppl を呼び出す必要があります。

同様に PBORCA_SccSetTarget は、SCC 処理に対して暗黙的にアプリケーション コンテキストを提供します。PBORCA_SccSetTarget を呼び出す場合は、PBORCA_SessionSetLibraryList および PBORCA_SetCurrentAppl は呼び出さないでください。

例

この例は、ORCA セッションを開きます。

```
lpORCA_Info->hORCASession = PBORCA_SessionOpen();
if (lpORCA_Info->hORCASession = NULL)
{
lpORCA_Info->lReturnCode = 999;
_tcscpy(lpORCA_Info->lpszErrorMessage,
_TEXT("Open session failed"));
}
```

関連項目

PBORCA_SessionClose
PBORCA_SessionSetLibraryList
PBORCA_SessionSetCurrentAppl

PBORCA_SessionSetCurrentAppl

機能

現行アプリケーションオブジェクトの ORCA セッションを確立します。

構文

INT PBORCA_**SessionSetCurrentAppl** (HPBORCA *hORCASession*,
LPTSTR *lpszApplLibName*, LPTSTR *lpszApplName*);

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
lpszApplLibName	アプリケーション ライブラリの名前を値に持つ文字列へのポインタ
lpszApplName	アプリケーション オブジェクトの名前を値に持つ文字列へのポインタ

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト
-2　PBORCA_DUPOPERATION	現行のアプリケーションはすでに設定済み
-3　PBORCA_OBJNOTFOUND	参照ライブラリが存在しない
-4　PBORCA_BADLIBRARY	ライブラリ名が正しくない
-5　PBORCA_LIBLISTNOTSET	ライブラリ リストが未設定
-6　PBORCA_LIBNOTINLIST	ライブラリ リストに参照ライブラリが見つからない

解説

現行のアプリケーションを設定する前にライブラリ リストを設定する必要があります。

コンパイルやオブジェクトのクエリを行う ORCA 関数を呼び出す前に、PBORCA_SessionSetLibraryList を呼び出してから PBORCA_SessionSetCurrentAppl を呼び出します。ライブラリ名は、どこにあるのかがわかるように完全パスで指定する必要があります。

アプリケーションの変更　ライブラリ リストと現行のアプリケーションの設定は、セッション中に一度だけ行うことができます。設定した後に現行のアプリケーションを変更する必要がある場合は、セッションを閉じてから新しいセッションを開く必要があります。

新しいアプリケーション　空のライブラリを使用して新しいアプリケーションを作成するには、アプリケーション ライブラリ名へのポインタを設定し、アプリケーション名を NULL に設定します。ORCA は内部でデフォルトのアプリケーションをセット アップします。

新しいアプリケーションの作成についての詳細は、20 ページの「新しいアプリケーションのブートストラップ」を参照してください。

例

この例は、現行のアプリケーション オブジェクトに MASTER.PBL ライブラリ中の demo というオブジェクトを設定します。

```
LPTSTR pszLibraryName;
LPTSTR pszApplName;
// ライブラリ名を指定
pszLibraryName =
    _TEXT("c:¥¥app¥¥master.pbl");
// アプリケーション名を指定
pszApplName = _TEXT("demo");
// 現行のアプリケーション オブジェクトを設定
lpORCA->lReturnCode = PBORCA_SessionSetCurrentAppl(
    lpORCA_Info->hORCASession,
    pszLibraryName, pszApplName);
```

