

Btrieve Classes for .NET

Version 4.00

Programming Manual

TechKnowledge

Table of Contents

TABLE OF CONTENTS	2
PRODUCT OVERVIEW	7
INTRODUCTION	7
FUNCTIONS NEW IN VERSION 4.0.....	7
FUNCTIONS NEW IN VERSION 3.0.....	7
FUNCTIONS NEW IN VERSION 2.0.....	7
FUNCTIONS NEW IN VERSION 1.2.....	8
FUNCTIONS NEW IN VERSION 1.1.....	8
TWO CLASS SYSTEMS	8
SUPPORTED LANGUAGES	9
CREATABLE APPLICATIONS	9
ABOUT DDFs.....	9
DATA TYPES	9
DATA SIZE	10
LICENSE AGREEMENT	10
USER SUPPORT.....	11
LIMITATIONS OF WARRANTY	11
SALES AND USER SUPPORT	12
DEVELOPMENT.....	12
INSTALLATION	14
SYSTEM REQUIREMENTS.....	14
USE ON WINDOWS 2000.....	14
SETUP	14
INSTALLED FILES	16
64-BIT OPERATING SYSTEM SUPPORT	17
REGISTERING AND UN-REGISTERING STRUCTURE BUILDER ON WINDOWS VISTA AND WINDOWS 7.....	17
TUTORIAL	19
Preparation of the Btrieve Database	19
Starting the ASP.NET Project.....	21
Setting of References to the Project	21
Name Space Declaration	23
WebForm Settings.....	23
Creating the Data Acquisition Code.....	24
Result of Execution	26

ABOUT THE SAMPLE PROGRAMS	27
OUTLINE	27
REFERENCING BTLIB	27
VISUAL STUDIO .NET PROJECT FILE VERSIONS	28
APPLICATION CONFIGURATION FILES IN A .NET 4.0 ENVIRONMENT.....	29
64-BIT EXECUTION OF THE SAMPLE PROGRAMS	29
STRUCTURE BUILDER	30
OVERVIEW	30
HOW TO START THE STRUCTURE BUILDER	31
HOW TO INSERT STRUCTURES.....	31
STRUCTURE BUILDER RESTRICTIONS.....	32
SELECTING VARIABLE DATA TYPE FOR CHARACTER STRING DATA.....	32
CLASS LIBRARY REFERENCE	35
DDF CLASS	35
Constructor	35
Property	36
CodePage	36
DDFDir	36
FloatSize	36
FillSpace	36
OwnerName	37
SignNibble	37
TableCount.....	37
TableNames	37
Method	39
GetRecord	39
Load	39
Login.....	39
Logout.....	39
Unload.....	40
EXTENDED CLASS	40
Constructor	42
Property reference	43
IgnoreCase	43
Index	43
IndexNumber	43
Lock	43
MaxRecords	44
Mode.....	44
ResultCount	44

SearchCond	44
SkipRecords	45
Method reference.....	45
AddField	45
ClearField	45
Fill.....	45
GetData	46
GetDataSet	46
GetDataTable	47
MoveFirst.....	47
MoveLast	47
MoveNext	48
MoveTo.....	48
Read	48
RecordExists	49
Step	49
EXCEPTION CLASS	51
Constructor	52
Properties.....	52
BtrieveStatus	52
ErrorCode	53
Method	53
ToString	53
NATIVE CLASS	54
Constructor	54
Property	54
Methods.....	54
BtrCall.....	54
BtrCallID	57
FixString	57
GetBoolean	58
GetBytes	58
GetDate	59
GetDecimal	60
GetDouble.....	60
GetInt16	61
GetInt32	61
GetInt64	62
GetSingle	62
GetTime	62
Trim	63
RECORD CLASS	64
Constructor	65

Properties.....	65
ColumnCount.....	65
DataFileName	65
FileFlag	65
Index	66
IndexNumber	66
IsOpen.....	67
Lock	67
NullKeyValue	68
OpenMode	68
PageSize.....	68
Methods.....	68
ClearData	68
Close	69
Create	69
Delete	70
GetBytes	70
GetData	71
GetDataColumn	71
GetDataSet	72
GetDataTable	73
GetNumOfRecords	74
GetPosition.....	74
GetRecordLength.....	74
Open.....	75
Query	75
Read	78
SetBytes	78
SetData.....	79
Step	79
Write	80
TRANSACTION CLASS.....	80
Constructor	82
Property	82
Lock	82
Method	82
Abort.....	82
Begin.....	82
BeginConCurrent	83
End.....	83
Reset	83
APPENDIX-A: CODE SAMPLES	84

VISUAL BASIC.NET SAMPLE CODE	84
SAMPLE CODE SHOWING THE USE OF IDICTIONARYENUMERATOR	86
GETDATASET C# SAMPLE	86
C# SAMPLE SHOWING DATABIND TO DATAGRID IN WEBFORM.....	87
NATIVE CLASS C# RECORD SCAN SAMPLE	87
NATIVE CLASS VB.NET RECORD SCAN SAMPLE	90
NATIVE CLASS C# INSERT SAMPLE CODE.....	92
SAMPLE FOR INPUTTING THE CHANGED PARTS OF THE DATASET INTO THE DATABASE	95
NATIVE CLASS C# SAMPLE FOR SPECIFYING THE DATA AREA IN THE STRUCTURE .	96
C# STRUCTURE DECLARING SAMPLE.....	97
NATIVE CLASS VB.NET SAMPLE FOR SPECIFYING THE DATA AREA IN THE STRUCTURE.....	98
VB.NET STRUCTURE DECLARING SAMPLE	99
APPENDIX-B: FAQS (FREQUENTLY ASKED QUESTIONS)	101
Status 94 at the time of Web execution	101
Sample does not operate on a domain controller.....	102
How to delete Structure Builder from VS.NET Add-in Manager	102
Structure Builder does not display on Visual Studio 2008 Startup	103
On re-installing Structure Builder does not display on the Tool Menu.....	103
What is a DDF file?.....	104
I want to place the DDF files and the data files in separate folders	104
Running Btrieve 6.15 under Windows 2000 or Windows XP.....	104
Is operation possible with Btrieve 6.10?	104
The character string field has been declared, but the leading 1 byte seems to be displaced.....	104
Moving from a Btrieve 5.x system	105
APPENDIX-C: ABOUT DATA TYPES	106
APPENDIX-D: EXCEPTION CLASS ERROR CODE TABLE	107
APPENDIX-E: BTRIEVE STATUS CODES.....	111

Product Overview

Introduction

Btrieve Classes for .NET are special .NET-language database access support classes for use with Pervasive.SQL V8, Pervasive PSQL v9, Pervasive PSQL Summit v10, and Pervasive PSQL v11. ADO.NET and OLE DB are the standard database access layers in .NET, but by offering a class library that does not work via these generic software layers, Btrieve Classes for .NET ensure a very high performance that is 3 to 20 times that of these standard classes.

Functions New in Version 4.0

In Version 4.0, the following functionality has been added:

1. Support for Pervasive PSQL v11.
2. Support for Visual Studio 2010.
3. Support for Windows 7.

Version 4.0 is backwards compatible with prior versions.

Functions New in Version 3.0

In Version 3.0, the following functionality has been added:

1. Support for Pervasive PSQL Summit v10.
2. Support for Visual Studio 2008.
3. LINQ support methods have been added to the Record class.
4. 64 Bit OS support (Windows 2008 Server 64bit, Windows Vista 64bit, Windows Server 2003 64bit)

Version 3.0 is backwards compatible with prior versions.

Functions New in Version 2.0

The class library now supports .NET framework 2.0, and Visual Studio .NET 2005 IDE. Version 2.0 is compatible with prior versions.

