Btrieve Classes for .NET

Version 1.20

Programming Manual

TechKnowledge

目次

‘

製品概要

はじめに

Btrieve Classes for .NET は Pervasive.SQL 2000i/Pervasive.SQL V8/ Pervasive .SQL V8.6 専用の.NET 言語用データベース･アクセス･サポートクラスです。.NET では ADO.NET/OLE DB が標準ですがあえて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3～20倍という非常に高いパフォーマンスを実現しています。

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/Pervaisve 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。

３系統のクラス

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

- Compat
  弊社製品 VBMan ActiveX Controls for Btrieve のメソッドとコンパチブルな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。

- 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。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出来ます。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 が無い場合や、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 はパラメータの多く、レコードバッファからアプリケーション・データへの取出しや、格納するコードが必要になるため、アプリケーション・コード煩雑は煩雑になります。

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。従いましてご利用いただける言語は Visual Basic.NET/Visual C#/Visual J#となります。(2002/12 現在)今後、.NET Framework 環境を前提とした言語が多数予定されています。これらにつきましては日本での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を利用できる以下のタイプのアプリケーションを作成できます。

- Windows アプリケーション
- ASP.NET 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Pervaisve.SQLのコントロール・センター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を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 classではAppendix-Cに記載されるデータ型変換に従い、pervasive.SQLのデータ型を.NET frameworkデータ型に変換します。データ型の変換には基本的には.NET frameworkの型変換メソッドで変換しますが、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、逆の場合、Recordクラスに.NET framework データからBtrieveデータに変換する場合、変換に失敗した場合は例外を発生させます。データ変換例外が発生した場合でも.NET frameworkデータ型からToStringメソッドで文字列型に変換してデータをセットした場合は変換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の製品パッケージを開梱したときに発効します。一度開梱した商品の返品には応じることはできません。
- 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て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のユーザー登録書しない場合。
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、Pervasive.SQL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
昭栄神田橋ビル 3F

電話: 03-3293-5300
FAX: 03-3293-5250
E-Mail: info@agtech.co.jp
URL: http://www.agtech.co.jp

開発元

㈱ 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: www.techknowledge.co.jp

商標登録

本マニュアルに記載される商標、登録商標は該当各社の商標または登録商標です。

インストール

Btrieve Clases 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 または Visual Studio.NET または Visual Studio .NET 2003
② Pervasive.SQL 2000i /Pervasive.SQL V8/Pervasive.SQL V8.6

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。そのよう場合は Pervasive.SQL SDK マニュアルやサンプル･プログラムをご参照ください。

Setup の実行

Btrieve Classes for .NET のインストールについて説明します。以下はインストールの手順です。

① インストール CD-ROM をドライブに挿入します。
② Windows のエクスプローラー等で CD-ROM のルートにある setup.exe を起動します。
③ setup.exe の質問に答えて導入ボタンをクリックすると自動的に導入が終了します。
④ 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･グループ作成されます。
⑤ README.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>とした場合に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 c:\program files\techknowledge\Btrieve classes for .NET になります。

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には「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	ライブラリ本体	可
<instdir>\bin\sbCore.dll	ストラクチャビルダー	不可
<instdir>\bin\sbAddIn.dll	ストラクチャビルダーアドイン	不可
<instdir>\bin\sbAddInUI.dll	ストラクチャービルダーアドインユーザーインタフェース	不可
<instdir>\man\btlib120.pdf	PDFマニュアル	不可
<instdir>\man\readme_jp.html	お読みください	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\csDataSetSamp	C# DataSetオブジェクトサンプル	不可
<instdir>\samples\vbDataSetSamp	VB.NET DataSetオブジェクトサンプル	不可
<instdir>\samples\csWebSamp*.*	C# Web Formsサンプル・プログラム	不可
<instdir>\samples\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

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。説明に使うアプリケーションは Pervasive.SQL の demodata データベースにある Person 表を 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を作成します。

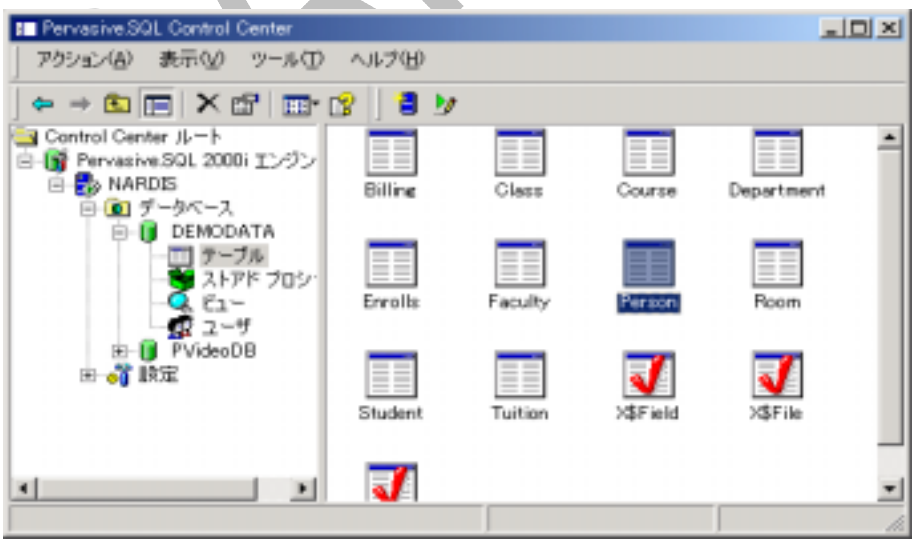
Btrieveデータベースの準備

Pervaisve.SQL のデータベース操作ツール「pervasive 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 Pervasive.SQL マニュアルをご参照ください。

