Btrieve Classes for .NET version 9.0

プログラミングガイド

TechKnowledge

目次

製品概要

はじめに

Btrieve Classes for .NETはActian Zen / PSQL専用の.NET言語用データベース・アクセス・サポートクラスライブラリです。.NETではADO.NET/OLE　DBが標準ですがジェネリックなソフトウェア層を経由しないクラスライブラリを提供することで高いパフォーマンスを実現しています。AG-TECH社の協力・監修により最新のActian Zen仕様に準拠しています。

バージョン9.0の新機能について

当クラスライブラリバージョン9.0では以下の機能を追加いたしました。

1. Actian Zen v15 サポート
2. Visual Studio 2022 サポート
3. Windows 11サポート
4. .NET Core 3.1 ～ 6.0 サポート
5. .NET Framework 4.7.2 ～ 4.8 サポート
6. BTRVEX APIによるExtendedクラスのパフォーマンス向上
7. ストラクチャビルターのVSIX対応とEntity class生成機能追加
8. データ型 AUTOTIMESTAMP とTIMESTAMP2をサポート
9. Record クラスに GetRawValue/SetRawValue メソッドの追加
10. Extended クラスにAddFields,FillAll,GetRawValueメソッド追加
11. Blazorサンプルなどを追加
12. 機能向上と軽微な不具合の訂正

過去のバージョン履歴はAppendix-Fをご覧ください。

クラス構成について

Btrieve Classes for .NETは３系統のクラスをサポートしています。

- DDF
  .NET Frameworkの仕様に添って設計された新しいクラス群です。Record/Extended/Transaction/Exception等のクラスで構成されます。RecordクラスにはLINQサポートメソッドが利用できます。カラム等へのアクセス、データ型の変換コードもスマートに記述することができま

す。設計が新しい分、開発効率はこのクラスが優れています。

- Native
  既存のアプリケーションにDDFがない場合や、既存のBtrieve APIで作成したコードを移行したい場合等にご利用ください。ご存知のように、Btrieve API は多くのパラメーターを持ち、レコードバッファからアプリケーション データを取り出したり格納したりするにはコードが必要になるため、アプリケーション コードは煩雑になります。

- Compat
  弊社製品VBMan ActiveX Controls for Btrieveのメソッドと互換性があるメソッドを提供するクラスです。既存のVBManアプリケーションを.NET環境に少ないワークロードで移行する場合にご利用ください。エラーコード等もVBManと互換性がございます。

対応言語について

当クラスライブラリは.NET Frameworkで導入されたマネージド クラスライブラリとして構成されています。したがって、ご利用いただける言語は Visual Basic.NET/Visual C#/C++ CLI となります。今後、.NET Framework環境に対応する言語が追加された場合には、普及状況等を考慮し、順次対応していく予定です。

作成可能アプリケーション

当クラスライブラリは.NET Frameworkまたは.NET Coreを利用する以下のタイプのアプリケーションを作成できます。

- Windows デスクトップアプリケーション
- ASP.NET Webアプリケーション・Webサービス・Blazor Web アプリケーション
- コンソール・アプリケーション

DDFについて

Btrieve Classes for .NETのCompatクラスとDDF/Record/ExtendedクラスはDDF情報を基にして動作します。 DDFはZen Control Center/PSQL Control Centerでテーブルデザイナーやテーブル作成ウィザードを使って定義することが可能です。

データ型について

Btrieve Classes for .NET の Record/Extended クラスでは、Appendix-C に記載されているデータ型変換に従い、Zen/PSQLのデータ型を.NET Frameworkデータ型に変換します。データ型の変換は、基本的には .NET Framework の型変換メソッドを使用して行いますが、該当する.NET Frameworkデータ型に変換できない場合にはString型としてデータを返します。また逆に、Record クラスの .NET Framework データを Btrieve データに変換する場合、変換に失敗すると例外を発生させます。データ変換の例外が発生した場合でも、ToString メソッドを使用して .NET Framework データ型から文字列型に変換してからデータをセットすれば、変換の例外を回避できる場合があります。

Nativeクラスについては基本的な.NET Framework型とbyte配列の間ではデータ変換するメソッドが提供されますが、本来のBtrieveデータ型を保持するbyte配列と.NET Frameworkデータ型の変換はコードで記述することが必要になります。
Compatクラスでは従来どおりString型でのデータの交換が基本になります。

データサイズについて

longvarchar/longvarbinary等の型については 1 レコードに収まるデータ長を上限としてサポートします。今回のバージョンでは複数レコードにまたがりチャンクオペレーションが必要となるようなサイズのデータはサポートされません。

使用許諾権

使用権とは、お客様が 1 台のパーソナル・コンピュータ・システムでBtrieve Classes for .NETの開発環境を利用することが出来る権利です。

- Btrieve Classes for .NETの使用権はいかなる方法によっても第三者に譲渡および貸与することは出来ません。
- 使用権はBtrieve Classes for .NETをインストールしたときに発効します。
- ランタイム・モジュールのライセンス料は無料です。お客様のアプリケーションと一緒に配布可能なファイルは当マニュアルの「インストール」にあるモジュール一覧をご覧ください。
- 当製品の利用によるお客様の損失等に関しましては弊社および、販社エージーテックは一切責任を負いませんのでご了承ください。
  使用権は以下のいずれかの事由が起こった場合に消滅します。

    1. 当ソフトウェアの不正な使用により弊社に著しい損害を与える場合。
    2. 購入者が使用規定に違反した場合。
    3. プログラム・ディスク、印刷物などを使用権の範囲外の目的で複製した場合。
    4. 購入者がBtrieve Classes for .NETのユーザー登録をしない場合。
    5. 当製品をリバース・エンジニアリングの対象として利用した場合。

ユーザーサポート

本製品のユーザーサポートは、Actian Zen/PSQLの製品サポートの一部として提供されます。詳細につきましてはパッケージに添付される説明文書をご参照ください。

保証規定

当製品、および付随する著作物に対して商品性及び特定の目的への適合性などについての保証を含むいかなる保証もそれを明記するしないに関わらず提供されることはありません。

当製品の著作者及び、製造、配布に関わるいかなる者も、当ソフトウェアの不具合によって発生する損害に対する責任は、それが直接的であるか間接的であるか、必然的であるか偶発的であるかに関わらず、負わないものとします。それは、その損害の可能性について、開発会社に事前に知らされていた場合でも同様です。

販売元・ユーザーサポート

株式会社エージーテック
〒101-0054
東京都千代田区神田錦町１－２１－１
ヒューリック神田橋ビル3F

電話: 03-3293-5300
FAX: 03-3293-5270
E-Mail: info@agtech.co.jp
URL: https://www.agtech.co.jp

開発元

（株）テクナレッジ

東京都世田谷区駒沢２丁目１６番１号　サンドービル９Ｆ
電話: 03-3421-7621
FAX: 03-3421-6691
E-Mail: info@techknowledge.co.jp
Web: https://www.techknowledge.co.jp

インストール

Btrieve Classes for .NETのインストールについて説明します。

システム条件

Btrieve Classes for .NETを動作させるには、以下の前提となるソフトウェア環境が必要となります。

1. .NET Framework 4.7.2 以降、.NET Core 3.1 以降、Visual Studio 2017～2022
2. PSQL v13 R2/Actian Zen v14/Actian Zen v15

アプリケーション開発においてはBtrieveについての詳しいプログラミング情報が必要になる場合が想定されます。そのような場合はActian Zen/PSQL SDKマニュアルやサンプル・プログラムをご参照ください。

インストーラーの実行

Btrieve Classes for .NET のインストールについて説明します。ダウンロードの詳細については、『製品ガイド』を参照してください。

1. エージーテックのサイトよりインストーラー ファイルをダウンロードします。
2. ダウンロードしたファイルを実行します。
3. インストールが正常に終了するとメニューに Btrieve Classes for .NET プログラム グループが作成されます。
4. readme_jp.html ファイルにはマニュアルには記述されていない最新情報が記述されています。インストールに関する最新情報が記述される場合もありますので、必ずご一読ください。

インストールされるファイルについて

OSのインストールディレクトリを<osdir>, Btrieve Classes for .NETのインストールディレクトリを<instdir>とした場合の導入されるファイルの一覧を以下に示します。デフォルトインストールでは<instdir>は以下です。

C:¥Program Files¥techknowledge¥Btrieve Classes for .NET 9.0

お客様の作成したアプリケーションに添付して配布するモジュールは「再配布」のカラムに「可」と記述されるものに限定されます。それ以外のモジュールを配布した場合、著作権法違反となりますので十分ご注意ください。

デフォルト・パスとファイル名	内訳	再配布
<instdir>\bin\btlib.dll	.NET Framework アセンブリ	可
<instdir>\bin\BtLibCore.dll	.NET Core アセンブリ	可
<instdir>\bin\ijwhost.dll	.NET Core ホスト DLL	可
<instdir>\bin\sb.vsix	ストラクチャービルダー (x86)	不可
<instdir>\bin\sb.dll	ストラクチャービルダー	不可
<instdir>\bin\sbCore2.dll	ストラクチャービルダーコア	不可
<instdir>\bin\sasb.exe	スタンドアロン ストラクチャービルダー	不可
<instdir>\bin*.dll	VSIXサポート関連DLL	不可
<instdir>\bin64\BtLib.dll	.NET Framework x64 アセンブリ	可
<instdir>\bin64\BtLibCore.dll	.NET Core x64 アセンブリ	可
<instdir>\bin64\ijwhost.dll	.NET Core x64 ホスト DLL	可
<instdir>\bin64\sb2x64.vsix	ストラクチャービルダー (x64)	不可
<instdir>\bin64\sb2x64.dll	ストラクチャービルダー	不可
<instdir>\bin64\sbCore2_x64.dll	ストラクチャービルダーコア (x64)	不可
<instdir>\bin64*.dll	VSIXサポート関連DLL	不可
<instdir>\man\bcn900jp.pdf	PDFマニュアル	不可
<instdir>\man\readme_jp.html	お読みくださいファイル	不可
<instdir>\man\bcn9.chm	ヘルプファイル	不可
<instdir>\samples\BlazorSample.zip	C# Blazor Server サンプル	不可
<instdir>\samples\csSamp*	C# Win formsサンプル・プログラム	不可
<instdir>\samples\vbSamp*	VB.NET win formsサンプル・プログラム	不可
<instdir>\samples\csDataSetCoreSample*	C# DataSet .NET Core サンプル	不可
<instdir>\samples\csDataSetSamp*	C# DataSetサンプル	不可
<instdir>\samples\vbDataSetSamp*	VB.NET DataSetサンプル	不可
<instdir>\samples\csWebSamp*	C# Web Formsサンプル・プログラム	不可

<instdir>\samples\csNativeCoreSample*	C# Native Class .NET Core サンプル	不可
<instdir>\samples\csNativeSamp*	C# Native Classサンプル	不可
<instdir>\samples\vbWebSamp*	VB.NET Web Formsサンプル・プログラム	不可
<instdir>\samples\vbNativeSamp*	VB.NET Native Classサンプル	不可
<instdir>\samples\csWPFSample*	C# WPF .NET Core サンプル	不可

前提となるソフトウェアの詳細について

以下が本製品の前提となるソフトウェアとそのダウンロードURLです。マニュアル執筆時のもので変更になる場合があります。

Visual C++ 再領布可能パッケージ
https://aka.ms/vs/17/release/vc_redist.x86.exe
https://aka.ms/vs/17/release/vc_redist.x64.exe

.NET Framework 以下のいずれか (4.7.2 ～ 4.8)

https://dotnet.microsoft.com/en-us/download/dotnet-framework/thank-you/net472-web-installer
https://dotnet.microsoft.com/en-us/download/dotnet-framework/net48

.NET Core を利用する場合以下のいずれか (3.1 ～ 6.0)

https://download.visualstudio.microsoft.com/download/pr/8286a41b-c787-4912-a5c9-50c40eba07d0/a48056f55cb8efb2eaf2fd2e5dd5f8c6/dotnet-sdk-3.1.420-win-x86.exe

https://download.visualstudio.microsoft.com/download/pr/b0cd1e2f-13da-4573-803b-68b31618d35b/7b485cb16a59e6f2ddab0d45821ca43a/dotnet-sdk-3.1.420-win-x64.exe

https://download.visualstudio.microsoft.com/download/pr/777a3e10-4027-47f7-83cb-73271430cfc7/a555d60b3595e57a0f2b964f1331c5a0/dotnet-sdk-6.0.106-win-x86.exe

https://download.visualstudio.microsoft.com/download/pr/569408e2-e6c7-4c7c-8564-6de9daedd9d7/eef50ddcf5e30843efbab355d4b88e53/dotnet-sdk-6.0.106-win-x64.exe

Windows Serverへのインストールについて

インストーラーが前提ソフトウェアのインストールに失敗するか前提ソフトウェアのインストールを実行しない場合があります。前項で示したソフトェアをインストールしてから本製品をインストールしてくだ

さい。サポートされる開発環境 (Visual Studio 2017～2022) を事前にインストールすることで要件を満たすことが可能です。

64bitオペレーティングシステム対応について

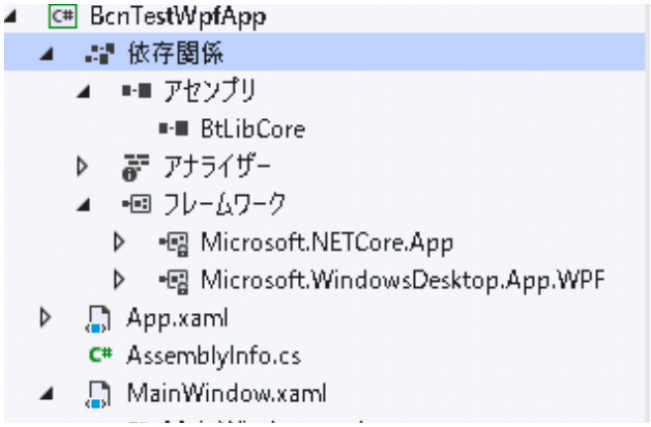
64bit版はインストールディレクトリ以下のbin64フォルダーに配布されています。

ランタイムに関しましては、Visual C++ 2015,2017,2019,2022 再領布可能パッケージ64bit版が必要になります。マイクロソフトサイトの以下からダウンロードしてインストールしてください。

https://docs.microsoft.com/ja-JP/cpp/windows/latest-supported-vc-redist?view=msvc-170#visual-studio-2015-2017-2019-and-2022

.NET Core 対応について

BtLibCore.dllは .NET Core 3.1を参照してビルドされたアセンブリです。.NET Core 3.1 ～ 6.0で動作確認しました。作成したアプリが実行時にSystem.BadImageFormatException例外が発生する場合には、CPUアーキテクチャを指定して(AnyCPUではなくx86又はx64)それに沿ったijwhost.dllを実行ファイルと同じフォルダーへ配置してください。プロジェクト依存関係は以下のようになります。Visual Studioのバージョンによって直接アセンブリを指定できないメニューが出る場合もありますがプロジェクト参照から参照マネジャーが起動できれば左の参照タブからBtLibCore.dllを指定できます。



.NET Coreの仕様により本製品のトライアル版では.NET Coreアセンブリは提供されません。

チュートリアル

ここでは、Btrieve Classes for .NETを使ったASP.NETアプリケーションの作成方法について説明します。 作成するアプリケーションは、Zen/PSQL の demodata データベースに含まれる Person テーブルを Table コントロールを使って表示する簡単なアプリケーションです。

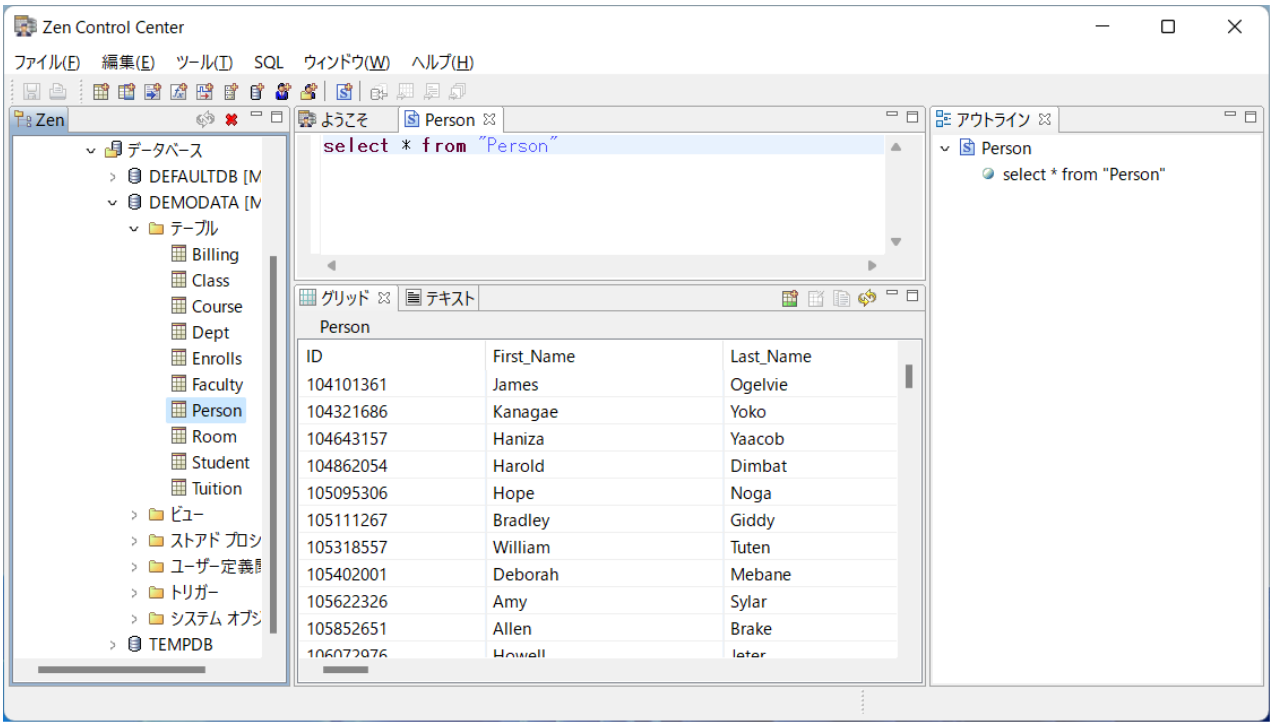
Btrieveデータベースの準備

Zen/PSQLのデータベース操作ツール「Zen/PSQL Control Center」でデータを用意します。データベースの作成方法等の詳細はZen/PSQLマニュアルをご参照ください。