Functions New in Version 1.2

In this version functionality has been added to support the new security functionality in Pervasive.SQL V8.5 and above. This includes support for using a Database URI in the Ddf class constructor and new Login/Logout methods.

Functions New in Version 1.1

This version features an added function for direct input and output of Btrieve and Pervasive data record images that utilizes a structure based on the byte alignment control functions offered by Microsoft .NET. The Native and Record classes support this structure. Furthermore, because the workload involved with declaring this structure to support byte alignment is thought to be very large, the product now includes a structure builder as add-in software for use with Microsoft Visual Studio .NET that automatically generates this type of structure from a set of DDF files. This version also features support for Visual Studio .NET 2003.

Two Class Systems

Btrieve Classes for .NET supports two class systems.

- DDF
These are new classes that are designed to comply with the .NET Framework specifications. These DDF classes are composed of Record, Extended, Transaction, and Exception classes, etc. They allow easy access to columns and data type conversion code. These classes allow very high development efficiency for new coding work.
- Native
The Pervasive PSQL SDK does not yet offer a method for accessing Btrieve level operations from .NET Managed Code. These Native classes enable access to Btrieve API level operations from Managed Code. These classes are particularly suited to cases where an existing application does not access DDFs or when you wish to migrate existing code based on the Btrieve API. However, due to the fact that the Btrieve API contains many parameters and due to the necessity of code for fetching and storing application data from the record buffer, the application can become complicated.

Supported Languages

The class library is constructed as a Managed Class Library introduced in the .NET Framework. Accordingly, the languages currently supported are Visual Basic .NET, Visual C#, and Managed C++ . A large number of other languages for which the .NET Framework environment is a prerequisite are scheduled for release in the future. As the use of these languages becomes widespread , it is planned to introduce support for these new languages too on a one by one basis.

Creatable Applications

This class library enables the creation of the following types of applications that can make use of the .NET Framework.

- Windows applications
- ASP.NET Web applications and Web Services
- Console applications

About DDFs

The DDF, Record, and Extended classes of Btrieve Classes for .NET operate based on DDF information. From Pervasive.SQL's Control Center, it is possible to define DDFs using the the table designer and table-creating wizard.

Data Types

Btrieve Classes for .NET's Record and Extended classes can be used to convert Pervasive.SQL data types to .NET Framework data types in accordance with the data type conversions described in Appendix-C. Data type conversions are mainly performed by the type conversion methods of the .NET Framework, but in cases where conversion to an existing .NET Framework data type is impossible, data is returned as String type. Conversely, when .NET Framework data in the Record class is converted to Btrieve data, an exception occurs if conversion fails. Even when a data conversion exception occurs, it

may be possible to avoid the exception by first converting the .NET Framework data type to a character string type by using the ToString method. For the Native classes, a basic method is provided for converting data between a .NET Framework type and a Byte array. However, Byte array and .NET Framework data type conversions that retain the original Btrieve data type must be described in code.

Data Size

For types like longvarchar and longvarbinary, , this product only supports record lengths up to length that can be read in a single Btrieve call (maximum 65,535 bytes). The current version does not support data with a size that necessitates chunk operations straddling multiple records.

License Agreement

The license agreement grants the customer the right to use the Btrieve Classes for .NET environment on one (1) personal computer system.

- The Btrieve Classes for .NET License Agreement cannot be transferred or lent to a third party in any way.
- The License Agreement goes into effect at the time the product package containing Btrieve Classes for .NET is opened. A once-opened product cannot be returned.
- No license fee is required for the runtime module. Please see the “Module List” feature in the “Installation” section of this manual for the files that the customer is allowed to distribute together with the customer’s application.
- TechKnowledge Inc.and the sales company AG-TECH shall in no event be liable for any losses whatsoever arising out of the use of this product.

The License Agreement will be nullified in any of the following instances.

1. If significant losses are inflicted upon TechKnowledge Inc.as a result of improper use of this Software.
2. If the purchaser violates the stipulations for use of the Software.
3. If the program disk, printed materials, etc., are duplicated for a purpose that falls outside the range of the License Agreement.
4. If the purchaser neglects to make a user registration for Btrieve Classes for .NET.
5. If this product is subjected to reverse engineering.

User Support

User support for this product is offered via the following email address:-
bcnsupport@agtech.co.jp.

Limitations of Warranty

TechKnowledge Inc.disclaims all warranty; either express or implied, such as implied warranties of merchantability, fitness for a particular purpose, etc., of this product and accompanying written materials.

The author of this product as well as any persons involved in the manufacture and distribution of it shall not be liable for any damages arising out of malfunctions of this software regardless of whether these damages are direct, indirect, consequential or incidental. This applies also even if the development company has been advised of the possibilities of such damages.

Sales and User Support



AG-TECH CORPORATION

Shoei Kanda-Bashi Building 3F,
1-21-1 Kanda Nishiki-cho, Chiyoda-ku, Tokyo 101-0054, Japan

Phone: +81-3-3293-5300

FAX: +81-3-3293-5250

E-Mail: bcnsupport@agtech.co.jp

URL: <http://www.agtech.co.jp>

Development



TechKnowledge Inc.

Sando Building 9F, 2-16-1 Komazawa, Setagaya-ku, Tokyo, Japan

Phone: +81-3-3421-7621

FAX: +81-3-3421-6691

E-Mail: info@techknowledge.co.jp

Web: www.techknowledge.co.jp

Trademarks

The trademarks and registered trademarks mentioned in this manual are trademarks or registered trademarks of their respective companies.

Installation

Installation of Btrieve Classes for .NET is explained in the following.

System Requirements

The following software environment is a prerequisite for the operation of Btrieve Classes for .NET.

- ① .NET Framework 2.0 or above, Visual Studio .NET 2005, 2008 or 2010.
- ② Pervasive.SQL V8.x or Pervasive PSQL v9 or Pervasive PSQL Summit v10 or Pervasive PSQL v11.

It is assumed that detailed Btrieve programming information will be required in connection with development of applications with this product. For this, please refer to the manuals and the sample programs of the Pervasive.SQL SDK.

Use on Windows 2000

When installing on Windows 2000, a failure to find a procedure entry point in Kernel32.dll error may be seen. This can be solved by applying Windows 2000 Service Pack 4 followed by Update Rollup 1 for Windows 2000 Service Pack 4. This can be done by running Windows Update.

Setup

Installation of Btrieve Classes for .NET is explained here. The installation procedure is as follows.

- ① Insert the Install CD-ROM into your CD-ROM drive.
- ② Execute setup.exe located in the root directory of the CD-ROM using Windows Explorer, etc.

- ③ Answer the questions asked by setup.exe, and then click the Install button to complete the installation automatically.
- ④ When the installation is completed correctly, a Btrieve Classes for .NET program group is created in the menu.
- ⑤ The README.html file contains the latest information that is not included in the manual. Be sure to read the file, as it may also contain recent information regarding the installation.

Installed Files

The installed files are shown in the following chart based on the assumption that the OS installation directory is named <osdir>, and the Btrieve Classes for .NET installation directory is named <instdir>. The default installation directory <instdir> is

c:\program files\techknowledge\Btrieve Classes for .NET 4.0.

The modules that the customer is granted the right to attach and distribute with customer-created applications are limited to those indicated by “Yes” in the “Re-distributable” column. Please note that distribution of any other modules will constitute an infringement of copyright.

