

SYBASE®

ORCA ガイド

PowerBuilder®

11.1

LAST REVISED: November 2007

Copyright © 1991-2008 by Sybase, Inc. All rights reserved.

本書は Sybase ソフトウェアの付属マニュアルであり、新しいマニュアルまたはテクニカル ノートで特に示されないかぎり、後続のリリースにも付属します。このマニュアルの内容は、予告なく変更されることがありますが、Sybase, Inc. およびその関連会社では内容の変更に関して一切の責任を負いません。このマニュアルに記載されているソフトウェアはライセンス契約に基づいて提供されるものであり、無断で使用することはできません。

予定したソフトウェアのリリース日にのみアップグレードを提供します。本書に記載されている内容は、Sybase, Inc. およびその関連会社の書面による事前許可を得ずに、電子的、機械的、手作業、光学的、またはその他のいかなる手段によっても複製、転載、翻訳することを禁じます。

Sybase の商標は [Sybase の商標ページ のサイト http://www.sybase.com/detail?id=1011207](http://www.sybase.com/detail?id=1011207) に記載されています。記載の Sybase およびマークは Sybase, Inc. の商標です。® はアメリカ合衆国における登録商標を示します。

Java およびすべての Java ベースのマークは、米国および他国における Sun Microsystems, Inc. の商標または登録商標です。

Unicode および Unicode のロゴは Unicode, Inc. の登録商標です。

本書で記載されている上記以外の社名および製品名は、各社の商標または登録商標の場合があります。

本書に記載されている内容は、将来予告なしに変更することがあります。また、本ソフトウェアおよび説明書を使用したことによる損害、または第三者からのいかなる請求についても、サイベース株式会社、その親会社である米国法人 Sybase, Inc. またはその関連会社は、一切の責任を負わないものとします。

目次

第 1 章	ORCA を使用する方法.....	11
	ORCA とは ?.....	11
	ORCA を使用してできること	12
	ORCA 呼び出しプログラムの開発者	13
	ORCA のインストール.....	14
	ORCA とライブラリ ペインタ	14
	PowerBuilder ライブラリ中のオブジェクト	15
	オブジェクトのソース コード	15
	PowerBuilder コマンドと ORCA 関数	16
	ORCA 関数について	17
	ORCA セッションの管理に関する関数.....	17
	PowerBuilder ライブラリの管理に関する関数.....	19
	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	20
	PowerBuilder オブジェクトのクエリ関数.....	20
	マシン コードと動的ライブラリの作成に関する関数.....	21
	EAServer ヘコンポーネントを配布する関数	22
	ソース管理の操作を管理する関数	22
	ORCA コールバック関数について.....	23
	コールバックを使用する ORCA 関数	23
	コールバックの働き	24
	コールバック関数の内容.....	25
	ORCA プログラムを書く	28
	ORCA プログラムの概要	28
	新しいアプリケーションのブートストラップ	31
	廃止された ORCA 関数の削除	32
第 2 章	ORCA 関数.....	35
	例について	35
	ORCA リターン コード	35
	PBORCA_ApplicationRebuild	37
	PBORCA_BuildProject	40
	PBORCA_BuildProjectEx.....	42
	PBORCA_CompileEntryImport	43

PBORCA_CompileEntryImportList.....	51
PBORCA_CompileEntryRegenerate.....	57
PBORCA_ConfigureSession.....	60
PBORCA_DeployWinFormProject.....	64
PBORCA_DynamicLibraryCreate.....	66
PBORCA_ExecutableCreate.....	68
PBORCA_LibraryCommentModify.....	74
PBORCA_LibraryCreate.....	75
PBORCA_LibraryDelete.....	77
PBORCA_LibraryDirectory.....	78
PBORCA_LibraryEntryCopy.....	82
PBORCA_LibraryEntryDelete.....	84
PBORCA_LibraryEntryExport.....	86
PBORCA_LibraryEntryExportEx.....	92
PBORCA_LibraryEntryInformation.....	94
PBORCA_LibraryEntryMove.....	98
PBORCA_ObjectQueryHierarchy.....	100
PBORCA_ObjectQueryReference.....	102
PBORCA_SccClose.....	104
PBORCA_SccConnect.....	105
PBORCA_SccConnectOffline.....	107
PBORCA_SccExcludeLibraryList.....	110
PBORCA_SccGetConnectProperties.....	112
PBORCA_SccGetLatestVersion.....	114
PBORCA_SccRefreshTarget.....	115
PBORCA_SccResetRevisionNumber.....	116
PBORCA_SccSetTarget.....	118
PBORCA_SessionClose.....	121
PBORCA_SessionGetError.....	122
PBORCA_SessionOpen.....	123
PBORCA_SessionSetCurrentAppl.....	124
PBORCA_SessionSetLibraryList.....	126
PBORCA_SetExeInfo.....	128

第3章

ORCA コールバック関数と構造体 131

オブジェクトをコンパイルするコールバック関数.....	132
PBORCA_COMPERR 構造体.....	133
EAServer へコンポーネントを配布するコールバック関数.....	135
PBORCA_BLDERR 構造体.....	136
PBORCA_LibraryDirectory のコールバック関数.....	137
PBORCA_DIRENTRY 構造体.....	138
PBORCA_ObjectQueryHierarchy のコールバック関数.....	139
PBORCA_HIERARCHY 構造体.....	140
PBORCA_ObjectQueryReference のコールバック関数.....	141

PBORCA_REFERENCE 構造体	142
PBORCA_ExecutableCreate のコールバック関数	143
PBORCA_LINKERR 構造体	144
PBORCA_SccSetTarget のコールバック関数	145
PBORCA_SCCSETTARGET 構造体	146

本書について

目的

このマニュアルでは、PowerBuilder Enterprise 版の一機能である Powersoft OpenLibrary API (ORCA) について説明します。

ORCA プログラムを開発する際に必要となる以下の情報を提供しています。

- 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 ペインタの中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

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 を使用して PowerBuilder のオブジェクトを操作したり管理するツールや関連製品を開発するツール ベンダや CODE パートナを対象としています。

制約

ORCA ユーザは、ORCA 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、[13 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。

本書は、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 アプリケーション構築で使用するための設計はされていません。また、本書は ORCA を使用したプログラムを実行する方法について記述してありません。そのようなプログラム用のドキュメントが提供されているはずです。

サポートについて

日本では、ORCA はサポート対象外です。

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 ペインタで行えるタスクと PowerBuilder ライブラリへ ORCA でプログラム的にやりたいタスク間のやり取り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？	11
ORCA のインストール	14
ORCA とライブラリ ペインタ	14
ORCA 関数について	17
ORCA コールバック 関数について	23
ORCA プログラムを書く	28
廃止された ORCA 関数の削除	32

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 関数へ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 で作成されたプログラム）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やライブラリ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た PowerBuilder オブジェクトを作成したり変更したりするために PowerBuilder のライブラリへプログラミ的な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、PBL にオブジェクト ソースを置くための ORCA 関数を使用する
- 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、オブジェクトをライブラリへ戻すために ORCA を再度使用する

サンプル ORCA アプリケーション

ORCA は、下記の PowerBuilder と一緒に使用する様々なツールで使われています。

- 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
- ユーティリティ (テキストを検索してライブラリ全体への置換や、ライブラリ中のオブジェクトをツリー ビューで表示など)
- PowerBuilder が直接サポートしていないソース コントロール 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、アプリケーションからプログラミ的行わせることができます。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 ペインタとライブラリ ペインタの幾つかの機能をカバーしています。

できることは以下のとおりです。

- 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付きのマシンコード (DLL)、または PowerBuilder 動的ライブラリの作成
- 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
- 新しいライブラリにアプリケーション全体を作成 (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い、EAServer コンポーネントの構築と配布を行う
- ソース コントロールから PowerBuilder のターゲットを開き、ターゲット オブジェクト上の様々なソース コントロール操作を実行する

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、以下の制約を知る必要があります。

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には、本セクションに記述する制約を理解して守る必要があります。

ORCA 使用の制約について PowerBuilder と ORCA は両方とも、PowerBuilder のコンパイラを使用します。しかし、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、ORCA を呼び出すプログラムの場合は PowerBuilder を実行することはできません。

ORCA を使用したツール プロバイダは、PowerBuilder 開発者が以下のような ORCA をベースにしたモジュールを呼ぶときのことを考慮して、注意深くプログラムを作成しなければいけません。

- 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場合は、PowerBuilder ライブラリが破損する恐れがあります。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 Windows と UNIX 環境下で、PowerBuilder ライブラリ中のオブジェクトを操作して管理する関連製品 (PowerBuilder と共に使用) を開発するツール ベンダと CODE パートナとお客様の方々が利用できます。

ORCA プログラムを実行するためには

ORCA を使用したプログラムを実行するためには、ORCA DLL (PowerBuilder バージョン 11 では、PBORC110.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ためには

ORCA を使用した C プログラムを開発するためには、幾つかのアイテムを必要とします。詳細は、株式会社アシスト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 ペインタで定義され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 ペインタは、PowerBuilder 開発者に PBL の内容を表示して保守を可能にします。このペインタは、PBL 中のオブジェクトを日付やコメントの変更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、エクスポートとインポートを行うことができます。またソース コントロール システムを使用したり、PowerBuilder 動的ライブラリと DLL を作成することができます。

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、視覚的にオブジェクトを見たり修正したり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、オブジェクトを視覚的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 中の視覚的な形式で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

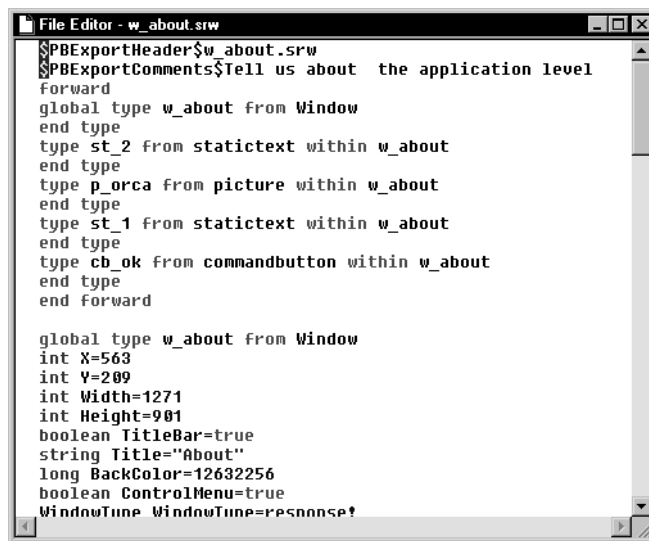
ライブラリ ペインタを使用して、ソース コードをエクスポートし、その内容を任意のテキスト エディタで調べて修正し、ライブラリに戻すためにインポートします。PowerBuilder は、ソース コードの有効性をチェックするためにインポートしたオブジェクトをコンパイルします。有効でない場合には、インポートしたオブジェクトのコンパイルは失敗します。