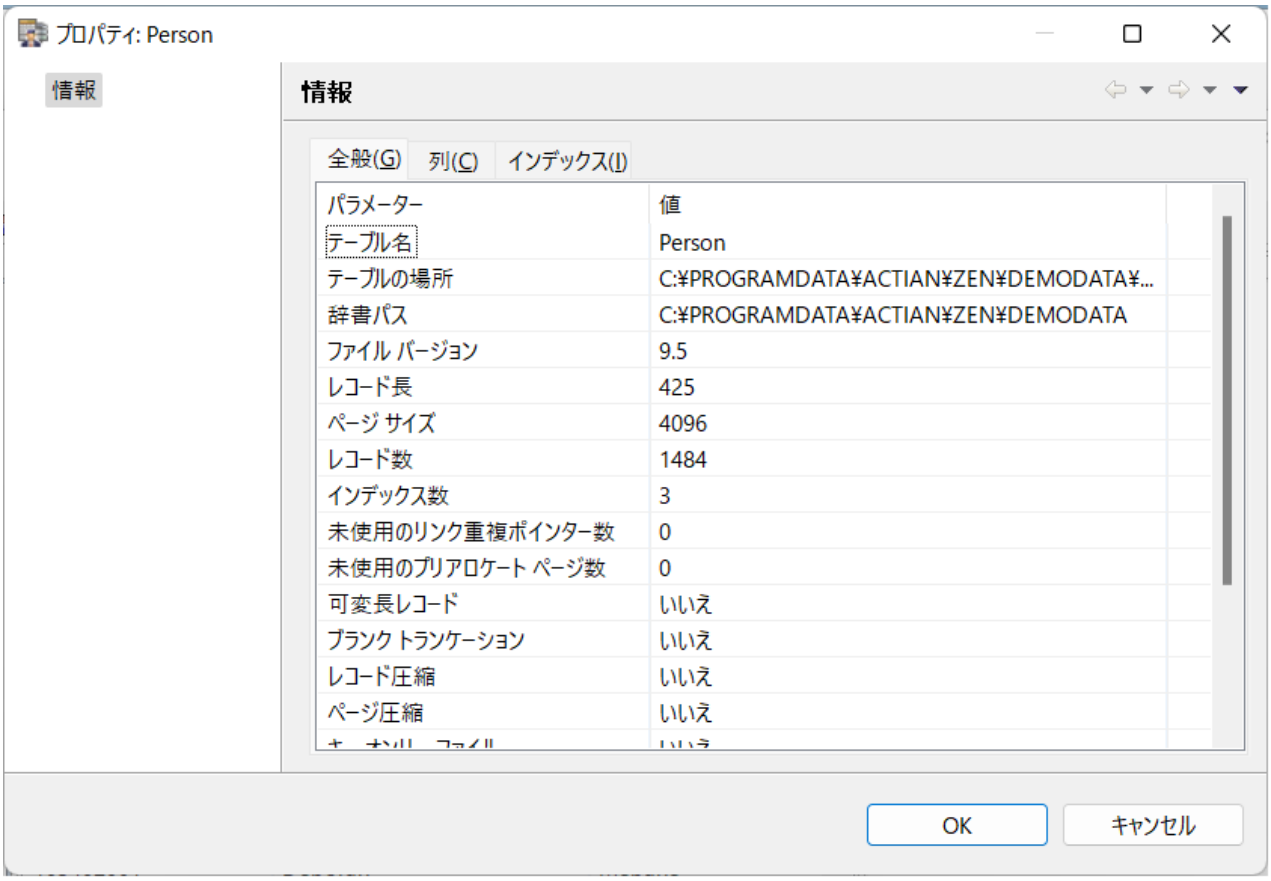
データベース作成情報で重要なのはデータベースがディスクの何処のフォルダーに作成されているかを知ることです。データベースが存在するディレクトリをDDFDirプロパティとしてプログラミング時に指定することが必要になるためです。

ここでは最初にZen/PSQLのデモ用データベースの存在位置を参照する方法を示します。

① データベース以下のDEMODATAからテーブルを選択します。左ペインに表示されるテーブル アイコンから「Person」を選択し、マウスを右クリックして「プロパティ」を選択します。



② 下図のようなプロパティ ウィンドウに表示される辞書パスが DDFDir になります。PSQL v12以降のデフォルトインストールではC:¥ProgramData¥Actian¥PSQL¥Demodataフォルダー、またはC:¥ProgramData¥Actian¥Zen¥Demodata)になります。



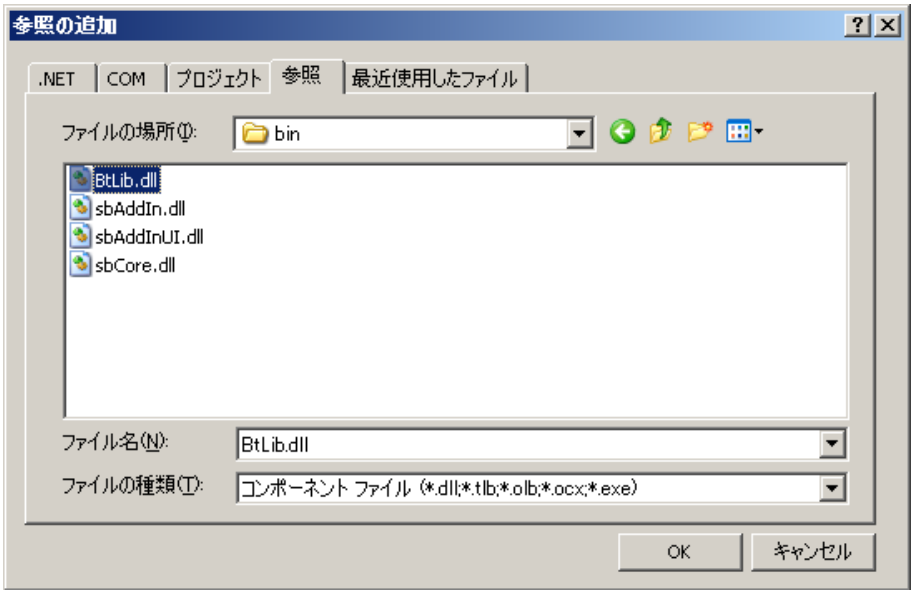
ASP.NETプロジェクトを開始

Visual Studioを起動して「新規作成」から「ウェブサイト」を選択します。ご利用になる言語を選択し、「ASP.NET Webアプリケーション」を選択します。このチュートリアルではC#言語を選択した例になります。

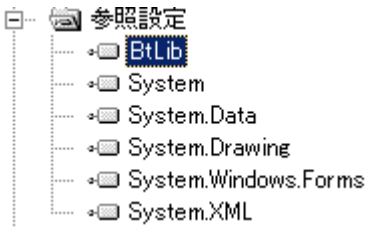
プロジェクトへの参照設定

Btrieve Classes for .NETをアプリケーションから利用するにはプロジェクトにBtLib.DLLまたはBtLibCore.DLLへの参照を追加する必要があります。参照の追加をするにはソリューション・エクスプローラータブで「参照設定」フォルダーを右クリックして、「参照の追加」を選択します。「参照の追加」

ダイアログが表示されたら、下図のように「参照」タブを開き、c:¥Program Files¥TechKnowledge¥Btreive Classes for .NET 9.0¥bin フォルダーへ移動して BtLib.DLLまたは BtLibCore.DLL を選択します。x64アプリの場合はbin64フォルダーの該当DLLを選択します。



以下は参照設定にBtLibを追加したC#プロジェクトのソリューション・エクスプローラー・タブの表示です。



Visual Basic.NETの場合はプロジェクトのプロパティから参照タブを選択すると参照設定を確認することができます。

ネームスペースの宣言

C#言語の場合

以下の行をソースファイルの先頭の宣言部に追加します。

```
using BtLib;
```

Visual Basic.NET言語

以下の行をソースファイルの先頭の宣言部に追加します。

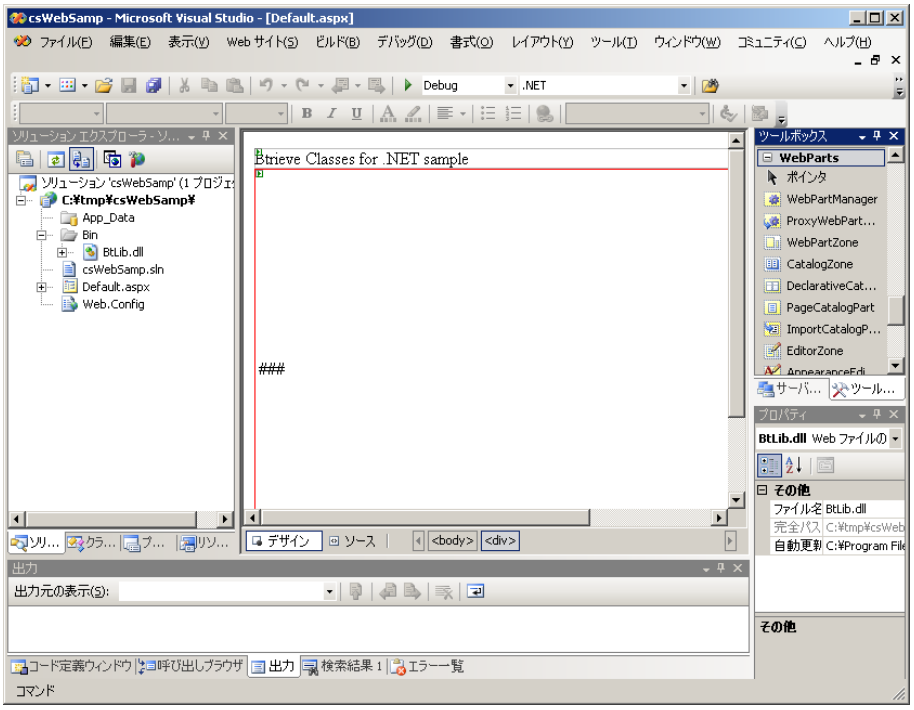
Import BtLib

WebForm設定

WebFormにデータを表示するテーブルを張ります。手順は以下のとおりです。

1. ソリューション・エクスプローラーからWebForm1.aspxを表示します。
2. ツールボックスを「Webフォーム」タブにします。
3. 「Table」をフォームにドラッグします。

上記手順後の画面は以下のようになります。



データ取得コードの作成

データをテーブルに表示するprivateメソッドを作成します。コードは以下のようになります。入力はWebFormをダブルクリックして表示されるWebForm1.aspx.csファイルに記述します。

```
private void fillTable()
{
  BtLib.Ddf demodata = new BtLib.Ddf(@"C:¥ProgramData¥Actian¥Zen¥Demodata");
  BtLib.Record person = demodata.GetRecord("Person");
  person.Open();
  short rc = person.Read(Operation.GetFirst);
  while(rc == 0)
  {
    TableRow tr = new TableRow();
    TableCell c1 = new TableCell();
    c1.Controls.Add(new LiteralControl(person["ID"].ToString()));
    TableCell c2 = new TableCell();
    c2.Controls.Add(new LiteralControl(person["First_Name"].ToString()));
    TableCell c3 = new TableCell();
    c3.Controls.Add(new LiteralControl(person["Last_Name"].ToString()));
    //
    tr.Cells.Add(c1);
    tr.Cells.Add(c2);
    tr.Cells.Add(c3);
    //
    Table1.Rows.Add(tr);
    //
    rc = person.Read(Operation.GetNext);
  }
  person.Close();
}
```

Tableコントロールはフォームの初期化時に毎回ビルドする必要がありますのでPage_Loadイベントから上記fillTableメソッドを呼び出します。

```
  private void Page_Load(object sender, System.EventArgs e)
  {
        fillTable();
  }
```

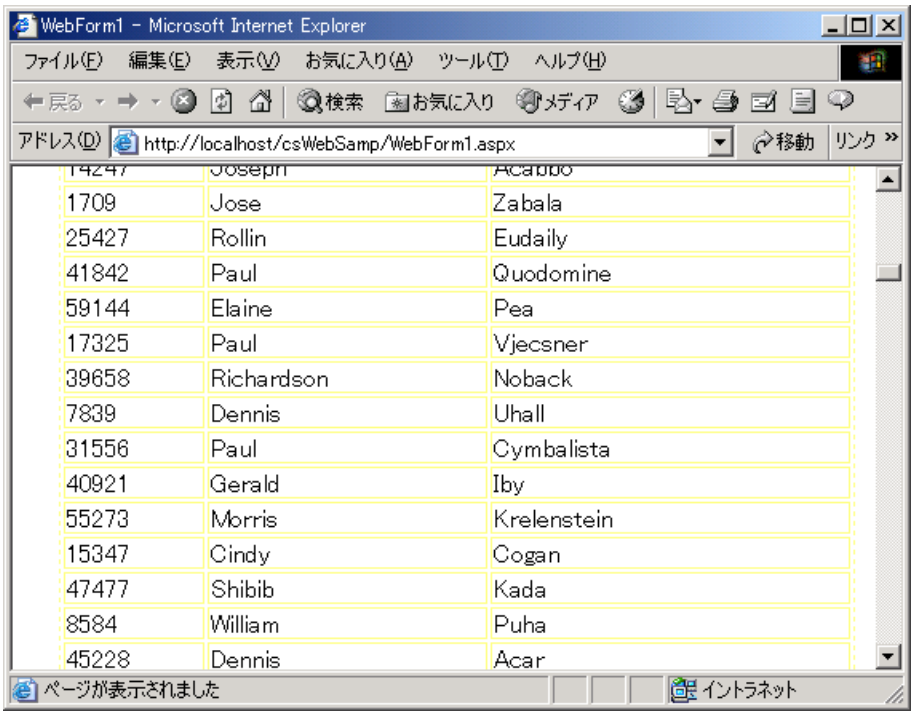
初期値ではなく、PostBack の結果としてテーブルを表示するような場合には、Page オブジェクトの IsPostBack を参照する以下のようなコードを記述します。

```
  private void Page_Load(object sender, System.EventArgs e)
  {
        if(IsPostBack)
        {
              fillTable();
        }
  }
```

尚、このコードはサンプルとして製品に収録されています。プロジェクト名はcsWebSampです。Visual Basic.NET言語のサンプルはvbWebSampです。

実行結果

プロジェクトをビルドして実行した結果は以下のようになります。



また、実行時に Open メソッドでステータス 94 の例外が発生する場合は、当マニュアルの Appendix-B の説明に従って、ASP.NET アプリケーションの実行ユーザーを変更することで対応してください。

サンプルプログラムについて

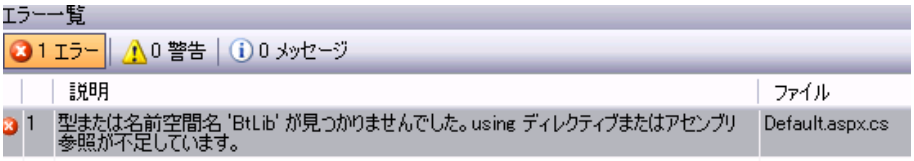
概要

サンプルプログラムを動作させる場合にはVisual Studio がインストールされていることが前提となります。サンプルプログラムは製品のメニューから選択するか、インストールディレクトリのsamplesディレクトリ以下にある*.slnファイルをエクスプローラーで選択することでVisual Studioに読み込むことができます。一部のサンプルはZIPで圧縮されていますので展開してご利用ください。

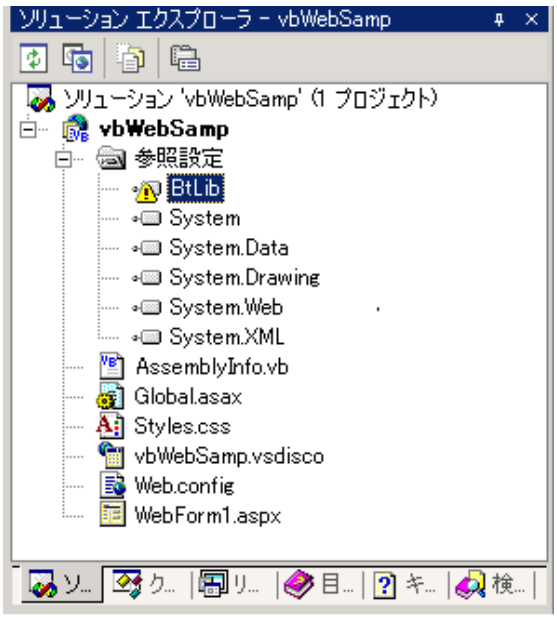
ウェブサンプルプログラムにつきましてもVisual　Studioではデバッグモードであればデフォルトでは、ASP.NET開発用のウェブサーバー(IIS Express）を起動しますのでIISへのサイトの登録なども不要です。

BtLibへの参照

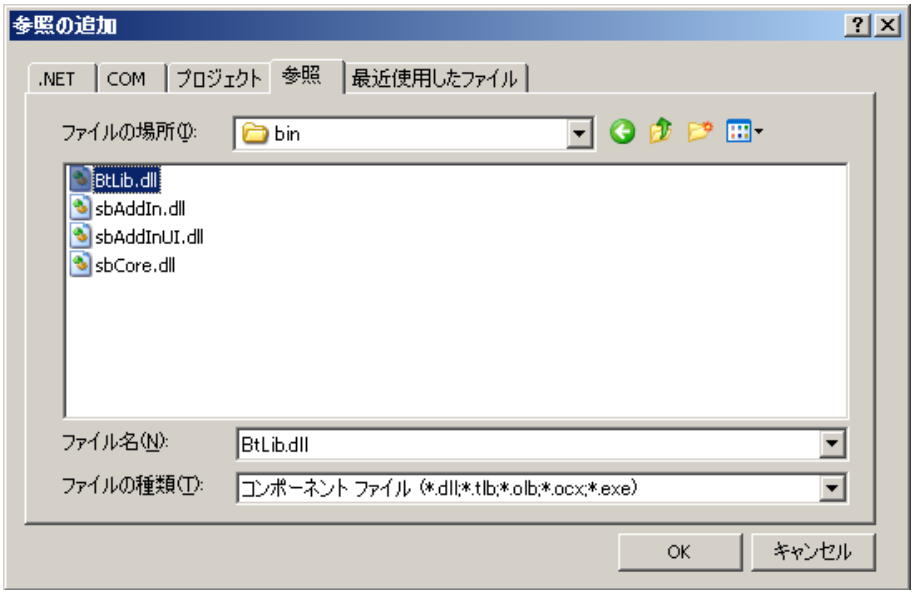
サンプル実行時、または、サンプルコンパイル時にBtLib.DLLへの参照ができない場合があります。



そのような場合には一旦BtLibへの参照を削除して再度参照を設定します。以下は参照できなくなった状態のソリューションエクスプローラ表示です。



参照設定から BtLib を選択して削除キーを押すと、参照を削除できます。この状態で「参照設定」を再度右クリックして、「参照の追加」を選択することが可能です。



上記画面で「参照」タブから、c:¥Program files¥Techknowledge¥Btrieve Classes for .NET 9.0\binフォルダーにあるBtLib.dllを指定します。CPUアークテクチャx64のプログラムをビルドする場合は同フォルダーのc:¥Program files¥Techknowledge¥Btrieve Classes for .NET 9.0¥bin64にあるBtLib.dllを指定します。.NET Coreアプリの場合はBtLibCore.dllを指定します。

64bit OSでのサンプルプログラム実行について

デスクトップ向けサンプルプログラムは基本的に32bitモードで動作するように調整されています。64bit OS上で64bitモードでサンプルプログラムを動作させる場合には以下の手順に従ってください。

1. Visual Studio で読み込んだサンプルプロジェクトのターゲットプラットフォームをx64を追加・変更します。
2. プログラムの参照設定でBtLib.dllまたはBtLibCore.dll 32bit版への参照を一旦削除します。
3. プログラムの参照設定bin64¥BtLib.dllまたはbin64¥BtLibCore.dllを追加します。
4. プログラムをビルドして実行します。

Visual Studio 2017-2019でのWebサンプルプログラム実行について

Visual Studio 2022からIDEが64bitに変更されているため、Webサンプルプログラムの設定はx64になっています。Visual Studio 2017/2019でWebサンプルプログラムを実行すると System.BadImageFormatExceptionが発生します。正常動作のためにはVisual Studioのメニューから [ツール]-[オプション]を開き[プロジェクトおよびソリューション] - [Web プロジェクト]内の[Web サイトおよびプロジェクト用 IIS Express の64ビットバージョンを使用]をチェックして下さい。

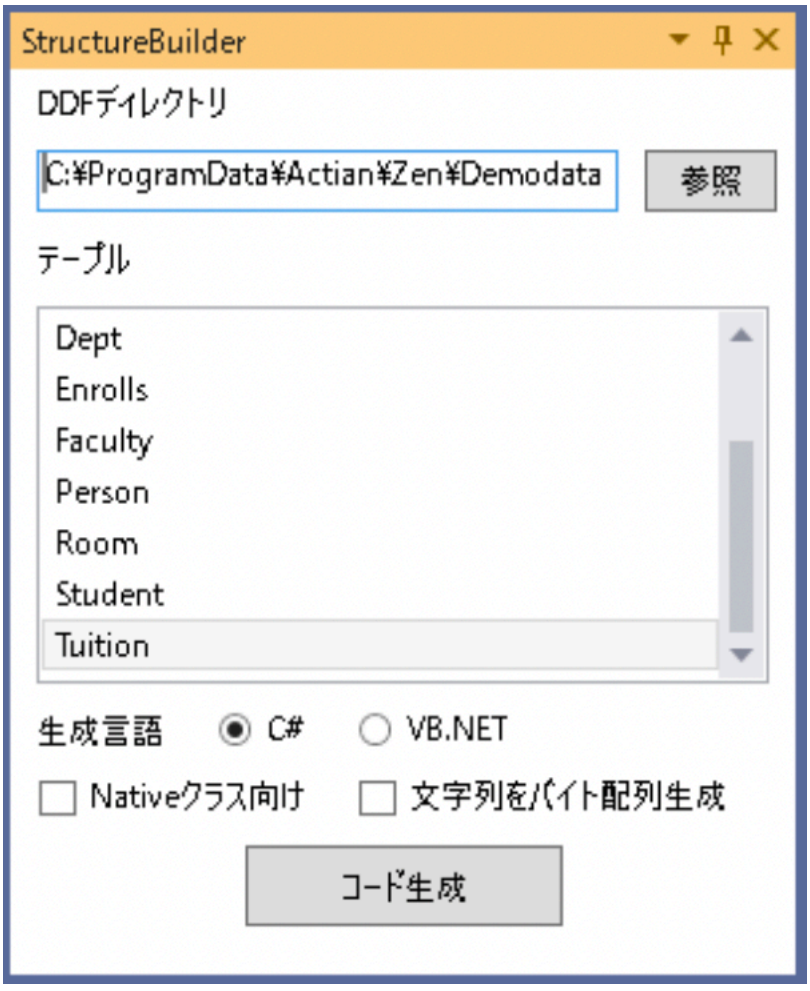
ストラクチャービルダー

概要

Microsoft.NETではVisual Basic 6.0で不可能だった構造体のアライメントを指定し、構造体の任意の位置にデータ領域を設定できるようになりました。この機能を使ってNative class のデータ交換構造体を定義する場合には各フィールドの属性を指定する必要があるデータ型が多数存在します。テーブル定義を見ながらこの構造体を定義することはカラム数の多いテーブルでは非常に時間と根気の必要な作業になります。