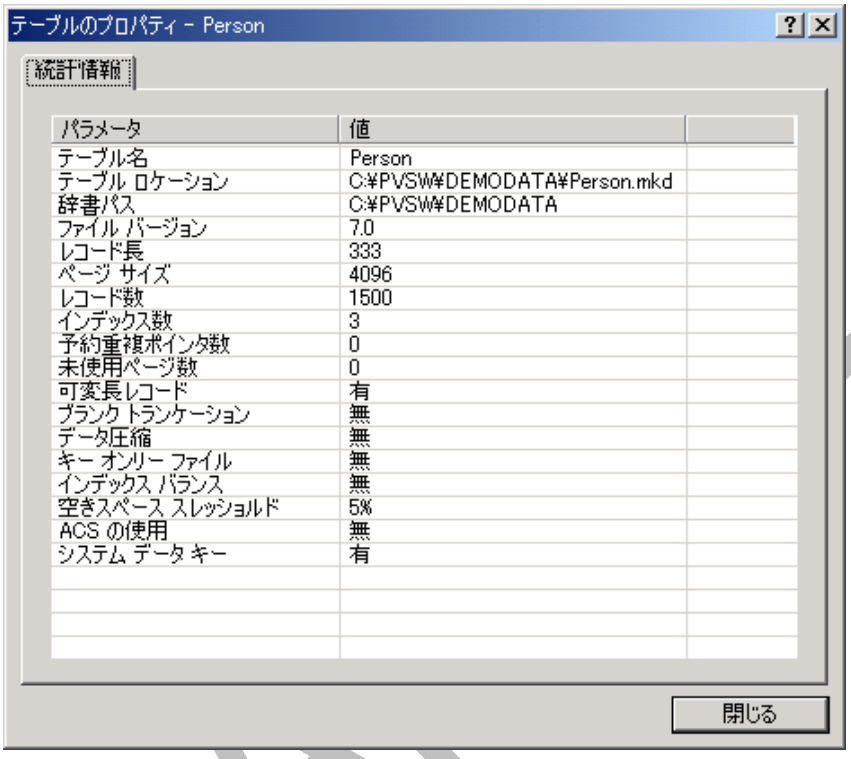
データ・ベース作成情報で重要なのはデータベースがディスクの何処のフォルダーに作成されているかを知ることです。データ・ベースが存在するディレクトリを DDFDir プロパティとしてプログラミング時に指定することが必要になるためです。

ここでは最初に Pervasive.SQL 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。左ベインに表示される、表アイコンから「Person」を選択しマウスの右クリックによりプロパティを選択します。



② 以下に表示されるプロパティウィンドウの辞書パスが DDFDir になります。デフォルトの Pervaisve.SQL インストールでは C:\PVSW\DEMODATA フォルダーになります。



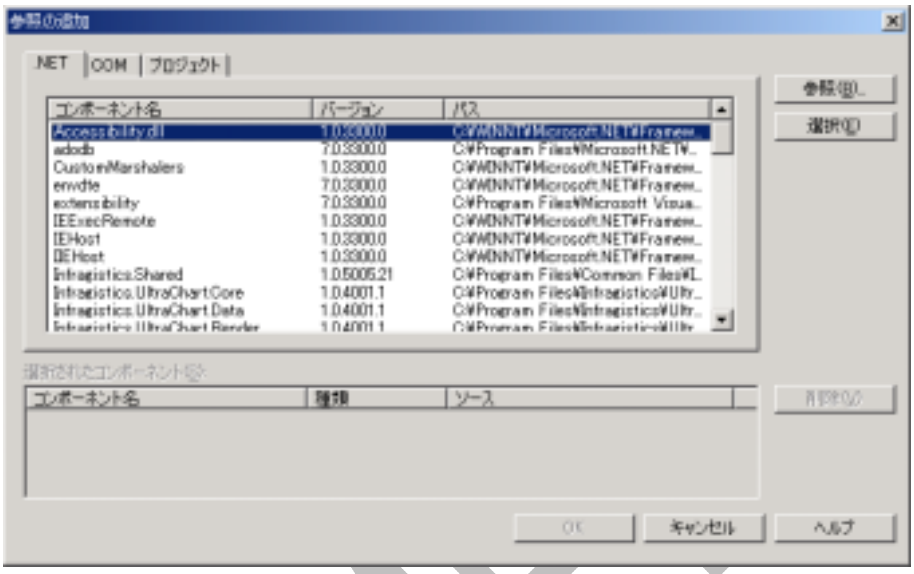
ASP.NETプロジェクトを開始

Visual Studio.NET を起動して新規プロジェクトを選択します。ご利用になる言語を左ペインで選択し、右ペインから「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。

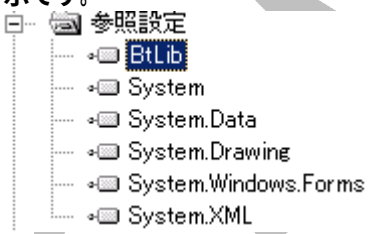
プロジェクトへの参照設定

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 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」フォルダを右クリックして、「参照の追加」を選択します。以下の「参照の追加」ダイアログが表示されましたら、「参