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_SessionSetLibraryList

PBORCA_SessionSetLibraryList

機能

ORCA セッションのライブラリ リストを確立します。ORCA は、オブジェクト参照を解決するためにリストの中のライブラリを検索します。

構文

```
INT PBORCA_SessionSetLibraryList ( HPBORCA hORCASession,
        LPTSTR *pLibNames,
        INT iNumberOfLibs );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
**pLibNames*	文字列へのポインタの配列へのポインタ。文字列の値は、ライブラリのファイル名です。どこからでも検索可能なように、各ライブラリは絶対パスで指定します。
iNumberOfLibs	配列 pLibNames が指し示すライブラリ名ポインタの数

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	不正なパラメータ リスト
-4　PBORCA_BADLIBRARY	ライブラリ名が正しくない、またはリスト上のライブラリが存在しない

解説

コンパイルやオブジェクト のクエリ を行う ORCA 関数を呼び出す前に、PBORCA_SessionSetLibraryList および PBORCA_SessionSetCurrentAppl を呼び出す必要があります。

ライブラリ名は、どこからでも検索できるようにするために絶対パスで指定します。

ライブラリ リストの変更　現行のアプリケーションとライブラリ リストの設定は、セッション中に一度だけ行うことができます。設定後にライブラリ リストまたは現行のアプリケーションの設定を変更する必要がある場合は、セッションを終了してから新しいセッションを開きます。

ORCA によるライブラリ リストの使用法　ORCA セッション中にオブジェクトの再生成またはクエリを実行する場合、参照オブジェクトを見つけるために検索パスが使用されます。PowerBuilder と同じように、ORCA は参照オブジェクトが見つかるまでライブラリ検索パスで指定した順番でライブラリ全体を探します。

ライブラリ リストが不要な関数　以下のライブラリ管理関数およびソース管理関数は、ライブラリ リストを設定せずに呼び出すことができます。

PBORCA_LibraryCommentModify
PBORCA_LibraryCreate
PBORCA_LibraryDelete
PBORCA_LibraryDirectory
PBORCA_LibraryEntryCopy
PBORCA_LibraryEntryDelete
PBORCA_LibraryEntryExport
PBORCA_LibraryEntryInformation
PBORCA_LibraryEntryMove

例

この例は、PockectBuilder のライブラリ ファイル名の配列を構築し、セッションのライブラリ リストを設定します。

```
LPTSTR lpLibraryNames[4];
// ライブラリ名を指定
lpLibraryNames[0] =
    _TEXT("c:¥¥qadb¥¥qadbtest¥¥qadbtest.pkl");
lpLibraryNames[1] =
    _TEXT("c:¥¥qadb¥¥shared_obj¥¥shared_obj.pkl");
lpLibraryNames[2] =
    _TEXT("c:¥¥qadb¥¥chgreqs¥¥chgreqs.pkl");
lpLibraryNames[3] =
    _TEXT("c:¥¥qadb¥¥datatypes¥¥datatypes.pkl");
lpORCA_Info->lReturnCode =
   PBORCA_SessionSetLibraryList(
    lpORCA_Info->hORCASession, lpLibraryNames, 4);
```

24 ページの「例について」で示されるように、例中のセッション情報は ORCA_Info データ構造体に保存されます。

関連項目

PBORCA_SessionSetCurrentAppl

PBORCA_SetDebug

機能

PBORCA_ConfigureSession が発行された後に、ORCA セッションの bDebug プロパティをリセットすることができます。PowerScript コンパイラを起動するメソッドは、bDebug 設定を使用して条件付きコンパイル ロジックを評価します。

構文

```
INT PBORCA_SetDebug ( HPBORCA hORCASession,
        BOOL bDebug );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
bDebug	DEBUG 条件付きコンパイラ ディレクティブの設定

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	hORCASession は無効

解説

ORCA セッションの間、bDebug 値をリセットすることができます。通常の ORCA アプリケーションでは、bDebug は、ORCA セッションを開いた直後に呼び出される PBORCA_ConfigureSession メソッドに設定されます。bDebug を PBORCA_ConfigureSession メソッドに設定すると、通常は後で PBORCA_SetDebug を呼び出す必要がありません。PBORCA_ConfigureSession または PBORCA_SetDebug を呼び出さない場合、bDebug 値はデフォルトで TRUE になります。

標準の PowerBuilder ターゲットでオブジェクトを構築または再生成するときに、PowerScript コンパイラは、bDebug 値を使用して DEBUG 条件付きコンパイラ ディレクティブを有効にするか無効にするかを決定します。Windows フォーム ターゲットでは bDebug 値を使用しません。これは、PBORCA_DeployWinFormProject メソッドが、これらのターゲットのプロジェクト オブジェクトの中の設定を使用して、DEBUG ディレクティブを有効にするか、無効にするかを決定するためです。