Btrieve Classes for .NETではVisual Studio用のVXISアドインとしてこの属性付の構造体定義をDDFから自動生成してソースコードの任意の位置に追加するストラクチャービルダーを提供しています。以下はストラクチャービルダーのサンプル画面です。DDFディレクトリにはZen/PSQLでサポートされるデータベースURIを指定することができます。

今回のVSIXバージョンからRecordクラスで利用できるEntity Classを生成する機能を追加しました。



ストラクチャービルダーインストール

Visual Studioが起動されていない状態で当製品のメニューから「Structure Builderインストール(x86)」を選択します。Visual Studio 2022をご利用の場合は「Structure Builder インストール(x64)」を選択します。

ストラクチャービルダー起動方法

Visual Studioの「表示」メニューから「その他のウィンドウ」から「ストラクチャービルダー」選択します。表示された画面はVisual Studioのウィンドウにドッキング可能です。

構造体の挿入方法

以下の手順でターゲットとなるソース・コードに構造体定義を挿入します。

1. DDFが存在するディレクトリを「参照ボタン」を押して指定します。セキュアデータベースをご利用の場合はデータベースURIを指定します。
2. 言語をクリックして選択します。
3. テーブル一覧リスト・ボックスにテーブル名が表示されるのでテーブルを選択します。
4. ソースコード上の挿入位置にカーソルを置きます。
5. コード生成ボタンをクリックします。
6. ソースコードに構造体が挿入されます。このときクリップボードにも同じ内容がコピーされます。それ以前のクリップボードの内容は破棄されますのでご注意ください。

ストラクチャービルダーの制約事項

1. 可変長データには対応していません。
2. ターゲット言語で予約語となるカラム名が存在する場合には生成した構造体のカラム名を予約語以外の文字列に変換する必要があります。
3. .NETに存在しないBtrieveデータ型はByte型に変換されます。Byte 型から .NET データ型への変換は、Native クラスの変換メソッドを使って行えます。
4. 「文字列をバイト配列生成」チェックはNativeクラスの生成時のみ有効です。

文字列データに対する変数型の選択について

例えば、C# 言語で構造体を以下のように宣言したとします。

```
[StructLayout(LayoutKind.Sequential,Pack=1,CharSet=CharSet.Ansi)]
public class PrimaryKey
{
  public byte null_flag1;
  [MarshalAs(UnmanagedType.ByValTStr,SizeConst=25)]
  public string Building_Name;
  public byte null_flag2;
  public Int32 Number;
}
```

Btrieveの文字列型データは後続がスペースで埋まる仕様なので、実行時コードで25バイトの領域を指定しようとして以下のようにコードしたと仮定します。

```
PrimaryKey pk = new PrimaryKey();
pk.Building_Name = "Eldridge Building        "; // 25 bytes string
```

ところが実際にはBuilding_Nameで確保された領域には24バイトだけセットされ最後の１バイトにはバイナリのヌルがセットされます。これはMicrosoft .NETのSystem.Runtime.InteropServicesの仕様と判断できます。一応この状況に対応するため強制的に文字列領域をBtrieve仕様に合わせるメソッドをNative Classに追加してあります。以下のサンプルコードでは3行目のFixString呼び出しです。

```
System.IntPtr keyPtr = Marshal.AllocCoTaskMem(Marshal.SizeOf(pk));
Marshal.StructureToPtr(pk,keyPtr,true);
Native.FixString(keyPtr,1,25);    //文字列をBtrieve仕様に！
```

この方法ですと簡単に文字列を指定出来ることは良いのですがIntPtr上で文字列のオフセットを渡す必要があり、プログラムのメンテナンス性がよくない場合もあります。

次にこのFixStringメソッドを使わないでデータベース上のString型の領域でもByte型で宣言して対応する方法を示します。構造体の定義は以下のようになります。

```
[StructLayout(LayoutKind.Sequential,Pack=1,CharSet=CharSet.Ansi)]
public class PrimaryKey
{
  public byte null_flag1;
   [MarshalAs(UnmanagedType.ByValArray,SizeConst=25,ArraySubType=UnmanagedType.U1)]
  public byte [] Building_Name = new byte[25];
  public byte null_flag2;
  public Int32 Number;
}
```

文字列をBuilding_Nameメンバ領域にセットするコード例は以下になります。

```
// get shift jis encoder
Encoding sje = Encoding.GetEncoding("shift-jis");
PrimaryKey pk = new primaryKey();
string s = "Eldridge Building         ";
pk.Building_Name = sje.GetBytes(s);
```

どちらのコード方法でも同じアンマネージドなデータ領域を作成することができますので、お客様の状況に合わせてコード方法をご選択ください。 文字列を Byte 配列で定義する構造体を出力したい場合には、ストラクチャービルダーの「文字列型をバイト配列で定義」オプションをオンにしてください。

Unicodeデータ型カラムの処理について

Unicode のデータ型の文字列型 NCHAR と NVARCHAR を使用する場合、ストラクチャービルダーによって生成されたテーブルストラクチャーの一部を手作業で修正する必要があります。

ストラクチャービルダーは文字列型を定義する2つのモードがあります。どちらのモードが使用されるかは"文字列型をバイト配列で定義"オプションによって決定されます。このオプションがオフの場合、ストラクチャービルダーはすべての文字列を ANSI 文字列として CharSet.Ansi に定義します。

これは、Unicode 文字列に不適切な設定です。テーブル内の文字列が Unicode のみの場合と Unicode と ANSI が混在する場合で次のような別々の修正が必要になります。

■ テーブル内のすべての文字列が Unicode の場合

1. ストラクチャーの CharSet 設定を CharSet.Ansi から CharSet.Unicode に変更する。
2. 各 Unicode 文字列フィールドの SizeConst = の値を 1/2 にした値に変更する。

■ Unicode と ANSIの文字列が混在する場合

1. CharSet.Ansi を使用して ANSI 文字列はそのまま文字列型で扱います。
2. Unicode 文字列をバイト配列として定義します。
3. Encoding.Unicode.GetString()メソッドで、そのバイト配列からUnicode文字列に変換して扱います。

混在する場合の修正を具体的に説明します。

次のSQLで定義された ANSI と Unicode の文字列が混在するテーブルがあるとします。

create table v12newbcntest (chartest char(100) NOT NULL , nchartest NCHAR(100) NOT NULL);

最初に、”文字列型をバイト配列で定義"オプションをオフにして、ストラクチャービルダーでこのテーブルの次のストラクチャーを生成します。

```
[StructLayout(LayoutKind.Sequential, Pack = 1, CharSet = CharSet.Ansi)]
public class v12newbcntest
{
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 100)]
  public string chartest;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 200)]
  public string nchartest;
}
```

このストラクチャーでは、Unicode のデータ型のフィールドは Charset.ANSI として定義されているため、その中のデータが破損します。これを回避するには、Unicode 文字列をバイト配列として再定義する必要があります。

文字列型をバイト配列で定義”オプションをオンにして、ストラクチャービルダーでテーブルのストラクチャーを再び生成すると、下記のストラクチャーが生成されます。

```
[StructLayout(LayoutKind.Sequential, Pack = 1, CharSet = CharSet.Ansi)]
public class v12newbcntest
{
  [MarshalAs(UnmanagedType.ByValArray, SizeConst = 100, ArraySubType =
UnmanagedType.U1)]
  public byte[] chartest = new byte[100];
  [MarshalAs(UnmanagedType.ByValArray, SizeConst = 200, ArraySubType =
UnmanagedType.U1)]
  public byte[] nchartest = new byte[200];
}
```

このストラクチャーは、Unicode 型だけでなく、ANSI 型もバイト配列で扱うことになるため、CharSet.Ansi を指定しているのに文字列型として扱えず、使い勝手が悪くなります。下記のように ANSI 型は上記の 1つめのストラクチャーのように文字列型、Unicode 型は 2つめのストラクチャーのバイト配列として扱うように手作業でマージすることで、Unicode 型と ANSI 型の両方を 1つのストラクチャーで扱うことができます。

```
[StructLayout(LayoutKind.Sequential, Pack = 1, CharSet = CharSet.Ansi)]
public class v12newbcntest
```

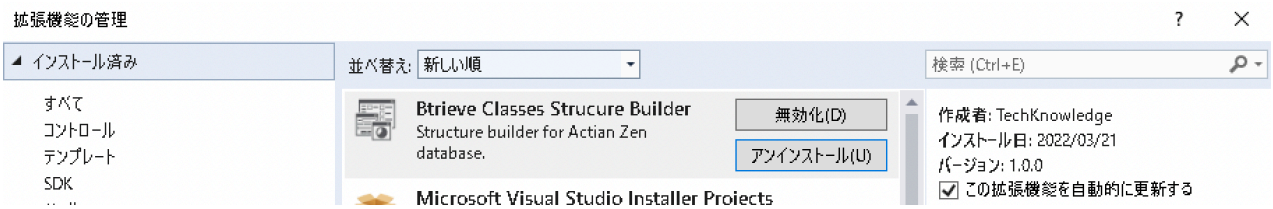
```
{
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 100)]
  public string chartest;
  [MarshalAs(UnmanagedType.ByValArray, SizeConst = 200, ArraySubType =
UnmanagedType.U1)]
  public byte[] nchartest = new byte[200];
}
```

プログラムの中でこのようなフィールドは、次の C＃の例のように使用することができます。

```
static v12newbcntest testv12Mstruct = new v12newbcntest();
string str = testv12Mstruct.chartest + Encoding.Unicode.GetString(testv12Mstruct.nchartest);
```

ストラクチャービルダーのアンインストール方法

Visual Studio の「拡張機能」メニューから「インストール済み」タブを選択しストラクチャービルダーを選択してアンインストールボタンをクリックしてください。プログラムと設定からBtrieve Classes 9.0をアンインストールする場合は事前にストラクチャービルダーVSIXを削除してください。



スタンドアロンストラクチャービルダー

概要

単体アプリとして実行可能なスタンドアロンストラクチャービルダーを同梱しています。基本機能はアドイン版と同じです。

スタンドアロンストラクチャービルダーの起動

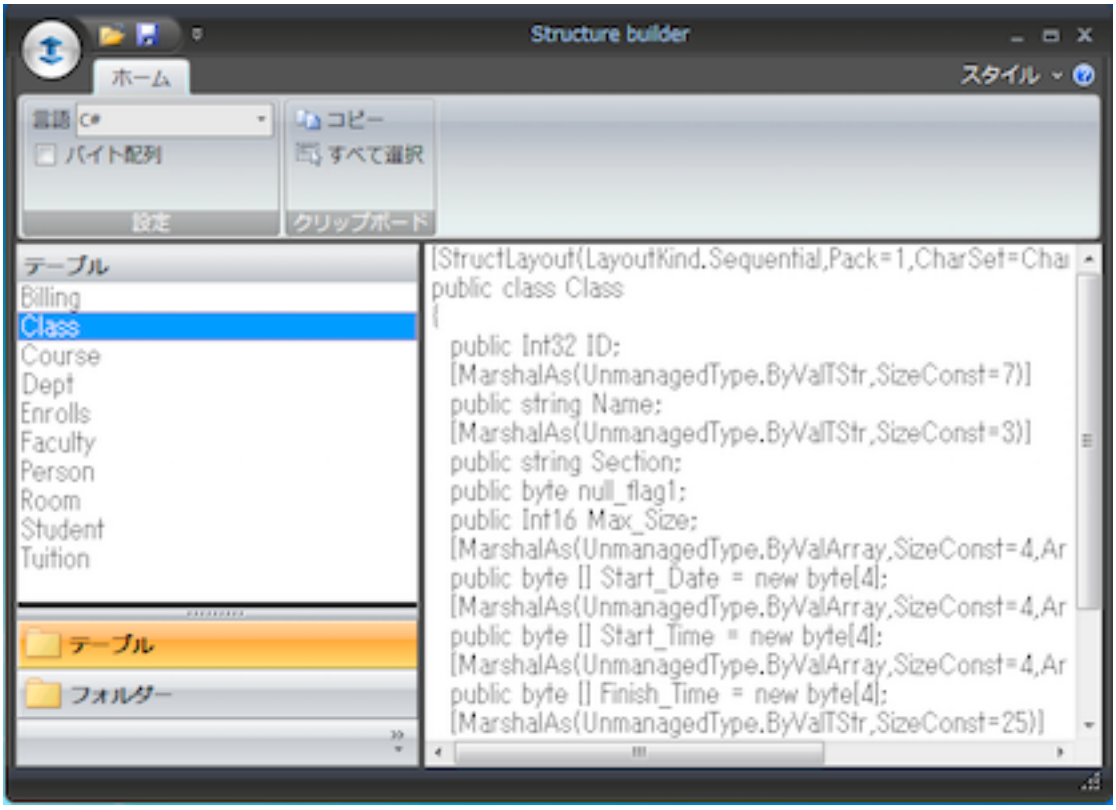
WindowsメニューのBtrieve　Classesメニュー内にあるストラクチャービルダーを選択することで起動できます。

利用方法

1. DDFファイルをリボンメニューから開きます。

2. 左パネルにDDF内に定義されるテーブル名のリストが表示されます。

3. ご利用の言語をリボンメニューから選択します。デフォルトはC#です。
4. テーブル名のリストから構造体を生成したいテーブルをマウスクリックで選択します。
5. 右パネルに生成された構造体が表示されます。



6. リボンメニューからコピーをクリックするとクリップボードに構造体がコピーされますので、Visual Studio の該当ソースにペーストしてください。ファイルメニューから言語ソースとして保存することもできます。

注意事項

1. 左ベインのフォルダーリストからDDFのあるフォルダーを選択するとDDFを読み込みます。このフォルダーリストでは隠し属性のフォルダーを扱うことはできません。

クラス・ライブラリ・リファレンス

Ddf Class

概要

DDFを指定しデータベース情報を得てレコードを管理するクラスです。

コンストラクタ

Ddf()
Ddf(short ClientID)
Ddf(DDFDir As String)
Ddf(DDFDir As String, short ClientID)
Ddf(DDFDir As String, MetadataVersion ver)
Ddf(DDFDir As String, MetadataVersion ver, short ClientID)
Ddf(DDFDir As String, ConnectionString As String)
Ddf(DDFDir As String, ConnectionString As String, short ClientID)
Ddf(DDFDir As String, ConnectionString As String, MetadataVersion ver)
Ddf(DDFDir As String, ConnectionString As String, MetadataVersion ver, short ClientID)

概要

データベースの DDF へのパスをパラメータとして指定する形式では、Ddf オブジェクトの初期化後、DDF が読み込まれ、その情報がオブジェクトの中に展開され保持されます。 Zen/PSQLのセキュアデータベースを扱う場合にはパラメータとしてコネクションストリングを指定することができます。コネクションストリングはbtrv://で始まるデータベースURI文字列を指定します。データベースURIの詳細についてはZen/PSQLマニュアル等をご参照ください。このコネクションストリングには&table指定、&dbfile指定は含めることができませんのでご注意ください。

DDF パスを指定しないコンストラクタは、Ddf オブジェクトの初期化のみを実行します。この場合は、DDFDir プロパティを指定して Load() メソッドを呼び出すことにより、データベース情報を Ddf オブジェクトに読込みます。

DDFのパスに”Btrv://”プリフィックスがある場合はログインオペレーションがコンストラクタ内で実行されます。ログイン状態はDDFのインスタンスが廃棄されるときに解除されます。LogOutメソッドでもログイン状態を解除できます。明視的にインスタンスを廃棄する場合にはプログラムの終了時などにDisposeメソッドを呼び出してください。

各コンストラクタの最終パラメータにてZen/PSQLメタデータバージョンを指定する事ができます。

各コンストラクタの最後にClient IDが指定可能となっています。Client IDを指定するコンストラクタを使った場合は指定したDDF利用中のBtrieve API呼出しにClient IDを含むメモリエリアを使用します。

プロパティ

DDFDir

データ型
String

概要
DDFファイルが存在するディレクトリを指定します。

FloatSize

データ型
Int16

概要
Float、Bfloat型のフィールドを文字列に展開する際の浮動小数点の桁数。この値の設定が小さい場合、4バイトのFloat型では桁落ちが発生する場合がありますので適当なサイズを設定してください。デフォルトは10桁です。最大は20桁です。変換の精度はマイクロソフトのコンパイラのランタイム・ライブラリに依存しています。

FillSpace

データ型
bool

概要
Btrieve String型のデータを取得する場合、後続するスペースの処理を指定します。trueの場合は後続するスペースがついたままになります。デフォルトはfalse設定で後続スペースは取り除かれます。

OwnerName

データ型
String

概要
DDFファイルのオーナー名を指定します。

SignNibble

データ型
Int16

概要
DECIMAL型、MONEY型の正数値を表すニブル値を設定します。設定可能な値は12または15です。デフォルト値は15となります。

Version

データ型
MetadataVersion

概要
Zen/PSQLデータベースのメタデータバージョンを指定します。ver2 を指定した場合はPVFILE.DDF 等のあるディレクトリをDDFDirとして指定します。

メソッド

GetRecord

書式
Record GetRecord(string TableName)

概要
パラメータで指定したテーブルに関連するレコードオブジクトを取得します。

Load

書式
void Load()

概要
DDFDirプロパティで指定されるデータベース情報をロードします。

LogIn

書式
void LogIn(ConnectionString As String)

概要
Pervasive.SQL V8.6からサポートされたセキュアデータベースにログインします。指定するパラメータはbtrv:から始まるデータベースURIです。データベースURIの詳細についてはZen/PSQLマニュアル等を参照してください。以下はコード例です。

Ddf.LogIn(“btrv://rasta@jamaica /demodata?pwd=reggae”)

LogOut

書式
void LogOut()

概要
Pervasive.SQL V8.6からサポートされたセキュアデータベースに接続している場合にデータベースからログアウトします。

Unload

書式
void Unload()

概要
読み込まれたDDF情報を解放します。Ddfオブジェクトは初期状態に戻ります。解放後はDdfクラスから得られたRecordオブジェクトは無効になります。

Extended Class

概要

Zen/PSQL/BtrieveのExtended系オペレーションを実行するクラスです。Extended系オペレーションはレコードの一部を取得してデータ転送量を抑えることが出来るため、非常に高速なデータ取得方法として知られています。また検索条件の設定により該当するデータをサーバー側で選択しクライアントに転送することも可能です。パフォーマンスやネットワークトラフィックでは非常に有利なExteded系オペレーションですが、唯一の欠点はフィールドのオフセットや長さや検索情報をセットする構造体が多く、検索結果の取得方法等、プログラミングが複雑になることです。構造体のアライメントが1バイト単位に簡単にセットできない言語を使っている場合はさらにトリッキーな手法を導入する必要があります。このクラスを使用すると、DDF から構造体情報を取得し、コレクションを使ってフィールドを指定するシンプルなメソッドを用いて、extended 系メソッドを簡単に実行できます。