照」ボタンを押し c:\Program Files\TechKnowledge\BtLib\bin フォルダーに移動して BtLib.DLL を選択します。



以下は参照設定に BtLib を追加したソリューション・エクスプローラー・タブの表示です。



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。

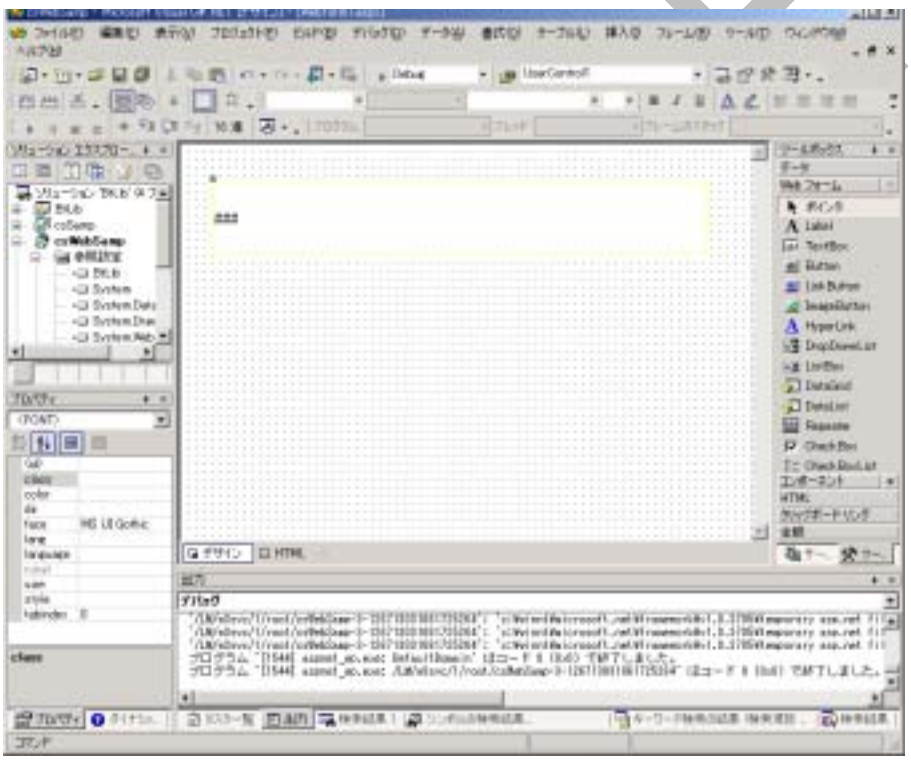
```
Import BtLib
```

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:\\pvsw\\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();
}
```

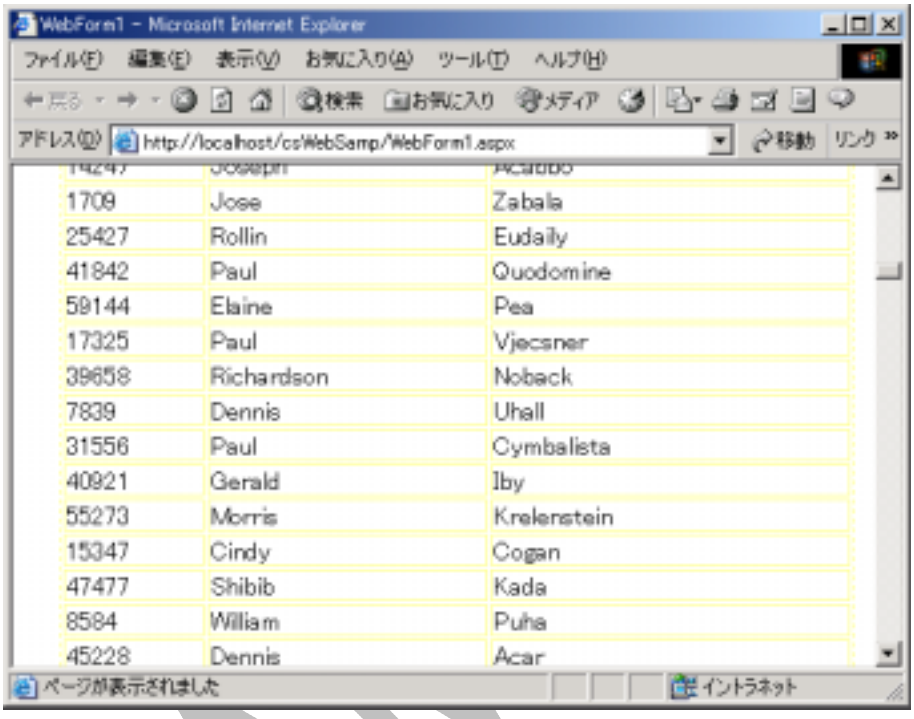
初期値ではなく PostBack の結果としてテーブルを表示するような場合には Page オブジェクトの IsPostBack を参照して以下のようなコードをします。

```
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 メソッドにて status 94 の例外が発生する場合は当マニュアルの Appendix-B のとおり 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。

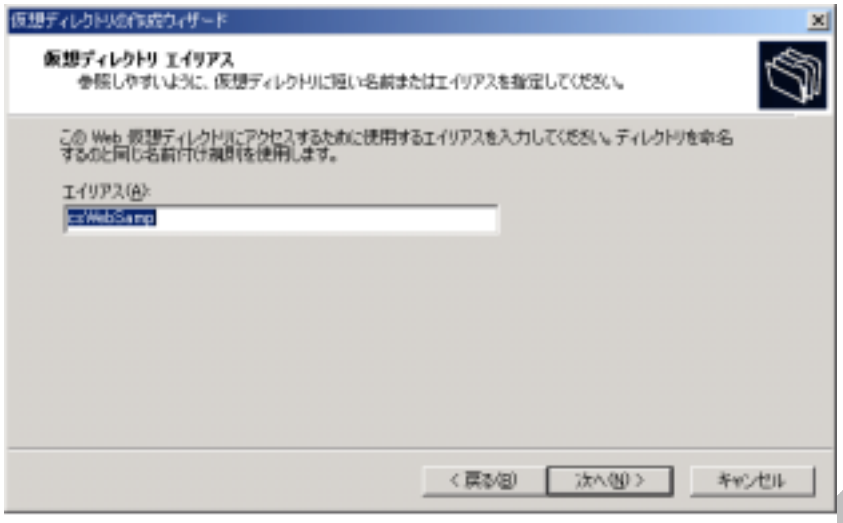
サンプルプログラムについて

概要

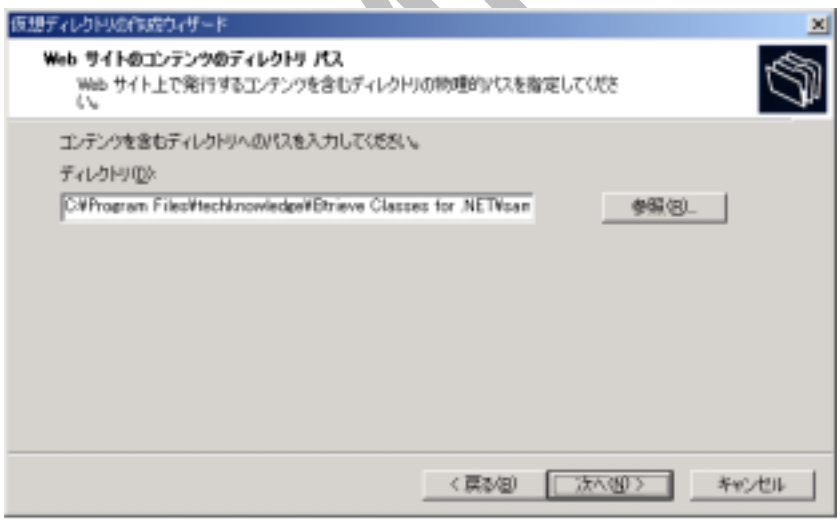
サンプルプログラムを動作させる場合には Visual Stdio.NET がインストールされていることが前提となります。サンプルプログラムで Windows Form で動作するものについては、製品のメニューから選択するか、インストールディレクトリの samples ディレクトリ以下にある*.vbproj ファイルまたは*.csproj ファイルをエクスプローラーで選択することで Visual Studio.NET に読み込むことができます。ただし、ASP.NET の Web アプリケーション用のサンプルにつきましては、プロジェクト・ファイルを Visual Studio.NET に読み込ませる前に IIS で仮想ディレクトリの設定をする必要があります。実行時に Open メソッドにて status 94 の例外が発生する場合は当マニュアルの Appendix-B にある 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。また、クラスライブラリへの参照が切れるような場合は一旦参照から BtLib.DLL を削除して再度指定しなおす必要があります。ここでは最初に ASP.NET 用 Web サンプルアプリケーションの設定の説明をします。