次の ORCA メソッドは PowerScript コンパイラを起動します。

- PBORCA_ApplicationRebuild
- PBORCA_CompileEntryImport
- PBORCA_CompileEntryImportList
- PBORCA_CompileEntryRegenerate
- PBORCA_SccGetLatestVersion

- PBORCA_SccRefreshTarget

PBORCA_LibraryEntryCopy および PBORCA_LibraryEntryMove は、PBL 内のオブジェクトを追加または置換しますが、これらは PowerScript コンパイラの起動も、追加するオブジェクトや置き換えるオブジェクトのコンパイル済み PCODE の変更も行いません。これらのメソッドを使用してオブジェクトを対象の PBL にコピーまたは移動する場合、これらのオブジェクトの DEBUG 条件付きコンパイルの設定は、不明とみなされます。

オブジェクトの PCODE コンポーネントが現行の bDebug 設定と一致するかどうか不確かな場合は、PBORCA_CompileEntryRegenerate を呼び出して現行の設定でオブジェクトを再生成できます。

PBORCA_SetDebug は、PBORCA_SessionOpen の後いつでも呼び出すことができます。PBORCA_SetDebug メソッドは、再コンパイルが必要なオブジェクトを記録しません。PBORCA_ApplicationRebuild メソッドは PowerScript コンパイラを呼び出しますが、PBORCA_INCREMENTAL_REBUILD オプションとともにそれを使用する場合、唯一の変更がその DEBUG ディレクティブのステータスであれば、オブジェクトを再構築しません。このため、DEBUG 条件付きコンパイル ロジックを含むターゲットに対しては PBORCA_INCREMENTAL_REBUILD オプションを使用するべきではありません。

同様に、PBORCA_SccRefreshTarget メソッドで PBORCA_INCREMENTAL_REBUILD オプションを使用するべきではありません。元のオブジェクトとリフレッシュしたオブジェクト間での唯一の違いがその DEBUG 条件付きコンパイル ステータスの中にある場合、このオプションが使用されるとオブジェクトはリフレッシュされません。

例

この例は、OrcaScript に set debug コマンドを実装するために OrcaScript インタプリタによって使用されます。

```
Int ParserActions::setDebug(HPBORCA hORCA, Bool bDebug)
{
int orcaResult = PBORCA_OK;
orcaResult = PBORCA_SetDebug( hORCA, bDebug);
if( orcaResult != PBORCA_OK )
    orcaError(PBTEXT("set debug "), orcaResult );
return (orcaResult == PBORCA_OK);
}
```

関連項目

PBORCA_ConfigureSession

PBORCA_SetExeInfo

機能

PBORCA_ExecutableCreate 呼び出しを行う前に、ユーザ指定の値をプロパティ フィールドに設定します。

構文

```
INT PBORCA_SetExeInfo ( HPBORCA hORCASession,
        PBORCA_EXEINFO *pExeInfo );
```

引数	説明
hORCASession	事前に確立した ORCA セッションへのハンドル
**pExeInfo*	実行可能なプロパティを含む構造体へのポインタ

戻り値

INT 型。一般的な戻り値は以下のとおりです。

戻り値	説明
0　PBORCA_OK	処理成功
-1　PBORCA_INVALIDPARMS	無効なパラメータ リスト（pExeInfo または hORCASession が NULL の場合）

解説

PBORCA_ExecutableCreate を呼び出す前にこの関数を呼び出します。

PowerBuilder　では、マシン　コード　コンパイルが要求されると PBORCA_SetExeInfo は動的ライブラリのプロパティも設定します。

PBORCA_EXEINFO 構造体は次のように定義されます。

```
typedef struct pborca_exeinfo
{
LPTSTR    lpszCompanyName;
LPTSTR    lpszProductName;
LPTSTR    lpszDescription;
LPTSTR    lpszCopyright;
LPTSTR    lpszFileVersion;
LPTSTR    lpszFileVersionNum;
LPTSTR    lpszProductVersion;
LPTSTR    lpszProductVersionNum;
} PBORCA_EXEINFO
```

ユーザは PBORCA_SetExeInfo を呼び出す前に、PBORCA_SessionOpen、PBORCA_SessionSetCurrentAppl、および PBORCA_SetLibraryList を発行しておく必要があります。