エクスポートされるソース コードには、ソース コードの前に 2 行のヘッダがついています。

```
$PBExportHeader$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、関数へ引き渡される lpszEntryName と lpszComments 引数を使用します。

PowerBuilder のテキスト 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。



```
File Editor - w_about.srw
$PBExportHeader$w_about.srw
$PBExportComments$Tell us about the application level
forward
global type w_about from Window
end type
type st_2 from statictext within w_about
end type
type p_orca from picture within w_about
end type
type st_1 from statictext within w_about
end type
type cb_ok from commandbutton within w_about
end type
end forward

global type w_about from Window
int X=563
int Y=209
int Width=1271
int Height=901
boolean TitleBar=true
string Title="About"
long BackColor=12632256
boolean ControlMenu=true
WindowTune WindowTune=response
```

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ドキュメントはありません。ソース コードが何に所属しているかを知る方法は、オブジェクトをエクスポートしてそのソースを調べるだけです。

ORCA とソース コード

ORCA には、エクスポートする関数があり、これで既存のオブジェクトを調査したり変更したりすることができます。PowerBuilder 10 以降では、開発者はメモリ バッファ、またはファイルへ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。

次のセクションでは、ORCA 関数とそれぞれの目的、及びそれらが PowerBuilder 開発環境内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、関数の呼び出し仕様を指定する WINAPI マクロを使用している外部 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。

本書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。本書中のコード例は、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場合、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
- マシン コード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 を起動して PowerBuilder セッションを開始し、PowerBuilder を終了して PowerBuilder セッションを終了するのと同じように、ORCA を使用するときセッションを開き、終了するとき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、まず初めに現行のアプリケーションがなければいけません。また、オブジェクトの参照や変更、実行モジュー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ています。ORCA の要件は同じですが、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。ソース管理の関数は、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_）と、それぞれの目的と、PowerBuilder 開発環境でそれらに相当するものを、ここにリストします。

関数 (接頭辞 PBORCA_)	目的	相当する PowerBuilder
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	システム ツリー内でターゲット ライブラリのプロパティを指定する
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	システム ツリー内でアプリケーションを指定する
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 ペインタでのコマンドによく似ています。これらのライブラリを管理する関数を使用すると、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、ライブラリ 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を、ここにリストします。

関数 (接頭辞 PBORCA_)	目的	相当する PowerBuilder
LibraryCommentModify	ライブラリに対するコメントの変更	[エントリ プロパティ]
LibraryCreate	新しいライブラリ ファイルの作成	[エントリ ライブラリ 作成]
LibraryDelete	ライブラリ ファイルの削除	[エントリ 削除]
LibraryDirectory	ライブラリのコメントと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 ライブラリ項目 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 ライブラリ項目 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 ライブラリ項目 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 ライブラリ項目 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、ライブラリ中に既に存在するエントリをコンパイルすることができます。

ソースコードとコンパイル済みの両方をライブラリのエントリにすることができます。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を、ここにリストします。

関数 (接頭辞 PBORCA_)	目的	相当するライブラリ ペインタ
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 ライブラリ項目 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再構築]

コンパイルする関数は、ライブラリから実行可能なモジュールを作成する関数ではありません。次の「[マシンコード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_）を、ここにリストします。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

マシン コード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、マシン コードと動的ライブラリ（PBD と DLL）を作成することができます。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_）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を、ここにリストします。

関数 （接頭辞 PBORCA_）	目的	相当するペインタ
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 ペインタでマシンコード生成を指定
SetExeInfo	生成される DLL と EXE に関連する追加的なファイルのプロパティを設定	プロジェクト ペインタ

EAServer へコンポーネントを配布する関数

これらの関数を使用して、EAServer コンポーネントの使用や、上書きや、プロジェクト オブジェクトの指定を配布することができます。

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
SetExcludeLibraryList	ターゲット ライブラリの一覧の中で、次に続く PBORCA_SccRefreshTarget 操作で同期を取りたく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、オブジェクトの最新バージョンをコピーする
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 プロパティを設定
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

幾つかの 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に、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

呼び出す回数が不明な処理を行う必要がある場合に、ORCA は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。

オプションが必須か

コールバック関数のうちの幾つかは、オブジェクトのコンパイルや実行ファイルの構築といったような、メインの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 リスト中の各項目といったような、関数呼び出し時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 や C++ といったようなポインタを使用する言語で書いてあるプログラムでのみ使用することが可能です。

ORCA コールバック関数を作成する場合、関数を呼び出す仕様を指定する 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 関数が呼び出す (接頭辞 <code>PBORCA_</code>)	コールバックの目的
<code>BuildProjectEx</code> <code>BuildProject</code>	各配布エラーにつき一度呼び出される
<code>CompileEntryImport</code> <code>CompileEntryImportList</code> <code>CompileEntryRegenerate</code>	各コンパイル エラーにつき一度呼び出される
<code>ExecutableCreate</code>	各リンク エラーにつき一度呼び出される

ORCA 関数が呼び出す (接頭辞 PBORCA_)	コールバックの目的
LibraryDirectory	各ライブラリ エントリ名につき一度呼び出される
ObjectQueryHierarchy	各先祖名につき一度呼び出される
ObjectQueryReference	エントリ中の参照される各オブジェクトにつき一度呼び出される
SccSetTarget	ライブラリ リスト中の各ライブラリにつき一度呼び出される

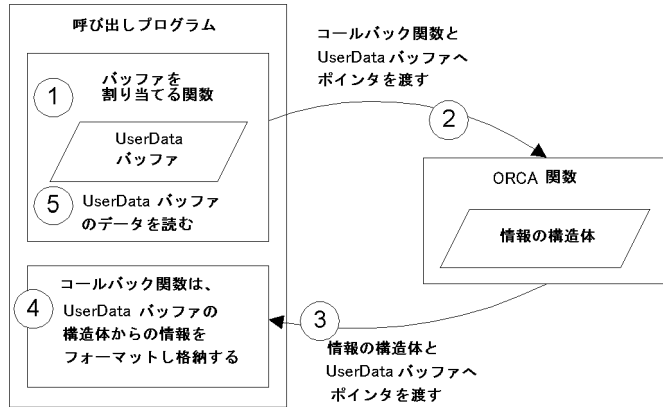
コールバックの働き

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を割り当てます。(UserData バッファ)
- 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 バッファへ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、UserData バッファ中の構造体から情報を読み出し格納します。

手順 3 と 4 は、ORCA がレポートをするために必要となるそれぞれの情報ごとに繰り返します。1 つの ORCA 関数がコールバックを呼び出すのは、一度だけかもしれませんが、数回、または一度も呼び出さないかもしれません。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。

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



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ている処理は、完全に開発者の責任です。本節では、そのハンドリングの方法について簡単に図解しています。

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、満杯になったメッセージ バッファの内容を管理します。

```
typedef struct ORCA_UserDataInfo {
    LPBYTE lpszBuffer;    // データ ストアのためのバッファ
    DWORD dwCallCount;    // バッファ中のメッセージ数
    DWORD dwBufferSize;   // バッファ サイズ
    DWORD dwBufferOffset; // 現在のバッファ中のオフセット
} 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、メッセージに必要な容量が分からないため、60000 バイト（任意のサイズ）を割り当てます。リンク エラーの収集時には足りるかもしれませんが、大きなライブラリのディレクトリ情報を必要とする場合には足りないかもしれません。

```
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 UserData 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、そして 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。

シンプルなコールバック シンプルなコールバックとは、下記の事柄を行うものです。

- 取り出された項目数を保持する
- メッセージを保持し、オーバーフローする場合には再割り当てを行う

下記のコード例では、PBORCA_ExecutableCreate に対して LinkErrors と呼ばれるコールバックを実行しています。


```
void CALLBACK LinkErrors(PPBORCA_LINKERR lpLinkError,
    LPVOID lpUserData)
{
    PORCA_USERDATAINFO lpData;
    LPBYTE lpCurrByte;
    LPTSTR lpCurrentPtr;
    int iNeededSize;
    lpData = (PORCA_USERDATAINFO) lpUserData;

    // リンク エラーのトラック数を保持する
    lpData->dwCallCount++;

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

    // 新しいメッセージの長さはどれくらいあるか?
    // メッセージの長さに改行と新規行を加える
    iNeededSize =
        (_tcslen(lpLinkError->lpszMessageText) + 2)*
        sizeof(TCHAR);

    // 必要があればバッファを再割り当てする
    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

本節では、セッションを開き起動する 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、何もないところからアプリケーションを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。

最初のステップ：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 内では呼び出しプログラムは使用できず構造体へマップされているためです。

サンプルコード

この C の記述例は、ORCA セッションを開いています。

```
HPBORCA WINAPI SessionOpen()
{
```

```

HPBORCA hORCASession;
hORCASession = PBORCA_SessionOpen();
return hORCASession;
}

```

オプションのステップ：ライブラリ リストと現行アプリケーションの設定

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

- プログラムがライブラリを管理する、ライブラリ中のエントリを移動する、エントリ中のソースを調べるだけの場合は、ほかを呼び出す必要はありません。ORCA セッションを継続することができます。
- プログラムがほかの ORCA 関数を呼ぶ場合は、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、PowerBulder 開発環境の必要要件とよく似ています。ライブラリ ペインタでは、たとえ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とも、それらのある PBL からほかへコピーすることができます。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。なお、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。

セッション中 1 回設定 ORCA セッション中一度だけライブラリ 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ためには、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

下記の 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;
    LPCTSTR lpLibraryNames[2] =
        {_TEXT("c:\\¥¥pbfiles\\¥¥demo\\¥¥master.pbl"),
         _TEXT("c:\\¥¥pbfiles\\¥¥demo\\¥¥work.pbl")};
}

```

```
// ORCA 関数呼び出し
nReturnCode = PBORCA_SessionSetLibraryList(
    hORCASession, lpLibraryNames, 2);
if (nReturnCode != 0)
    return nReturnCode; // 失敗時の戻り