仮想ディレクトリ設定方法

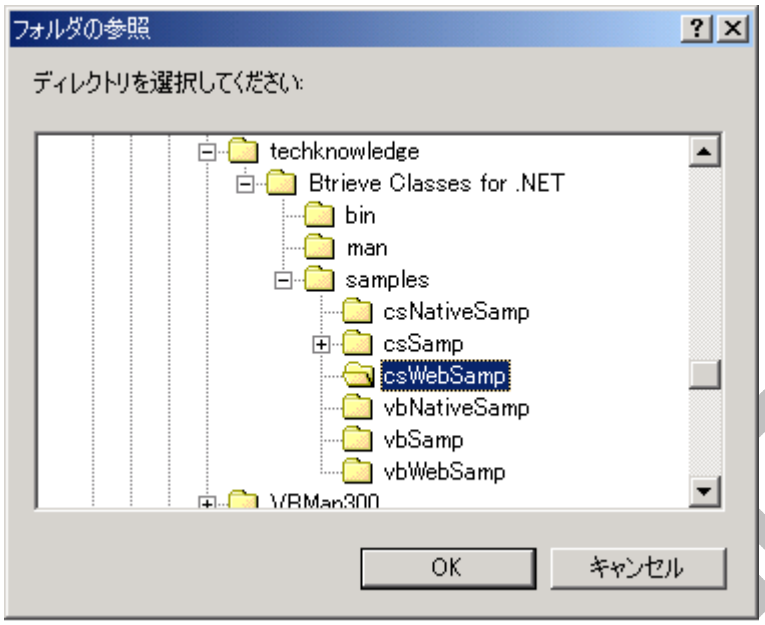
1. コントロールパネルから「管理ツール」を選択し、「コンピュータの管理」を選択します。
2. インターネットインフォメーションサーバーの「既定 Web サイト」を選択します。
3. マウスの右クリックで「新規作成」から「仮想ディレクトリ」を選択します。
4. 最初の画面で「次へ」ボタンを押して表示される以下の画面で C#サンプルの場合は csWebSamp,VB.NET サンプルの場合は vbWebSamp と入力します。



5. 以下の画面ではサンプルの存在するディレクトリを指定します。デフォルトインストールでは \program files\techknowledge\btrieve classes for .NET\samples 下の csWebSamp または vbWebSamp フォルダーを指定します。



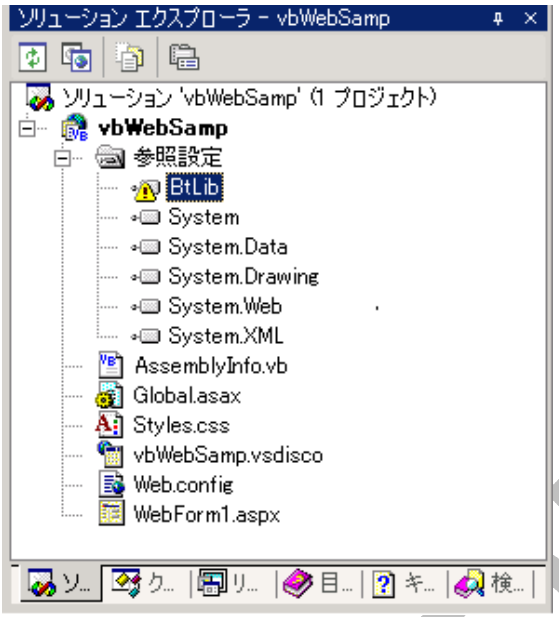
以下はインストールフォルダーの指定画面です。



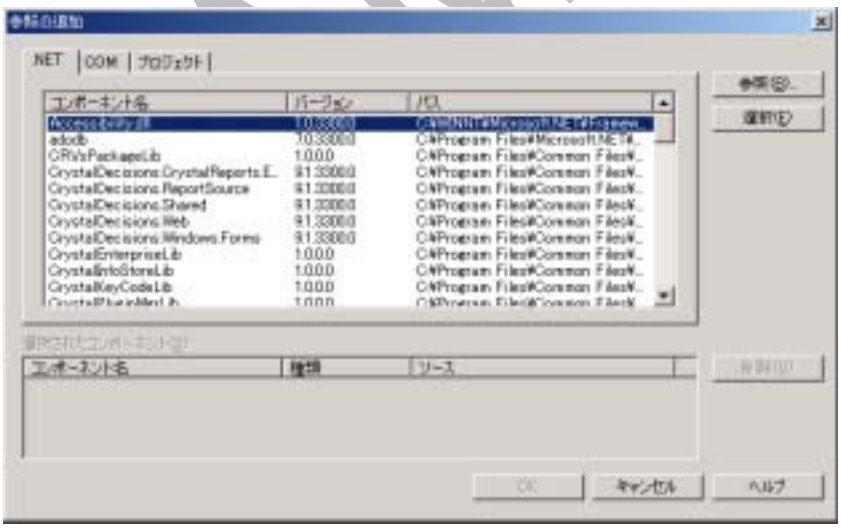
6. 「次へ」ボタンを押して仮想ディレクトリ指定を完了します。

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が削除できます。この状態で「参照設定」を再度右クリックして「参照の追加」を選択することで可能です。