Default Path and File Name	Item	Re-distributable
<instdir>\bin\btLib.dll	Library main body	Yes
<instdir>\bin\sbCore.dll	Structure builder	No
<instdir>\bin\sbAddIn.dll	Structure builder add-in	No
<instdir>\bin\sbAddInUI.dll	Structure builder add-in user interface	No
<instdir>\bin64\Btlib.zip	64 bit library (Please unzip the zip to use)	Yes
<instdir>\man\bcn400en.pdf	PDF manual	No
<instdir>\man\readme.html	Readme file	No
<instdir>\samples\csSamp*.*	C# Win forms sample program	No
<instdir>\samples\vbSamp*.*	VB.NET win forms sample program	No
<instdir>\samples\csDataSetSamp	C# DataSet object sample	No
<instdir>\samples\vbDataSetSamp	VB.NET DataSet object sample	No
<instdir>\samples\csWebSamp*.*	C# Web Forms sample program	No
<instdir>\samples\csNativeSamp*.*	C# Native Class sample	No

<installdir>%samples%\vbWebSamp%*.*	VB.NET Web Forms sample program	No
<installdir>%samples%\vbNativeSamp%*.*	VB.NET Native Class sample	No

64-bit Operating System Support

The 64-bit version of this product is to be found in a zip in the bin64 folder in the install directory of this product. To use simply unzip on a 64-bit OS and make use of the DLL inside. In order to run programs making us of this DLL , VC++ 2008 64-bit runtime libraries are necessary. This can be obtained by downloading and installing the following from Microsoft:-

Microsoft Visual C++ 2008 Redistributable Package (x64)

<http://www.microsoft.com/downloads/details.aspx?familyid=BD2A6171-E2D6-4230-B809-9A8D7548C1B6&displaylang=en>

Registering and un-registering Structure Builder on Windows Vista and Windows 7

In order to use the Structure Builder utility, it must be first registered with the OS.

When registering and un-registering Structure Builder on a Windows Vista or later OS, it is necessary to run with administrator rights. This process can be performed as follows:

1. Run the Command Prompt as an administrator
2. Change to the directory C:\Program Files\TechKnowledge\Btrieve Classes for .NET 4.0\bin
3. To register Structure Builder as an application run the install_sb.bat within the above folder.
4. To un-register Structure Builder as an application run uninstall_sb.bat.

If an error like that below is seen, please check that you ran the Command Prompt as an administrator:

The module ”%sbAddIn.dll” was loaded but the call to DllRegister failed with error code 0x 80070005.

If an error like that below is seen, please confirm that the current directory is C:\Program Files\TechKnowledge\Btrieve Classes for .NET 4.0\bin

The module ".\sbAddIn.dll" failed to load.

Make sure the binary is stored the specified path or debug it to check for problems with the binary or dependant .DLL files.

Tutorial

The following describes the procedure for creating an ASP.NET application using Btrieve Classes for .NET. The application used for this explanation is a simple application using the Table control to show the Person table in the DEMODATA database of Pervasive.SQL.

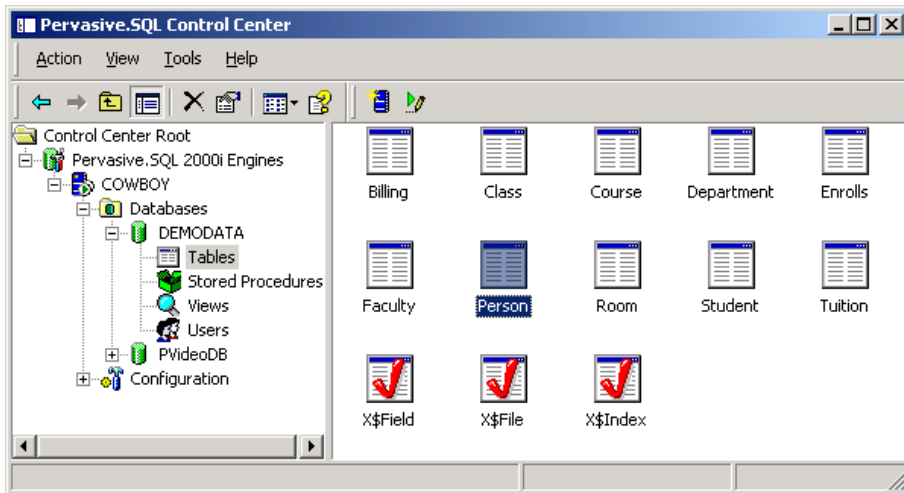
Preparation of the Btrieve Database

In this section data is prepared using the Pervasive.SQL database operation tool “Pervasive Control Center.” For details on how to use this tool for creating databases etc., please refer to the Pervasive.SQL manuals.

Knowing in which folder the database is created is important information in the case of database creation. This is because it is necessary to specify the directory in which the database exists in the DDFDir property when programming.

The following describes the method for obtaining the location of the demonstration database of Pervasive.SQL.

- ① From Pervasive.SQL’s Pervasive Control Center, select Tables from beneath DEMODATA in the Databases folder. With the tables are shown in the left pane, select the “Person” icon, and then right-click the mouse and select Properties.



- ② The dictionary path in the Table Properties window shown below is the directory to which DDFDir should be set. For a default Pervasive.SQL installation, this folder is C:\PVS\W\DEMODATA .

Table Properties - Person ? X

Statistics

Parameter	Value
Table Name	Person
Table Location	C:\PVSW\DEMODATA\Person.mkd
Dictionary Path	C:\PVSW\DEMODATA
File Version	7.0
Record Length	333
Page Size	4096
Number of Records	1500
Number of Indexes	3
Number of Duplicate Ptrs	0
Number of Unused Pages	0
Variable Records	Yes
Var Rec Blank Truncation	No
Data Compression	No
Key Only File	No
Index Balancing	No
Freespace Threshold	5%
Uses Alternate Collating Seq.	No
System Data Key	Yes

Close

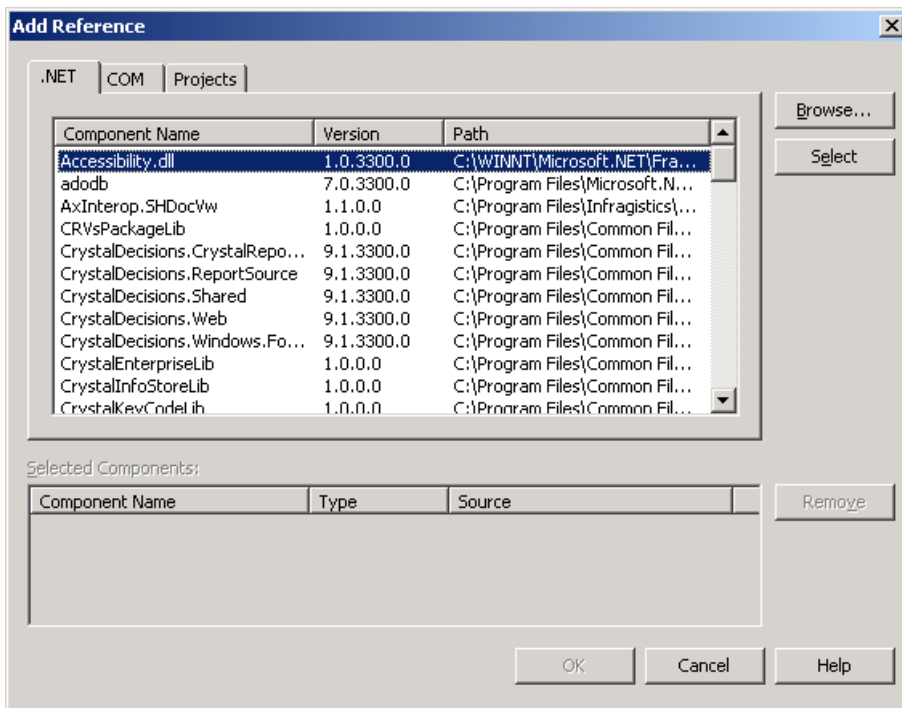
Starting the ASP.NET Project

Start Visual Studio .NET, and select the new project option. Select the language to use in the left pane, and select “ASP.NET Web application” from the right pane. In this tutorial, the example is one in which the C# language is selected.