以下はExtendedクラスを利用したサンプル・コードです。Zen/PSQLのdemodataにあるPersonテーブルからFirst_Nameフィールドを抽出しています。

```
BtLib.Ddf ddf = new BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata");
BtLib.Record r = ddf.GetRecord("Person");
r.Open();

r.Index = "ID";
BtLib.Extended ex = r.GetExtended(100,32767);
ex.SearchCond = "@First_Name > Geroge";
ex.AddFields(new string[] {“First_Name”,”Last_Name”});
rc = r.Read(BtLib.Operation.GetFirst);

while(rc != 9)
{
  rc = ex.Read(BtLib.Operation.GetNextExtended);
  if( ex.ResultCount == 0)
    break;
  for(i=0; i < ex.ResultCount; i++)
  {
    tmp = ex["First_Name"].ToString() + ex["Last_Name"].ToString();
    listBox1.Items.Add(tmp);
    ex.MoveNext();
  }
}
r.Close();
```

またExtendedクラスは.NET FrameworkのSystem.Data.DataSetオブジェクトを簡単に生成できます。DataSet オブジェクトを使用すると、Visual Studio で提供される DataGrid 等のデータ　バウンド コン

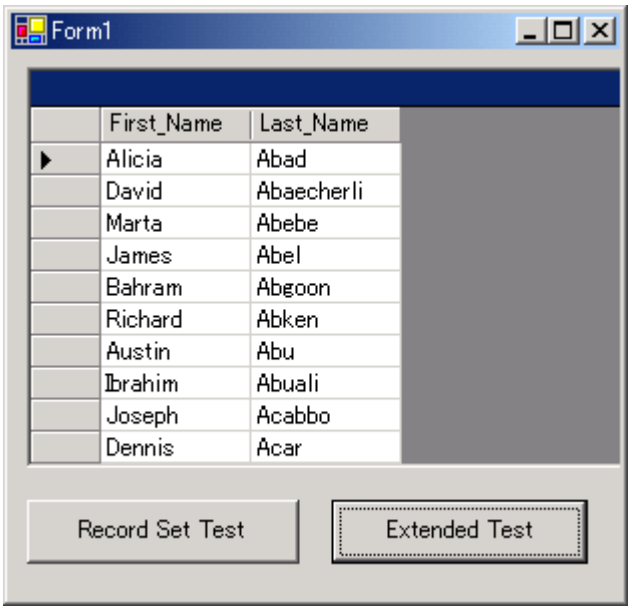
トロールに簡単にデータを連結できます。以下はDataSetオブジェクトを生成し、DataGrid(インスタンス名はdataGrid1)に連結するサンプル・コードです。

```
short rc;
BtLib.Ddf ddf = new BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata");
BtLib.Record r = ddf.GetRecord("Person");
r.Open();

r.Index = "Names";
BtLib.Extended ex = r.GetExtended(100,32767);
ex.SearchCond = "@First_Name > Adachi";
ex.AddFields(new string[] {“First_Name”,”Last_Name”);
DataSet ds = ex.GetDataSet();
rc = r.Read(BtLib.Operation.GetFirst);
do
{
   rc = ex.Read(BtLib.Operation.GetNextExtended);
   if( ex.ResultCount > 0)
   {
       ex.Fill(ds);
   }
} while( rc != 9 );
r.Close();

 dataGrid1.SetDataBinding(ds,"person");
```

上記コードのWindows formでの実行結果は以下のようになります。



コンストラクタ

Extended();

概要
当クラスはRecordクラスのGetExtendedメソッドを使ってインスタンスを生成してください。当オブジェクトのインスタンスをnewで生成する必要はありません。

プロパティ・リファレンス

IgnoreCase

データ型
bool

概要
SearchCondition プロパティで指定した検索条件をデータと照合する際、この設定が true の場合はケース（大文字/小文字）を無視して検索します。 この設定は英文字のみに有効です。

Index

データ型
String

概要
GetNextExtended/GetPreviousExtendedをReadメソッドで実行する際に採用されるインデックス名を設定します。

Lock

データ型
LockBias

概要
Extended系オペレーション実行によりレコードロック制御が必要な場合には当プロパティにロック・バイアス値を設定します。

MaxRecords

データ型
short

概要
抽出レコードの最大数を指定します。

Mode

データ型
ExtendedMode

概要
ReadメソッドまたはStepメソッドの検索結果にカレントレコードを含める場合はExtendedMode.UCを指定します。含めない場合はExtendedMode.EGを指定します。

ResultCount

データ型
short

概要
抽出データ数を保持します。

SearchCond

データ型
string

概要
レコード抽出条件を設定します。単一の文字列に@の後にフィールド名、比較演算子、値の順に指定します。比較演算子は以下を指定できます。

=	等しい
>	より大きい
<	より小さい
<>	等しくない
>=	より大きいか等しい
<=	より小さいか等しい

複合検索をする場合は検索条件を＆（AND)または｜(OR)で結合します。比較対象を即値で指定する場合は、スペースが区切り文字になります。スペースが含まれる即値を指定する場合は一重引用符、二重引用符、スラッシュで文字列を囲みます。以下に検索文字列の例を示します。

@income >= 100 & @income <= 1000
@製品 = “PSQL” | @製品 = “Btrieve”
@maker <> /Microsoft/ & @maker <> /Apple/

TIMESTAMP,TIMESTAMP2,AUTOTIMESAMP型のカラムに対しては8バイト整数で値を指定できます。

SkipRecords

データ型
short

概要
レコード抽出条件に合致しないレコード最大数を設定します。

AddField

書式
void AddField(string ColName);

概要
Extendedオペレーションで抽出するカラムを指定します。

パラメータ
抽出するカラム名。

戻り値
なし。

AddFields

書式
void AddFields(string [] ColNames);

概要
Extendedオペレーションで抽出するカラムを文字列配列で指定します。

パラメータ
抽出するカラム名の配列。

戻り値
なし。

ClearField

書式
void ClearField();

概要
AddFieldメソッドで指定した抽出対象となるカラムをすべて削除します。

戻り値
なし。

Fill

書式
void Fill(DataSet ds);

概要
ReadまたはStepの結果をパラメータで指定したDataSetオブジェクトに追加します。

戻り値
なし。

FillAll

書式
short FillAll(DataSet ds);

概要
現在のレコードポジションからGET NEXT EXTENDEDを実行し結果をDataSetオブジェクトに追加します。

戻り値
Btrieve ステータスコード

サンプルコード

```
// dgv はDataGridView
try
{
  short rc;
  BtLib.Ddf ddf = new BtLib.Ddf(dsn);
  BtLib.Record r = ddf.GetRecord("Person");
  r.Open();
  r.Index = "ID";
  rc = r.Read(BtLib.Operation.GetFirst);

  BtLib.Extended ex = r.GetExtended(100,32767);
  ex.AddFields(new string[] { "ID", "First_Name", "Last_Name" });
```

```
    DataSet ds = ex.GetDataSet();
    rc = ex.FillAll(ds);
    dgv.DataSource = ds.Tables[0];
    r.Close();
    ddf.LogOut();
  }
  catch (System.Exception er)
  {
    System.Diagnostics.Debug.WriteLine(er.ToString());
  }
```

GetData

書式
void GetData(IntPtr pStructure);

概要
IntPtr でポイントされるメモリ領域に、拡張オペレーションで抽出したフィールド データ領域（先頭から 6 バイト目以降）を転送します。この領域に構造体を定義することで容易にアプリケーションから抽出したフィールドデータを扱うことが出来るようになります。定義する構造体はメモリアライメントを考慮して、抽出するカラムの領域のみ定義する必要があります。

パラメータ
拡張オペレーションで抽出したデータを格納するIntPtrでポイントされるメモリ領域。

戻り値
なし

GetDataSet

書式
DataSet GetDataSet();

概要
FillAll/Read/Stepの結果を保持するためのDataSetオブジェクトを作成して返します。

戻り値
DataSetオブジェクト

GetDataTable

書式
DataSet GetDataTable ();

概要
Extended オブジェトにセットされた抽出カラム情報に基づいて、DataTable オブジェクトを返します。

戻り値
DataTableオブジェクト

GetRawValue

書式
object GetRawValue(String columnName);

概要
Extendedオペレーション実行結果の確定した位置から指定されたカラムの生の値を返します。.NET の DateTime型への変換で桁落ちするTIMESTAMP2, AUTOTIMESTAMP 型を補完します。これらの型の戻り値はULong64になります。

戻り値
カラムデータオブジェクト

MoveFirst

書式
void MoveFirst();

概要
Extendedオペレーション実行結果の先頭データに移動します。移動が成功した場合にはExtendedオブジェクト・コレクションに抽出したデータの先頭の値をセットします。

戻り値
なし。

MoveLast

書式
void MoveLast();

概要
Extendedオペレーション実行結果の最終データに移動します。移動が成功した場合にはExtendedオブジェクト・コレクションに抽出したデータの値をセットします。

戻り値
なし。

MoveNext

書式
void MoveNext();

概要
Extendedオペレーション実行結果の次データに移動します。移動が成功した場合にはExtendedオブジェクト・コレクションに抽出したデータの値をセットします。

戻り値
なし。

MoveTo

書式
void MoveTo(int Index);

概要
Extendedオペレーション実行結果データの指定行に移動します。移動が成功した場合にはExtendedオブジェクト・コレクションに抽出したデータの値をセットします。

パラメータ
抽出結果の指定行。0ベースで指定します。

戻り値
なし。

Read

書式
short Read(Operation Op);

概要
キーに関連するExtendedオペレーションを実行します。指定できるオペレーションは
Operation.GetNextExtended または Operation.GetNextPreviousです。

パラメータ
Btrieveオペレーション・コードを指定します。

戻り値
Btrieveステータスコード。

RecordExsists

書式
bool RecordExsists(short value);

概要
Extendedオペレーションの実行結果ステータスを判断して検索結果レコードが存在するステータス・
コードの場合はtrueを返します。Falseの場合は検索結果が存在しないと判断できます。

パラメータ
Extended系オペレーション実行メソッドReadからのBtrieveステータス・コードを指定します。

戻り値
bool

Step

書式
short Step(Operation Op);

概要
Step系Extendedオペレーションを実行します。指定できるオペレーションは
Operation.StepNextExtended, Operation.StepPreviousExtendedです。

パラメータ
Btrieveオペレーション・コードを指定します。

戻り値
Btrieveステータスコード。

サンプル・コード

```
short rc;
string tmp;
int i;

listBox1.Items.Clear();
BtLib.Ddf ddf = new BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata");
BtLib.Record r = ddf.GetRecord("test");
r.Open();
BtLib.Extended ex = r.GetExtended(100,32767);
ex.SearchCond = "@ID > 2";
ex.AddFields(new string[] {“ID”, “dt”});
rc = r.Step(BtLib.Operation.StepFirst);
while(rc != 9)
{
  rc = ex.Step(BtLib.Operation.StepNextExtended);
  if( ex.ResultCount == 0)
    break;
  for(i=0; i < ex.ResultCount; i++)
  {
    tmp = ex["ID"].ToString() + " " + ex["dt"].ToString();
    listBox1.Items.Add(tmp);
    ex.MoveNext();
  }
}
r.Close();
```

Exception Class

概要
当クラスはSystem.Exceptionから導出されており、Btireve Classes for .NETの以下のクラスについて当クラスライブラリのエラー時には例外を発生させます。当クラス(BtLib.Exception)が例外情報としてスローされます。

Ddf
Record
Extended

上記以外のクラスでスローされる例外はシステム例外で当Exceptionクラス例外がスローされることはありません。
また、注意しなければならないのは上記クラスでBtrieveオペレーションによるステータスは例外としてスローされないということです。当クラスライブラリではBtrieveステータスはエラーではなくステータスなので例外として扱っていません。

以下は例外処理コード例です。

```
try
{
  BtLib.Ddf d = new BtLib.Ddf();
  BtLib.Record r = d.GetRecord("Person");
}
catch(Exception ex)
{
  MessageBox.Show(ex.ToString());
}
```

Ddfはまだロードされていないので、メッセージボックスに以下のように表示されます。

数値 106 は BtLib.Exception.Errors コレクションの値で、エラーを表すコードです。Stataus:以下にはBtrieveのステータスが0以外の場合に表示されます。「DDFがロードされていません」はエラーコード106の詳細説明です。

デフォルトのエラー表示を変更したい場合は以下のようなコードでメッセージを変更することができます。

```
try
{
  BtLib.Ddf d = new BtLib.Ddf();
  BtLib.Record r = d.GetRecord("Person");
}
catch(BtLib.Exception ex)
{
  if( ex.ErrorCode.Equals(BtLib.Exception.Errors.DdfNotLoaded))
  {
    System.Diagnostics.Debug.WriteLine("DDFをロードしてください");
  }
}
```

コンストラクタ

Exception();
Exception(Errors n);
Exception(Errors n, short BtrieveStatus);

当クラスのインスタンスを作成して例外を発生させることは通常のアプリケーション利用では意味がありませんのでご注意ください。

プロパティ

BtrieveStatus

概要
例外発生時にZen/PSQL/Btrieveからのステータスが0以外のもので報告するものがある場合にはこのプロパティに保持されます。当プロパティの値の詳細につきましてはZen/PSQL/Btrieveのマニュアルをご参照ください。

データ型
Int16

ErrorCode

概要
エラーの発生原因を示すコードです。例外発生時には必ずセットされます。エラーコードの意味につきましては当マニュアルの巻末Apeendix-Dに記載されていますのでご参照ください。発生した例外について技術サポートをご利用になる場合にはこのプロパティの値とBtrieveStatusプロパティの値も添えて技術サポートにご連絡ください。

データ型
Exception.Errors

メソッド

ToString

書式
string ToString();

概要
エラーコード、Btrieveステータスコード、エラーの説明を文字列として取得します。説明は概要だけになりますので、詳細な情報が必要な場合はBtrieveStatus/ErrorCodeプロパティの値から該当マニュアルを参照してエラーの診断をしてください。

戻り値
エラーコード、Btrieveステータスコード、エラーの説明テキストを含む文字列。

Native Class

概要
当クラスはマネージド言語環境からBtrieve APIを容易に呼び出すことを目的として作成されたクラスです。マネージド言語からDLLを呼び出すことは.NET Framework Libraryの機能で可能ですが、DLLの宣言やマネージドからアンマネージドへのデータ変換等が必要でプログラムは煩雑になることが多いため、簡単に処理できるようにNative クラスとして機能をまとめました。また、Encoder.GetString での文字列処理仕様を補助したり、マネージド データを Byte 配列に変換するヘルパー メソッドを提供しています。

Btrieve API呼び出しに関連するメソッドはすべてstaticメソッドのため、当クラスはインスタンスを作成しないでメソッドを呼び出してご利用ください。当クラスの利用サンプル・コードはAppendix-Aに記載されています。

コンストラクタ

存在しません。

プロパティ

存在しません。

メソッド

BtrCall

書式

```
static Int16 BtrCall(Operation op,
                     Byte posBlock[],
                     Byte dataBuffer[],
                     Int16 dataLength,
                     Byte keyBuffer[],
                     Int16 keyBufferLength,
                     Int16 keyNumber);

static Int16 BtrCall(Operation op,
                     Byte posBlock[],
                     IntPtr dataBuffer,
                     Int16 dataLength,
                     Byte keyBuffer[],
                     Int16 keyBufferLength,
                     Int16 keyNumber);


static Int16 BtrCall(Operation op,
                     Byte  posBlock[],
                     IntPtr dataBuffer,
                     Int16  dataLength,
                     IntPtr keyBuffer,
                     Int16 keyBufferLength,
                     Int16 keyNumber);
```

概要

Btrieve API BtrCallを呼び出します。パラメータはBtrCall APIと同じですので、詳細はZen/PSQL SDKマニュアル等をご参照ください。２番目のオーバーロード定義はデータをアンマネージドメモリに指定することができます。３番目のオーバーロード定義はキー領域もアンマネージド メモリを指定できる呼び出しについては、System.Runtime.InteropServices を使って構造体を指定することができます。構造体の宣言方法についてはAppendix-Aにて解説していますのでご参照ください。

戻り値

Btrieveステータスコード。

サンプル・コード

```
// データ領域をバイト配列で指定する例。
data = new Byte[334];
dataLength = (short)data.Length;
keyBuf = new Byte[128];
rc = Native.BtrCall(Operation.GetFirst,
                    posblk,data,
                    ref dataLength,
                    keyBuf,
                    keyBufLen,
                    0);


// データ領域を構造体で指定する例。
dataLength = (short)Marshal.SizeOf(dept);
System.IntPtr ptr = Marshal.AllocCoTaskMem(dataLength);
Marshal.StructureToPtr(dept,ptr,true);

rc = Native.BtrCall(Operation.GetFirst,
                    pb,
                    ptr,
                    ref dataLength,
                    keyBuf,
                    (short)keyBuf.Length,
                    0);
Marshal.PtrToStructure(ptr,dept);
```

BtrCallID

書式

```
static Int16 BtrCallId(Operation op,
                       Byte posBlock[],
                       Byte dataBuffer[],
                       Int16 dataLength,
                       Byte keyBuffer[],
                       Int16 keyBufferLength,
                       Int16 keyNumber,
                       Int32 Id);
```

概要

Btrieve API BtrCallIDを呼び出します。パラメータはBtrCall APIと同じですので、詳細はZen/PSQL SDKマニュアル等をご参照ください。

戻り値

Btrieveステータスコード。

FixString

書式

static void FixString (IntPtr mem, int offsetFrom, int offsetTo)

概要

IntPtr で指定された領域の、offsetFrom と offsetTo で指定される範囲にある文字列データのヌル バイトをスペースに置き換えます。Btrieveの文字列型の後続ブランク設定をする場合に使います。このメソッドの利用方法の解説は「ストラクチャービルダー」にもありますのでご参照ください。

パラメータ

第1パラメータはMarshall.StructureToPtr呼び出し等で得られるアンマネージドデータ領域を指定します。通常は構造体の領域が指定されると考えて、第 2、第 3 パラメータでは、指定された領域内で変換処理を開始する位置と終了する位置をオフセットで指定します。オフセット指定のベースは0です。

戻り値

変換されたIntPtr領域

GetBoolean

書式
static Boolean GetBoolean(Byte [] b, int offset)

概要
Byte配列の指定したオフセットにあるデータを.NET FrameworkのBooleanデータ型として返します。

パラメータ
Btrieve APIから戻されたByte型データの配列を第1パラメータに指定します。第2パラメータはByte配列内の該当データへのオフセットです。

戻り値
変換されたBoolean型のデータ。

GetBytes

書式
static Byte [] GetBytes(Boolean b);
static Byte [] GetBytes(Char c);
static Byte [] GetBytes(DataTime d);
static Byte [] GetBytes(DateTime d, BtrieveTypes targetTp);
static Byte [] GetBytes(Decimal d);
static Byte [] GetBytes(Double d);
static Byte [] GetBytes(Int16 n);
static Byte [] GetBytes(Int32 n);
static Byte [] GetBytes(Int64 n);
static Byte [] GetBytes(Single s);
static Byte [] GetBytes(Decimal d,BtrieveTypes targetTp,int size,int dec);

概要
.NET Framework データ型をByte配列に格納します。各.NET Frameworkデータ型をサポートするため、メソッドはオーバーロードして定義されています。Native ClassでBtrCall呼び出しの前にByte配列にデータをセットする必要がある場合に使います。文字列型は.NET FrameworkのEncodingクラスを使えばbyte型配列に変換できますので、ここでは提供していません。

パラメータ

第1パラメータは変換元のデータです。変換元データ型に対して複数のBtrieveデータ型がマッピングされる場合にはターゲットとなるBtrieveデータ型を第２パラメータで指定します。Decimal型に限りサイズ、小数点以下桁数をそれぞれ第３、第４パラメータとして指定します。

戻り値
変換されたByte配列。

サンプル・コード

```
// Int32変換例。
Int32 id = Convert.ToInt32(txtID.Text);
byte [] byteId = Native.GetBytes(id);
Buffer.BlockCopy(byteId,0,data,1,4);

 // Date変換例
DateTime dt = System.DateTime.Now;
byte [] byteDate = Native.GetBytes(dt);
Buffer.BlockCopy(byteDate,0,data,69,byteDate.Length);

 // numeric 変換例
Decimal d = 12.3M;
byte []byteNumeric = Native.GetBytes(d,BtrieveTypes.numeric,4,1);
Buffer.BlockCopy(byteNumeric,0,data,74,4);
```

GetDate

書式
static DateTime GetDate(Byte [] b, int offset)