上記画面で「参照」ボタンを押し、c:\program files\techknowledge\Btrieve Classes for .NET\bin フォルダにある btlib.dll を指定します。

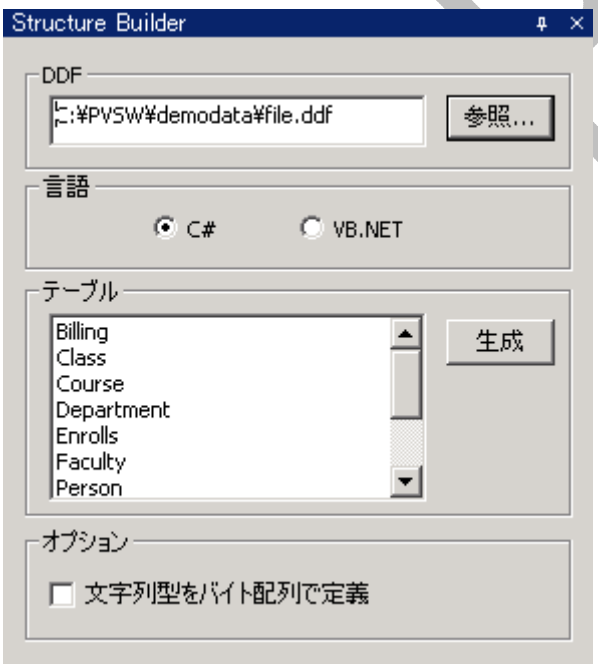
Visual Studio .NET 2003 でのサンプル利用方法

プロジェクト・ファイルやソリューションファイルの形式が Visual Studio .NET 2003 では変更になり、上位バージョン(2003 年版)で作成されたプロジェクトファイルおよびソリューションファイルは下位バージョン（2002 年版）では開くことが出来なくなりました。そのため当製品のサンプルはすべて Visual Studio .NET で作成されています。製品サンプルを Visual Studio.NET 2003 から利用する場合はプロジェクトファイルを読み込む際に新形式に変換するオプションを選択して読み込むことで、問題なく動作することを確認しております。

ストラクチャビルダー

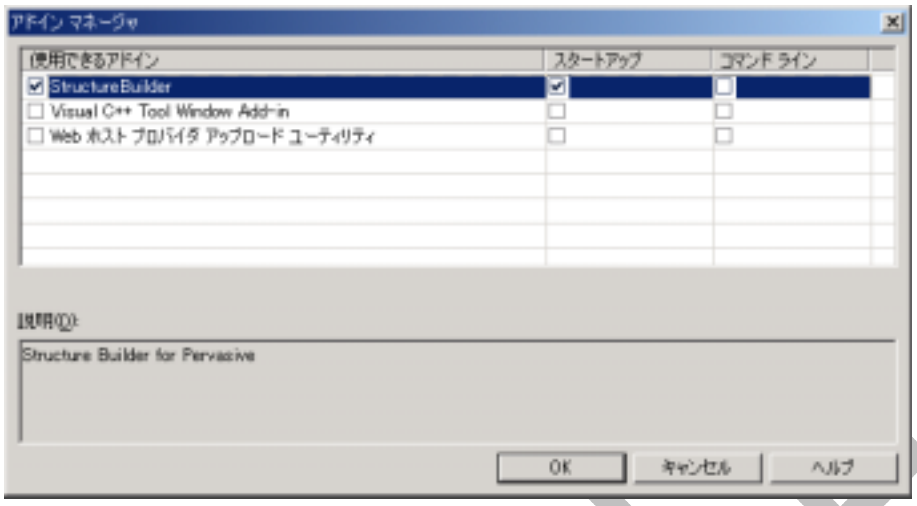
概要

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.NET 用の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ツール、ストラクチャビルダーを提供しています。以下はストラクチャビルダーのサンプル画面です。また DDFDir には Pervasive.SQL V8.6 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。



ストラクチャービルダーの起動方法

Visual Studio.NET の「ツール」メニューから「アドイン」を選択すると以下の画面が表示されます。



使用できるアドインの「StructureBuilder」のチェックを入れると StructureBuilder のツール・ウィンドウが Visual Studio.NET に追加されます。任意の場所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ソールを置きます。
⑤ 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 データ型に変換するには Native クラスの変換メソッドを使うことで変換できます。

文字列データに対する変数型の選択について

たとえば C#言語で構造体を以下のように宣言した場合に、

```
[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にスペースが埋まる仕様なので、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 の仕様と判断出来ます。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。(3 行目の FixString 呼び出し)

```
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はよくない場合があると思われます。

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=Unm
anagedType.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出来ますので、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」にチェックしてください。

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(*DDFDirOrConnectionString* As String)
Ddf()

概要
パラメータとしてデータベースを指す DDF へのパスを指定した最初の形式は Ddf オブジェクトの初期化後に DDF を読み込み情報を展開しオブジェクトに保持します。Pervasive.SQL V8.6 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 Pervasive.SQL V8.6 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 パスを指定しない第3の形式は Ddf オブジェクトの初期化を実行します。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でデータベース情報を Ddf オブジェクトに読み込みます。

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。

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 でサポートされる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 Pervasive.SQL V8.6 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 でサポートされる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 情報を開放します。Ddf オブジェクトは初期状態に戻ります。開放後は 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
Btrieve/Pervasive.SQL 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。当クラスは構造体情報を DDF から取得することでコレクションを使ってフィールド指定するシンプルなメソッドで簡単に extended 系メソッドを実行します。

以下は Extended クラスを利用したサンプル・コードです。Pervasive.SQL 2000i 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:\\pvsw\\demodata");
BtLib.Record r = ddf.GetRecord("Person");
r.Open();

r.Index = "ID";
BtLib.Extended ex = r.GetExtended();
ex.SearchCond = "@First_Name > Geroge";
ex.MaxRecords = 100;
ex.SkipRecords = 100;
```

```
ex.AddField("First_Name");
ex.AddField("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 Microsoft .NET framework の System.Data.DataSet オブジェクトを簡単に生成できます。DataSet オブジェクトは Visual Studio.NET 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連結が可能になります。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。