PBORCA_EXEINFO 構造体の情報は内部の ORCA 制御構造体にコピーされ、PBORCA_SetExeInfo 呼び出しが完了したら、ただちに呼び出し元がこのメモリを解放できるようにします。

実行ファイルのバージョン情報は、PBORCA_SessionClose 処理中に削除されます。このため、ORCA プログラムが多数の ORCA セッションを作成する場合、それぞれ個別のセッションが PBORCA_SetExeInfo を呼び出して、PBORCA_EXEINFO 構造体のすべての要素を再び割り当てる必要があります。

FileVersionNum および ProductVersionNum 文字列は、メジャー バージョン番号、マイナー バージョン番号、修正バージョン番号、およびビルド番号の整数値をコンマで区切って表した 4 つの整数値で構成されます。たとえば、「12,0,0,0001」と指定します。

例

この例は、PowerBuilder アプリケーションの実行ファイルの情報を設定します。

```
memset(&ExeInfo, 0x00, sizeof(PBORCA_EXEINFO));
ExeInfo.lpszCompanyName = _TEXT("Sybase"));
ExeInfo.lpszProductName = _TEXT("PowerBuilder 12.5
   DBAuto"));
ExeInfo.lpszDescription =
   _TEXT("Batch Automation for QADB Test Suite"));
ExeInfo.lpszCopyright = _TEXT("2011"));
ExeInfo.lpszFileVersion = _TEXT("12.5.0.001");
ExeInfo.lpszFileVersionNum = _TEXT("12,5,0,001");
ExeInfo.lpszProductVersion =_TEXT("12.5.0.001");
ExeInfo.lpszProductVersionNum = _TEXT("12,5,0,001");
LpORCA_Info->lReturnCode = PBORCA_SetExeInfo(
lpORCA_Info->hORCASession, &ExeInfo );
   lpORCA_Info->hORCASession, lpLibraryNames, 2);
```

関連項目

PBORCA_DynamicLibraryCreate
PBORCA_ExecutableCreate

第 3 章 ORCA コールバック関数と構造体

この章について

この章では、代表的ないくつかの ORCA 関数で使用されるコールバック関数とそれらの関数に渡される構造体について説明しています。これらのプロトタイプは、PBORCA.H で宣言されています。

内容

項目	ページ
オブジェクトをコンパイルするコールバック関数	120
PBORCA_COMPERR 構造体	121
EAServer にコンポーネントを配布するコールバック関数	123
PBORCA_BLDERR 構造体	124
PBORCA_LibraryDirectory のコールバック関数	125
PBORCA_DIRENTRY 構造体	126
PBORCA_ObjectQueryHierarchy のコールバック関数	127
PBORCA_HIERARCHY 構造体	128
PBORCA_ObjectQueryReference のコールバック関数	129
PBORCA_REFERENCE 構造体	130
PBORCA_ExecutableCreate のコールバック関数	131
PBORCA_LINKERR 構造体	132
PBORCA_SccSetTarget のコールバック関数	133
PBORCA_SCCSETTARGET 構造体	134

オブジェクトをコンパイルするコールバック関数

機能

ライブラリ中のオブジェクトをコンパイルした際に発生するエラーを後で表示するために、エラー発生のたびに呼び出され、エラーを格納します。

このコールバック形式を使用する関数は以下のとおりです。

PBORCA_ApplicationRebuild
PBORCA_CompileEntryImport
PBORCA_CompileEntryImportList
PBORCA_CompileEntryRegenerate

構文

```
typedef void (CALLBACK *PBORCA_ERRPROC)
        ( PPBORCA_COMPERR, LPVOID );
```

引数	説明
PPBORCA_COMPERR	PBORCA_COMPERR 構造体へのポインタ（次で説明）
LPVOID	ユーザ データへの Long 型のポインタ

戻り値

なし。

解説

コールバック関数のコードを記述します。通常、コールバック関数は PBORCA_COMPERR 構造体に渡されたエラー情報を読み取り、必要な情報を抽出し、LPVOID が指し示すユーザ データ バッファに合わせてフォーマットします。