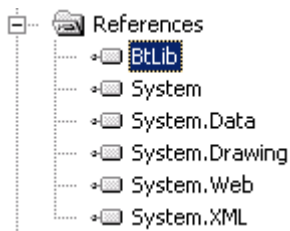
Setting of References to the Project

To enable use of the Btrieve Classes for .NET from an application, it is necessary to add to the project a reference to BtLib.DLL. To add this reference,

right-click the “References” folder in the Solution Explorer tab, and then select “Add Reference”. When the following “Add Reference” dialog appears, click the “Browse” button and move to the c:\Program Files\techknowledge\Btrieve Classes for .NET\bin folder. Then select BtLib.DLL and hit OK.



The following shows the view of the Solution Explorer tab when BtLib is added to the References.



Name Space Declaration

C# Language

Add the following line to the beginning of the declarative section of the source file.

```
using BtLib;
```

Visual Basic .NET Language

Add the following line to the beginning of the declarative section of the source file.

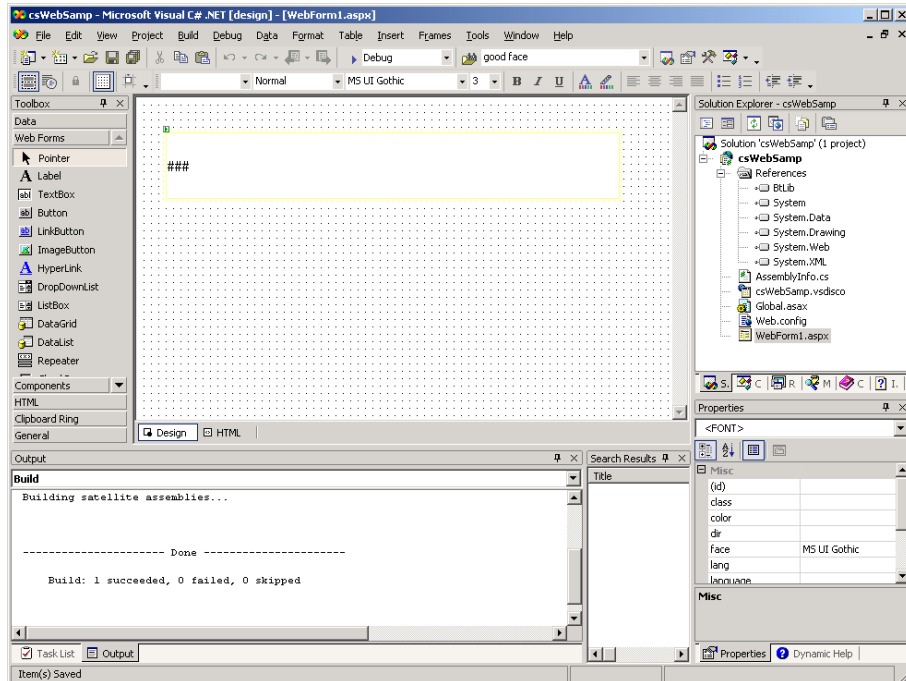
```
Import BtLib
```

WebForm Settings

First we will place a table to show the data on the WebForm. The procedure to do this is as follows.

- ① Display WebForm1.aspx from Solution Explorer.
- ② Click the “Web Forms” tab in the Toolbox.
- ③ Drag “Table” to the form.

The screen will appear as follows after performing the above procedure.



Creating the Data Acquisition Code

Next a private method for displaying data as a table should be created. The code for this is as follows below. This is inputted by double-clicking the WebForm and then entering the code below in the displayed WebForm1.aspx.cs file.

```
private void fillTable()
{
    BtLib.Ddf demodata = new BtLib.Ddf("C:\¥pvsww¥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();
}

```

As the Table control has to be built every time the form is initialized, the fillTable method above is invoked from the Page_Load event:-

```

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

```

To show the table as the result of aPostBack instead of with initial values, code like the following can be used referencing IsPostBack of the Page object .

```

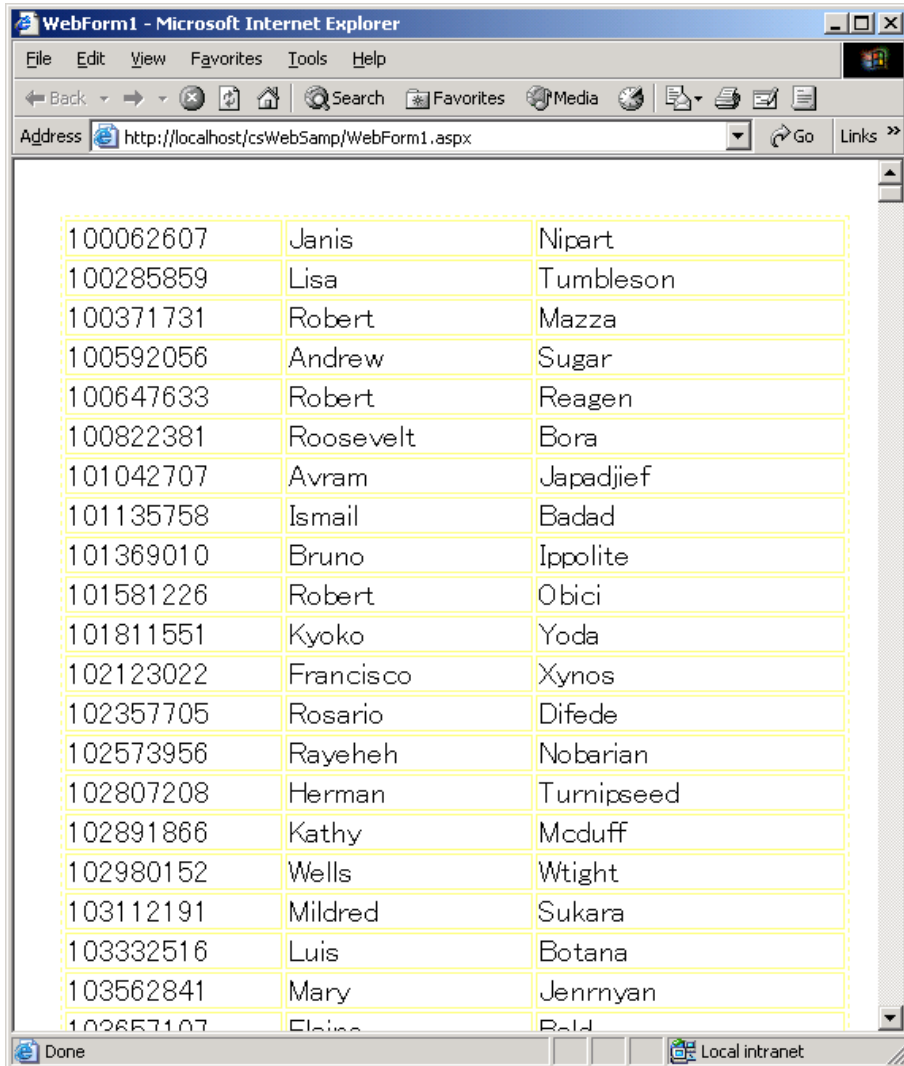
private void Page_Load(object sender, System.EventArgs e)
{
    if(IsPostBack)
    {
        fillTable();
    }
}

```

This code is included with the product as a sample. The project name is csWebSamp. The Visual Basic .NET language sample is named vbWebSamp.