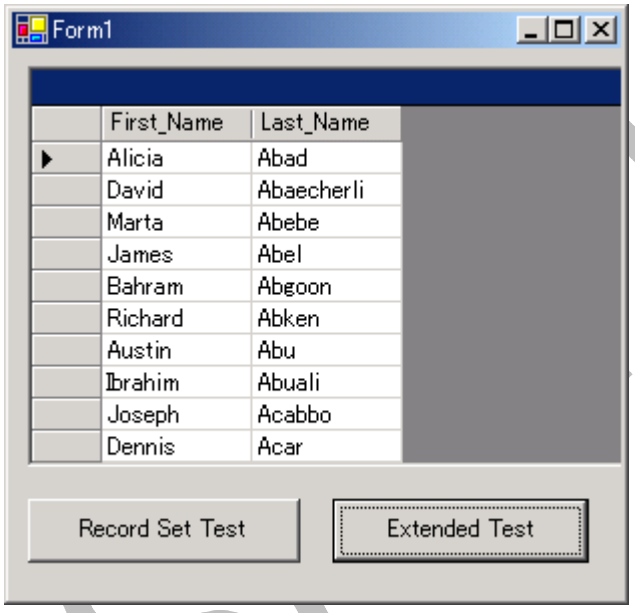
```
short rc;
BtLib.Ddf ddf = new BtLib.Ddf("c:\\pvsw\\demodata");
BtLib.Record r = ddf.GetRecord("Person");
r.Open();

r.Index = "Names";
BtLib.Extended ex = r.GetExtended();
ex.SearchCond = "@First_Name > Adachi";
ex.MaxRecords = 100;
ex.SkipRecords = 100;
ex.AddField("First_Name");
ex.AddField("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に true 設定の場合はケース (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。

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ディリミッタになります。スペースが含まれる即値を指定する場合はシングル･クォート、ダブル･クォート、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 = “VBMan” ¦ @製品 = “Visual Basic”
@maker <> /Microsoft/ & @maker <> /Borland/
```

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。

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全て削除します。

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

GetData

書式
void GetDataSet(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を転送します。(先頭の6バイト以降を転送します) 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
Read または 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もとずく DataTable オブジェクトを返します。

戻り値
DataTable 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 になります。

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 になります。

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:\\pvsw\\demodata");
BtLib.Record r = ddf.GetRecord("test");
r.Open();
BtLib.Extended ex = r.GetExtended();
```

```
ex.SearchCond = "@ID > 2";
ex.MaxRecords = 100;
ex.SkipRecords = 100;
ex.AddField("ID");
ex.AddField("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。

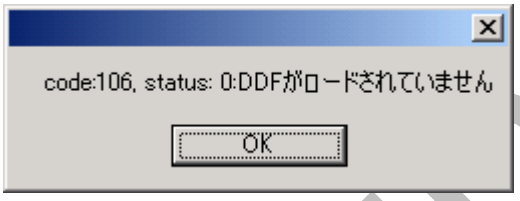
Ddf
Record
Extended

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あらわすコードです。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出来ます。

```
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をロードして下さい");
  }
}
```

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 Pervasive.SQL/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 Pervasive.SQL/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 Managed 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。Managed 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 Managed から UnManaged 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、Managed データを Byte 配列に変換するするヘルパー・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全て 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 Pervasive.SQL 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出来ます。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に指定することが出来ます。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 を使い構造体を指定することが出来ます。構造体の宣言方法については Appendix 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 Pervasive.SQL 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 指定された領域の offsetFrom,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
第一パラメータは 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パラメータで指定された領域の中での変換処理開始位置、変換処理終了位置をオフセットで指定します。(ベース0指定)

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一パラメータに指定します。第二パラメータは 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

第一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けデータを.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全て 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一パラメータに指定します。第二パラメータは 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一パラメータに指定します。第二パラメータは 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一パラメータに指定します。
第二パラメータは 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一パラメータに指定します。
第二パラメータは 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一パラメータに指定します。
第二パラメータは 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一パラメータに指定します。
第二パラメータは 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一パラメータに指定します。
第二パラメータは 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 型データの日付け部分は現在の日付けが設定されます。

パラメータ
Btrieve API から戻された Byte 型データの配列を第一パラメータに指定します。第二パラメータは 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
たとえば、

```
data = new Byte[334];
//
// btrieve operation によるデータ取得
// …
string str = Encoder.GetString(data,9,16);
```

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に領域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。(上記のサンプルの場合 14) このようにして得られた stirng は後続ヌルのためたとえば文字列の連結等が正常に機能しなくなります。当メソッドで Trim することで後続のヌルを削除して.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にアクセスすることが出来ます。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へのアクセス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"] = "鈴木";
rec["Last_Name"] = "パパイヤ";
```

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全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.GetFirest) == 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 null pointer excepotion が発生します。(HashTable オブジェクトのデフォルト例外) また、アクセサー文字列はケースセンシティブです。たとえば、Perviasive 2000i の DEMODATA の場合 Person 表の"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

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 Pervasive.SQL 2000i の Person 表で 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:\\pvsw\\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 Pervasive.SQL 2000i の Person 表で 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:\\pvsw\\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当クラスは Close メソッドを呼び出すことにより関連資源を解放済みとす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。レコード読込み時にロックをかける場合に 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で既存のロックを解除させる以外の方法としては Unlock メソッドで可能です。

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 Pervasive.SQL/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 Pervasive.SQL/Btrieve のマニュアルをご参照ください。

メソッド

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存在しないことが必要です。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。たとえば 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以前に Read または Step メソッドにより Btrieve/Pervasive データを読み込まないと内部バッファにある不定なデータを返しますのでご注意ください。（通常は初期値は 0 になります）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
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 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:\\pvsw\\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();
```

GetDataSet

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

概要
Microsoft .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 オブジェクトを返します。

GetDataTable

書式
DataTable GetDataTable ();

概要
Microsoft .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

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。取得した物理位置は Read メソッドのパラメータとして指定してレコードを読み込みます。

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

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。

Read

書式
short Read(Operation *op*);
short Read(Int32 *PhysicalPosition*);

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･コードによりレコードを読み込みます。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み込みます。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。

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。

戻り値
Btrieve ステータスコード。

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 Btrieve/Pervasie.SQL 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。（たとえば日付け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 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。