ユーザ データ バッファは呼び出しプログラム中で割り当てられ、必要な形に組み立てることができます。それには、エラーを数える構造体、および、すべてのエラーに関する情報をフォーマットした配列やテキスト ブロックが含まれることもあります。

コールバック関数のコード例については、13 ページの「ORCA コールバック関数について」を参照してください。

PBORCA_COMPERR 構造体

機能

ライブラリ中のオブジェクトをインポートおよびコンパイルしようとしたときに発生したエラーに関する情報をレポートします。

以下の関数は、コールバック関数に PBORCA_COMPERR 構造体を渡します。

PBORCA_CompileEntryImport
PBORCA_CompileEntryImportList
PBORCA_CompileEntryRegenerate

構文

```
typedef struct pborca_comperr {
        int iLevel;
        LPTSTR lpszMessageNumber;
        LPTSTR lpszMessageText;
        UINT iColumnNumber;
        UINT iLineNumber;
} PBORCA_COMPERR, FAR *PPBORCA_COMPERR;
```

メンバー	説明
iLevel	エラーの重大度を示す番号。値は以下のとおりです。 0 オブジェクトやスクリプト名などのコンテキスト情報 1 CM_INFORMATION_LEVEL 2 CM_OBSOLETE_LEVEL 3 CM_WARNING_LEVEL 4 CM_ERROR_LEVEL 5 CM_FATAL_LEVEL 6 CM_DBWARNING_LEVEL
lpszMessageNumber	メッセージ番号を値に持つ文字列へのポインタ
lpszMessageText	エラー メッセージ テキストを値に持つ文字列へのポインタ
iColumnNumber	エラーが発生したソース コードの行の文字番号
iLineNumber	エラーが発生したソース コードの行番号

解説

1 つのエラーが、いくつかのコールバック関数を呼び出すきっかけとなることもあります。最初のメッセージは、エラーが発生したオブジェクトとスクリプトをレポートします。次に、1 つ以上のメッセージで実際のエラーがレポートされます。

たとえば、IF-THEN-ELSE ブロックで END IF がない場合、以下のようなエラーが生成されます。

レベル	番号	メッセージ テキスト	カラム	行
0	null	オブジェクト：f_boolean_to_char	0	0
0	null	関数のソース	0	0

レベル	番号	メッセージ テキスト	カラム	行
4	null	(0002): Error C0031: 構文エラー	0	2
4	null	(0016): Error C0031: 構文エラー	0	16
4	null	(0017): Error C0031: 構文エラー	0	17

EAServer にコンポーネントを配布するコールバック関数

機能

後でそのエラーを表示できるように、エラーを格納し、EAServer へのオブジェクトの配布でエラーが発生すると毎回呼び出されます。

このコールバック形式を使用する関数は以下のとおりです。

PBORCA_BuildProject
PBORCA_BuildProjectEx

構文

```
typedef PSCALLBACK (void, *PPBORCA_BLDPROC)
        ( PBORCA_BLDERR, LPVOID );
```

引数	説明
PPBORCA_BLDERR	PBORCA_BLDERR 構造体へのポインタ（次で説明）
LPVOID	ユーザ データへの Long 型のポインタ

戻り値

なし。

解説

コールバック関数のコード例については、13 ページの「ORCA コールバック関数について」を参照してください。

PBORCA_BLDERR 構造体

機能

EAServer にオブジェクトを配布しようとして発生したエラーに関する情報をレポートします。

PBORCA_BLDERR 構造体をコールバック関数へ渡すのは、以下の関数です。

PBORCA_BuildProject
PBORCA_BuildProjectEx

構文

```
typedef struct pborca_blderr {
        LPTSTR lpszMessageText;
} PBORCA_BLDERR, FAR *PPBORCA_BLDERR;
```

メンバー	説明
lpszMessageText	エラー メッセージ テキストを値に持つ文字列へのポインタ

PBORCA_LibraryDirectory のコールバック関数

機能

エントリに関する情報を後で表示することができるように格納するために、ライブラリ中の各エントリごとに呼び出されます。

構文

```
typedef void (CALLBACK *PBORCA_LISTPROC)
        ( PPBORCA_DIRENTRY, LPVOID );
```