Result of Execution

When the project is built and executed, the result will be like the following.



The screenshot shows a Microsoft Internet Explorer browser window titled "WebForm1 - Microsoft Internet Explorer". The address bar displays "http://localhost/csWeb5amp/WebForm1.aspx". The main content area contains a table with three columns and 20 rows of data. The table is highlighted with a yellow border. The status bar at the bottom shows "Done" and "Local intranet".

100062607	Janis	Nipart
100285859	Lisa	Tumbleson
100371731	Robert	Mazza
100592056	Andrew	Sugar
100647633	Robert	Reagen
100822381	Roosevelt	Bora
101042707	Avram	Japadjief
101135758	Ismail	Badad
101369010	Bruno	Ippolite
101581226	Robert	Obici
101811551	Kyoko	Yoda
102123022	Francisco	Xynos
102357705	Rosario	Difede
102573956	Rayehev	Nobarian
102807208	Herman	Turnipseed
102891866	Kathy	Mcduff
102980152	Wells	Wtight
103112191	Mildred	Sukara
103332516	Luis	Botana
103562841	Mary	Jenrnyan
103657107	Flaine	Bald

If a status 94 exception is generated by the Open method at the time of execution, remedy by changing the execution user of the ASP.NET application as described in Appendix-B of this manual.

About the Sample Programs

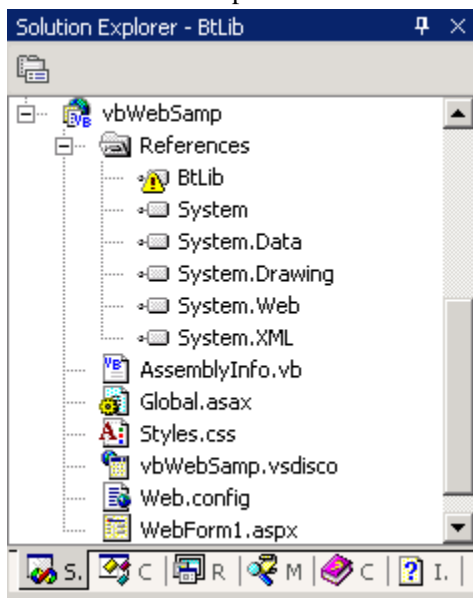
Outline

To operate the sample programs, it is a prerequisite that Visual Studio 2008 or 2010 is installed. The sample programs that operate in a Windows Form can be read into Visual Studio by selecting them from this product's menu, or they can be selected as *.vbproj files or *.csproj files from the Samples directory in the install directory of this product using Explorer.

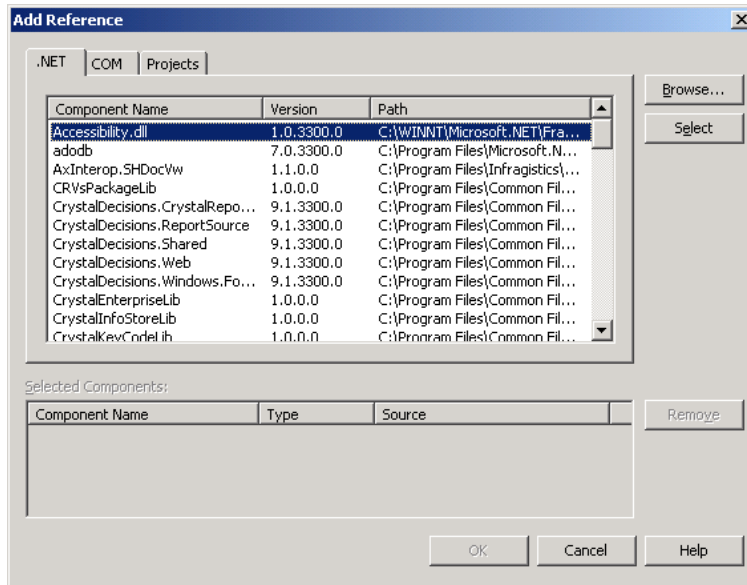
For the web sample programs too, so long as Visual Studio is in debug mode, the ASP.NET development web server will be automatically loaded, so it is unnecessary to register the site with IIS etc.

Referencing BtLib

It may not be possible to reference BtLib.DLL during execution of a sample, or during sample compiling. In this case, the reference to BtLib should be deleted once and then the reference setting should be made again. The following shows a view of Solution Explorer when referencing has become impossible.



Select BtLib from the References and press the Delete key to remove the reference. Then right-click “References” again and select “Add Reference.”



Next click the “Browse...” button on the above screen to specify the btlib.dll file in the folder c:\program files\techknowledge\Btrieve Classes for .NET 4.0\bin.

Visual Studio .NET project file versions

The format of all project files and solution files in this product is Visual Studio .NET 2005. This means that they cannot be opened in lower versions of Visual Studio (such as 2002,2003 versions).

When sample programs are read in by the higher versions of Visual Studio 2008/2010, the Visual Studio Conversion wizard runs. By following the instructions on the wizard screens, the sample programs included with this product can be converted into Visual Studio 2008/2010 format. The sample programs have been checked for correct building and running in both Visual Studio 2008/2010.

Application Configuration Files in a .NET 4.0 environment

In a Microsoft .NET 4.0 environment, if a .NET 2.0 component is used then it is necessary to create and add an Application Configuration File to the project in order to build or distribute. Usually this file is named App.config and contains the following content:-