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み込みます。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
ステップ関連 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*);

概要
指定した Insert または Update オペレーション･コードによりレコードの書込を実行します。書き込むフィールド値は Record コレクションとして当メソッド実行前に設定しておきます。

パラメータ

Insert または Update Btrieve オペレーションコード。

戻り値
Btrieve ステータスコード。

Transaction Class

概要

Ddf Class にて接続した Btrieve/Pervasive.SQL 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:\\pvsw\\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

Btrieve/Pervasive.SQL 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

Comat();

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 で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に 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で既存のロックを解除させる以外の方法としては Unlock メソッドで可能です。

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 Btreive/Pervasive.sql 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。それ以外の正の値 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。それ以外の正の値 Btrieve のステータス・コードです。

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。値は 100,200,300,400,500 を指定可能です。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。値は 100,200,300,400 を指定可能です。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

とのデータを交換するメモリ・エリアです。

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時には関連する 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･ユーザーで使用する場合は一つのクライアントから DbClose 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。すでに Btrieve ファイルが存在するような場合は上書きされます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。たとえば、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。たとえば、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) 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	リターン値
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 データを読み込みます。

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
この関数によって得られたレコードの新アクセス・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
データベースのフィールドの値を指定されたテーブルのデータ・バッファから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おこなわれます。たとえば時間型は、"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 Stirng,
NumOfField As Integer) As Integer

概要
指定されたテーブル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 Stirng,
NumOfIndex As Integer) As Integer

概要
指定されたテーブル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 Stirng,
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
現在読みこんでいる 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されている時にのみ有効となりますのでこのメソッドを呼び出す際には Btrieve ファイルがオープンされていることが必須となります。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はデータはセットできません。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、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 の誤り、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 が読みこ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時は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・コントロールの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・コントロールの 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にすでにオープン中のものについては Open オペレーションは実行されません。また、メソッドはオープン中のエラーを返すこともありません。

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すでに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おこなわれます。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 Stirng,
NewFileName As Stirng) 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 ビルダーによる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移行の DbAccess/DbGetDirect メソッドに反映されます。DbAccess メソッドの第一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 の誤り、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:\pvsw\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:\\pvsw\\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:\\pvsw\\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:\\pvsw\\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:\\pvsw\\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 VC#です。先頭からレコードを読み込みます。

```
byte  [] posblk = new byte[128];
byte  [] data   = Encoding.ASCII.GetBytes("\0\0");
Int16 dataLength = 0;
byte  [] keyBuf =
    Encoding.ASCII.GetBytes("c:\\pvsw\\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 オペレーションで指定したレコードからデータを読み込みます。

```
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:\pvsw\demodata\person.mkd")
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ましたので、Native Class にヘルパーメソッドとして.NET framework の各データタイプから Byte 配列に変換する GetBytes メソッドを提供します。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:\\pvsw\\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 クラスは変更分を GetChanges メソッドを呼び出すことで取得することが出来ます。

```
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:\\pvsw\\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:\\pvsw\\demodata\\Dept.mkd");
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。Pervasive.SQL 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。ストラクチャビルダーで自動生成することが出来ます。