// アプリケーション名を含む文字列のセットアップ
_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;
}
```

次のステップ : ORCA セッションを継続する

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 関数を呼ぶことができます。関数呼び出しの多くはとても簡単です。幾つかは、コールバックが必要なように少々複雑です。

コールバック関数の詳細に関しては、23 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。

最後のステップ : セッションを閉じる

ORCA プログラムの最後のステップでは、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するすべてのリソースを解放し後始末をします。

下記のサンプルの C 関数は、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 10.0 で開始するときに、オブジェクトのソース コードから全アプリケーション用のライブラリを作成するために ORCA を使用することができます。既存の PBL を使って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ためには、通常は、既存のアプリケーションにある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 オブジェクトとして NULL 値を設定した場合、ORCA はアプリケーション オブジェクトをインポートするために臨時のアプリケーション オブジェクトを使用します。しかし、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 オブジェクトに現行アプリケーションはありません。

❖ 新しいアプリケーションをブートストラップする

- 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 と共に、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へインポートします。

ほかのオブジェクトはインポートしません。

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、良い考えではありません。ブートストラップのセッション中、標準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になります。アプリケーション オブジェクトが何かに依存するような場合（例えば、グローバル変数の参照）は、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が新しいアプリケーションを開始します。処理を完了するためには、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。

セッション中一度だけライブラリ 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。この時点で使用したいアプリケーション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方法

- 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へ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とき

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。しかし、正確なアプリケーション オブジェクトが最新の場合、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 アクセス方法を導入しました。ソース管理と共に動作する ORCA 関数は廃止されましたが、ORCA 8 API では削除はしていませんでした。

PowerBuilder 9 では、新しい ORCA ソース管理関数を追加し、古い ORCA ソース管理関数は ORCA API から削除しました。このため、既存の ORCA アプリケーションから以下の関数を使っている呼び出しをすべて取り除く必要があります。

- 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

本章について

本章では、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	35
ORCA リターンコード	35
ORCA 関数 (アルファベット順)	37

例について

本章内の例では、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。例中では、変数 `lpORCA Info` は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 バッファ長
    long lReturnCode; // リターン コード
    HINSTANCE hLibrary; // ORCA ライブラリのハンドル
    PPBORCA_CONFIG_SESSION pConfig; // セッション構成
} ORCA_INFO, FAR *PORCA_INFO;
```

ORCA リターン コード

ヘッダ ファイルである PBORCA.H に、以下のリターン コードを定義しています。

リターンコード	説明
0 PBORCA_OK	処理成功

リターン コード	説明
-1 PBORCA_INVALIDPARMS	不正なパラメータ リスト
-2 PBORCA_DUOPERATION	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	不正なライブラリ名
-5 PBORCA_LIBLISTNOTSET	ライブラリ リストが未設定
-6 PBORCA_LIBNOTINLIST	ライブラリがライブラリ リスト中にある
-7 PBORCA_LIBIOERROR	ライブラリの 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 サイズが小さ過ぎる
-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_PBD COUNTERERROR	不正な PBD 数
-17 PBORCA_PBD CREATERERROR	PBD 作成エラー
-18 PBORCA_CHECKOUTERROR	ソース管理エラー (廃止)
-19 PBORCA_CB CREATEERROR	ComponentBuilder クラスのインスタンス化は不可
-20 PBORCA_CB INITERROR	コンポーネント ビルダの Init メソッド失敗
-21 PBORCA_CB BUILDERROR	コンポーネント ビルダの BuildProject メソッド失敗
-22 PBORCA_SCCFAILURE	ソース管理への接続不可
-23 PBORCA_REGREADERERROR	レジストリの読み出し不可
-24 PBORCA_SCCLOADDLLFAILED	DLL のロード不可
-25 PBORCA_SCCINITFAILED	SCC 接続の初期化不可
-26 PBORCA_OPENPROJFAILED	SCC プロジェクト開けず
-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 ファイルの読み出し不可
-29 PBORCA_GETINTERFACEERROR	SCC インタフェースへのアクセス不可
-30 PBORCA_IMPORTONLY_REQ	SCC 接続オフラインは 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の中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
<i>hORCASession</i>	事前に確立した ORCA セッションのハンドル
<i>eRebldType</i>	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。値の種類は以下のとおりです。 PBORCA_FULL_REBUILD PBORCA_INCREMENTAL_REBUILD PBORCA_MIGRATE
<i>pCompErrorProc</i>	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は、PBORCA_COMPERR 型の構造体に格納されているエラー レベル、メッセージ番号、メッセージ テキスト、行番号、カラム番号です。オブジェクト名とスクリプト名はメッセージ テキストの一部です。 コールバック関数を使用したくない場合には、 <i>pCompErrorProc</i> に 0 を設定します。
<i>pUserData</i>	コールバック関数の PBORCA_CompileEntryImport へ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、コールバック関数がバッファ サイズの情報とエラー情報を格納するバッファへのポインタ、またはバッファの中に含まれます。 呼び出し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	現行アプリケーションの未設定

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お互いを参照している場合、単純なコンパイルでは失敗します。オブジェクトの相互関係によるエラーを解決するためには、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する
- **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する
- **マイグレート** 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。旧バージョンでのオブジェクトの構築が適切な場合だけ対象となります。

例

下記は、現行ライブラリ リスト上にあるライブラリ リスト中のすべてのオブジェクトを再コンパイルする例です。

エラーが発生するたびに、PBORCA_ApplicationRebuild はコールバック関数の CompileEntryErrors を呼び出します。CompileEntryErrors には、lpUserData を指し示すバッファにエラーメッセージを格納するためのコードを記述します。

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

コールバックのためのデータ バッファの設定に関する更なる情報は、[25 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。

これらのサンプル中では、[35 ページの「例について」](#)で紹介しているように、セッション情報は ORCA_Info 構造体に格納されます。

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, LPTSTR lpszProjectName, PBORCA_BLDPROC pBuildErrProc, LPVOID pUserData);`

引数	説明
<i>hORCASession</i>	事前に確立されている ORCA セッションのハンドル
<i>lpszLibraryName</i>	プロジェクト エントリを含むライブラリのファイル名
<i>lpszProjectName</i>	配布情報を含むプロジェクト オブジェクト
<i>pBuildErrProc</i>	エラーのコールバック関数である PBORCA_BuildProject へのポインタ コールバック関数を使用したくない場合には、 <i>pBuildErrProc</i> に NULL を設定します。
<i>pUserData</i>	コールバック関数へ渡されるユーザ 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	エラーにより配布失敗

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である 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下記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。指定された引数値で、サーバとポートのプロパティを上書きします。

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

引数	説明
<i>hORCASession</i>	事前に確立している ORCA セッションへのハンドル
<i>lpszLibraryName</i>	プロジェクト エントリを含むファイル名
<i>lpszProjectName</i>	配布情報を含むプロジェクト オブジェクト
<i>pBuildErrProc</i>	エラーのコールバック関数である PBORCA_BuildProject へのポインタ コールバック関数を使用したくない場合には、 <i>pBuildErrProc</i> に NULL を設定します。
<i>lpszServerName</i>	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。
<i>iPort</i>	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。
<i>pUserData</i>	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 ビルタークラス 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_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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszLibraryName</i>	オブジェクトにインポートしたいライブラリのファイル名である文字列値へのポインタ
<i>lpszEntryName</i>	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
<i>otEntryType</i>	PBORCA_TYPE カタログ データ型の値は、インポートしたエントリのオブジェクトの種類を指定します。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
<i>lpszComments</i>	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
<i>lpszEntrySyntax</i>	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。 <i>lpszEntrySyntax</i> 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。
<i>lEntrySyntaxBuffSize</i>	<i>lpszEntrySyntax</i> 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
<i>pCompErrorProc</i>	<p>コールバック関数である 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都度呼び出されるコールバック関数です。</p> <p>PBORCA_COMPERR 型構造体に格納し、ORCA がコールバック関数に渡す情報は、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、カラム番号です。オブジェクト名とスクリプト名は、メッセージ テキストの一部にあります。</p> <p>コールバック関数を使用したくない場合には、<i>pCompErrorProc</i> に 0 を設定します。</p>
<i>pUserData</i>	<p>コールバック関数である PBORCA_CompileEntryImport へ渡されるユーザ データへのポインタ</p> <p>通常ユーザ データには、バッファまたは、エラー情報とバッファ サイズに関する情報を格納するコールバック関数中のバッファへのポインタを含んでいます。</p> <p>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。</p>

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中に保存できない
-6 PBORCA_LIBNOTINLIST	ライブリがリスト中に存在しない
-8 PBORCA_COMPERROR	コンパイル エラー
-9 PBORCA_INVALIDNAME	名前が PowerBuilder の命名規則に従っていない
-13 PBORCA_CURRAPPLNOTSET	現行アプリケーションの未設定

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。

PowerBuilder

PowerBuilder 10 以降では、インポートされ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

PBORCA_CompileEntryImport の 2 つのそれぞれの呼び出しは、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。二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。

エラー発生時 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へ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、編集するためにペインタで開くことができます。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のコンパイルする順番による場合には、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、インポートの前に古いエントリが削除されます。インポートが失敗した場合、古いファイルは既に削除されています。

エラーを処理するコールバック関数についての説明は、PBORCA_CompileEntryImportList を参照してください。

例

下記の例では、d_labels という名前のデータウィンドウをライブラリ名 DWOBJECTS.PBL へインポートしています。ソース コードは、szEntrySource という名前のバッファへ格納しています。

エラーが発生するたびに、PBORCA_CompileEntryImport は CompileEntryErrors というコールバック関数を呼びます。CompileEntryErrors 用に記述することは、lpUserData が指し示すバッファにエラーメッセージを格納することです。

```
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);
```

この例については、35 ページの「例について」のデータ構造体 ORCA_Info に保存されるセッション情報を参照してください。

この例ではソース 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断し、PBL ヘインポートします。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 の二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 ヘインポートします。
// 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。
```

```

#ifdef (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:\\pb11.0\\main\\pbls\\qadb\\qadbtest\\qadbtest.pbl");
    pszLibraryName[1] =
        _TEXT("c:\\pb11.0\\main\\pbls\\qadb\\shared_obj\\shared_obj.pbl");
    pszLibraryName[2] =
        _TEXT("c:\\pb11.0\\main\\pbls\\qadb\\datatypes\\datatype.pbl");
    pszLibraryName[3] =
        _TEXT("c:\\pb11.0\\main\\pbls\\qadb\\chgreqs\\chgreqs.pbl");
    pszLibraryName[4] =
        _TEXT("c:\\pb11.0\\main\\orca\\testexport\\work.pbl");
    pszImportFile =
        _TEXT("c:\\pb11.0\\main\\pbls\\qadb\\qadbtest\\w_edit_connect.srw");
    ;
    memset(&Config, 0x00, sizeof(PBORCA_CONFIG_SESSION));