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0"/>
  </startup>
</configuration>
```

64-bit execution of the sample programs

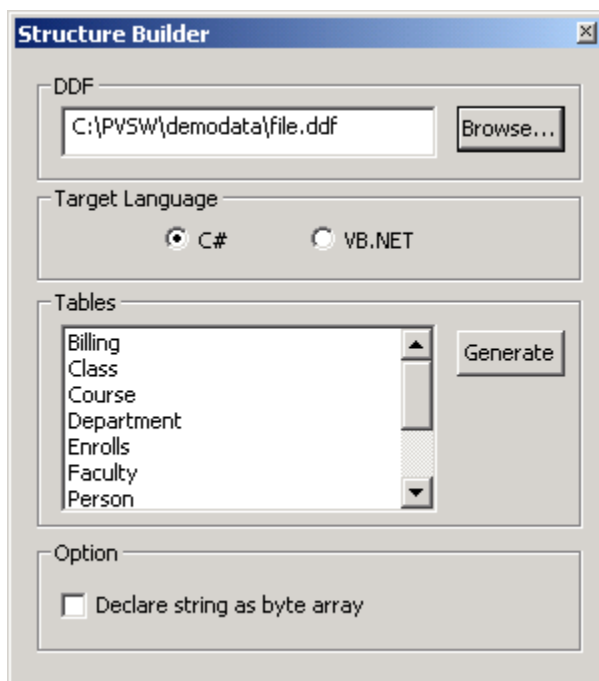
After install, the sample programs included with this product are set to run in 32-bit mode. In order to run the sample programs in 64-bit mode on a 64-bit OS, please carry out the following steps:-

1. Install a 64-bit version of Pervasive PSQL.
2. Beneath the install directory of this product in the bin64 directory there is a Btlib.zip, containing the 64-bit version of Btlib.dll. Unzip this.
3. After if necessary taking a backup of the existing 32-bit version of btlib.dll, copy the 64-bit version of Btlib.dll to the bin folder of the install directory of this product.
4. On loading a sample project into Visual Studio change the Platform Target to x64.
5. Build and run the sample program.

Structure Builder

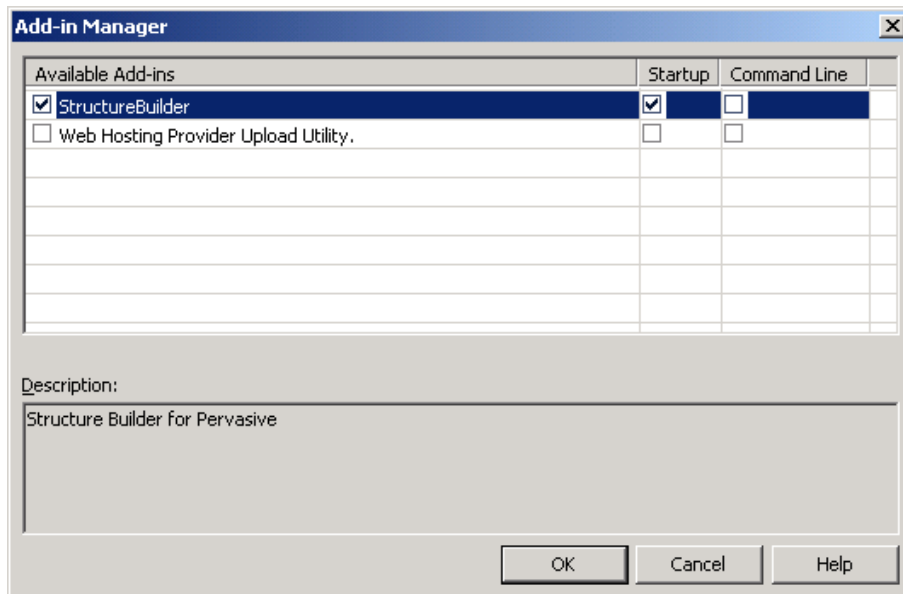
Overview

In Microsoft .NET, it has become possible to specify structure alignment and set the data area in a desired location in the structure. This was not possible in Visual Basic 6.0. When using this function to define the structure, there are a number of data formats that must be specified for the attributes of each field. In the case of tables with many columns, the definition of this structure requires much work, consuming both time and energy. Btrieve Classes for .NET provides a Structure Builder as a Visual Studio .NET add-in. This is a tool that automatically generates the structure declaration with added attributes from a set of DDF files and adds it to the source code at a desired location. With Pervasive.SQL V8.5 and above, it is possible to set a Database URI for a secure database as the location of the DDF files. The following shows a sample screen of the Structure Builder.



How to Start the Structure Builder

The following screen appears when “Add-in Manager” is selected from the “Tools” menu in Visual Studio .NET.



When the checkbox of the “StructureBuilder” add-in is checked, the Structure Builder’s Tools window is added to Visual Studio .NET. The window can then be dragged to a desired location.

How to Insert Structures

The following procedure is used to insert the structure declaration into the target source code.

- ① Click the “Browse...” button to specify the directory in which the DDFs are located.
- ② Select the Target Language.
- ③ From the table names displayed in the Tables list box select the table for which you wish to generate a structure.
- ④ Place the cursor at the location in the source code where you wish the structure to be inserted.

- ⑤ Click the Generate button.
- ⑥ The structure is inserted into the source code. At the same time, the structure is also copied to the Clipboard. Please note that the previous content of the Clipboard is destroyed by this.

Structure Builder Restrictions

- ① Variable length data is not supported.
- ② When a column name that is a reserved word in the target language exists, the column name in the generated structure must be converted to a character string that is not a reserved word.
- ③ The Btrieve data formats that are not found in .NET are converted to Byte format. The Native class conversion method can be used to convert from Byte format to .NET data format.

Selecting Variable Data Type for Character String Data

For example, if the structure is declared as follows in C# ,

```
[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;
}
```

because in Btrieve the character string type data is padded with spaces, code as follows might be assumed to be OK in order to declare a 25-byte area within the code at execution time .

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

However, in reality, the area reserved for the `Building_Name` is set as 24 bytes only, and the last byte is set as a binary null. This can be concluded by referring to the specifications of `System.Runtime.InteropServices` of Microsoft .NET. To cope with this, a method for forcing the character string area to match the `Btrieve` specification has been added to the Native Class. (See calling the `FixString` method on the third line below)

```
System.IntPtr keyPtr =
Marshal.AllocCoTaskMem(Marshal.SizeOf(pk));
Marshal.StructureToPtr(pk, keyPtr, true);
Native.FixString(keyPtr, 1, 25); //Match character string to
Btrieve specifications!
```

It is an advantage that it is simple to declare the character string by this method. However, it is necessary to pass the character string offset on `IntPtr`, and it is thought that the maintainability of the program may not always be good as a consequence.

The following shows a method that allows declaration using the `Byte` data type also of a `String` data type field in the database without using this `FixString` method. Declaration of the structure becomes as follows.

```
[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;
}
```

A code example with the character string set in the `Building_Name` field area is shown below.

```
// get the appropriate encoder for the encoding of the data
in your Pervasive.SQL database
Encoding asce = Encoding.GetEncoding("us-ascii");
PrimaryKey pk = new primaryKey();
string s = "Eldridge Building      ";
pk.Building_Name = asce.GetBytes(s);
```

As the same unmanaged data area can be created by either of the code methods outlined above, the customer is free to select the coding method that best fits the customer's needs. To output a structure where a character string in the database is defined as a Byte array, select the "Declare string as Byte array" checkbox.

Class Library Reference

Ddf Class

Overview

This is the class for specifying DDFs, acquiring database information, and managing records.

Constructor

```
Ddf(DDFDir or Database URI)  
Ddf()
```

Overview

If the path to the DDFs of the database is specified as a parameter of the constructor, the DDFs are read, and the information therein is expanded and retained following initialization of the Ddf object . If you are using a Pervasive.SQL V8.5 or above secure database, instead of a path to the DDFs, a database URI starting with “btrv://” can be specified. In this case do not specify either the dbfile or the table parameters of the database URI. See the following example:--

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

For more information on database URIs refer to the Pervasive.SQL V8.5 and above manuals. If neither the DDF path nor the database URI is specified, only initialization of the Ddf object is executed. In this case, the database information can then be read into the Ddf object by specifying the DDFDir property and invoking the Load() method.

If the DDF path specified starts with the “btrv://” prefix, a Login Btrieve operation is implicitly performed within the constructor code. This login state is automatically released when the DDF class instance is discarded. You can also release the login state by executing the Logout method. To explicitly discard an instance of the DDF Class, at the end of the program etc., please use the Dispose method.

Property

CodePage

Data type

Int16

Overview

Specifies the Windows codepage of the data contained within the Pervasive database.

DDFDir

Data type

String

Overview

Specifies the directory in which the DDF files are located.

FloatSize

Data type

Int16

Overview

This is the number of decimal places when a Float or Bfloat type field is expanded into a character string. If this setting is too small, digits may be dropped in the case of 4-byte Float type, so an appropriate size should be set. The default is 10 digits. The maximum is 20 digits. The conversion accuracy depends on the runtime library of the Microsoft compiler.

FillSpace

Data type

Boolean

Overview

Specifies the trailing space processing when Btrieve String type data is acquired. In the case of "True," the trailing spaces will remain attached as they

are in the database. In case of the default setting “False,” the trailing spaces are discarded.

OwnerName

Data type

String

Overview

Specifies the owner name of the DDF files.

SignNibble

Data type

Int16

Overview

Sets the nibble value for expressing positive values in the DECIMAL and MONEY data types. The values 12 or 15 can be set. The default value is 15.

TableCount

Data type

Int32

Overview

Returns the number of tables in the database. For sample code please refer to the sample code of the TableNames property below.

TableNames

Data type

Indexed Array of Type String

Overview

An indexed array of strings containing the table names of the tables in the database. See sample code below.

Sample code