引数	説明
PPBORCA_DIRENTRY	PBORCA_DIRENTRY　構造体へのポインタ（次で説明）
LPVOID	ユーザ データへの Long 型のポインタ

戻り値

なし。

解説

コールバック関数のコードを記述します。通常コールバック関数は PBORCA_DIRENTRY 構造体に渡されたライブラリ エントリ情報を読み、必要な情報を抽出し、LPVOID が指し示すユーザ データ バッファに合わせてフォーマットします。

ユーザ データ バッファは呼び出しプログラム中で割り当てられ、必要な形に組み立てることができます。それには、エントリを数える構造体、および、すべてのエントリに関する情報をフォーマットした配列やテキスト ブロックが含まれることもあります。

コールバック関数のコード例については、13 ページの「ORCA コールバック関数について」を参照してください。

PBORCA_DIRENTRY 構造体

機能

ライブラリ中のエントリに関する情報をレポートします。

PBORCA_LibraryDirectory 関数は、PBORCA_DIRENTRY 構造体をコールバック関数に渡します。

構文

```
typedef struct pborca_direntry {
        TCHAR szComments[PBORCA_MAXCOMMENT + 1];
        LONG lCreateTime;
        LONG lEntrySize;
        LPTSTR lpszEntryName;
        PBORCA_TYPE otEntryType;
} PBORCA_DIRENTRY, FAR *PPBORCA_DIRENTRY;
```

メンバー	説明
szComments	ライブラリ中に格納してあるオブジェクトのコメント
lCreateTime	オブジェクトが作成された時間
lEntrySize	ソース コードとコンパイル済みオブジェクトを含むオブジェクトのサイズ
lpszEntryName	情報が返されるオブジェクトの名前
otEntryType	オブジェクトのデータ型を指定するのカタログデータ型 PBORCA_TYPE の値

PBORCA_ObjectQueryHierarchy のコールバック関数

機能

調査されるオブジェクトの階層中の各先祖オブジェクトに対して呼び出されます。コールバック関数では、後で表示するために先祖名を保存することができます。

構文

```
typedef void (CALLBACK *PBORCA_HIERPROC)
        ( PPBORCA_HIERARCHY, LPVOID );
```

引数	説明
PPBORCA_HIERARCHY	PBORCA_HIERARCHY 構造体へのポインタ（次で説明）
LPVOID	ユーザ データへの Long 型のポインタ

戻り値

なし。

解説

コールバック関数のコードを記述します。一般に、コールバック関数は　PBORCA_HIERARCHY　構造体に渡された先祖名を読み取り、LPVOID が指し示すユーザ データ バッファに保存します。

ユーザ データ バッファは呼び出しプログラム中で割り当てられ、必要な形に組み立てることができます。それには、先祖の数を数える構造体、および名前を格納する配列やテキスト ブロックが含まれることもあります。

コールバック関数のコード例については、13 ページの「ORCA コールバック関数について」を参照してください。

PBORCA_HIERARCHY 構造体

機能

クエリが実行されたオブジェクトの先祖オブジェクト名をレポートします。

PBORCA_ObjectQueryHierarchy 関数は、PBORCA_HIERARCHY 構造体をコールバック関数に渡します。

構文

```
typedef struct pborca_hierarchy {
        LPTSTR lpszAncestorName;
} PBORCA_HIERARCHY, FAR *PPBORCA_HIERARCHY;
```

メンバー	説明
lpszAncestorName	先祖オブジェクトの名前へのポインタ

PBORCA_ObjectQueryReference のコールバック関数

機能

調査されるオブジェクト中の各参照オブジェクトに対して呼び出されます。コールバック関数では、参照オブジェクト名を保存して後で表示することができます。

構文

```
typedef void (CALLBACK *PBORCA_REFPROC)
        ( PPBORCA_REFERENCE, LPVOID );
```

引数	説明
PPBORCA_REFERENCE	PBORCA_REFERENCE 構造体へのポインタ（次で説明）
LPVOID	ユーザ データへの Long 型のポインタ