    PbOrca = PBORCA_SessionOpen();
    // work.pbl を削除して再生成します。
    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 つの呼び出しを行います。
if (Config.eImportEncoding == PBORCA_UNICODE)
{
    LPWSTR pszUniBinHeader;
    LPWSTR pUniBinStart;
    pszUniBinHeader = "PowerBuilder バイナリ データ セク
セッション 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
セッション 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 エンコーディングで呼び出すために ORCA
// セッションの環境を設定します。
iErrCode = PBORCA_ConfigureSession(hPbOrca, &Config);

```

```
// エントリに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("test embedded OLE object"),
    (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。この例は、ORCA クラスの
// メソッドではなく、プログラムで提供しています。
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へインポートしてコンパイルします。インポートされるそれぞれのオブジェクト名は、配列中に保持されます。配列中には、そのほかに、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ています。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
<i>hORCASession</i>	事前に確立している ORCA セッションへのハンドル
<i>*pLibraryNames</i>	関連するオブジェクトをインポートしたいライブラリ ファイル名を持つ string 型配列へのポインタ
<i>*pEntryNames</i>	関連するライブラリにインポートしたいオブジェクト名を持つ string 型配列へのポインタ
<i>*otEntryTypes</i>	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 タイプ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
<i>*pComments</i>	関連するオブジェクトのコメントを持つ string 型配列へのポインタ
<i>*pEntrySyntaxBuffers</i>	関連するオブジェクトのソース コードを持つ string 型配列へのポインタ

引数	説明
<i>*pEntrySyntaxBuffSizes</i>	<i>*pEntrySyntaxBuffers</i> が指し示す文字列長の値を持つ long 型配列へのポインタ
<i>iNumberOfEntries</i>	インポートされたエントリの数であり、すべての配列引数の配列長と同じである
<i>pCompErrorProc</i>	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 が渡す情報は、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、カラム番号であり、PBORCA_COMPERR 型の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 テキストに含まれています。 コールバック関数を使用したくない場合は、 <i>pCompErrorProc</i> に 0 を設定します。
<i>pUserData</i>	コールバック関数の PBORCA_CompileEntryImportList へ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たくない場合は、 <i>pUserData</i> 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	リスト中がないライブラリ
-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 命名規則に従っていない
-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、相互に関連する幾つかのオブジェクト（例えば、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 タイプの定義をコンパイルします。エラーがない場合は、ORCA は全ライブラリ リスト中の全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リストには、先祖オブジェクトを最初にインポートするために、子孫オブジェクトの前に先祖オブジェクトを記述します。

オブジェクトのリストには、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ユーザ オブジェクトを記述します。

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を格納 インポートしたオブジェクトの各々の情報は、幾つかの並列した配列に含まれています。例えば、`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 番目のエレメントにある場合、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 タイプはオブジェクト タイプ配列の 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` を使用してエントリ タイプを指定する OLE オブジェクトのような埋め込みバイナリ情報を含むインポートやエクスポート時に、この `PBORCA_TYPE` のカタログデータ型を使用します。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の記述例は、45 ページの「例」を参照してください。

エラー発生時 オブジェクトのインポート処理中にエラーが発生すると、オブジェクトはライブラリへ入りますが編集が必要かもしれません。オブジェクトのエラーが些細な場合は、ペインタを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のコンパイルする順番による場合には、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、インポートの前に古いエントリが削除されます。インポートが失敗した場合、古いファイルは既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 コンパイル処理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。

どれくらい多くのエラーが発生するか分からないため、*pUserData* バッファ サイズを見積もることは困難です。コールバック関数は、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

以下の例では、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 に対して記述する中で、lpUserData が指し示すバッファ中にエラーメッセージを格納します。例では、lpUserData バッファは既に設定済みです。

```
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¥¥pb11.0¥¥demo¥¥windows.pbl");
lpLibraryNames[1] =
    _TEXT("c:¥¥sybase¥¥pb11.0¥¥demo¥¥windows.pbl");
lpLibraryNames[2] =
    _TEXT("c:¥¥sybase¥¥pb11.0¥¥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 タイプ配列を設定
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] = (LPSTR) szWindow1;
lpSourceBuffers[1] = (LPSTR) szWindow2;
lpSourceBuffers[2] = (LPSTR) 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 );
```

コールバックのためのデータ バッファの設定に関する情報については、25 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。

これらのサンプル中では、35 ページの「例について」で紹介しているように、セッション情報は ORCA_Info 構造体に格納されます。

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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszLibraryName</i>	コンパイルされるオブジェクトを含むライブラリファイル名の文字列値を指し示すポインタ
<i>lpszEntryName</i>	コンパイルされるオブジェクト名の文字列値を指し示すポインタ
<i>otEntryType</i>	コンパイルされるオブジェクトのオブジェクト タイプを PBORCA_TYPE のカタログデータ型で指定します。この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
<i>pCompErrorProc</i>	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、 <i>pCompErrorProc</i> に 0 を設定します。

引数	説明
<i>pUserData</i>	PBORCA_CompileEntryRegenerate コールバック関数へ渡されるユーザ データへのポインタ 一般的にユーザ 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。

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 リストの未設定
-6 PBORCA_LIBNOTINLIST	ライブラリがライブラリ リスト中に存在しない
-7 PBORCA_LIBIOERROR	ライブラリ I/O エラー
-11 PBORCA_COMPEXPORTERROR	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。

エラー発生時 再生成する間に発生したエラーを修正するためには、ソース コードをエクスポートし、エラーを修正し、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。

時には、PowerBuilder ペインタ内で些細なエラーのあるオブジェクトを開いて修正できることもありますが、大きなエラーの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。

エラー用のコールバック処理の情報については、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ています。

エラーが発生する度に、PBORCA_CompileEntryRegenerate はコールバック関数の CompileEntryErrors を呼び出します。CompileEntryErrors 用の記述の中で、lpUserData が指し示すバッファ中にエラーを格納します。例中では、lpUserData バッファは既に設定されています。


```
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 );
```

これらのサンプル中では、35 ページの「例について」で紹介しているように、セッション情報は ORCA_Info 構造体に格納されます。

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 の柔軟性を増加し、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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>pSessionConfig</i>	ORCA クライアントに、後続する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全ての設定を指定します。

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	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。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もできます。

```
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; // 既存ファイルを上書きしますか？
    PBORCA_ENCODING eExportEncoding;
    // 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; // バイナリを含む
    BOOL bExportCreateFile; // ソースをファイルへエクスポート
    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
    PVOID filler1; // 将来使用するために予約
    PVOID filler2;
    PVOID filler3;
    PVOID filler4;
} PBORCA_CONFIG_SESSION, FAR *PPBORCA_CONFIG_SESSION;

```

メンバ変数	説明
<i>eClobber</i>	<p>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ついて指定します。この値は以下のとおりです。</p> <p>PBORCA_LibraryEntryExport PBORCA_LibraryEntryExportEx PBORCA_DynamicLibraryCreate PBORCA_ExecutableCreate PBORCA_LibraryDelete</p> <p>環境設定セッションに、以下の <i>eClobber</i> 値のいずれかを使用することが可能です。</p> <ul style="list-style-type: none"> • PBORCA_NOCLOBBER 既存ファイルを上書きしません • PBORCA_CLOBBER 既存ファイルが書き込み不可でない場合は上書きします • PBORCA_CLOBBER_ALWAYS 既存ファイルが書き込み不可でも上書きします • PBORCA_CLOBBER_DECIDED_BY_SYSTEM 以前の ORCA リリースと同様に動作するように前述の関数が発生します

メンバ変数	説明
<i>eExportEncoding</i>	<p>PBORCA_LibraryEntryExport を使用してソースのエンコーディングを指定します。</p> <ul style="list-style-type: none"> • PBORCA_UNICODE Unicode ORCA クライアント用のデフォルト • PBORCA_ANSI_DBCS ANSI ORCA クライアント用のデフォルト • PBORCA_UTF8 • PBORCA_HEXASCII
<i>bExportHeaders</i>	<p>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。</p>
<i>bExportIncludeBinary</i>	<p>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。</p>
<i>bExportCreateFile</i>	<p>この変数に TRUE を設定すると、PBORCA_LibraryEntryExport はファイルへソースをエクスポートします。生成されるファイル名は、PowerBuilder オブジェクト エントリ名に <i>.sr?</i> ファイル拡張子が付きます。デフォルト値は FALSE です。</p>
<i>pExportDirectory</i>	<p>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。</p>
<i>elmportEncoding</i>	<p>ソース エンコーディング。</p> <p>PBORCA_CompileEntryImport と PBORCA_CompileEntryImportList への連続した呼び出しは、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。</p>

例 この例は、PBORCA_CONFIG_SESSION 構造体に環境を設定しています。

```

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;
iErrCode = PBORCA_ConfigureSession(
    lpORCA_Info->hORCASession,
    lpORCA_Info->pConfig);
return iErrCode;
}
```

関連項目

[PBORCA_CompileEntryImportList](#)
[PBORCA_ApplicationRebuild](#)

PBORCA_DeployWinFormProject

機能 Windows フォーム プロジェクトを生成、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,
 LPTSTR lpszProjectName,
 LPTSTR lpszIconFileName,
 PBORCA_DOTNETPROC pDotNetProc
 LPVOID pData);`

引数	説明
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszLibraryName</i>	プロジェクト エントリを含むファイル名の文字列値のポインタ
<i>lpszProjectName</i>	配布情報を含むプロジェクト オブジェクト
<i>lpszIconFileName</i>	アプリケーション アイコン ファイルの名前
<i>pDotNetProc</i>	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る。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。
<i>pUserData</i>	PBORCA_DOTNETPROC コールバック関数へ渡されるユーザ 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_CBBUILDERERROR	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
{
    PBORCA_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）や 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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszLibraryName</i>	PBD や DLL を構築するライブラリ ファイル名の文字列値のポインタ
<i>lpszPBRName</i>	PBD や DLL に取り込みたいオブジェクトの PowerBuilder リソース ファイル名の文字列値へのポインタ。アプリケーションにリソース ファイルがない場合には、ポインタに 0 を指定します。
<i>lFlags</i>	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 <i>lFlags</i> 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_PBDCREATERERROR	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幾つかの動的ライブラリを含める計画がある場合には、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（拡張子のみ異なります）で作成されます。例えば、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 で出力すると、C:¥DIR1¥DIR2¥PROG.PBD