概要
Byte配列の指定したオフセットにある日付データを.NET FrameworkのDateTimeデータ型として返します。返されるDateTime型データの時間部分にはすべて0がセットされます。

パラメータ
Btrieve APIから戻されたByte型データの配列を第1パラメータに指定します。第2パラメータはByte配列内の該当データへのオフセットです。

戻り値
変換されたDateTime型のデータ。

GetDecimal

書式
static Decimal GetDecimal (Byte [] b,
int offset,
int size,
int dec,
BtrieveTypes tp)

概要
Byte配列の指定したオフセットにあるBtrieve数値型データを.NET FrameworkのDoubleデータ型として返します。対象となる Btrieve 数値型データは Numeric、Numericsa、Numericsa、Decimal、Money です。

パラメータ
Btrieve APIから戻されたByte型データの配列を第1パラメータに指定します。第2パラメータはByte配列内の該当データへのオフセットです。第3パラメータは対象データのサイズ、第4パラメータは小数点以下桁数を指定します。第5パラメータにはByte配列上の対象となるBtrieveデータ型を指定します。

戻り値
変換されたDecimal型のデータ。

GetDouble

書式
static Double GetDouble (Byte [] b, int offset)

概要
Byte配列の指定したオフセットにある浮動小数点データ(サイズ8byte)を.NET FrameworkのDoubleデータ型として返します。

パラメータ
Btrieve APIから戻されたByte型データの配列を第1パラメータに指定します。第2パラメータはByte配列内の該当データへのオフセットです。

戻り値
変換されたDouble型のデータ。

GetInt16

書式
static Int16 GetInt16(Byte [] b, int offset)

概要
Byte配列の指定したオフセットにある２バイト整数データを.NET FrameworkのInt16データ型として返します。

パラメータ
Btrieve APIから戻されたByte型データの配列を第1パラメータに指定します。第2パラメータはByte配列内の該当データへのオフセットです。

戻り値
変換されたInt16型のデータ。

GetInt32

書式
static Int32 GetInt32(Byte [] b, int offset)

概要
Byte配列の指定したオフセットにある4バイト整数データを.NET FrameworkのInt32データ型として返します。

パラメータ
Btrieve APIから戻されたByte型データの配列を第1パラメータに指定します。第2パラメータはByte配列内の該当データへのオフセットです。

戻り値
変換されたInt32型のデータ。

GetInt64

書式
static Int64 GetInt64(Byte [] b, int offset)

概要
Byte配列の指定したオフセットにある8バイト整数データを.NET FrameworkのInt64データ型として返します。

パラメータ
Btrieve APIから戻されたByte型データの配列を第1パラメータに指定します。第2パラメータはByte配列内の該当データへのオフセットです。

戻り値
変換されたInt64型のデータ。

GetSingle

書式
static Single GetSingle(Byte [] b, int offset)

概要
Byte配列の指定したオフセットにある4バイト浮動小数点データを.NET FrameworkのSingleデータ型として返します。

パラメータ
Btrieve APIから戻されたByte型データの配列を第1パラメータに指定します。第2パラメータはByte配列内の該当データへのオフセットです。

戻り値
変換されたSingle型のデータ。

GetTime

書式
static DateTime GetTime(Byte [] b, int offset)

概要
Byte配列の指定したオフセットにある時間型データを.NET FrameworkのDateTimeデータ型として返します。返されるDateTime型データの日付部分は現在の日付が設定されます。

パラメータ
Btrieve APIから戻されたByte型データの配列を第1パラメータに指定します。第2パラメータはByte配列内の該当データへのオフセットです。

戻り値
変換されたDateTime型のデータ。

Trim

書式
static string Trim(string src);

概要
例えば、
data = new Byte[334];
//
// btrieve オペレーション によるデータ取得
// …
string str = Encoder.GetString(data,9,16);

上記のようなコードで得られたレコード バッファから文字列を取得する際は、領域内の後続ヌルも含めて string が作成され、文字列のサイズは常に GetString で指定した領域のサイズになります（このサンプルの場合は 14）。このようにして得られた string には後続ヌルが含まれているため、文字列の連結等が正常に機能しないことがあります。Trim メソッドにより後続のヌルを取り除き、.NET Framework で正常に認識できる文字列に変換します。

戻り値
変換後文字列

サンプル・コード

```
  // get shift jis encoder
  Encoding sje = Encoding.GetEncoding("shift-jis");

  // data領域にレコードを読み込むコード(省略)
  // …
  // …
  string firstName = Native.Trim(sje.GetString(data,9,16)); //
  string lastName = Native.Trim(sje.GetString(data,26,26)); //
```

Record Class

概要

Ddf クラスから得られる Record クラスにより、データをレコード単位でアクセスすることができます。Record クラスは System.Collections.HashTable から導出されており、Record に含まれるフィールドには以下のようにカラム名でアクセスします。

```
Record rec = ddf.GetRecord(“Person”);       // Recordオブジェクトを取得
rec[“First_Name”] = “太郎”;
rec[“Last_Name”] = “田中”;
```

HashTableコレクションへセットしたデータはInsert/Update等の登録系Btrieveオペレーションで参照され、データベースに出力されます。

また、キー参照が必要なGetEqualやGetGreater等のBtrieveオペレーション時にはキーに該当するフィールド値をセットしてこれらのキー参照オペレーションを実行してください。

HashTableコレクションはGet/Step系のオペレーションが正常に終了した場合にはすべてレコードから読み込んだ値がセットされます。Get/Set系オペレーション前にセットしていたフィールド値は読み込んだ値に変更されます。以下はGetFirstを実行して値を参照するコード・サンプルです。

```
Record rec = ddf.GetRecord(“Person”);
if(rec.Read(Operation.GetFirst) == 0)
{
  listBox1.Items.Add(rec[“First_Name”]);
}
```

レコードオブジェクトへのアクセサーは文字列型です。アクセサーとして指定した文字列がDDF定義に存在しない場合はHashTableオブジェクトのデフォルト例外でnull pointer excepotion が発生します。また、アクセサー文字列は大文字小文字を区別します。例えば、Zen/PSQLのDEMODATAの場合Personテーブルの”Comments”カラムを”comments”（すべて小文字）”COMMENTS”(すべて大文字)で指定した場合はnull pointer exceptionが発生します。

コンストラクタ

Record()

当クラスはDdfクラスのGetRecordメソッドを使ってインスタンスを生成してください。当オブジェクトのインスタンスをnewで生成した場合はAttachメソッドでテーブル情報をセットします。

プロパティ

ColumnCount

データ型
Int32

概要
レコードに存在するカラムの数を保持します。

DataFileName

データ型
string

概要
Create/Openメソッド実行時に参照されます。実行時に動的にデータ・ファイル名を設定したい場合には
Create/Openメソッド実行前にこのプロパティを変更します。

FileFlag

データ型
short

概要
Createメソッド実行時に参照されます。Createオペレーション時にファイル・フラグ値を指定します。

Index

データ型
String

概要
Readメソッドでキーに従った読込みを実行する際のインデックス名を指定します。

サンプル・コード
以下はZen/PSQLのPersonテーブルでGET EQUALオペレーションを実行する例です。Indexプロパティにインデックス名”Names”をセットしています。Namesインデックスはセグメント・キーを構成しているので、First_Name、Last_Name の両カラムにデータをセットしています。

```
BtLib.Ddf d = new BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata");
BtLib.Record r = d.GetRecord("person");
r.Open();
r.Index = "Names";
r["First_Name"] = "Koichi";
r["Last_Name"] = "Adachi";
short rc = r.Read(BtLib.Operation.GetEqual);
```

IndexNumber

データ型
Int32

概要
Readメソッドでキーに従った読込みを実行する際のインデックス番号を指定します。インデックス番号はBtrieveのキー番号です。IndexNumberプロパティが設定されるとIndexプロパティは自動的にヌルに再設定されます。

サンプル・コード
以下はZen/PSQLのPersonテーブルでGET EQUALオペレーションを実行する例です。IndexNumberプロパティにキー番号値0をセットしています。Namesインデックスはセグメント・キーを構成しているので、First_Name、Last_Name の両カラムにデータをセットしています。

```
BtLib.Ddf d = new BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata");
BtLib.Record r = d.GetRecord("person");
r.Open();
r.IndexNumber = 0;
r["First_Name"] = "Koichi";
r["Last_Name"] = "Adachi";
short rc = r.Read(BtLib.Operation.GetEqual);
```

IsOpen

データ型
bool

概要
レコードが関連するデータ・ファイルのオープン状態を保持します。Openメソッドが呼び出され正常終了するとtrueがセットされます。Closeメソッドが正常終了するとfalseがセットされます。アプリケーション終了時には、このクラスの Close メソッドを呼び出して関連資源を解放されることをお勧めします。当プロパティの参照により例外発生時等にCloseメソッドを呼び出す必要性を判断することができます。

Lock

データ型
LockBias

概要
Read/Stepオペレーションで参照されます。レコード読み込み時にロックをかける場合は、NoLock 以外の値をセットします。ロックの解除は、シングル レコード ロック方式で既存のロックを解除する方法以外に、Unlock メソッドによって行うこともできます。

NullKeyValue

データ型
String

概要
Createメソッド実行時に参照されます。キーのヌル値を設定します。デフォルト値は32です。

OpenMode

データ型
short

概要
Openメソッドで参照され、Btrieveデータファイルをオープンするときのモードを設定します。詳細はZen/PSQL/Btreiveのオープン・モードを参照してください。

PageSize

データ型
short

概要
Createメソッドで参照されます。Btrieveデータファイルのページサイズをセットします。デフォルトは4096です。当プロパティの値の詳細につきましてはZen/PSQL/Btrieveのマニュアルをご参照ください。

メソッド

ClearData

書式
short ClearData();

概要
データバッファを初期化した状態にします。数値カラムの値はゼロに、文字列はすべてスペースか長さのない文字列へ初期化されます。

戻り値
なし。

サンプルコード

```
  Dim ddf As BtLib.Ddf = New BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata")
  Dim rec As BtLib.Record = ddf.GetRecord("Room")

  rec.Open()
  rec.ClearData()
  rec.Index = "Building_Number"
  Dim rc As Short = rec.Write(BtLib.Operation.Insert)
  If rc <> 0 Then
    MsgBox(CStr(rc))
  End If
rec.Close()
```

Close

書式
short Close();

概要
Btrieveデータ・ファイルをクローズし関連する資源を解放します。当メソッドの呼び出し前にOpenメソッドが呼び出されて正常終了している必要があります。

戻り値
Btrieveステータスコード。

Create

書式
short Create();

概要
Btrieveデータ・ファイルを新たに生成します。対象となるデータ ファイルは、クローズ状態であるか、または存在しない名前であることが必要です。生成時に資源を確保出来ない場合は例外でBtrieve Statusが通知されますので、この値を参照してエラーの診断をしてください。

戻り値
Btrieveステータスコード。

Delete

書式
short Delete();

概要
カレント・レコードを削除します。カレント・レコードはReadまたはStepメソッドにて事前にセットされている必要があります。例えばOpen直後やDeleteメソッド実行直後はカレント・レコードが存在しないので注意が必要です。

戻り値
Btrieveステータスコード。

GetBytes

書式
Byte [] GetBytes(String colName);

概要
指定したカラムデータをByte配列に返します。このメソッドを呼び出す前に Read または Step メソッドにより Zen/PSQL/Btrieve データを読み込んでおかないと、内部バッファに入っている不定なデータを返すので注意してください（通常、初期値は 0 です）。また、今回のバージョンでは可変長レコードには対応していません。

パラメータ
カラム名。

戻り値
Byte配列。

GetData

書式
void GetData(IntPtr pStructure);

概要
IntPtrでポイントされるメモリ領域にカレント・レコード・イメージを転送します。レコードイメージを持つ構造体にデータを転送する場合に使います。構造体はメモリアライメントを考慮して定義することが必要になります。構造体定義サンプルはAppendix-Aにありますのでご参照ください。

パラメータ
レコードイメージを格納するIntPtrでポイントされるメモリ領域。

戻り値
なし

サンプルコード

```
BtLib.Ddf d = new BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata");
BtLib.Record r = d.GetRecord("Department");
r.Open();
r.Index = "Dept_Name";
short rc = r.Read(Operation.GetFirst);
Department dept = new Department();
System.IntPtr ptr = Marshal.AllocCoTaskMem(r.GetRecordLength());
Marshal.StructureToPtr(dept,ptr,true);
r.GetData(ptr);
```

```
Marshal.PtrToStructure(ptr,dept);
System.Diagnostics.Debug.WriteLine(dept.Name);
Marshal.FreeCoTaskMem(ptr);
r.Close();
```

GetDataColumn

書式
DataColumn GetDataColumn(Int32 Index);

概要
.NET Framework の DataColumn オブジェクトを取得します。

パラメータ
取得するカラムを指定するゼロベースのインデックスを指定します。この数値は、テーブル定義（DDF）におけるカラムの順序に対応します。

戻り値
DataColumn オブジェクト

サンプルコード

```
BtLib.Ddf d = new BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata");
BtLib.Record r = d.GetRecord("Person");

for(int i=0; i<r.ColumnCount; i++)
{
  System.Data.DataColumn col= r.GetDataColumn(i);
  System.Diagnostics.Debug.WriteLine(col.ColumnName.ToString());
}
```

GetDataSet

書式
DataSet GetDataSet();
DataSet GetDataSet(String [] fields);
DataSet GetDataSet(Int32 maxRecords);
DataSet GetDataSet(String [] fields, Int32 maxRecords);

概要
.NET FrameworkのDataSet オブジェクトを返します。パラメータ指定をしない呼び出しではすべてのカラムとすべてのレコードを DataSet に返します。抽出するカラム名をString型の配列に指定する呼び出しは指定したカラムのみ DataSet に返します。Int32型の値としてゼロ以外の数を指定した場合にはこの値をレコードの上限として DataSet を作成して返します。

パラメータ
抽出するカラム名の文字列配列、上限レコード数を指定します。

戻り値
DataSetオブジェクト

GetDataTable

書式
DataTable GetDataTable ();

概要
.NET FrameworkのDataTableオブジェクトを返します。返されるDataTableオブジェクトはDDFからのカラム情報を含みます。Rowは空の状態となります。

戻り値
DataTableオブジェクト

GetExtended

書式
Extended GetExtended();
Extended GetExtended(short skipRecords, short maxRecords);

概要
レコードに関連するExtended オブジェクトを返します。

パラメータ
Extended オペレーションのスキップレコード数と最大レコード数を指定できます。

戻り値
Extendedオブジェクト

GetNumOfRecords

書式
Int16 GetNumOfRecords(Int32 records);

概要
オープンしているデータファイルに含まれるレコード数を返します。

パラメータ
レコード数。

戻り値
Btrieveステータス

GetPosition

書式
Int16 GetPosition(Int32 PhysicalPosition);

概要
現在のレコードの物理位置を取得します。取得した物理位置を Read メソッドのパラメータとして指定して、その位置のレコードを直接読み込めます。

パラメータ
物理位置が戻されます。C#言語ではパラメータにOut属性を指定する必要があります。

戻り値
Btrieveステータス

GetRawValue

書式
Object GetRawValue(String columnName);

概要
レコードに格納されているデータを.NETデータに変換せずに返します。AUTOTIMESTAMP, TIMESTAMP2データ型でDateTimeデータへの変換で失われる精度を補完します。

パラメータ
データを得るカラム名。

戻り値
レコードからの生データ。TIMESTAMP/TIMESTAMP2の場合はUint64データで返します。

GetRecordLength

書式
Int32 GetRecordLength();

概要
レコードクラスが関連しているデータファイルのレコード長を返します。レコードイメージと同じサイズのメモリ領域を確保する場合に便利です。

パラメータ
なし。

戻り値
レコード長

Open

書式
short Open();

概要
Btrieveデータ・ファイルをオープンします。オープンする対象となるデータ・ファイルはDdfクラスのGetRecordメソッドのパラメータとして指定したテーブルに関連するデータファイルです。

戻り値
Btrieveステータスコード。

Query

書式

List<T> Query<T>;
List<T> Query<T>(short keyNo);
List<T> Query<T>(short keyNo, string extendedConditions);

概要
レコードのカラム名と同じメンバーを型として指定することでその型のジェネリックリストを得ることができます。取得したジェネリックリストはLinqから利用することができます。

パラメータ

T
レコードクラスが関係するテーブルのカラムを受取るクラス。プロパティ名をカラム名と一致させます。テーブルのすべてのカラムをプロパティとしてクラスに定義する必要はなく、データとして受け取りたいカラムを選択して定義することができます。

keyNo
Btrieve読込み系オペレーションを発行する場合のキー番号。

extendedConditions
結果ジェネリックリストを絞り込むための検索条件。当クラスライブラリのExtendedクラスのフィルタリング条件と同じ形式で文字列を指定します。

現在のバージョンでは 3 つのオーバーロードを提供しています。最初の形式では、Step 系オペレーションでデータを読込みます。2 番目の形式では、キー順にデータを読み込むオペレーションでデータをリストします。3 番目の形式では、Extended 系オペレーションでキー順にデータを取得します。

戻り値

指定した型のジェネリックリスト

サンプル・コード

Linqを使いデータから特定のレコードを抽出します。Zen/PSQLのdemodataにあるPersonテーブルを対象としています。

```
try
{
  BtLib.Ddf d = new BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata");
  BtLib.Record r = d.GetRecord("person");

  r.Open();
  var query = from p in r.Query<Person>(0)
              where p.First_Name == "William"
              select p;
```

```
    foreach (var person in query)
    {
      listBox1.Items.Add(person.First_Name + " " + person.Last_Name + " " +
person.Perm_Street);
    }
    r.Close();
  }
  catch (System.Exception er)
  {
    System.Diagnostics.Debug.WriteLine(er.ToString());
  }
```

上のサンプルコードでは以下のようなPersonクラスを参照しています。プロパティ名がPersonテーブルのカラム名と一致することにご注意ください。

```
  public class Person
  {
    private string _first_Name;
    private string _last_Name;
    private string _perm_Street;

    public String First_Name
    {
      get { return _first_Name; }
      set { _first_Name = value; }
    }

    public String Last_Name
    {
      get { return _last_Name; }
      set { _last_Name = value; }
    }

    public string Perm_Street
    {
      get { return _perm_Street; }
      set { _perm_Street = value; }
    }
}
```

Read

書式
short Read(Operation op);
short Read(Int32 PhysicalPosition);
short Read(Operation op, Object obj);

概要

第 1 オーバーロード形式では、指定したキー読込み系オペレーションでレコードを読込みます。

第２オーバーロード形式ではパラメータとして物理位置を指定してレコードを読込みます。読み込んだレコードのデータをRecordオブジェクトのコレクションとして保持します。このメソッドではLockプロパティが参照され該当のレコードロックが同時に実行されます。

第 3 オーバーロード形式では、指定したオペレーションでデータを読込み、第 2 パラメータで指定したクラス インスタンスでそのデータを受け取ります。指定するクラスのプロパティ名はテーブルのカラム名と一致している必要があります。テーブル内の一部のカラム名を指定してデータを読み込むこともできます。
パラメータ
インデックス依存の参照系Btrieveオペレーションコードまたは GetPositionメソッドにより取得したレコードの物理位置。第3オーバーロード形式の第2パラメータはレコードデータを受取るクラスインスタンスを指定します。

戻り値
Btrieveステータスコード。

サンプルコード
Zen DemodataのDeptを読むC#サンプルコードです。Deptクラスはストラクチャビルダーで生成したものにToStringを追加し以下のようになります。

```
public class Dept
{
        public string Name { get; set; }
        public Double Phone_Number { get; set; }
        public string Building_Name { get; set; }
        public Int32 Room_Number { get; set; }
        public Int64 Head_Of_Dept { get; set; }
        public override string ToString()
        {
            return "dept: " + Name + " " + Building_Name;
        }
```

```
    }
}
```

上のクラスにデータを読むサンプルコード。

```
short rc;
Dept dept = new Dept();
BtLib.Ddf ddf = new BtLib.Ddf(@"C:\ProgramData\Actian\Zen\Demodata");
BtLib.Record rec = ddf.GetRecord("Dept");
rc = rec.Open();
rc = rec.Read(BtLib.Operation.GetFirst, dept);
while (rc == 0)
{
   Console.WriteLine(dept.ToString());
   rc = rec.Read(BtLib.Operation.GetNext, dept);
}
rec.Close();
```