```

// retrieve colum names.
    BtLib.Ddf d = new BtLib.Ddf("c:\¥¥pvsww¥¥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());
    }

// retrieve table names.
    BtLib.Ddf d = new BtLib.Ddf("c:\¥¥pvsww¥¥demodata");
    for(int i=0; i< d.TableCount; i++)
    {
        System.Diagnostics.Debug.WriteLine( d.TableNames(i));
    }

```

Method

GetRecord

Format

```
Record GetRecord( string TableName )
```

Overview

Gets the record object related to the table specified by the parameter.

Load

Format

```
void Load()
```

Overview

Loads the database information specified by the DDFDir property.

Login

Format

```
void Login(ConnectionString As String)
```

Overview

This method allows a Login to be performed to a Pervasive.SQL V8.5 or above secure database. Specify the database URI starting with btrv: as detailed in the Pervasive.SQL V8.5 or above manuals. A code example is included below:

```
Ddf.Login("btrv://rasta@jamaica /demodata?pwd=reggae")
```

Logout

Format

```
void Logout()
```

Overview

This method allows a Logout to be performed when connected to a Pervasive.SQL V8.5 or above secure database.

Unload

Format

```
void Unload()
```

Overview

Releases the read-in DDF information. The Ddf object returns to its initial state. After releasing, the Record object acquired from the Ddf class becomes invalid.

Extended Class

Overview

This is the class that executes the Extended operations of Btrieve and Pervasive.SQL. Extended operations are known as methods that are extremely fast for data acquisition because they make it possible to limit the amount of data transferred because just a part of a record can be acquired. By means of the search condition settings, it is also possible to have the appropriate data selected on the server side and then transferred to the client. Extended operations are extremely convenient in terms of performance and network traffic; and the only drawback is that the methods for acquiring search results and the programming required are complicated because there are many structures to be set for field offset, length, and search information. When using a language that does not allow structure alignment to be set simply in 1-byte units, it becomes necessary to introduce even more tricky techniques. By acquiring the structure information from a set of DDFs, this class enables easy execution of the extended methods by using a collection and specifying the fields required.

The following is sample code using this Extended class. The First_Name field is extracted from the Person table found in the demodata folder of Pervasive.SQL.

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

```

The Extended class can also generate the System.Data.DataSet object of the Microsoft .NET Framework easily. The DataSet object can easily be linked to the data by means of data bound control like DataGrid, etc., provided by Visual Studio .NET. The following is sample code that generates the DataSet object and links it to DataGrid (the instance name is dataGrid1).

```

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
{

```

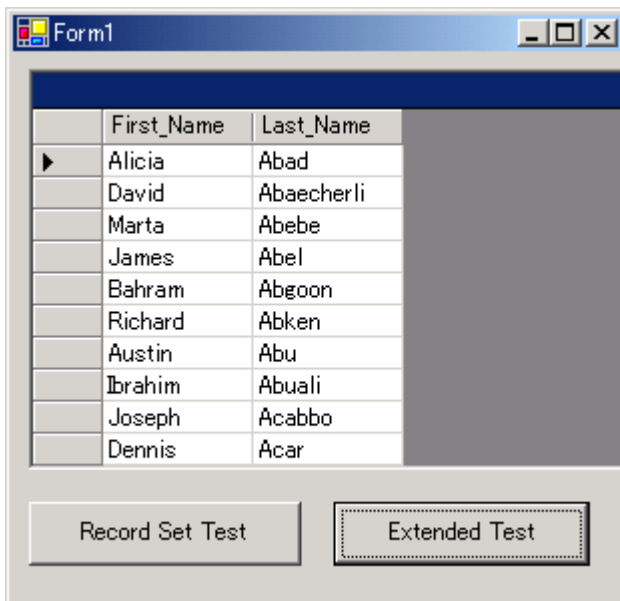
```

rc = ex.Read(BtLib.Operation.GetNextExtended);
if( ex.ResultCount > 0)
{
    ex.Fill(ds);
}
} while( rc != 9 );
r.Close();

dataGridView1.SetDataBinding(ds,"person");

```

Shown in a Windows form, the result of executing the above code is as follows.



Constructor

```
Extended();
```

Overview

For this class, generate instances using the GetExtended method of the Record class. There is no need to generate an instance of this object using New.

Property reference

IgnoreCase

Data type

Boolean

Overview

When data is collated using the search conditions specified by the SearchCond property, the search is performed while ignoring case (upper case or lower case characters) when this property is set to “true.” This setting is only valid for letters of the English alphabet.

Index

Data type

String

Overview

Sets the name of the key to be used when GetNextExtended/GetPreviousExtended is executed by the Read method.

IndexNumber

Data type

Short

Overview

Sets the index number of the key to be used when GetNextExtended/GetPreviousExtended is executed by the Read method. Setting the IndexNumber property causes the Index property to be set to Null.

Lock

Data type

LockBias

Overview

When you wish to use record lock control during execution of an Extended operation, the lock bias value is set in this property.

MaxRecords

Data type

short

Overview

Specifies the maximum number of extracted records.

Mode

Data type

ExtendedMode

Overview

Specifies when using the Read or Step method whether or not the current record should be included. To include the record, set this property to ExtendedMode.UC, for it not to be included set it to ExtendedMode.EG.

ResultCount

Data type

short

Overview

Holds the extracted data record count.

SearchCond

Data type

string

Overview

Sets the record extraction conditions.

SkipRecords

Data type

short

Overview

Specifies the maximum number of records to be skipped that do not agree with the record extraction conditions.

Method reference

AddField

Format

```
void AddField(string ColName);
```

Overview

Specifies a column to be extracted by an Extended operation.

Parameter

Name of column to be extracted.

Returned value

None

ClearField

Format

```
void ClearField();
```

Overview

Deletes all the columns specified by the AddField method to be extracted.

Returned value

None

Fill

Format

```
void Fill(DataSet ds);
```

Overview

Adds the Read or Step results to the DataSet object specified by the parameter.

Returned value

None

GetData

Format

```
void GetData(IntPtr pStructure);
```

Overview

Transfers the field data area extracted by an Extended operation to the memory area pointed to by IntPtr. (Bytes from the sixth byte onwards are transferred.) Declaring the structure in this area makes it easy to handle the field data extracted from the application. For the structure to be declared it is only necessary to consider the memory alignment and declare only the area of the column to be extracted.

Parameter

The memory area pointed to by IntPtr and used for storing the data extracted by the Extended operation.

Returned value

None

GetDataSet

Format

```
DataSet GetDataSet();
```

Overview

Creates and returns a DataSet object to hold the Read or Step results.

Returned value

DataSet object

GetDataTable

Format

`DataSet GetDataTable ();`

Overview

Returns a DataTable object based on the extraction column information set in the Extended object.

Returned value

DataTable object

MoveFirst

Format

`void MoveFirst();`

Overview

Moves to the first row of data in the results returned from the executed Extended operation. If the move is successful, the first value of the extracted data is set in the Extended object collection.

Returned value

None

MoveLast

Format

`void MoveLast();`

Overview

Moves to the last row of data in the results returned from the executed Extended operation. If the move is successful, the value of the extracted data is set in the Extended object collection.

Returned value

None

MoveNext

Format

```
void MoveNext();
```

Overview

Moves to the next row of data in the results returned from the executed Extended operation. If the move is successful, the value of extracted data is set in the Extended object collection.

Returned value

None

MoveTo

Format

```
void MoveTo(int Index);
```

Overview

Moves to the specified row of data in the results returned from the executed Extended operation. If the move is successful, the value of extracted data is set in the Extended object collection.

Parameter

Index specifying the desired row in the results of the data extraction. Specified in 0 base.