- マシーン コードで出力すると、C:¥DIR¥DIR2¥PROG.DLL

eClobber の設定 ファイル システム中に PBD や DLL が既に存在する場合、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既に存在していた場合、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、読み込みのみの属性を持つ場合は処理は失敗します。
PBORCA_CLOBBER_ALWAYS	既存ファイルの属性に関わらず、処理は成功します。

例

この例では、PROCESS.PBL からマシーン コードの DLL を構築します。トレース情報やエラー情報付き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 );
```

35 ページの「例について」で紹介しているように、例中のセッション情報は ORCA_Info 構造体に保存しています。

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、幾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。既に構築されている 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 IFlags );
```

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の文字列値へのポインタ
lpszIconName	アイコン ファイル名の文字列値へのポインタ。既存のアイコン ファイルでなければいけません。
lpszPBRName	PowerBuilder リソース ファイル名の文字列値へのポインタ。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。
pLinkErrProc	コールバック関数の 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たくない場合は、pLinkErrProc に 0 を設定します。
pUserData	コールバック関数である PBORCA_ExecutableCreate へ渡すユーザ データへのポインタ 通常ユーザ データには、バッファまたは、エラー情報とバッファ サイズに関する情報を格納するコールバック関数中の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
<i>iPBDFlags</i>	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。 <ul style="list-style-type: none"> • 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、既に PBD または PowerBuilder DLL であり、実行ファイルには含めない
<i>iNumberOfPBDFlags</i>	<i>iPBDFlags</i> 配列中のエレメント数であり、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。
<i>IFlags</i>	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 <i>IFlags</i> 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、以下の解説にて説明しています。

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
-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色々なオプションに関しては、『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 リスト中の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、また、どれが実行ファイルの中に取り込まれて構築されるのかを指定します。
iPBDFlags 配列の各整数は、ORCA のライブラリ リスト上にある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) に組み込まれて構築されます。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て構築されます。

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、個々のビットを設定することで、様々なマシン コード生成オプションの設定が可能です。下記の表は、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 コードを取得するためには、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 コードを取得するためには、PBORCA_MACHINE_CODE および PBORCA_MACHINE_CODE_16 を使用します。 16- ビット Pcode を取得するためには、PBORCA_P_CODE_16 を使用します。 PowerBuilder 7 以降はサポート対象外 PowerBuilder は、Windows 3.x 16- ビット プラットフォームを今後サポートしません。
2	0 = Open Server なし 1 = Open Server あり	Open Server 実行ファイルを構築するためには、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しません。

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ためには、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ためには、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ためには、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ためには、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_CONTROL

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

```
IFlags = PBORCA_P_CODE;
```

色々なマシン コードのオプションへの IFlags を設定すると、ビット フラグの定数は、希望する組み合わせを OR 条件で判断されます。

```
IFlags = 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 コンテキスト情報付きのネイティブ 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 PBD や DLL が既に存在する場合、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既に存在していた場合、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、読み込みのみの属性を持つ場合は処理は失敗します。
PBORCA_CLOBBER_ALWAYS	既存ファイルの属性に関わらず、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 コードを構築しています。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。例中では、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 );

```

コールバック用のデータバッファの設定に関する更なる情報については、25 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。

35 ページの「例について」で紹介しているように、例中のセッション情報は `ORCA_Info` 構造体に保存しています。

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	コメントを変更したいライブラリ名の文字列値へのポインタ
lpszLibComments	新しいライブラリ コメントの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

この例では、MASTER.PBL ライブラリのコメントを変更しています。

```
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¥¥pb11.0¥¥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);
```

35 ページの「例について」で紹介しているように、例中のセッション情報は ORCA_Info 構造体に保存しています。

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 *lpzLibraryName*,
LPTSTR *lpzLibraryComments*);

引数	説明
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpzLibraryName</i>	作成するライブラリのファイル名の文字列値へのポインタ
<i>lpzLibraryComments</i>	新しいライブラリを説明するコメントの文字列値へのポインタ

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	オブジェクトが既に存在する
-9 PBORCA_INVALIDNAME	ライブラリ名が不正である

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することができます。

例

この例では NEWLIB.PBL という新しいライブラリを作成し、説明のコメントをつけています。

```
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¥¥pb11.0¥¥demo¥¥newlib.pbl");
```

```
pszLibraryComments =  
    _TEXT("PBL は XYZ アプリケーションの先祖オブジェクトを含み  
    ます");  
  
// ライブラリを作成  
lpORCA_Info->lReturnCode =  
    PBORCA_LibraryCreate(lpORCA_Info->hORCASession,  
        pszLibraryName, pszLibraryComments);
```

35 ページの「例について」で紹介しているように、例中のセッション情報は ORCA_Info 構造体に保存しています。

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 *pszLibraryName*);

引数	説明
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>pszLibraryName</i>	削除するライブラリ ファイル名の文字列値へのポインタ

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。読み込みのみの属性を持つ PowerBuilder ライブラリを削除したい場合には、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 この例では、ライブラリ名 EXTRA.PBL を削除しています。

```
LPTSTR pszLibraryName;

// ライブラリ名を指定
pszLibraryName =
    _TEXT("c:¥¥sybase¥¥pb11.0¥¥demo¥¥extra.pbl");

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

35 ページの「例について」で紹介しているように、例中のセッション情報は ORCA_Info 構造体に保存しています。

関連項目 PBORCA_ConfigureSession
PBORCA_LibraryCreate

PBORCA_LibraryDirectory

機能 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。ディレクトリ中のオブジェクトリストも含みます。

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

引数	説明
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszLibName</i>	必要なディレクトリ情報のライブラリのファイル名文字列値へのポインタ
<i>lpszLibComments</i>	ライブラリと共に格納されるコメントを出力する ORCA 中のバッファへのポインタ
<i>iCmntsBuffLen</i>	<i>lpszLibComments</i> が指し示すバッファ長 (TCHAR で指定)。推奨値は、PBORCA_MAXCOMMENTS + 1 です。
<i>pListProc</i>	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毎に呼び出されます。 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。これらは、PBORCA_DIRENTRY 型の構造体に格納されています。
<i>pUserData</i>	コールバック 関数の PBORCA_LibraryDirectory へ渡すユーザ データへのポインタ ユーザ データは一般的に、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 バッファに各々のオブジェクトごとのコメントを格納することができます。

ライブラリ エントリに関する情報 ライブラリ中の各々のエントリに関して戻る情報は、コールバック関数中で行う処理に依存します。構造体 PBORCA_DIRENTRY 中のライブラリ エントリに関する情報を ORCA はコールバック関数に渡します。コールバック関数は、バッファ中のどの情報であっても pUserData が指し示す構造と内容を調べることができます。

PBORCA_LibraryDirectory を呼び出すとき、ライブラリ中のエントリ数はわかりません。2つのアプローチをとることができます。

- メモリを適度な大きさのブロック サイズで割り当て、オーバーフローした場合には再度割り当てます (23 ページの「ORCA コールバック関数について」で図解)。
- lpUserDataBuffer をリンク リストの先頭に位置づけます。各 PBORCA_DIRENTRY の戻りは、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

この例では、リンク リスト ヘッダを定義しています。

```
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;
```

以下のように定義をして、各 DirectoryProc コールバック関数の呼び出しは、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 の中で長さを指定
        fpDirectoryProc,
        pHead);
    // PBORCA_LibraryEntryInformation の例を参照
    if (lpORCA_Info->lReturnCode == PBORCA_OK)
        GetEntryInfo(pHead);
    CleanUp(pHead);

    // CleanUp - 割り当てたメモリの解放
    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 を使用してコールバック処理
    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;
```

35 ページの「例について」で紹介しているように、例中のセッション情報は ORCA_Info 構造体に保存しています。

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ほかのライブラリへコピーします。

構文 INT **PBORCA_LibraryEntryCopy** (HPBORCA *hORCASession*,
LPTSTR *lpzSourceLibName*,
LPTSTR *lpzDestLibName*,
LPTSTR *lpzEntryName*,
PBORCA_TYPE *otEntryType*);

引数	説明
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpzSourceLibName</i>	オブジェクトを含むソース ライブラリのファイル名の文字列へのポインタ
<i>lpzDestLibName</i>	コピーするオブジェクトの格納先ライブラリ ファイル名の文字列値へのポインタ
<i>lpzEntryName</i>	コピーされるオブジェクト名の文字列値へのポインタ
<i>otEntryType</i>	コピーされるエントリのオブジェクト タイプを指定する 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て、次にオブジェクトが存在する場合には、オブジェクトのバイナリ コンポーネントをコピーします。

例

この例では、`d_labels` という名前のデータウィンドウを `SOURCE.PBL` から `DESTIN.PBL` へコピーしています。

```
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、ポインタの `lpszSourceLibraryName`、`lpszDestinationLibraryName`、および `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 タイプであると想定しています。

```
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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszLibName</i>	オブジェクトを含むライブラリ ファイル名の文字列 値へのポインタ
<i>lpszEntryName</i>	削除されるオブジェクト名の文字列へのポインタ
<i>otEntryType</i>	削除されるエントリのオブジェクト タイプを指定する PBORCA_TYPE のカタログ データ型の値。この値 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 この例では、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ています。

```
l rtn = PBORCA_LibraryEntryDelete(  
lpORCA_Info->hORCASession,  

```

```
_TEXT("c:¥¥app¥¥source.pbl"),  
_TEXT("d_labels"), PBORCA_DATAWINDOW);
```