SetBytes

書式
```
short SetBytes(String colName, byte [] b);
```

概要
指定したカラムのデータをbyte配列で設定します。このメソッド呼び出し後にWriteを呼び出すことで実際にZen/PSQL/Btrieveデータベースに反映されます。Byte配列で指定するデータは指定したカラムのデータ型の仕様に沿った形式でbyte配列に正しいサイズで設定されている必要があります。誤ったデータを書き込んだ際には、読み出すアプリケーションによっては予期しない動作の原因になることがありますので十分ご注意ください。例えば日付型に不正なデータをセットした場合、Control Centerでは読み込めないというエラーが出て、それ以降データを表示することができない場合があります。また、今回のバージョンでは可変長レコードには対応していません。

パラメータ
カラム名。

戻り値
なし。

SetData

書式
void SetData(IntPtr pStructure);

概要
IntPtrでポイントされるメモリ領域をカレント・レコード・イメージに転送します。レコードイメージを持つ構造体を出力する場合に使います。構造体はメモリアライメントを考慮して定義することが必要になります。構造体定義サンプルはAppendix-Aにありますのでご参照ください。

パラメータ
レコードイメージを保持するIntPtrでポイントされるメモリ領域。

戻り値
なし。

SetRawValue

書式
void SetRawValue(String columnName, System.Object data);

概要
columnName で指定されるカラムへデータを設定します。指定するデータはZenデータ形式に沿って指定します。例えばTIMESTAMPやTIMESTAMP2型のカラムの場合はUint64データ型で指定します。

パラメータ

columnName
データを設定するカラム名。

data
レコードバッファへ設定するデータ

戻り値
なし。

Step

書式
short Step(Operation op);

概要
指定したStep系オペレーション・コードによりレコードを読込みます。読み込んだレコードのデータをRecordオブジェクトのコレクションとして保持します。

パラメータ
Step 系 Btrieveオペレーションコード。

戻り値
Btrieveステータスコード。

Write

書式
short Write(Operation op);
short Write(Operation op, System.Object obj);

概要

第 1 形式のオーバーロードでは、指定した Insert または Update オペレーション コードに従ってレコードの書き込みを実行します。書き込むフィールド値は、当メソッドを実行する前に Record コレクションとして設定しておきます。

第 2 形式のオーバーロードでは、指定した Insert または Update オペレーション コードに従ってレコードの書き込みを実行します。第 2 パラメータで指定したクラス定義を基に、現在の Record コレクションにデータを書き込みます。クラスのプロパティ名はテーブルのカラム名と一致している必要があります。すべてのカラムをプロパティで定義する必要はありませんが、設定されていないカラムには、現在のレコード バッファ内の値が書き込まれることになります。

パラメータ
Insert または Update に対応する Btrieve オペレーション コード。第 2 オーバーロードの第 2 パラメータは、書き込むデータが設定されたクラス インスタンス。

戻り値
Btrieveステータスコード。

Transaction Class

概要

Ddf Classにて接続したZen/PSQL/Btrieveに関するトランザクション制御を実行します。トランザクション制御は当クラスに定義されたstaticなメソッドを呼び出します。

サンプル・コード

```
BtLib.Ddf d = new BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata");
BtLib.Record r = d.GetRecord("test");
r.Open();

r["ID"] = "100";
r["name"] = "George";
r["dt"] = "2002/7/22";
r["tm"] = "22:10:12";
Transaction.Begin();
short rc = r.Write(Operation.Insert);
if(rc != 0)
{
  Transaction.Abort();
  r.Close();
  MessageBox.Show("error " + Convert.ToString(rc));
  return;
}
Transaction.End();
r.Close();
```

コンストラクタ

概要
トランザクションクラスにはコンストラクタは存在しません。

プロパティ

Lock

データ型
LockBias

概要
LockBiasの値を保持するプロパティです。Begin()またはBeginConCurrent()メソッドのパラメータを省略した場合にこのプロパティ値が参照されます。

メソッド

Abort

書式
static short Abort();

概要

実行中のトランザクションを中断します。

Begin

書式
static short Begin();
static short Begin(LockBias lock);

概要

トランザクションを開始します。

BeginConCurrent

書式
static short BeginConCurrent();
static short BeginConCurrent(LockBias lock);

概要

コンカレントトランザクションを開始します。

End

書式
static short End();

概要

トランザクションまたはコンカレントトランザクションをコミットします。

Reset

書式
static short Reset();

概要

Zen/PSQL/BtrieveのResetを実行します。

Compat Class

Btrieve Classes for .NETで既存のVBMan Controls for Btrieveアプリケーションを.NET環境に移行する場合に便利なクラスです。VBMan Controls for BtrieveのメソッドでExtended系とSafeArrayを使うメソッド以外をCompatクラスに移植しました。ここでは元のアプリケーションはVisual Basicで作成されていることを想定していますので、メソッド表記はVisual Basic形式にしています。

コンストラクタ

Compat();

概要
プロパティをデフォルト値で初期化します。パラメータはありません。

プロパティ

VBMan Controls for BtrieveではVBMan.INIファイルに設定されていたシステム設定を当クラスではプロパティとして設定します。

DDFDir

データ型
String

概要
DDFが存在するディレクトリへのパスを指定します。通常はローカルドライブを含めたパスやUNC形式でサーバー名を含めたディレクトリへのパスを指定します。Pervasive.SQL V8.6からサポートされるセキュアデータベースを利用する場合にはbtrv://で始まるデータベースURIを指定します。この場合データベースURIには&table=や&dbfile=の指定は除外した文字列を指定してください。

FileFlag

データ型
short

概要
Createメソッド実行時に参照されます。Createオペレーション時にファイル・フラグ値を指定します。

Lock

データ型
LockBias

概要
DbAccessオペレーションで参照されます。レコード読込み時にロックをかける場合は、NoLock 以外の値をセットします。ロックの解除は、シングル レコード ロック方式で既存のロックを解除する方法以外に、Unlock メソッドによって行うこともできます。

NullKeyValue

データ型
String

概要
Createメソッド実行時に参照されます。キーのヌル値を設定します。デフォルト値は32です。

OpenMode

データ型
short

概要
DbOpen/DbAllOpenメソッドで参照されます。詳細はZen/PSQL/Btreiveのオープン・モードを参照してください。

メソッド

既存のアプリケーションがVisual Basicで記述されていると思われますので当クラスのメソッドの表記はVisual Basic形式としています。

DbAbortTransaction

Object.DbAbortTransaction() As Integer

概要
トランザクションの中止を宣言します。

パラメータ
なし。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外は Btrieve のステータス コードが返されます。

DbAccess

```
Object.DbAccess(BtrieveOpCode As Integer,
                TableName As String,
                KeyName   As String) As Integer
```

概要
BtrieveOpCodeで指定されるBtrieveの機能を呼び出します。

パラメータ
BtrieveOpCode
Btrieveのオペレーションコードを指定します。

TableName
Btrieveオペレーションを発行するテーブルの名前を指定します。

KeyName
Btrieveオペレーションに関連するキーの名前を指定します。

戻り値
正常終了ならば0が返ります。負の値はVBManエラーコード一覧を参照してください。それ以外は Btrieve のステータス コードが返されます。

Visual Basicサンプル

```
Dim rc%
With VBManDb1.
  rc% = .DbSetFieldData(“従業員”,”社員番号”,"066217")
  rc% = .DbAccess(BTR_GET_EQUAL,”従業員”,”社員キー”)
```

```
  If rc% <> 0 Then
    MsgBox "Btrieveの呼び出しに失敗しました" + Str$(rc%)
    Exit Sub
  End If
End With
```

DbBeginConCurTransaction

Object.DbBeginConCurTransaction(LockBias As Integer) As Integer

概要
コンカレント・トランザクションの開始を宣言します。

パラメータ
LockBias

トランザクション ロックを指定します。指定できる値は 100、200、300、400 のいずれかです。詳細はBtrieve SDKマニュアルを参照してください。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

DbBeginTransaction

Object.DbBeginTransaction(LockBias As Integer) As Integer

概要
トランザクションの開始を宣言します。

パラメータ

LockBias

トランザクション ロックを指定します。指定できる値は 100、200、300、400 のいずれかです。詳細はBtrieve SDKマニュアルを参照してください。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

DbClearFieldBuffer

Object.DbClearFieldBuffer(TableName As String) As Integer

概要
指定したテーブルのデータ・バッファを初期化します。データ バッファは、Btrieve とデータ交換を行うメモリ領域です。

パラメータ

TableName
テーブルの名前を指定します。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

DbClose

Object.DbClose(TableName As String) As Integer

概要

指定されるテーブルに関連するBtrieveファイルをクローズします。このメソッドを呼び出す以前にBtrieveファイルはオープンされている必要があります。

パラメータ

TableName
テーブルの名前を指定します。

戻り値

正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

DbCloseAll

Object.DbCloseAll() As Integer

概要

DDFに定義されたBtrieveファイルをすべてクローズします。

パラメータ
なし

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

DbCreate

Object.DbCreate(TableName As String) As Integer

概要
TableNameで指定されるテーブルに関連するBtrieveファイルを生成(Create)します。レコード長、インデックスの構成などはDDFの定義を参照します。このメソッドを呼び出すときには関連するBtrieveファイルはクローズされていることが必要です。サーバーにある Btrieve ファイルをマルチ ユーザーで使用している場合は、1 つのクライアントから DbClose を実行しても他でオープンしていれば、このメソッドは成功しません。既存の Btrieve ファイルに関連するテーブル名を指定すると、ファイルが上書きされるので注意してください。生成されるBtrieveファイルのページ・サイズは4,096となります。

パラメータ

TableName
テーブルの名前を指定します。

戻り値

正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

Visual Basicサンプル

```
Dim rc As Integer
rc = VBMan1.DbClose(“月次ファイル”)
Kill “c:\data\月次.btr”
rc = VBMan1.DbCreate(“月次ファイル”)
If rc <> 0 Then
    MsgBox “Btrieve createステータス “ & CStr(rc)
    Stop
End If
rc = VBMan1.DbOpen(“月次ファイル”)
If rc <> 0 Then
    MsgBox “Btrieve openステータス “ & CStr(rc)
    Stop
End If
```

DbEndTransaction

Object.DbEndTransaction() As Integer

概要
トランザクションの終了を宣言します。

パラメータ
なし。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

DbFindPercentage

```
Object.DbFindPercentage(TableName As String,
                        KeyName As String,
                        PhysicalPosition As Long,
                        Percentage As Integer) As Integer
```

概要
レコード位置をパーセントで取得します。

パラメータ
TableName
テーブル名を指定します。
KeyName
インデックス名を指定します。ヌルを指定した場合は物理位置で取得します。
PhysicalPosition
パーセンテージを得る物理位置を指定します。このパラメータを有効にするためには、KeyNameにヌル文字列を設定します。
Percentage
取得する位置。例えば、80パーセントの位置の場合、整数値で8000が返ります。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

03	FILE NOT OPEN
06	Invalid key number
07	Different key number
08	Invalid positioning
09	End of file
22	Data buffer length
41	Operation not allowed
43	Invalid data record address
82	Lost position

注意
Btrieveファイルはバージョン6.X以降の形式でなければ使用できません。

DbGetByPercentage

```
Object.DbGetByPercentage(TableName As String,
                         KeyName As String,
                         Percentage As Integer) As Integer
```

概要
パーセント指定でレコードを取得します。

パラメータ
TableID
テーブル名を指定します。

KeyName
インデックス名を指定します。ヌル文字列を指定した場合は物理位置でレコードを取得します。

Percentage
取得する位置をパーセントで指定します。例えば、80パーセントの位置の場合、8000を指定します。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

03	FILE NOT OPEN
06	Invalid key number
07	Different key number
08	Invalid positioning
09	End of file
22	Data buffer length
41	Operation not allowed
82	Lost position

注意
Btrieveファイル形式はバージョン6.x以降であることが必要です。

DbGetDataSize

Object.DbGetDataSize(TableName As String, FieldName As String) As Integer

概要
指定したフィールドのデータ・サイズ（バイト単位）を返します。

パラメータ
TableName
テーブル名を指定します。

FieldName
フィールド名を指定します。

戻り値
正常ならばフィールドのデータ・サイズが返されます。負の値はCompat Classエラーコード一覧を参照してください。

DbGetDataType

Object.DbGetDataType(TableName As String,
FieldName As String) As Integer

概要
指定したフィールドのBtrieveデータ型を返します。

パラメータ
TableName
テーブル名を指定します。

FieldName
フィールド名を指定します。

戻り値
正常ならばフィールドのデータ型が返されます。負の値はCompat Classエラーコード一覧を参照してください。

データ型	戻り値
String	0
Integer	1
Float	2
Date	3
Time	4
Decimal	5
Money	6
Logical	7
Numeric	8
Bfloat	9
Lstring	10
Zstring	11
Note	12

Lvar	13
Unsinged Binary	14
Auto increment	15
Named Index	255

DbGetDirect

Object.DbGetDirect(TableName As String,
Pos As Long,
NewIndexName As String) As Integer

概要
指定された物理レコード位置から、Btriveデータを読込みます。

パラメータ
TableName
テーブル名を指定します。
Pos
物理レコード位置を設定します。DbGetPosition関数で取得する、4バイトの整数です。
NewIndexName
このメソッドで取得したレコードの、新しいアクセス パスをインデックス名で指定します。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

DbGetFieldData

Object.DbGetFieldData(TableName As String,
FieldName As String) As String

概要
指定されたテーブルのデータ バッファから、データベースのフィールドの値を文字列データとして返します。

パラメータ
TableName
テーブル名を指定します。
FieldName

フィールド名を指定します。

戻り値
フィールドの値が文字列で返されます。データベースのフィールドから文字列データ型への変換はこの関数内部で行われます。例えば、時刻型は "hh:mm:ss" の形で返されます。エラーが発生した場合は、ヌル文字列が返されます。ヌル文字列が返されるのは、テーブル名、フィールド名がこのメソッドの関連するDDF定義に存在しない場合です。

DbGetFieldName

Object.DbGetFieldName(TableName As String,
FieldID As Integer,
FieldName As String) As Integer

概要
テーブル名,フィールドIDで指定したフィールド名を返します。

パラメータ
TableName
テーブル名を指定します。
FieldID
フィールドIDを指定します。0ベースで指定します。
FieldName
フィールド名が返されます。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。

DbGetIndexName

Object.DbGetIndexName(TableName As String,
IndexID As Integer,
IndexName As String) As Integer

概要
インデックスIDで指定したインデックスが設定されているフィールド名を返します。

パラメータ
TableName
テーブル名を指定します。
IndexID
インデックスIDを指定します。0ベースで指定します。
IndexName
インデックスが設定されているフィールド名が返されます。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。

注意
インデックス名（NamedIndex）はこのメソッドでは得ることができません。

DbGetNumOfField

Object.DbGetNumOfField(TableName As String, NumOfField As Integer) As Integer

概要
指定されたテーブルに定義されているフィールド数を返します。

パラメータ
TableName
テーブル名を指定します。

NumOfField
定義されているフィールドの数が返されます。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。

DbGetNumOfIndex

Object.DbGetNumOfIndex (TableName As String, NumOfIndex As Integer) As Integer

概要
指定されたテーブルに定義されているインデックス数を返します。

パラメータ
TableName
テーブル名を指定します。

NumOfIndex
定義されているインデックスの数が返されます。セグメント・キーが含まれる場合はその構成メンバーの数も加算された値が返されます。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。

DbGetNumOfRecords

Object.DbGetNumOfRecords(TableName As String, NumOfRec As Long) As Integer

概要
指定されたテーブルに存在するレコード数を返します。

パラメータ
TableName
テーブル名を指定します。

NumOfRec
レコード数が返されます。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。この関数内部ではBtrieveのstatオペレーションを発行します。

DbGetNumOfTable

Object.DbGetNumOfTable(NumOfTable As Integer) As Integer

概要
現在読み込まれている DDF に存在するテーブル数を返します。

パラメータ

NumOfTable
テーブル数が返ります。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。

DbGetPosBlock

Object.DbGetPosBlock(TableName As String, PosBlock(0 To 127) As Byte) As Integer

概要
指定されたテーブルに関連するBtrieveのポジション・ブロックを返します。

パラメータ
TableName
テーブル名を指定します。
PosBlock
BtrieveのPosBlockを保持するByte型の配列を指定します。配列のサイズはBtrieveの仕様により128バイトを割振る必要があります。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。

注意
ポジション ブロックは、Btrieve ファイルがオープンしている時にのみ有効となるため、このメソッドを呼び出す際には Btrieve ファイルがオープンしていることが必須です。ポジション ブロックは Btrieve が管理する領域なので、通常のアプリケーションでデータをセットしないでください。このメソッドで得られるポジション ブロックを利用して Btrieve API を発行し、DDF に合致しないようなデータを登録した場合は、他のコントロールやメソッドでのオペレーションに障害が出る可能性があります。弊社では動作を保証できませんので、Btrieve のデータ型、オペレーション、プログラミングを十分理解した上でのご利用をお願いします。

DbGetPosition

Object.DbGetPosition(TableName As String) As Long

概要
指定されたテーブルの現在のレコードの物理位置を返します。

パラメータ
TableName
テーブル名を指定します。

戻り値
正常ならば物理レコード位置(4バイト）が返されます。テーブル ID が正しくない、またはデータベースがオープンされていない場合は -1 が返されます。

注意
戻り値はシリアルな値ではなく、Btrieveで管理されるユニークな値です。整数値でレコードを識別したい場合は、AutoIncrement型のフィールドを利用します。

DbGetRecordLength

Object.DbGetRecordLength(TableName As String,
RecLen As Integer) As Integer

概要
指定されたテーブルに関連するBtrieveファイルのレコード長をバイト単位で返します。

パラメータ
TableName
テーブル名を指定します。
RecLen
レコード長が返される2バイト長の整数を指定します。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。

注意
当メソッドが呼び出される時点でDDFが読み込まれている必要があります。

DbGetTableName

Object.DbGetTableName(TableID As Integer, TableName As String)As Integer

概要
テーブルIDを指定してテーブル名を取得します。

パラメータ
TableID
テーブルIDを指定します。0ベースの値を指定します。
TableName
テーブル名が返されます。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。

注意
当メソッドが呼び出される時点でDDFが読みこまれている必要があります。

DbIsOpen

Object.DbIsOpen(TableName As String) As Integer

概要
指定されたテーブルに関連するBtrieveファイルのオープン状態を返します。

パラメータ
TableName
テーブル名を指定します。

戻り値
オープンしているなら値1が返されます。オープンしていない場合は0を返します。負の値はCompat Classエラーコード一覧を参照してください。

DbLoadDDF

Object.DbLoadDDF() As Integer

概要
DDFDirプロパティで指定されたDDFをロードします。実行時に参照するDDFを切り替えることができます。

パラメータ
なし

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

注意
デザイン時にDDFDirプロパティに設定したDDFと構造が異なるDDFを実行時に指定する場合は、該当コントロールにデータ・バインドするコントロールのDbField、DbTable,DbListTable、DbListFieldsプロパティの値に注意してください。新たに指定したDDFに定義されていないDbField、DbTable、DbListTable、DbListFieldsプロパティ値が設定されたコントロールの動作は保証されません。

DbOpen

Object.DbOpen(TableName As String) As Integer

概要
指定されたテーブルに関連するBtrieveファイルをオープンします。

パラメータ
TableName
テーブル名を指定します。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

注意
このメソッドを呼び出すときはBtrieveファイルはクローズしている必要があります。オープン・モードはこのメソッドが関連しているCompat ClassデータベースのOpenModeプロパティによって指定されます。OwnerNameについても同様です。

DbOpenAll

Object.DbOpenAll() As Integer

概要
DDFに定義されたBtrieveファイルをすべてオープン状態にします。

パラメータ
なし。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。それ以外はBtrieveのステータス・コードが返されます。

注意
オープン・モードはこのメソッドが関連しているCompat ClassデータベースのOpenModeプロパティによって指定されます。OwnerNameについても同様です。 DDF に定義されているファイルのうち、既にオープン中のファイルについては Open オペレーションは実行されません。また、オープン中である旨のエラーも返されません。

DbReset

Object.DbReset() As Integer

概要
Btrieveリセットオペレーションを発行します。

パラメータ
なし。

注意
リセット オペレーションはオープン中のファイルをすべてクローズするので、アプリケーションでオープンしているファイルが存在する場合には注意が必要です。リセットオペレーションの詳細についてはBtrieveのマニュアルをご参照ください。