```
[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=Unma
nagedType.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:\\pvsw\\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。Pervasive.SQL 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。ストラクチャビルダーで自動生成することが出来ます。

```
<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

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。(英語版 Pervasive.SQL V8.5 初期版利用時等) 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 Pervasive より利用可能ですので Pervasive サポートまでお問い合わせください。

Web実行でのStatus 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、初回の Btrieve アクセスが発生した時点で Btrieve status 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 Pervasive.SQL/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. 新しくユーザーを定義する。
2. 以下のグループに所属させる。
   Administrators
   <マシン名> admins
   <マシン名> authers
   <マシン名> browsers
   VS Developers
3. <windir>\Microsoft.NET\Framework\v1.0.3705\config にある machine.config を編集。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"/>

ユーザー設定の情報は以下を参考にしました。

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetHT01.asp

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ことが原因で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 の構成方法で解決できます。以下を参考にして IIS を構成してください。

http://support.microsoft.com/default.aspx?scid=kb;en-us;315158

ストラクチャービルダーをVS.NETのアドインマネジャーから削除したい

レジストリ操作が必要ですが以下を削除するとストラクチャビルダーを VS.NET のアドインマネジャーから削除することができます。VS.NET は停止した状態で以下のレジストリが存在する場合は削除してください。

VS.NET 2003 の場合

HKEY_LOCAL_MACHINE\Software\Microsoft\VisualStudio\7.1\AddIns\SbAddIn.Connect

HKEY_CURRENT\USER\Software\Microsoft\VisualStudio\7.1\AddIns\SbAddIn.Connect

VS.NET の場合

HKEY_LOCAL_MACHINE\Software\Microsoft\VisualStudio\7.0\AddIns\SbAddIn.Connect

HKEY_CURRENT\USER\Software\Microsoft\Visual Studio\7.0\AddIns\SbAddIn.Connect

再インストールでストラクチャビルダーがツールメニューに表示されない

Btrieve Classes for .NET をアンインストール後に残っている VS.NET 開発環境のツールメニューから「Structure Builder」を削除した場合、Btrieve classes for .NET を再インストールしても VS.NET 開発環境のツール・メニューの下に「Structre Builder」が設定されない場合があります。そのような場合はお手数ですが、再度 Btrieve Classes for .NET をアンインストールした状態で以下のレジストリの削除をお願いいたします。レジストリ操作は間違えた部分を削除するとシステムに影響が出る場合がございますので十分ご注意の上操作をお願いいたします。

CURRENT_USER\Software\Microsoft\VisualStudio\7.0\PreloadAddinState\SbAddin.Connect

上記レジストリは弊社インストーラーで作成するものではなく Visual Studio.NET がアドインを初回ロードしたときに作成するため、Btrieve classes for .NET のインストーラでは削除対象に含まれないため、手動での削除が必要になります。

DDFファイルとは？

Pervasive/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つのファイルが同一のディレクトリに存在することが必要となります。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 Pervasive.SQL で読み込むことができますが、逆に新しい Pervsive.SQL 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。

Btrieve 6.15をWindows2000/XPで動作させる

Btrieve 6.15 がサポートする動作環境には WindowsXP/2000 は存在しません。正式にメーカー保証のある動作環境でお使いになりたい場合には Pervasive.SQL 2000i/V8 をご利用ください。

Btrieve 6.10で動作可能か

NetWare に添付されていた Btrieve 6.10 は 32Bit のプログラミング･インターフェースがサポートされていないので、32Bit モジュールとして動作する当製品からはご利用いただけません。Btrieve 6.15 以降で 32Bit プログラミング･インターフェースがサポートになりました。ただし現在では Btrieve 6.15 は販売停止になっていますので新規にお買い求めになる場合は上位バージョンである Pervasive.SQL 2000i/Pervasive.SQL V8 を入手してください。

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

Pervasive.SQL 2000i/Pervasive.SQL V8 はカラムを「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。

Btrieve 5.xのシステムから移行したい

32BIT OS で動作するアプリケーションを作成するには 32Bit プログラミング環境がサポートされている Btrieve データ･ベース･エンジンが必要になります。現在購入可能なバージョンは、Pervasive.SQL 2000i/Peravsive.SQL V8 になります。Pervasive.SQL では Btrieve データ･ファイルの形式は 5.x, 6.x, 7.x と過去バージョンのファイル形式もサポートされています。従って、Btrieve 5.x で動作しているシステムのデータはデータ形式を変換することなく、Pervasive.SQL で動作させることが可能です。

Appendix-C Data Type について

Pervasive.SQL 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 Btrieve の Record/Extended クラスににおける.NET Framework データ型へのマッピングを以下に示します。

p.sql data type	Btrieve data type	.NET Data type(*)
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
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（円貨記号）を２重に記載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、Btrieve/Pervasive.SQL 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 読み込みエラー」DDF が何らかの原因で不整合な状態になっています。再度 DDF ファイルが Pervasive Control Center 等で正しく読み書きできるか確認してください。DdfOpenError と同様に Btrieve/Pervasive.SQL 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 Btrieve/Pervasive 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無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 short file name 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 データ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 Manged 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	Data 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 メソッドで指定された DataSet が Extended オブジェクトで返された DataSet 以外のオブジェクトが指定されていると思われます。
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 Btrieve/Pervasive.SQL のステータスコードが保持されていますので、この値を Pervasive.SQL 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 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 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されるべきメソッド/関数がオープン中のファイルに対して実行された。
VBM_ERR_NOT_OPEN	-3	Btrive ファイルがオープンされている時に実行されるべきメソッド/関数がオープンされていないファイルに対して実行された。
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 に存在しない。
VBM_ERR_INVALID_FIELD_NAME	-5	フィールド名が存在しない。
VBM_ERR_INVALID_KEY	-6	インデックス名が存在しない。
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ある。
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ある。
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した。
VBM_ERR_CONTROL_NOT_FOUND	-11	DbAttachControl 関数で最初のパラメータがただしくない。
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た。
VBM_ERR_INVALID_FIELD_ID	-13	フィールド ID が正しくない。
VBM_ERR_INVALID_KEY_ID	-14	インデックス ID が正しくない。
VBM_ERR_INVALID_TABLE_ID	-15	テーブル ID が正しくない。
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 を読みこ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 Btrieve ステータス･コード

以下に Btrieve のステータス･コードと概要を記述します。詳細は Pervasive.SQL マニュアルなどに記載があります。

2	I/O ERROR
3	File not opened
4	Key Value not found
5	Duplicate key value
6	Invalid key number
7	Different key number
8	Invalid Positioning
9	End of file
10	Modifiable Key value error
11	invalid file name
12	file not found
13	Extended file error
14	Pre-Image open error
15	Pre-Image I/O error
16	Expansion error
17	Close error
18	Disk Full
19	Unrecoverable error
20	Record Manager Inactive
21	Key Buffer Error
22	Key Buffer Error
23	Position Block Error
24	Page Size Error
25	Create I/O Error
26	Number of Keys
27	Key Position
28	Record Length

29	Key Length
30	Btrieve file name
31	Extended Error
32	Extended I/O error
34	Extend Name
35	Directory Error
37	Begin Transaction error
38	Transaction Control File
39	End/Abort Error
40	Transaction Max Files
41	Operation not allowed
42	Incomplete accelerated access
43	Invalid data record address
44	Null key path
45	Inconsistent Key flags
46	Access denied
47	Maximum open files
48	Invalid Alternate sequence definition
49	Key type error
50	Owner already set
51	Invalid Owner
52	Error Writing cache
53	Invalid Interface
54	Variable Page Unreadable
55	Autoincrement error
56	Incomplete index
57	Expanded Memory Error
58	Compression buffer too short
59	File Already Exists
60	Reject Count Reached
61	Work space too small
62	Incorrect descriptor

63	Invalid extended insert buffer
64	Filter limit reached
65	Incorrect field offset
74	Automatic transaction abort
75	Server Routing list too small
76	File server list too small
77	Wait lock error
78	Deadlock detected
80	Conflict
81	Lock error
82	Lost position
83	Read Outside Transaction
84	Record in use
85	File in use
86	File full
87	Handle full
88	Mode Error
90	Redirected Device table full
91	Server Error
92	Transaction table full
93	Incomplete lock type
94	Permission error
95	Session no longer valid
96	Communication environment error
97	Data message too small
98	Internal transaction error
1001	Invalid lock parameter
1002	Invalid memory parameter
1003	Insufficent memory for parameters specified
1004	Invalid page size parameter
1005	Invalid pre-image device parameter
1006	Invalid pre-image memory parameter

1007	Invalid file parameter
1008	Incorrect parameter
1009	Invalid transaction parameter
1010	Unable to access btrieve file for transaction recovery
1011	Invalid compression parameter
1012	Number of files in transaction should be between 1 to 18
1013	Task list is full
1014	File is open or transaction is active
1016	Already initialized
2001	Insufficent Memory
2002	Parameter Invalid or Out of Range
2003	No Local Access Allowed
2004	SPX is Not installed
2005	Wrong version of SPX Installed
2006	No Available SPX Connection
2007	Invalid Parameter

Btrieve Classes for .NET Version 1.20
第2版
2004年6月16日第1刷発行