戻り値

なし。

解説

コールバック関数のコードを記述します。一般に、コールバック関数は PBORCA_REFERENCE 構造体に渡された参照オブジェクト名を読み取り、LPVOID が指し示すユーザ データ バッファに格納します。

ユーザ データ バッファは呼び出しプログラム中で割り当てられ、必要な形に組み立てることができます。それには、参照オブジェクト数を数える構造体、および名前を格納する配列やテキスト ブロックが含まれます。

コールバック関数のコード例については、13 ページの「ORCA コールバック関数について」を参照してください。

PBORCA_REFERENCE 構造体

機能

クエリを実行されるオブジェクトが参照しているオブジェクト名をレポートします。

PBORCA_ObjectQueryReference 関数は、PBORCA_REFERENCE 構造体をそのコールバック関数に渡します。

構文

```
typedef struct pborca_reference {
        LPTSTR lpszLibraryName;
        LPTSTR lpszEntryName;
        PBORCA_TYPE otEntryType;
} PBORCA_REFERENCE, FAR *PPBORCA_REFERENCE;
```

メンバー	説明
lpszLibraryName	参照されるオブジェクトを含むライブラリのファイル名を値に持つ文字列へのポインタ
lpszEntryName	参照されるオブジェクト名を値に持つ文字列へのポインタ
otEntryType	参照されるオブジェクトの型を指定するカタログデータ型 PBORCA_TYPE の値

PBORCA_ExecutableCreate のコールバック関数

機能

実行ファイルを構築する際に発生する各リンク エラーに対して呼び出されます。

構文

```
typedef void (CALLBACK *PBORCA_LNKPROC)
        ( PPBORCA_LINKERR, LPVOID );
```

引数	説明
PPBORCA_LINKERR	PBORCA_LINKERR 構造体へのポインタ（次で説明）
LPVOID	ユーザ データへの Long 型のポインタ

戻り値

なし。

解説

コールバック関数のコードを記述します。一般に、コールバック関数は PBORCA_LINKERR 構造体に渡されたエラー情報を読み取り、LPVOID が指し示すユーザ データ バッファに合わせてメッセージ テキストをフォーマットします。

ユーザ データ バッファは呼び出しプログラム中で割り当てられ、必要な形に組み立てることができます。それには、エラーを数える構造体、およびメッセージ テキストをフォーマットした配列やテキスト ブロックが含まれます。

コールバック関数のコード例については、13 ページの「ORCA コールバック関数について」を参照してください。

PBORCA_LINKERR 構造体

機能

実行ファイルを構築する際に発生するリンク エラーに関するメッセージ テキストをレポートします。

PBORCA_ExecutableCreate 関数は、PBORCA_LINKERR 構造体をコールバック関数に渡します。

構文

```
typedef struct pborca_linkerr {
        LPTSTR lpszMessageText;
} PBORCA_LINKERR, FAR *PPBORCA_LINKERR;
```

メンバー	説明
lpszMessageText	エラー メッセージのテキストへのポインタ

PBORCA_SccSetTarget のコールバック関数

機能

ターゲットのライブラリ リスト中の各ライブラリに対して 1 度呼び出されます。

構文

```
typedef PBCALLBACK (void, *PBORCA_SETTGTPROC)
        ( PPBORCA_SETTARGET, LPVOID );
```

引数	説明
PPBORCA_SETTARGET	PBORCA_SCCSETTARGET 構造体へのポインタ
LPVOID	ユーザ データへの Long 型のポインタ

戻り値

なし。

解説

このコールバック関数を使用するとデフォルト でリフレッシュされるライブラリ がわかります。また、特定の共有ライブラリ が事前のタスクでリフレッシュされていることが確実な場合に PBORCA_SccExcludeLibraryList を呼び出す機会を得ることができます。

PBORCA_SCCSETTARGET 構造体

機能

ターゲットのライブラリ リスト中のライブラリ名を絶対パスの付いたファイル名でレポートします。

構文

```
typedef struct pborca_sccsettarget {
        LPTSTR lpszLibraryName;
} PBORCA_SETTARGET, FAR *PPBORCA_SETTARGET;
```

メンバー	説明
lpszLibraryName	ターゲットのライブラリ リスト中のライブラリ名へのポインタ