DbSetFieldData

Object.DbSetFieldData(TableName As String,
FieldName As String,
Data As String) As Integer

概要
Btrieveデータベースへ登録するフィールドのデータを、指定されたテーブルのデータ・バッファに設定します。

パラメータ
TableName
テーブル名を指定します。

FieldName
フィールド名を指定します。

Data
フィールドの値を文字列で指定します。文字列型データからデータベースのフィールドのデータ型への変換はこの関数内部で行われます。Integer型などのバイナリ型も文字列で指定します。Date型は、"YY/MM/DD"または”YYYY/MM/DD”の形で指定します。Time型は”HH:MM:SS”の形で指定します。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。

サンプル
```
Dim rc%
rc% = DbSetFieldData(“給与”,”欠勤”,"10")
If rc%<> 0 then
  ' エラー処理
End If
```

DbSetFileName

Object.DbSetFileName(TableName As String, NewFileName As String) As Integer

概要
テーブルに関連するBtrieveファイル名を指定します。DDF定義を実行時に変更します。ファイルがオープンされている状態では変更はできません。

パラメータ
TableName
テーブル名を指定します。

NewFileName
Btrieveファイル名を指定します。ドライブ、パスまで含めることができます。ドライブ、パスを省略した場合はDDFが存在するディレクトリにあるBtrieveファイルを扱います。

戻り値
正常ならば0が返されます。負の値はCompat Classエラーコード一覧を参照してください。

DbSetLockBias

Object.DbSetLockBias(BiasValue As Integer) As Integer

概要
ロック・バイアス値を指定します。当メソッドで指定したロック・バイアス値はこのメソッド呼び出し以降のDbAccess/DbGetDirectメソッドに反映されます。DbAccessメソッドの第1パラメータにロック・バイアス値を毎回加算するコードを記述する必要がなくなります。

パラメータ
BiasValue
ロック・バイアス値を指定します。以下の値が指定可能です。ロック動作の詳細はBtrieveのマニュアル記載をご参照ください。

0	通常の状態
100	シングルウェイトロック
200	シングルノーウェイトロック
300	マルチウェイトロック
400	マルチノーウェイトロック

戻り値
正常終了の場合は0が返されます。負の値に関してはCompat Classエラーコード一覧をご参照ください。

サンプル・コード

```
Dim rc As Integer
Const TableName = “商品”
Const IndexName = “商品コード”

With VBMan
  rc = .DbSetLockBias(400) ‘ multi no wait lock
  If rc <> 0 Then Stop

  ‘全レコードをロックする。
  rc = .DbAccess(BTR_GET_FIRST,”商品”,”商品コード”)
  Do
    If rc <> 0 Then Exit Do
    Rc = .DbAccess(BTR_GET_NEXT,”商品”,”商品コード”)
  Loop
  rc = .DbSetLockBias(0)        ‘ バイアス解除
```

```
  Debug.Print rc
  rc = .DbUnlock("商品",-2)     'ロック解除
  Debug.Print rc
End With
```

DbUnlock

```
Object.DbUnlock(TableName As String,
                UnlockType As Integer) As Integer
```

概要
指定されたテーブルのレコード・ロックを解除します。

パラメータ
TabldName
テーブル名を指定します。
UnlockType
以下の値が有効です。

アンロックタイプ	詳細
0	シングル・レコード・ロックを解除する
-1	マルチ・レコード・ロックされている現在のレコードのみロックを解除する。
-2	マルチ・レコード・ロックのすべてを解除

戻り値
正常ならば0が返されます。テーブル ID が正しくない、またはデータベースがオープンされていない場合は -1 が返されます。

Appendix-A コードサンプル

Visual Basic.NETサンプルコード

チュートリアルで示されたC＃サンプル・コードのVisual Basic.NET版です。

```
Private Sub FillTable()
  Dim rc As Integer
  Dim demodata As New BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata")
  Dim person As BtLib.Record
  '''
  person = demodata.GetRecord("Person")
  person.Open()
  rc = person.Read(Operation.GetFirst)

  While (rc = 0)
    Dim tr As New TableRow()
    Dim c1 As New TableCell()
    Dim lt1 As New LiteralControl(person("ID").ToString())
    c1.Controls.Add(lt1)

    Dim c2 As New TableCell()
    Dim lt2 As New LiteralControl(person("First_Name").ToString())
    c2.Controls.Add(lt2)
    Dim c3 As New TableCell()
    Dim lt3 As New LiteralControl(person("Last_Name").ToString())
    c3.Controls.Add(lt3)
    '//
    tr.Cells.Add(c1)
    tr.Cells.Add(c2)
    tr.Cells.Add(c3)
    '//
    Table1.Rows.Add(tr)
    '//
    rc = person.Read(Operation.GetNext)
  End While
  person.Close()
End Sub
```

IdictionaryEnumerator利用方法サンプル・コード

以下のVisual　C#サンプルではRecordクラスのすべてのカラムについてデータを取得する方法を示します。RecordクラスはHashTableから導出されているのでIdictionaryEnumeratorを使ってすべてのカラムの名前と値を得ることができます。

```
BtLib.Ddf d = new BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata”);
BtLib.Record r = d.GetRecord("alltypes");
r.Open();
short rc = r.Step(BtLib.Operation.StepFirst);
listBox1.Items.Clear();
while(rc==0)
{
   IDictionaryEnumerator en = r.GetEnumerator();
   while(en.MoveNext())
   {
      listBox1.Items.Add(en.Value.ToString());
   }
   rc = r.Read(BtLib.Operation.StepNext);
}
r.Close();
```

Compat Class サンプル・コード

以下はVisual C#によるCompat Classサンプルコードです。

```
short rc;
BtLib.Compat vbm = new BtLib.Compat();
vbm.DDFDir = @“C:¥ProgramData¥Actian¥Zen¥Demodata";
rc = vbm.DbLoadDDF();
if( rc != 0 )
{
  MessageBox.Show(“load error “ + Convert.ToString(rc));
}
rc = vbm.DbOpen("Person");
if( rc != 0 )
{
  MessageBox.Show(“open error “ + Convert.ToString(rc));
}
//
```

```
rc = vbm.DbSetFieldData("Person","First_Name","Koichi");
rc = vbm.DbSetFieldData("Person","Last_Name","Adachi");
rc = vbm.DbAccess(Operation.GetEqual,"Person","Names");

listBox1.Items.Clear();
while(rc == 0)
{
  String fn = vbm.DbGetFieldData("Person","First_Name");
  String ln = vbm.DbGetFieldData("Person","Last_Name");
  listBox1.Items.Add(fn + " " + ln);
  rc = vbm.DbAccess(Operation.GetNext,"Person","Names");
}
//
rc = vbm.DbAllClose();
```

GetDataSet C#サンプル

Visual C#でWindows FormのDataGridにデータを表示するサンプルです。RecordクラスのGetDataSetには表示するカラム名と表示するレコード数を指定することができます。

```
try
{
   string [] cols = { "ID", "First_Name", "Last_Name" };
   BtLib.Ddf d = new BtLib.Ddf(@"C:¥ProgramData¥Actian¥Zen¥Demodata");
   BtLib.Record r = d.GetRecord("person");
   r.Open();
   DataSet ds = r.GetDataSet(cols,100);
   dataGrid1.SetDataBinding(ds,"person");
   r.Close();
}
catch( System.Exception er)
{
   System.Diagnostics.Debug.WriteLine(er.ToString());
}
```

WebFormにおけるDataGridとのDataBind C#サンプル

WebFormでよく利用される.NET Framework のDataGridにも簡単にデータバインド可能です。以下はC#でのサンプルコードです。

```
private void Page_Load(object sender, System.EventArgs e)
{
  if( !IsPostBack )
  {
    try
    {
      BtLib.Ddf d = new BtLib.Ddf(@“C:¥ProgramData¥Actian¥Zen¥Demodata");
      BtLib.Record r = d.GetRecord("person");
      r.Open();
      DataSet ds = r.GetDataSet();
      r.Close();
      DataGrid1.DataSource = ds;
      DataGrid1.DataMember = "person";
      DataBind();
    }
    catch( System.Exception er)
    {
      System.Diagnostics.Debug.WriteLine(er.ToString());
    }
  }
}
```

Native Class C#レコード・スキャン・サンプル

Native Classを使ったBtrieve API呼び出しサンプル・コードです。言語はC#です。先頭からレコードを読込みます。

```
byte [] posblk = new byte[128];
byte [] data   = Encoding.ASCII.GetBytes("\0\0");
Int16 dataLength = 0;
byte [] keyBuf = Encoding.ASCII.GetBytes(@“C:¥ProgramData¥Actian¥Zen¥Demodata¥person.mkd\0”);
Int16 keyNum = 0;  // normal open.
Int16 rc;
short keyBufLen = 128;

// get shift jis encoder
Encoding sje = Encoding.GetEncoding("shift-jis");
// open...
rc = Native.BtrCall(Operation.Open,
```

```
                            posblk,data,
                            ref dataLength,
                            keyBuf,
                            (short)keyBuf.Length,
                            keyNum);
// get first
data = new Byte[334];
dataLength = (short)data.Length;
keyBuf = new Byte[128];
rc = Native.BtrCall(Operation.GetFirst,
                            posblk,data,
                            ref dataLength,
                            keyBuf,
                            keyBufLen,
                            0);

listBox1.Items.Clear();

while(rc == 0)
{
  //
  string firstName = Native.Trim(sje.GetString(data,9,16)); //
  string lastName = Native.Trim(sje.GetString(data,26,26)); //
  listBox1.Items.Add(firstName + " " + lastName);
  // get next.
  rc = Native.BtrCall(Operation.GetNext,
                              posblk,data,
                              ref dataLength,
                              keyBuf,
                              keyBufLen,
                              0);
}
// close!
rc = Native.BtrCall(Operation.Close,
                            posblk,data,
                            ref dataLength,
                            keyBuf,
                            keyBufLen,
                            0);
```

Native Class VB.NET レコード・スキャン・サンプル

Native Classを使ったBtrieve API呼び出しサンプル・コードです。言語はVB.NETです。GET EQUAL オペレーションを使って、指定したレコードからデータを読込みます。

```
Imports System
Imports System.Text
Imports System.Text.Encoding
Imports BtLib
‘…
‘
Dim posblk(128) As Byte
Dim data(334) As Byte
Dim dataLength As Int16

Dim keyBuf(128) As Byte
Dim fname() As Byte

Dim keyNum As Int16
Dim rc As Int16
Dim keyBufLen As Int16
Dim i As Integer
Dim firstName As String, lastName As String
Dim tmp As String
Dim c() As Char
Dim sje As Encoding

keyBufLen = 128

' get shift jis encoder
sje = System.Text.Encoding.GetEncoding("shift-jis")

fname = Encoding.ASCII.GetBytes(“C:¥ProgramData¥Actian¥Zen¥Demodata")
data(0) = 0
dataLength = 0

' open...
rc = Native.BtrCall(Operation.Open, _
                     posblk, _
                    data, _
                    dataLength, _
                    fname, _
                    fname.Length, _
```

```
                    keyNum)

' get first
dataLength = data.Length

tmp = "Adachi"
keyBuf(0) = 0

c = tmp.ToCharArray()

For i = 0 To 5
    keyBuf(i + 1) = AscW(c(i))
Next

tmp = "Koichi"
c = tmp.ToCharArray()
For i = 0 To 5
   keyBuf(i + 28) = AscW(c(i))
Next

keyNum = 1
rc = Native.BtrCall(Operation.GetEqual, _
                    posblk, _
                    data, _
                    dataLength, _
                    keyBuf, _
                    keyBufLen, _
                    keyNum)

ListBox1.Items.Clear()

While rc = 0
  firstName = Native.Trim(sje.GetString(data, 9, 16))
  lastName = Native.Trim(sje.GetString(data, 26, 26))

    ListBox1.Items.Add(firstName + " " + lastName)
   ' get next.
   rc = Native.BtrCall(Operation.GetNext, _
                       posblk, _
                       data, _
                       dataLength, _
```

```
                    keyBuf, _
                    keyBufLen, _
                    keyNum)
End While
' close!
rc = Native.BtrCall(Operation.Close, _
                            posblk, _
                            data, _
                            dataLength, _
                            keyBuf, _
                            keyBufLen, _
                            0)
```

Native Class C#インサート・サンプル・コード

Native Classを使ったBtrieve API呼び出しサンプル・コードです。レコードを登録します。.NETデータ型をByte型配列に変換してセットする部分のコードがポイントになります。Dobule 型等をバイト配列に変換するのは、通常の .NET Framework の機能では困難と思われたため、Native Class にヘルパー メソッドとして、.NET Framework の各データ型から Byte 配列へ変換するための GetBytes メソッドをご用意しました。以下のサンプルでも文字列などはEncodingクラスのGetBytesメソッドでByte配列を得ていますが、Double型はNativeクラスのGetBytesメソッドを使っています。

```
byte[] posblk = new byte[128];
byte[] data   = Encoding.ASCII.GetBytes("\0\0");
Int16 dataLength = 0;
byte[] keyBuf = Encoding.ASCII.GetBytes(@"C:¥ProgramData¥Actian¥Zen¥Demodata¥test.mkd\0");
Int16 keyNum = 0;  // normal open.
Int16 rc;
short keyBufLen = 128;

// get shift jis encoder
Encoding sje = Encoding.GetEncoding("shift-jis");

// open...
rc = Native.BtrCall(Operation.Open,
                    posblk,
                    data,
                    ref dataLength,
                    keyBuf,
```

```
                    (short)keyBuf.Length,
                    keyNum);

// insert
data = new byte[80];
dataLength = (short)data.Length;
keyBuf = new byte[128];

// setting up data
Int32 id = Convert.ToInt32(txtID.Text);

byte [] byteId = Native.GetBytes(id);
Buffer.BlockCopy(byteId,0,data,1,4);

byte [] byteName = sje.GetBytes(txtName.Text);
Buffer.BlockCopy(byteName,0,data,6,byteName.Length);

byte [] byteDesc = sje.GetBytes(txtDesc.Text);
Buffer.BlockCopy(byteDesc,0,data,37,byteDesc.Length);

DateTime dt = System.DateTime.Now;
byte [] byteDate = Native.GetBytes(dt);
Buffer.BlockCopy(byteDate,0,data,69,byteDate.Length);
// 76, 4, 1 numeric
Decimal d = 12.3M;
byte [] byteNumeric = Native.GetBytes(d,BtrieveTypes.numeric,4,1);
Buffer.BlockCopy(byteNumeric,0,data,74,4);

rc = Native.BtrCall(Operation.Insert,
                    posblk,
                    data,
                    ref dataLength,
                    keyBuf,
                    (short)keyBuf.Length,
                    keyNum);
if( rc != 0 )
{
  MessageBox.Show("insert error rc = " + rc.ToString(),"error");
}
// close!
rc = Native.BtrCall(Operation.Close,
```

```
                posblk,
                data,
                ref dataLength,
                keyBuf,
                keyBufLen,
                0);
```

差分DataSetをデータベースに反映するサンプル

DataSet クラスに対して行われた変更は、GetChanges メソッドを呼び出すことで取得できます。

```
m_ds = (DataSet)Session["ds"];
DataSet uds = m_ds.GetChanges(DataRowState.Modified);

if(uds.HasErrors) // DataSetのエラーチェック
{
  // エラーの表示コード（省略）
}
try
{
  BtLib.Ddf d = new BtLib.Ddf(@"C:¥ProgramData¥Actian¥Zen¥Demodata");
  BtLib.Record r = d.GetRecord("person");
  r.Open();
  r.Index = "PersonID";
 BtLib.Transaction.Begin();
  int i;
  int j;
  short rc;
  DataTable dt = uds.Tables["Person"];
  for(i=0; i < dt.Rows.Count; i++)  // 変更されたレコード分ループ
  {
    // キーのみ転送
    r[dt.Columns[0].ColumnName.ToString()]
      =dt.Rows[i][0].ToString();
    // get equal を実行
    rc = r.Read(BtLib.Operation.GetEqual);
    // データを転送
    for(j=0; j < dt.Columns.Count ; j++)
    {
        r[dt.Columns[j].ColumnName.ToString()]
```

```
            = dt.Rows[i][j].ToString();
      }
      rc = r.Write(BtLib.Operation.Update);
    }
    BtLib.Transaction.End();
    r.Close();
}
catch (BtLib.Exception ex)
{
    System.Diagnostics.Debug.WriteLine(ex.ToString());
}
```

構造体でデータ領域を指定するNative Class C# サンプル

構造体をデータ領域として指定してNative Classでデータを読み込むC#サンプルです。

```
byte [] pb = new Byte[128];
byte [] data = new byte[1];
Int16 dataLength = 0;
byte [] keyBuf = Encoding.ASCII.GetBytes(@"C:¥ProgramData¥Actian¥Zen¥Demodata");
Int16 keyNum = 0;
Int16 rc;
Department dept = new Department();

rc = Native.BtrCall(Operation.Open,
                    pb, data,
                    ref dataLength,
                    keyBuf,
                    (short)keyBuf.Length,
                    keyNum);
if(rc != 0)
{
    return;
}

// clear the list box
listBox1.Items.Clear();

// get first.
dataLength = (short)Marshal.SizeOf(dept);
```

```
System.IntPtr ptr = Marshal.AllocCoTaskMem(dataLength);
Marshal.StructureToPtr(dept,ptr,true);

rc = Native.BtrCall(Operation.GetFirst,
                    pb,
                    ptr,
                    ref dataLength,
                    keyBuf,
                    (short)keyBuf.Length,
                    0);

while(rc != 9)
{
   Marshal.PtrToStructure(ptr,dept);
   string line = dept.Name + "\t" + dept.Billing_Name + "\t" +
Native.GetDecimal(dept.Phone_Number,0,6,0,BtrieveTypes.@decimal) + "\t" +
dept.Head_Of_Dept + "\t" + dept.Room_Number;

   listBox1.Items.Add(line);
   // get next
   rc = Native.BtrCall(Operation.GetNext,
                    pb,
                    ptr,
                    ref dataLength,
                    keyBuf,
                    (short)keyBuf.Length,
                    0);
}

Marshal.FreeCoTaskMem(ptr);
// close
rc = Native.BtrCall(Operation.Close,
                   pb,data,
                   ref dataLength,
                   keyBuf,
                   (short)keyBuf.Length,0);
```

C# 構造体定義サンプル

上記サンプルで使った構造体の定義例です。Zen/PSQLのdemodataのDepartmentサンプルデータにつ

いて定義した構造体です。ストラクチャービルダーで自動生成することができます。

```
[StructLayout(LayoutKind.Sequential, Pack=1,CharSet=CharSet.Ansi)]
public class Department
{
   [MarshalAs(UnmanagedType.ByValTStr, SizeConst=20)]
   public string  Name;         // char 20
   public byte    nf1;          // null flag for Phone_Number
   [MarshalAs(UnmanagedType.ByValArray,SizeConst=6,ArraySubType=UnmanagedType.U1)]
   public byte [] Phone_Number = new byte[6];   // decimal 6
   [MarshalAs(UnmanagedType.ByValTStr, SizeConst=25)]
   public string  Billing_Name; // char 25
   public int     Room_Number;  // unsigned 4
   public Int64   Head_Of_Dept; // unsinged 8
}
```