この例では、ポインタの `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タイプであると想定しています。

```
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 またはファイルへ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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszLibraryName</i>	エクスポートしたいオブジェクトを含むライブラリ ファイル名の文字列値へのポインタ
<i>lpszEntryName</i>	エクスポートされるオブジェクト名の文字列値へのポインタ
<i>otEntryType</i>	エクスポートされるエントリのオブジェクト タイプ 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
<i>lpszExportBuffer</i>	PBORCA_CONFIG_SESSION プロパティの bExportCreateFile が FALSE の場合に、エクスポートされるソースのコードを格納する ORCA 中のデータ バッファへのポインタ。bExportCreateFile が TRUE の場合は、この引数は NULL にすることができます。
<i>lExportBufferSize</i>	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

戻り値	説明
-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 へ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、この関数の動作を変更することができます。しかし、下位互換性のために、デフォルトでは動作は変更しません。

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 バッファの中に戻ります。bExportCreateFile プロパティが TRUE の場合、pConfigSession->pExportDirectory が指し示すディレクトリの中のファイルにソースは書き込まれます。

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 ヘッダを書きます。バッファ内で、エクスポートされたソース コードは改行 (Hex 0D) と新規行 (Hex 0A) というキャラクターを各表示行ごとの最後に取り込みます。

ソース コード エンコーディング PowerBuilder は 4 種類の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 PBORCA_ANSI_DBCS 形式でソースをエクスポートし、Unicode クライアントは PBORCA_UNICODE 形式でソースを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 を設定すると、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 コンポーネントを自動的に含めるように明示的に要求することができます。新しい環境では、この設定をしておくことを推奨します。以前の ORCA のバージョンではこの機能をサポートしていないため、以前からの方法もまだサポートしています。

下位互換性テクニック

以前のバージョンのように、各 PBORCA_LibraryEntryExport 要求のあとに、*otEntryType* に PBORCA_BINARY を持つ 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、*otEntryType* に PBORCA_BINARY を設定している二番目の PBORCA_LibraryEntryExport 呼び出しをすることができます。下位互換性のため、*otEntryType* に PBORCA_BINARY を設定すると、pConfigSession->bExportHeaders = TRUE と pConfigSession->bExportIncludeBinary = TRUE の環境設定プロパティは無視されます。

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ためには、まず最初に 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 のサイズをバイトで表します。

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、全ての必要なデータの変換を行います。十分なバッファ容量で無い場合には、PBORCA_BUFFERTOOSMALL リターンコードが返されます。lExportBufferSize が必要な長さと等しいサイズの場合 PBORCA_LibraryEntryExport は成功しますが、エクスポートされたデータに Null 区切り文字を追加しません。lExportBufferSize が十分なサイズの場合 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知りたい場合には、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全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 ファイルがまだ存在しない場合、eClobber の設定に関わらず PBORCA_LibraryEntryExport は PBORCA_OK を返します。以下の表では、エクスポート ファイルが既に存在する場合に eClobber の設定が PBORCA_LibraryEntryExport の動作をどのように変更するのかについて説明しています。PBORCA_OBJEXISTS の戻り値は、既存ファイルを上書きしないということを意味します。

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	読み込み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

この例では、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ます。そして、szEntrySource というバッファに PBORCA_UTF8 ソース コードを入れます。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 バッファへ書き込む
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);
```

この例では、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ます。PBORCA_UNICODE ソース コードを c:¥¥app¥d_labels.srd へ書きます。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;
```

```
// ファイルへ書き出す
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);
// 実際にエクスポート进行处理する
lpORCA_Info->lReturnCode = PBORCA_LibraryEntryExport(
    lpORCA_Info->hORCASession,
    _TEXT("c:¥¥app¥¥source.pbl"),
    _TEXT("d_labels"), PBORCA_DATAWINDOW,
    NULL, 0);
```

この例では、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。ウィンドウは、埋め込み OLE オブジェクトを含みます。ソースコードとバイナリオブジェクトは両方ともに、`c:¥¥app¥¥w_connect.srw` へエクスポートされます。エクスポートヘッダを含み、ソースは PBORCA_ANSI_DBCS 形式で書いてあります。

```
// ANSI_DBCS ソース エンコーディングを示す
lpORCA_Info->pConfig->eExportEncoding =
    PBORCA_ANSI_DBCS;
// ファイルへエクスポートする
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);
// 実際にエクスポート进行处理する
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 バッファへ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
<i>hORCASession</i>	事前に確立した ORCA セッションのハンドル
<i>lpszLibraryName</i>	エクスポートしたいオブジェクトを含むライブラリのファイル名である文字列値へのポインタ
<i>lpszEntryName</i>	エクスポートされるオブジェクト名の文字列値へのポインタ
<i>otEntryType</i>	エクスポートされるエントリのオブジェクト タイプを指定する 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
<i>lpszExportBuffer</i>	PBORCA_CONFIG_SESSION プロパティの bExportCreateFile が FALSE の場合に、エクスポートされるソースのコードを格納する ORCA 中のデータバッファへのポインタ。bExportCreateFile が TRUE の場合は、この引数は NULL にすることができます。
<i>lExportBufferSize</i>	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。
<i>*plReturnSize</i>	エクスポートされるソース バッファやファイルのサイズ (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 へ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 で呼び出し元へ返すという点以外は、PBORCA_LibraryEntryExport と同じ動作をします。

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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszLibraryName</i>	情報が欲しいオブジェクトを含むライブラリ ファイル名の文字列値へのポインタ
<i>lpszEntryName</i>	情報が欲しいオブジェクト名の文字列値へのポインタ
<i>otEntryType</i>	エントリのオブジェクト タイプを指定する PBORCA_TYPE カタログ データ型。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
<i>pEntryInformationBlock</i>	求められた情報を格納する ORCA 中の PBORCA_ENTRYINFO 構造体（下記解説を参照）

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、以下の構造体中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 サイズをバイトで
    LONG lSourceSize; // ソース サイズをバイトで
} PBORCA_ENTRYINFO, FAR *PPBORCA_ENTRYINFO;
```

ソース コード サイズの使用 `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) を見積もるためにしばしば使用されます。エクスポートされるソースのサイズは、有効な `ConfigureSession` の設定により変化します。以下の表では、`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。 <code>PBORCA_LibraryEntryInformation</code> は常に Unicode ソースに必要なバイト数を返します。
bExportHeaders=TRUE	<i>otEntryType</i> が <code>PBORCA_BINARY</code> でない場合、 <code>lSourceSize</code> 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が付加されます。
bExportIncludeBinary=TRUE	<i>otEntryType</i> が <code>PBORCA_BINARY</code> でない場合、 <code>lSourceSize</code> はバイナリ オブジェクトで Unicode での表示を生成するために必要なバイト数が付加されます。

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せずに前もって計算することはできません。開発者は、割り当てる適切なバッファ サイズを開発者自身で見積もります。たとえば、エントリのソースが全て 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 区切り文字として 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コントロールを含んでいるかを判断するために、PBORCA_BINARY の *otEntryType* と共に 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されるバイナリ データの表示を格納するために必要なバッファ サイズを決定します。PowerBuilder 11.0 では下位互換性のためにまだこの機能をサポートします。しかし、エントリのソースとバイナリ コンポーネントの両方に十分なバッファ サイズを取得するためには、pConfigSession->bExportIncludBinary = TRUE を設定するほうがより効果的です。

例 この例では、PBL 中の各オブジェクトの情報を取得しています。これは 78 ページの「PBORCA_LibraryDirectory」の例の延長にあります。

```
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ほかのライブラリへ、PowerBuilder ライブラリ エントリを移動します。

構文 INT **PBORCA_LibraryEntryMove** (PBORCA *hORCASession*,
 LPTSTR *lpzSourceLibName*,
 LPTSTR *lpzDestLibName*,
 LPTSTR *lpzEntryName*,
 PBORCA_TYPE *otEntryType*);

引数	説明
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpzSourceLibName</i>	オブジェクトを含むソース ライブラリ ファイル名の文字列値へのポインタ
<i>lpzDestLibName</i>	移動したいオブジェクトの格納先ライブラリとなるファイルの文字列値へのポインタ
<i>lpzEntryName</i>	移動されるオブジェクト名の文字列値へのポインタ
<i>otEntryType</i>	削除されるエントリのオブジェクト タイプを指定する PBORCA_TYPE のカタログ データ型。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、次にオブジェクトが存在する場合には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 から DESTIN.PBL へ移動します。

```
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 用のポインタは、有効なライブラリとオブジェクト名を指し示しており、また、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 継承階層内の PowerBuilder オブジェクトの一覧を取得するクエリです。ウィンドウ、メニュー、ユーザ オブジェクトのみ取得できます。

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

引数	説明
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszLibraryName</i>	取得するオブジェクトを含むライブラリ ファイル名の文字列値へのポインタ
<i>lpszEntryName</i>	取得するオブジェクト名の文字列値へのポインタ
<i>otEntryType</i>	取得す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、以下のとおりです。 PBORCA_WINDOW PBORCA_MENU PBORCA_USEROBJECT
<i>pHierarchyProc</i>	PBORCA_ObjectQueryHierarchy コールバック関数へのポインタ。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へ渡す情報は、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。
<i>pUserData</i>	PBORCA_ObjectQueryHierarchy コールバック関数へ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、コールバック関数がバッファサイズの情報とエラー情報を格納するバッファへのポインタ、またはバッファの中に含まれます。

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
-5 PBORCA_LIBLISTNOTSET	ライブラリ リストが未設定

戻り値	説明
-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 ライブラリ中のオブジェクト w_processdata の先祖の一覧を取得しています。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 ObjectQueryHierarchy コールバック関数を呼び出します。ObjectQueryHierarchy の中で、lpUserData が指し示すバッファに先祖名を格納します。例では、lpUserData バッファは既に設定されています。

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

コールバック用のデータ バッファの設定に関する更なる情報については、25 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。

35 ページの「例について」で紹介しているように、例中のセッション情報は ORCA_Info 構造体に保存しています。

関連項目

PBORCA_ObjectQueryReference

PBORCA_ObjectQueryReference

機能 PowerBuilder オブジェクトがほかのオブジェクトを参照するリストを取得するためのクエリです。

構文 INT **PBORCA_ObjectQueryReference** (HPBORCA *hORCA*Session, LPTSTR *lpszLibraryName*, LPTSTR *lpszEntryName*, PBORCA_TYPE *otEntryType*, PBORCA_REFPROC *pRefProc*, LPVOID *pUserData*);

引数	説明
<i>hORCA</i> Session	事前に確立した ORCA セッションへのハンドル
<i>lpszLibraryName</i>	取得されるオブジェクトを含むライブラリ ファイルの文字列値へのポインタ
<i>lpszEntryName</i>	取得するオブジェクト名の文字列値へのポインタ
<i>otEntryType</i>	取得す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。指定する値は以下のとおりです。 <div>PBORCA_APPLICATION PBORCA_DATAWINDOW PBORCA_FUNCTION PBORCA_MENU PBORCA_QUERY PBORCA_STRUCTURE PBORCA_USEROBJECT PBORCA_WINDOW PBORCA_PIPELINE PBORCA_PROJECT PBORCA_PROXYOBJECT</div>
<i>pRefProc</i>	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へ渡す情報は、PBORCA_REFERENCE 型の構造体に格納される参照オブジェクト名、ライブラリ、オブジェクト タイプです。
<i>pUserData</i>	コールバック関数 PBORCA_ObjectQueryReference へ渡されるユーザ データへのポインタ 通常ユーザ データは、コールバック関数がバッファ サイズの情報とエラー情報を格納するバッファへのポインタ、またはバッファの中に含まれます。

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、ライブラリ WINDOWS.PBL 中のウィンドウ オブジェクト w_processdata が参照するオブジェクトの一覧を取得するクエリです。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の ObjectQueryReference を呼び出します。ObjectQueryReference に関する記述の中で、lpUserData が指し示すバッファ中のオブジェクト名を格納します。例では、lpUserData バッファは既に設定済みです。