構造体でデータ領域を指定するNative Class VB.NETサンプル

構造体をデータ領域として指定してNative Classでデータを読み込むVB.NETサンプルです。

```
Dim pb(128) As Byte
Dim data(1) As Byte
Dim dataLength As Int16 = 0
Dim keyBuf() As Byte
  = Encoding.ASCII.GetBytes("C:¥ProgramData¥Actian¥Zen¥Demodata¥Dept.mkd")
Dim keyNum As Int16 = 0
Dim rc As Int16
Dim line As String
Dim dept As Department = New Department()
Dim ptr As System.IntPtr

rc = Native.BtrCall(Operation.Open, _
                    pb, data, _
                    dataLength, _
                    keyBuf, _
                    keyBuf.Length, _
                    keyNum)
If rc <> 0 Then
  Exit Sub
```

```
End If

' clear the list box
ListBox1.Items.Clear()

' get first.
dataLength = Marshal.SizeOf(dept)
ptr = Marshal.AllocCoTaskMem(dataLength)
Marshal.StructureToPtr(dept, ptr, True)

rc = Native.BtrCall(Operation.GetFirst, _
                    pb, _
                    ptr, _
                    dataLength, _
                    keyBuf, _
                    keyBuf.Length, _
                    0)

While rc <> 9
  Marshal.PtrToStructure(ptr, dept)
  line = dept.Name & vbTab & dept.Billing_Name & vbTab &
Native.GetDecimal(dept.Phone_Number, 0, 6, 0, BtrieveTypes.decimal) & vbTab &
CStr(dept.Head_Of_Dept) & vbTab & CStr(dept.Room_Number)
  ListBox1.Items.Add(line)
  ' get next
  rc = Native.BtrCall(Operation.GetNext, _
                      pb, _
                      ptr, _
                      dataLength, _
                      keyBuf, _
                      keyBuf.Length, _
                      0)
End While

Marshal.FreeCoTaskMem(ptr)

' close
rc = Native.BtrCall(Operation.Close, _
                    pb, data, _
                    dataLength, _
                    keyBuf, _
```

```
                keyBuf.Length, 0)
```

VB.NET構造体定義サンプル

上記サンプルで使った構造体の定義例です。Zen/PSQLのデモデータのDepartmentサンプルデータについて定義した構造体です。ストラクチャービルダーで自動生成することができます。

```
<StructLayout(LayoutKind.Sequential, pack:=1, CharSet:=CharSet.Ansi)> _
Public Class Department
  <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=20)> _
  Public Name As String              ' char 20
  Public nf1 As Byte                 ' null flag for Phone_Number
  <MarshalAs(UnmanagedType.ByValArray, SizeConst:=6, ArraySubType:=UnmanagedType.U1)>
_
  Public Phone_Number(6) As Byte     ' decimal 6
  <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=25)> _
  Public Billing_Name As String      ' char 25
  Public Room_Number As Int32        ' unsigned 4
  Public Head_Of_Dept As Int64       ' unsinged 8
End Class
```

Appendix-B FAQ – よくあるご質問

この章では、アプリケーション・プログラミングやシステム・セットアップに共通の問題、疑問などの解説をします。

Pervasive.SQL V8.6セキュアデータベースについて

英語版Pervasive.SQL V8.5初期版利用時等にはDdfクラス生成時にオーナーエラー(51)が発生する場合があります。SP2以降にアップグレードするか、オーナー名をクリアするツール等が利用可能ですのでサポートまでお問い合わせください。

Web実行でのステータス 94について

ASP .NET Web アプリケーションを実行すると、レコード クラスの Open メソッド等により Btrieve アクセスが初めて発生した時点で、Btrieve ステータス 94 の例外が発生することがあります。ASP.NETの実行ユーザーがAdministratorとしてZen/PSQL/Btrieveに認識されることが原因です。ASP.NETの実行ユーザーを変更することでこの問題を回避できます。具体的には新しくASP.NET実行ユーザーを登録してそのユーザーをmachine.configファイルのprocessModel タグに指定します。以下は手順です。

1. 新しいユーザーを定義します。
2. 以下のグループに所属させます。
   Administrators
   <マシン名> admins
   <マシン名> authers
   <マシン名> browsers
   VS Developers
3. <windir>\Microsoft.NET\Framework\<.net version>\configにあるmachine.configを編集します。processModelタグにあるuserNameとpasswordを上記で定義したユーザーIDとして編集し保存します。
4. パソコンを再起動します。

以下はmachine.configファイルの設定例です。

<processModel enable="true" timeout="Infinite" idleTimeout="Infinite"
shutdownTimeout="0:00:05" requestLimit="Infinite" requestQueueLimit="5000"
restartQueueLimit="10" memoryLimit="60" webGarden="false" cpuMask="0xffffffff"
userName="pvsw" password="password" logLevel="Errors" clientConnectedCheck="0:00:05"
comAuthenticationLevel="Connect" comImpersonationLevel="Impersonate"

responseRestartDeadlockInterval="00:09:00" responseDeadlockInterval="00:03:00"
maxWorkerThreads="25" maxIoThreads="25"/>

ドメインコントローラーでサンプルが動作しない

ドメイン コントローラー特有のアカウントでサービスを動作させると、ドメイン コントローラーで当製品のウェブ サンプルが動作しない場合があります。この問題は弊社製品のサンプルに特有の問題ではなくIISの構成で解決できます。設定の詳細マイクロソフトのサポートURLが削除されましたので、マイクロソフトサポートにお問いわせいただくか、ドメインコントローラー以外のサーバーにあるIISでご確認ください。

DDFファイルとは？

Zen/PSQL/Btrieveデータベースの情報を保持するファイルです。 具体的には、FILE.DDF、FIELD.DDF、INDEX.DDF の 3 つのファイルは同一のディレクトリに存在する必要があります。この３つのファイル自体もBtrieveデータファイルです。古いバージョンのBtrieveエンジンで作成したDDFファイルについては上位バージョンのZen/PSQLで読み込むことができますが、逆に新しいZen/PSQLのファイル形式で作成したDDFを古いBtrieveエンジンで読み込むような場合にはステータス30が返されることがありますのでご注意ください。

DDFファイルとデータファイルを別フォルダーに置きたい

データファイル名にパス名を含めることで基本的に可能です。ただしDDFの仕様ではデータファイルを格納する領域が64バイトしかありませんので、最近の長いファイル名等を使っている場合には注意が必要です。

文字列のフィールドを定義したが先頭1バイトがずれているようだ

Zen/PSQLでカラムを「ヌル値許可」で作成すると先頭に 1 バイト、「真のヌル値」(True Nullable)を保持する領域をレコード上に確保します。Btrieve 6.15等古いDDFの形式はこのような仕様はあてはまりません。GET_EQUAL等のキーを参照するオペレーションを発行する際にもキーバッファの先頭には1バイト、「真のヌル値」を格納することが必要です。

Appendix-C Data Typeについて

Zen/PSQLにおけるデータ型とそのBtrieveデータ型へのマッピング、Btrieve Classes for .NET のRecord/Extendedクラスにおける.NET Frameworkデータ型へのマッピングを以下に示します。

Zen/PSQL データ型	Btrieve データ型	.NET データ型(*)
AutoTimestamp	AutoTimestamp	DateTime
Bfloat4	Bfloat	Single
Bfloat8	Bfloat	Double
Bigint	Integer	Int64
Binary	Char	Byte []
Bit	Bit	Boolean
Char	Char	String
Currency	Currency	Decimal
Date	Date	Datetime
Decimal	Decimal	String
Double	Float	Double
Float	Float	Single
Identity	Autoinc	Int32
Integer	Integer	Int32
Longvarbinary	Blob	Byte
Longvarchar	Blob	String
Money	Money	Decimal
Numeric	Numeric	String
Numericsa	Numericsa	String
Numericsts	Numericsts	String
Real	Float	Single
Smallidentity	Autoinc	Int32
Smallint	Integer	Int32
Time	Time	Datetime
Timestamp	Timestamp	Datetime
Timestamp2	Timestamp2	DateTime
Tinyint	Integer	Byte
Ubigint	Unsigned	Int64
Usmallint	Unsigned	Int32
Utinyint	Unsigned	Byte
Varchar	Zstring	String

(*) Btrieve Classes の動作ではStringに変換される場合もありますので適宜Convertクラスを利用してターゲット・データ型に変換してください。

Appendix-D Exception Class エラーコード一覧

この章では、Exception Classのエラーコードを解説します。エラーコードは製品の改良のために予告なく追加、変更される場合がありますのであらかじめご了承ください。

OutOfMemroy	100	「一時的なメモリを確保できません」システムのメモリを増やす、スワップを増やす、同時に稼動しているアプリケーションやサービスを止めることで、状況を回避できる場合があります。
DdfOpenError	101	「DDFファイルをオープンできません」Ddfクラスに指定しているDDFファイルへのパスを確認してください。C#の場合バックスラッシュ（円記号）を２重に記述していない場合等が考えられます。DDFファイルへのパスが正しいと思われる場合、Zen/PSQL/Btrieve環境の設定に問題があると思われます。スローされた例外のBtrieveStatusプロパティにBtrieveエンジンからのステータスが保持されていますので、この値を参照してBtrieveの設定に関する問題を解決してください。
DdfReadError	102	「DDF読込みエラー」DDFが何らかの原因で不整合な状態になっています。Zen/PSQL Control Center 等で DDF ファイルを正しく読み書きできるか確認してください。DdfOpenErrorと同様にZen/PSQL/Btrieveエンジンから0以外の値が返される場合にはスローされた例外インスタンスのBtrieveStatusプロパティを参照することでエラーの詳細情報を得られます。
DdfCloseError	103	「DDFクローズエラー」DDFファイルをクローズするときにBtrieveから0以外のステータスが戻されました。
DdfInvalid	104	「DDFが不正です」Loadメソッド実行時にDdfDirプロパティが指定されていませんでした。
DdfAlreadyLoaded	105	「DDFはすでにロードされています」DDFがロードされている状態で２度目のLoadメソッド呼び出しが実行されています。
DdfNotLoaded	106	「DDFがロードされていません」DDFがロードされていない状態でGetRecordメソッド等、DDF情報が必要とされるメソッド呼び出しやプロパティ参照が実行されました。LoadメソッドでDDFをロードしてください。
OwnerNameTooLong	107	「オーナー名長が不正です」オーナー名がZen/PSQL/BtrieveのDDF仕様で定められているサイズより長い文字列がセットされています。
InvalidKey	108	「キーが不正です」データベースのテーブル定義にないキー名が指定されました。
InvalidTableName	109	「テーブル名が不正です」データベース定義にないテーブル名が指定されました。

InvalidFieldName	110	指定したカラムが指定したテーブルに含まれません。
InvalidRecordSize	111	「レコード長が正しくありません」DDF定義と実際のBtrieveデータのレコード長が合致していません。レコード定義を今一度ご確認ください。
InvalidOperation	112	「オペレーションコードが不正です」指定したオペレーション・コードは指定したメソッドでは使えません。
ExpandFileName	113	「ファイル名を変換できません」Btrieveデータファイルを短いファイル名に変換する際にWin32 APIからエラーが返されました。
OpenDataFile	114	「データファイルをオープンできません」Btrieveファイルをオープンできませんでした。Btrieve　Openオペレーションに失敗しています。Btrieveステータスコードを調べて、状況を解釈して対応してください。一般的には他のプロセスで排他モードですでにデータ・ファイルがオープン状態にある場合が多いと思われます。
StringConversion	115	「文字列変換に失敗しました」呼び出し言語のマネージドコードにある文字列をアンマネージドに変換する場合にエラーが発生しました。
DataTypeNotSupported	116	「サポートされていないデータ型です」DDFに定義されいてるデータ型は当製品ではサポートされていません。
InvalidSearchCond	117	「検索条件が不正です」extended系のオペレーション実行時にSearchCondプロパティに指定した文字列が正しくありません。今一度ご確認ください。
DuplicateFieldName	118	「フィールド名が重複しています」AddFieldメソッド等ですでに同じ名前のフィールドが追加済みです。
VariableLenghtNotSupported	119	「Extended系オペレーションで可変フィールドはサポートされません」
InvalidOperator	120	「オペレータが不正です」extended系オペレーションの検索条件指定時に演算子の指定が正しくありません。正しい演算子を指定してください。
NoFieldSpecified	121	「フィールドが指定されていません」Extended系オペレーション実行時に取得するフィールドが指定されていません。
InvalidMaxRecords	122	「最大レコード数が不正です」Extended系オペレーション実行時にMaxRecordの指定が正しくありません。
DataConversion	123	データ型変換ができませんでした。変換に関連するデータ型はサポートされていません。
InvalidParameter	124	メソッドに指定したパラメータ値が不正です。範囲指定が存在するパラメータの場合は今一度マニュアルでご確認ください。
InvalidDataTable	125	Fill メソッドに指定された DataSet は、Extended オブジェクトで返された DataSet 以外のオブジェクトであると思われます。

BtrieveError	126	Btrieveから処理を継続することのできない致命的なエラーが返されました。例外のパラメータに含まれるBtrieveStatusに処理を中断する原因になったZen/PSQL/Btrieveのステータスコードが保持されていますので、この値をZen/PSQLのマニュアル等で詳細を調べて対処してください。
LogInFail	127	Pervasive.SQL V8.6以降のセキュアデータベースに対してログインAPIを実行しましたが、正常なステータスが返されませんでした。例外クラスのBtrieveStatusプロパティ等を参照してBtrieveデータベースから返されるステータスを調査し対応してください。
LogOutFail	128	Pervasive.SQL V8.6以降のセキュアデータベースに対してログインAPIを実行しましたが、正常なステータスが返されませんでした。例外クラスのBtrieveStatusプロパティ等を参照してBtrieveデータベースから返されるステータスを調査し対応してください。
InvalidLockBias	129	トランザクション開始時に指定したトランザクション・ロック・バイアス値が正しくありません。パラメータを見直して正しい値を指定してください。

Appendix-E Compat Classエラーコード

Compatクラスでは、メソッドや関数の戻り値にマイナスの値を返す場合があります。以下はこれらのVBManと互換性があるエラーコードについての説明です。

エラー・シンボル	値	詳細
VBM_ERR_GENERIC	-1	一般的なエラー。これ以外のエラーに含めることができないもの。
VBM_ERR_ALREADY_OPEN	-2	Btrieve　ファイルがオープンしていない時に実行する必要のあるメソッド/関数が、オープンしているファイルに対して実行されました。
VBM_ERR_NOT_OPEN	-3	Btrieve　ファイルがオープンしている時に実行する必要のあるメソッド/関数が、オープンしていないファイルに対して実行されました。
VBM_ERR_INVALID_TABLE_NAME	-4	テーブル名がDDFDirプロパティで指定されるDDFに存在しません。
VBM_ERR_INVALID_FIELD_NAME	-5	フィールド名が存在しません。
VBM_ERR_INVALID_KEY	-6	インデックス名が存在しません。
VBM_ERR_INVALID_OPCODE	-7	Btrieveのオペレーション・コードとして指定できない値を設定しました。
VBM_ERR_INVALID_EXCOND	-8	Extended系のBtrieveオペレーションを発行するメソッド/関数で検索条件の指定が不正でした。
VBM_ERR_INVALID_EXTRACTOR	-9	Extended系のBtrieveオペレーションを発行するメソッド/関数で抽出するフィールドの指定が不正です。
VBM_ERR_LOCK_BIAS	-10	トランザクション関連のメソッド/関数などで、ロック値として不正な値を設定されました。
VBM_ERR_CONTROL_NOT_FOUND	-11	DbAttachControl関数で最初のパラメータが正しくありません。
VBM_ERR_NOT_ATTACHED	-12	DbAttachControl関数が呼び出されていない状態でバージョン1.xコンパチブル関数が呼び出されました。
VBM_ERR_INVALID_FIELD_ID	-13	フィールドIDが正しくありません。
VBM_ERR_INVALID_KEY_ID	-14	インデックスIDが正しくありません。
VBM_ERR_INVALID_TABLE_ID	-15	テーブルIDが正しくありません。
VBM_ERR_EXPAND_FILE_NAME	-16	DDFDirを短いファイル名に変換する際にエラーが発生しました。ファイル・システムの破損が考えられます。ScanDisk等でエラーが発生するファイル・システムを修復してください。
VBM_ERR_DATA_TYPE	-17	配列で変数を指定する仕様のメソッドにおいてパラメータのデータ型が不正です。

VBM_ERR_ARRAY_DIMS	-18	１次元配列が指定されたメソッドのパラメータにそれ以外の次元の配列が指定されました。
VBM_ERR_OUT_OF_RANGE	-19	データを受取る配列のサイズが不充分です。またはデータを配列で設定するメソッドの場合、配列のインデックス範囲がフィールドIDの範囲を越えています。
VBM_ERR_DDF_LOAD	-20	DDFがロードされていない状態でDDF情報が必要とされるメソッド、コントロールが利用されています。アプリケーションはDbLoadDDFメソッドで事前にDDFを読み込ませる必要があります。
VBM_ERR_INVALID_DDF_DIR	-21	DbLoadDDFメソッドが呼び出されましたが、DDFDirプロパティに正しい値が設定されていませんでした。
VBM_ERR_ARRAY_ACCESS	-22	指定された配列をアドレスに変換するWin32 APIがエラーを返しました。Visual Basic等の言語では通常起こり得ないシステム・エラーです。言語やシステムの再インストールをお勧めします。
VBM_ERR_INVALID_ARRAY_TYPE	-23	配列で変数を指定する仕様のメソッドにおいてパラメータのデータ型が不正です。
VBM_ERR_INVALID_ARRAY_DIM	-24	１次元配列が指定されたメソッドのパラメータにそれ以外の次元の配列が指定されました。

Appendix-F バージョン履歴

バージョン8.0の新機能について

当クラスライブラリバージョン8.0では以下の機能を追加いたしました。

1. Actian Zen v14, PSQL v13 サポート
2. Visual Studio 2017, Visual Studio 2019 サポート
3. Windows 10 / Windows Server 2016/2019 サポート
4. Metadata V2 サポート
5. DDFクラスにて CLIENT ID サポート
6. Extended クラスにてマルチスレッド対応
7. 機能向上と軽微な不具合の訂正

バージョン7.0の新機能について

当クラスライブラリバージョン7.0では以下の機能を追加いたしました。

1. PSQL v12をサポート
2. PSQL v12 のUnicode文字列データ型をサポート
3. Visual Studio 2015をサポート
4. スタンドアロン ストラクチャービルダーを追加
5. 機能向上と軽微な不具合の訂正

当クラスライブラリは以前のバージョンと上位互換性を保っています。

バージョン6.0の新機能について

当クラスライブラリバージョン6.0では以下の機能を追加いたしました。

1. PSQL Vx Server 11をサポート
2. .NET Framework 4.0ベースに変更
3. Visual Studio 2013 をサポート
4. Windows 8.1 をサポート
5. 機能向上と軽微な不具合の訂正

当クラスライブラリは以前のバージョンと上位互換性を保っています。

バージョン5.0の新機能について

当クラスライブラリバージョン5.0では以下の機能を追加いたしました。

1. PSQL v11 SP3 をサポート
2. Visual Studio 2012 をサポート
3. Windows 8 / Server 2012 をサポート
4. 機能向上と軽微な不具合の訂正

当クラスライブラリは以前のバージョンと上位互換性を保っています。

バージョン4.0の新機能について

当クラスライブラリバージョン4.0では以下の機能を追加いたしました。

1. PSQL v11 をサポート
2. PSQL summit v10 SP3 をサポート
3. Visual Studio 2010 をサポート
4. Windows 7をサポート

当クラスライブラリは以前のバージョンと上位互換性を保っています。

バージョン3.0の新機能について

当クラスライブラリバージョン3.0では以下の機能を追加いたしました。

1. PSQL summit v10 をサポート
2. Visual Studio 2008 をサポート
3. RecordクラスにLINQサポートメソッドを追加
4. 64Bit OSサポート (Windows Server 2008 64bit, Windows Vista 64bit, Windows Server 2003 64bit)

当クラスライブラリは以前のバージョンと上位互換性を保っています。

バージョン2.0の新機能について

当クラスライブラリバージョン2.0は .NET Framework バージョン2.0をサポートいたします。また、Visual Studio 2005 IDEをサポートします。以前のバージョンとは完全に互換性を保っています。

バージョン1.2の新機能について

バージョン1.2ではPervasive.SQL V8.6のセキュリティ機能に対応いたしました。Ddfクラスのコンストラクタにコネクションストリングが指定できるように変更になり、LogIn/LogOutメソッドが追加されました。

バージョン1.1の新機能について

バージョン1.1では主にMicrosoft .NETで提供されるバイトアライメント制御機能を用いた構造体を使いBtrieve/PSQLデータのレコードイメージを直接入出力する機能を追加しました。この構造体サポートに関連するクラスはNative/Recordとなります。また、このバイトアライメントをサポートする構造体を定義するのは非常にワークロードを使う作業と思われましたので、DDFからこのタイプの構造体を自動生成するMicrosoft Visual Studio.NET用のアドインソフト、ストラクチャービルダーを添付しました。また当バージョンではVisual Studio .NET 2003での動作を確認しサポート環境といたしました。

Btrieve Classes for .NET Version 9.0

プログラミングガイド

第1版
2022年9月30日 第1刷発行