```

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 バッファの設定に関する更なる情報については、25 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。

35 ページの「[例について](#)」で紹介しているように、例中のセッション情報は ORCA_Info 構造体に保存しています。

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 <i>hORCASession</i>);				
	<table><tr><th>引数</th><th>説明</th></tr><tr><td><i>hORCASession</i></td><td>事前に確立した ORCA セッションへのハンドル</td></tr></table>	引数	説明	<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
引数	説明				
<i>hORCASession</i>	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斷するために 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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>*pConfig</i>	事前に割り当てた構造体へのポインタ。通常で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	ソース管理へ接続できない
-23 PBORCA_REGREADERERROR	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 構造体が適用する接続情報に基づいたソース管理のセッションを初期化します。PBORCA_SCC 構造体は、以下のように定義をします。

```
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、または、指定されたワークスペース ファイルと関連する接続情報を取得するために PBORCA_SccGetConnectProperties を呼び出すことができます。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 オブジェクトを開く
- 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存在す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)、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ためには、SCC 接続を終了し、ORCA セッションを閉じて、その上で新しい ORCA セッションを開く必要があります。

例

以下の例は、PBNative ソース管理へ接続しています。

```
PBORCA_SCC  sccConfig;
memset(&sccConfig, 0x00, sizeof(PBORCA_SCC));
// PBNative への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¥¥orcasc.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いて、リフレッシュしてオフラインを再構築します。

構文

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

引数	説明
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>*pConfig</i>	事前に割り当てられた構造体へのポインタ。通常で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	ソース管理へ接続できない
-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_IMPORTONLY 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必要があるオブジェクトは全て、ローカルプロジェクトパス上に既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 プロバイダへ接続しません。

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くに役立ちます。ネットワークに接続している間、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。そして時間外に、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可能です。

例

この例は、現行の作業ディレクトリに置かれた 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 構造体を用意しています。そして、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。同期が取られていないオブジェクトのみリフレッシュされます。現行ユーザによりチェックアウトしたオブジェクトは、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 の操作で同期を取らないターゲット 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名
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 を共有し、ライブラリ リストのライブラリが前の PBORCA_SccRefreshTarget 処理でリフレッシュに成功していると確信する場合、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。しかし、対象外のライブラリは、アプリケーションのフル再構築で使用されます。

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 へ接続します
// ...
_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 [] pExcludeArray[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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>pWorkspaceFile</i>	PowerBuilder のワークスペース ファイルの相対パス名、または絶対パス名 (PBW)
<i>*pConfig</i>	事前に割り当てた構造体へのポインタ。通常で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 接続のプロセスを簡素化します。PBORCA_SccGetConnectProperties 呼び出しの引数中に含まれる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、SCC プロバイダとプロジェクトへの接続が成功します。これらのプロパティの一部を上書きすることができます。

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 構造体中の変数は、以下の表のとおり定義されています。

Member	説明
<i>hWnd</i>	親ウィンドウのハンドル値で、通常は NULL
<i>szProviderName</i>	SCC プロバイダ名

Member	説明
<i>*plCapabilities</i>	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います。
<i>szUserID</i>	ソース管理オブジェクト用のユーザ ID
<i>szProject</i>	ソース管理オブジェクトの名前
<i>szLocalProjPath</i>	プロジェクト用のローカルなルート ディレクトリ
<i>szAuxPath</i>	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たい全ての文字列を含むことができます。SCC プロバイダからダイアログボックスを開くことなく、PBORCA_SccGetConnectProperties は、暗黙的な接続を可能にするための値を戻します。
<i>szLogFile</i>	SCC 接続用のログ ファイル名
<i>pMsgHandler</i>	SCC メッセージ用のコールバック関数
<i>*pCommentLen</i>	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。
<i>lAppend</i>	SCC ログ ファイルを追加する (<i>lAppend</i> =1) か、または上書きする (<i>lAppend</i> =0) かを決める
<i>pCommBlk</i>	内部使用のための予約語

PBORCA_SccGetConnectProperties 関数を呼び出した後に PBORCA_SCC 構造体へ追加されるプロパティ値は、szProviderName、szUserID、szProject、szLocalProjPath、szAuxPath、szLogFile、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	検索されるファイル数
*ppFileNames	検索されるファイル名

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り出します。一般的に、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。たとえば、BMP、JPG、ICO、DOC、HLP、HTM、JSP、および PBR ファイルを含みます。

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用に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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>eRebldType</i>	アプリケーションの再構築の方法を指定することを許可する（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。リフレッシュの操作は、それぞれの PowerBuilder ライブラリにオブジェクトをインポートしてコンパイルします。

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、リフレッシュの操作には含まれません。

PBORCA_REBLD_TYPE 引数は、PBORCA_SccRefreshTarget を呼び出すときにアプリケーションをどのように再構築するかを指定します。

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

関連項目 PBORCA_SccClose
PBORCA_SccConnect
PBORCA_SccExcludeLibraryList
PBORCA_SccSetTarget

PBORCA_SccResetRevisionNumber

機能 オブジェクトのバー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 への **SccQueryInfoEx** エクステンションを実装している 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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszLibraryName</i>	リビジョン番号をリセットしたいオブジェクトを含む PBL ファイルの相対パスまたは絶対パス
<i>lpszEntryName</i>	<i>.sr?</i> 拡張子を含めないオブジェクトの名前の文字列値へのポインタ
<i>otEntryType</i>	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
<i>lpszRevisionNum</i>	文字列値または Null。Null を指定すると、PBL 中の現行リビジョンを削除します。

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 (<i>lpszLibraryName</i> または <i>lpszEntryName</i> が Null の場合)
-7 PBORCA_LIBIOERROR	読み/書きアクセス用で PBL が開けない

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* 引数で指定した PowerBuilder ライブラリ中の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上のオブジェクト 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 リスト中に存在していなくてもかまいません。

ORCA プログラムが内部的に PBL のオブジェクト ソースを変更し、かつ、以下のいずれかが true の場合、一般的には PBORCA_SccResetRevisionNumber を呼び出します。

- ORCA プログラムは、PBORCA_CompileEntryImport 呼び出しを介してオブジェクトの個別のリビジョンを PBL の中へインポートします。ORCA プログラムがインポートされる正確なリビジョン番号が分かる場合は、*lpszRevisionNum* 引数の中でそのリビジョン番号を指定します。正確なリビジョン番号が分からない場合は、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 呼び出しを介して PBL から既存のオブジェクト ソースをエクスポートすることで SCC にチェックインするのと同様の処理を内部的に行い、SCC リポジトリ中のオブジェクト ソースをチェックします。ジョブを実行するためには、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 へ渡し、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 IFlags, PBORCA_SETTGTPROC pSetTgtProc, LPVOID pUserData);`

引数	説明
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>pTargetFile</i>	ターゲットのファイル名
<i>IFlags</i>	ターゲットの操作の振る舞いをコントロールすることを許可する（解説の項を参照）
<i>pSetTgtProc</i>	ユーザ定義のコールバック関数へのポインタ
<i>pUserData</i>	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一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たかが分かります。また、ある特定の共有ライブラリが事前のタスクで既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。

ソース管理から検索するターゲット ライブラリ上でリフレッシュを行う設定をするために、IFlags 引数を割り当てます。

フラグ	説明
PBORCA_SCC_OUTOFDATE	<p>PBL 中に存在するオブジェクトの同期が取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。</p> <p>PBORCA_SCC_IMPORTONLY が設定されていない場合、SCC リポジトリの日付外のオブジェクトのみがリフレッシュされます。</p> <p>PBORCA_SCC_OUTOFDATE と PBORCA_SCC_REFRESH_ALL は相互に排他的です。</p>
PBORCA_SCC_REFRESH_ALL	<p>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 が使用されている場合、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は、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、次にターゲット ライブラリへインポートされます。</p>
PBORCA_SCC_IMPORTONLY	<p>ローカルなプロジェクト パス上に既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全てのオブジェクトを指し示します。SCC ベンダの管理ツールを使用してあらかじめローカルなパスをリフレッシュした場合にこのフラグを使用します。この ORCA セッション中にあらかじめ PBORCA_SccConnectOffline を呼び出す場合、PBORCA_SCC_IMPORTONLY が必要です。PBORCA_SCC_IMPORTONLY は特定の SCC ラベルやプロモーション グループからターゲットを再構築する場合に特に有用です。</p>
PBORCA_SCC_EXCLUDE_CHECKOUT	<p>ユーザの介入を必要としないバッチ ジョブでローカルなターゲットのリフレッシュを行う仕組みを提供します。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します。PBORCA_SccConnectOffline と共に使用する場合、チェックアウト ステータスはワークスペース名 .PBC から取得します。オフライン処理に関しては、ワークスペース名は事前に呼び出した PBORCA_SccGetConnectProperties から取得します。</p>

PBORCA_SccConnect で指定したローカルなプロジェクト パス中にターゲット ライブラリやディレクトリが存在しない場合、PBORCA_SccSetTarget を動的に呼び出すとこれらのディレクトリと PBL ファイルは作成されます。

SccSetTarget は、暗黙の PBORCA_SessionSetLibraryList と PBORCA_SessionSetCurrentAppl を行います。PBORCA_SccSetTarget (及びおそらく PBORCA_SccRefreshTarget も) 呼び出した後に、PBD や EXE を作成するといったような、現行アプリケーションの要求やライブラリ リストの初期化などのほかの処理を行うことができます。これは PBORCA_SccClose を呼び出すよりもより効率的であり、その次に、PBD や EXE を作成するために現行アプリケーションの設定やライブラリ リストの初期化を行います。

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
<i>hORCASession</i>	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場合は、PowerBuilder DLL が割り当てたメモリが解放されないために、結果的にメモリ 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 セッションを閉じています。

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

35 ページの「例について」で紹介しているように、例中のセッション情報は ORCA_Info 構造体に保存しています。

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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszErrorBuffer</i>	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
<i>iErrorBufferSize</i>	<i>lpszErrorBuffer</i> が指し示すバッファへのポインタ。PBORCA_MSGBUFFER の定数は、通常バッファ サイズに 256 を設定します。ORCA のヘッダファイル PBORCA.H に定義されます。

戻り値 なし。

解説 ほかの ORCA 関数でエラーが返るときはいつでも PBORCA_SessionGetError を呼ぶことができます。コードのリストは、35 ページの「ORCA リターン コード」を参照してください。また、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。

現在エラーがない場合には、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へ格納します。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);
```

35 ページの「例について」で紹介しているように、例中のセッション情報は ORCA_Info 構造体に保存しています。

PBORCA_SessionOpen

機能	ORCA セッションを確立し、一連の 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	<p>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。</p> <p>ORCA セッションを開いたままにするために必要となるオーバーヘッドや資源はありません。このため、いったんセッションを確立したら、必要なだけ長い間開いたままにすることができます。</p> <p>インポートやオブジェクトの検索や実行ファイルの構築などいくつかの 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内容を提供するために PBORCA_SessionSetLibraryList と PBORCA_SessionSetCurrentAppl を呼び出す必要があります。</p> <p>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 を呼び出す予定の場合には、PBORCA_SessionSetLibraryList および PBORCA_SetCurrentAppl は呼び出しません。</p>
例	<p>この例では、ORCA セッションを開いています。</p> <pre>lpORCA_Info->hORCASession = PBORCA_SessionOpen(); if (lpORCA_Info->hORCASession = NULL) { lpORCA_Info->lReturnCode = 999; _tcscpy(lpORCA_Info->lpszErrorMessage, TEXT("Open session failed")); }</pre>
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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>lpszApplLibName</i>	アプリケーション ライブラリ名の文字列値へのポインタ
<i>lpszApplName</i>	アプリケーション オブジェクト 名の文字列値へのポインタ

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既に設定済み
-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	参照ライブラリ リストに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検索を行う 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、どこにあるのかが分かるように完全修飾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。一度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には、いったん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ためには、アプリケーションの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関する情報については、31 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

この例では、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);
```

35 ページの「例について」で紹介しているように、例中のセッション情報は ORCA_Info 構造体に保存しています。

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、オブジェクトの参照を解決するためにリスト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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>*pLibNames</i>	文字列へのポインタ配列のへのポインタ。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、各ライブラリは完全修飾パスで指定します。
<i>iNumberOfLibs</i>	pLibNames の配列が指し示すポイント中のライブラリ名の数

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、またはリスト上に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検索を行う 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完全修飾パスで指定します。

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には、いったん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 のライブラリ リストの使い方 ORCA セッション中にオブジェクトを再生成または検索するときに、参照するオブジェクトを見つけるために ORCA はパスを検索します。PowerBuilder と同じように、ORCA は参照オブジェクトが見つかるまで指定されたライブラリ探索パス内に順番に探します。

ライブラリ リストが不要な関数 ライブラリ リストを設定することなく、以下のライブラリを管理する関数とソースをコントロールする関数を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、PocketBuilder 用のライブラリ ファイル名配列を構築し、セッションのライブラリ リストを設定しています。

```

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);

```

35 ページの「例について」で紹介しているように、例中のセッション情報は ORCA_Info 構造体に保存しています。

関連項目

PBORCA_SessionSetCurrentAppl

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
<i>hORCASession</i>	事前に確立した ORCA セッションへのハンドル
<i>*pExeInfo</i>	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	不正なパラメータ 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たときに PBORCA_SetExeInfo も動的ライブラリ用にプロパティを設定します。
PBORCA_EXEINFO 構造体は以下のように定義します。

```
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、PBORCA_SetExeInfo 呼び出しが完了したら即座に呼び出し元がこのメモリを解放することができるようにするために、内部の ORCA 制御構造体へコピーされます。

実行可能なバージョン情報は、PBORCA_SessionClose 処理中に削除されます。したがって、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 構造対中の全ての要素を再割り当てる必要があります。

FileVersionNum と ProductVersionNum の文字列は、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数値をそれぞれコンマで区切って表した 4 つの数値で構成されます。例えば、「11,0,,0,4621」と指定します。

例

この例では、PowerBuilder アプリケーションの実行ファイルの情報を設定しています。

```
memset(&ExeInfo, 0x00, sizeof(PBORCA_EXEINFO));
ExeInfo.lpszCompanyName = _TEXT("Sybase");
ExeInfo.lpszProductName = _TEXT("PowerBuilder 11.0
    DBAuto"));
ExeInfo.lpszDescription =
    _TEXT("Batch Automation for QADB Test Suite");
ExeInfo.lpszCopyright = _TEXT("2006");
ExeInfo.lpszFileVersion = _TEXT("11.0.0.001");
ExeInfo.lpszFileVersionNum = _TEXT("11,0,0,001");
ExeInfo.lpszProductVersion = _TEXT("11.0.0.001");
ExeInfo.lpszProductVersionNum = _TEXT("11,0,0,001");
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

本章について

本章では、代表的な幾つかの ORCA 関数で使用す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。これらは、PBORCA.H で宣言しています。

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	132
PBORCA_COMPERR 構造体	133
EAServer へコンポーネントを配布するコールバック関数	135
PBORCA_BLDERR 構造体	136
PBORCA_LibraryDirectory のコールバック関数	137
PBORCA_DIRENTRY 構造体	138
PBORCA_ObjectQueryHierarchy のコールバック関数	139
PBORCA_HIERARCHY 構造体	140
PBORCA_ObjectQueryReference のコールバック関数	141
PBORCA_REFERENCE 構造体	142
PBORCA_ExecutableCreate のコールバック関数	143
PBORCA_LINKERR 構造体	144
PBORCA_SccSetTarget のコールバック関数	145
PBORCA_SCCSETTARGET 構造体	146

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、エラー発生の際に呼び出され、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 型ポインタ

戻り値 なし

解説 コールバック関数への記述を行います。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、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に割り当てられ、必要な形式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、全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。

そのほかの情報とコールバック関数に記述例については、23 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して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;
```

引数	説明
<i>iLevel</i>	エラーの深刻度を示す Number 型の値 0 オブジェクトやスクリプト名といったコンテキスト情報 1 CM_INFORMATION_LEVEL 2 CM_OBSOLETE_LEVEL 3 CM_WARNING_LEVEL 4 CM_ERROR_LEVEL 5 CM_FATAL_LEVEL 6 CM_DBWARNING_LEVEL
<i>lpszMessageNumber</i>	メッセージ番号の文字列値へのポインタ
<i>lpszMessageText</i>	エラー メッセージのテキスト文字列値へのポインタ
<i>iColumnNumber</i>	エラーが発生したソース コードの行文字数
<i>iLineNumber</i>	エラーが発生したソース コードの行番号

解説

一つのエラーが、幾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、オブジェクトとスクリプトのどこでエラーが発生したかをレポートします。その次に、1 つ以上のメッセージが、実際のエラーを説明します。

例えば、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 Function Source	0	0
4	null	(0002): Error C0031: シンタックス エラー	0	2
4	null	(0016): Error C0031: シンタックス エラー	0	16
4	null	(0017): Error C0031: シンタックス エラー	0	17

EAServer へコンポーネントを配布するコールバック関数

機能

EAServer へオブジェクトを配布する際にエラーが発生するたびに呼び出されます。これにより、エラーを格納し、後でそのエラーを表示することができます。

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 型ポインタ

戻り値

なし。

解説

そのほかの情報とコールバック関数の記述例については、23 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 へオブジェクトを配布することを試み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。

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	説明
<i>lpszMessageText</i>	エラー メッセージ テキストの文字列値へのポインタ

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 コールバック関数への記述を行います。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 バッファに形を合わせて納めます。

ユーザ データ バッファは呼び出しプログラム中に割り当てられ、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、全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。

そのほかの情報とコールバック関数の記述例については、[23 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へ渡します。

構文

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

メンバ	説明
<i>szComments</i>	ライブラリ中に格納してあるオブジェクトのコメント
<i>ICreateTime</i>	オブジェクトが作成された時間
<i>IEntrySize</i>	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
<i>lpszEntryName</i>	情報が返されているオブジェクトの名前
<i>otEntryType</i>	オブジェクトのデータ タイプを指定する PBORCA_TYPE のカタログデータ型の値

PBORCA_ObjectQueryHierarchy のコールバック関数

機能

検査しているオブジェクトの階層中の各先祖オブジェクトごとに呼びだされます。コールバック関数の中で、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

コールバック関数への記述を行います。通常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、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に割り当てられ、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。

そのほかの情報とコールバック関数の記述例については、23 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

検索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へ渡します。

構文

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

メンバ	説明
<i>lpszAncestorName</i>	先祖オブジェクト名へのポインタ

PBORCA_ObjectQueryReference のコールバック関数

機能

調査中オブジェクトの階層中の各参照オブジェクトごとに呼び出されます。コールバック関数の中で、参照オブジェクト名を保存し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

コールバック関数への記述を行います。通常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、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に割り当てられ、必要な形式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。

そのほかの情報とコールバック関数の記述例については、23 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 検索された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。

構文 PBORCA_ObjectQueryReference 関数は、PBORCA_REFERENCE 構造体をコールバック関数へ渡します。

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

メンバ	説明
<i>lpszLibraryName</i>	参照されるオブジェクトを含むライブラリ ファイル名の文字列値へのポインタ
<i>lpszEntryName</i>	参照されるオブジェクト名の文字列値へのポインタ
<i>otEntryType</i>	参照されるオブジェクトの型を指定する PBORCA_TYPE カタログデータ型の値

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 エラーごとに呼び出されます。

構文

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

引数	説明
PBORCA_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 コールバック関数への記述を行います。通常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、LPVOID が指し示すユーザ データ バッファにメッセージ テキストを格納します。

ユーザ データ バッファは呼び出しプログラム中に割り当てられ、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。

そのほかの情報とコールバック関数の記述例については、23 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。

構文 PBORCA_ExecutableCreate 関数は、PBORCA_LINKERR 構造体をコールバック関数へ渡します。

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

メンバ	説明
<i>lpszMessageText</i>	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ごとに一度呼び出されます。

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標準でリフレッシュされるライブラリを知ることができ、また、確実に共有ライブラリが事前のタスクで既にリフレッシュされている場合に 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完全修飾した
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	詳細
<i>lpzLibraryName</i>	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