Returned value

None

Read

Format

```
short Read(Operation Op);
```

Overview

Executes Extended operations that use a key. The operations that can be specified are Operation.GetNextExtended and Operation.GetPreviousExtended.

Parameter

Specifies the Btrieve operation code.

Returned value

Btrieve status code

RecordExists

Format

```
bool RecordExists(short value);
```

Overview

Determines the status of the results of the execution of an Extended operation, and in the case of a status code in which search result records exist it returns “true”. If “false” is returned, it means that there were no search result records.

Parameter

Specifies the Btrieve status code from the Extended-related operation execution method Read.

Returned value

Boolean

Step

Format

```
short Step(Operation Op);
```

Overview

Executes Step-related Extended operations. The operations that can be specified are Operation.StepNextExtended and Operation.StepPreviousExtended.

Parameter

Specifies the Btrieve operation code.

Returned value

Btrieve status code

Sample code

```
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

Overview

This class is derived from System.Exception, and for the following Btrieve Classes for .NET classes an exception is generated in the case of an error in this class library. This class (BtLib.Exception) is thrown as exception information.

Ddf
Record
Extended

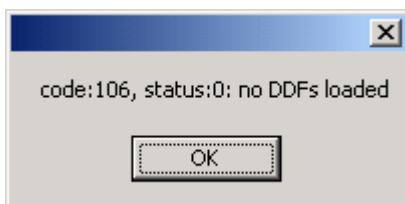
Exceptions thrown in classes other than those mentioned above are system exceptions and are not thrown by this Exception class.

It must also be noted that the status of Btrieve operation by the above classes is not thrown as an exception. (In this class library, the Btrieve status is not an error, but a status and is therefore not treated as an exception.)

The following is an example of an exception handling code.

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

As the Ddfs are not loaded yet, the message box will show a message like the following.



The numeric value 106 above is the value of the `BtLib.Exception.Errors` collection and it is a code that indicates the error. The `Btrieve` status is indicated after “Status:” when the status is anything but 0. “no DDFs loaded” is the detailed explanation of the error code 106.

When the default error indications should be changed, code similar to the following can be used to change the message.

```
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("Load DDF");
    }
}
```

Constructor

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

Please note that the things that generate an exception and create the instances of this class have no meaning during normal use of the application.

Properties

BtrieveStatus

Overview

If the status is reported as anything but 0 from `Pervasive.SQL/Btrieve` at the time an exception is generated, this is retained in this property. For details on the value of this property, see the `Pervasive.SQL/Btrieve` manuals.

Data type

Int16

ErrorCode

Overview

This code indicates the reason for the occurrence of the error. It is always set when an exception occurs. For the meaning of the error codes, refer to Appendix-D at the end of this manual. If you want to contact our technical support regarding an occurred exception, please be ready to inform our technical support about the value of this property and the value of the BtrieveStatus property.

Data type

Exception.Errors

Method

ToString

Format

```
string ToString();
```

Overview

Acquires the error code, Btrieve status code, and an explanation of the error as a character string. The explanation is only a summary, and if detailed information is necessary the error should be diagnosed by noting the value of the BtrieveStatus and ErrorCode properties after which the relevant manuals should be consulted.

Returned value

Character string including error code, Btrieve status code and explanatory text on error.

Native Class

Overview

This class has been created for the purpose of making it easy to call the Btrieve API from a managed language environment. It is possible to call a DLL by means of the .NET Framework's Library function from the managed language, but the program will often end up complicated since DLL declaration and the conversion of data from managed to unmanaged are necessary. Consequently, this has been gathered together as a Native class that makes such operations simple. Furthermore, in order to support the character string operation specifications of Encoder.GetString, this class also offers a helper method for conversion of managed data to a Byte array. Since all the methods related to calling Btrieve API are static methods, do not use this class to create instances, but for calling methods. Sample code using this class can be found in Appendix-A.

Constructor

Does not exist.

Property

Does not exist.

Methods

BtrCall

Format

```
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);

```

Overview

Calls the Btrieve API's BtrCall. As the parameters are the same as for the BtrCall API, please refer to Pervasive.SQL SDK manuals for details on the parameters. With the second overload declaration it is possible to specify the data buffer in unmanaged memory. With the third overload declaration it is possible to specify both the key and the data buffer memory areas in unmanaged memory. For calls that specify unmanaged memory areas, the structure can be specified by using System.Runtime.InteropServices. For how to declare this structure, please refer to the explanation found in the Appendix.

Returned value

Btrieve status code

Sample code

```

// Example with data area specified as a Byte array.
data = new Byte[334];
dataLength = (short)data.Length;
keyBuf = new Byte[128];
rc = Native.BtrCall(Operation.GetFirst,
    posblk,data,
    ref dataLength,
    keyBuf,
    keyBufLen,
    0);

// Example with the data area specified as a structure.
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

Format

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

Overview

Calls the Btrieve API's BtrCallID. As the parameters are the same as the BtrCall API, please refer to the Pervasive.SQL SDK manuals for details.

Returned value

Btrieve status code

FixString

Format

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

Overview

Replaces the null bytes of the character string data found in the IntPtr specified area specified by offsetFrom and offsetTo with spaces. This is used for setting trailing blanks in Btrieve character string type data. For an explanation of how to use this method, please refer to the section on "Structure Builder."

Parameters

The first parameter specifies the unmanaged data area acquired by calling Marshall.StructureToPtr. Normally, a structure area is thought to be specified and the position for the start of the conversion and the position for the end of the conversion within this area are specified by the offset second and third parameters. (Base 0 specification)

Returned value

The converted IntPtr area

GetBoolean

Format

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

Overview

Returns the data at the offset, in the specified Byte array, as a .NET Framework Boolean data type.

Parameters

The Byte type array data returned from the Btrieve API is specified in the first parameter. The second parameter is the offset of the relevant data within the Byte array.

Returned value

Converted Boolean type data

GetBytes

Format

```
static Byte [] GetBytes(Boolean b);  
static Byte [] GetBytes(Char c);  
static Byte [] GetBytes(DateTime d);  
static Byte [] GetBytes(DateTime d, BtrieveTypes targetType);  
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 targetType, int  
size, int dec);
```

Overview

Stores .NET Framework data types as Byte arrays. The method is overloaded in order to support each of the .NET Framework data types. Used before calling BtrCall in the Native Class when it is necessary to set data in a Byte array before use. When the Encoding class of the .Net Framework is used, the character string type can be converted to a byte-type array. Accordingly, this function is not offered here.

Parameters

The first parameter is the original data for conversion. When it is possible to map to multiple Btrieve data types from the original data type for conversion, the Btrieve data type to be used at the target is specified in the second parameter. For the Decimal type only, the size and decimal places are set in the third and fourth parameter, respectively.

Returned value

Converted Byte array.

Sample code

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

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

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

GetDate

Format

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

Overview

The date data at the offset specified in the Byte array is returned as a .NET Framework DateTime data type. The time portion of the returned DateTime type data is set to 0.

Parameters

The array of Byte type data returned from the Btrieve API is specified in the first parameter. The second parameter is the offset to the relevant data in the Byte array.

Returned value

The converted DateTime type data

GetDecimal

Format

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

Overview

Returns the Btrieve decimal type data at the offset specified in the Byte array as a .NET Framework Double data type. Supported Btrieve decimal data types are Numeric, Numericsa, Numericsts, Decimal, and Money.

Parameters

The array of Byte type data returned from the Btrieve API is specified in the first parameter. The second parameter is the offset to the relevant data in the Byte array. The third parameter is the size of the target data, and the fourth parameter specifies the number of digits following the decimal point. The fifth parameter specifies the Btrieve data type in the Byte array.

Returned value

Converted Decimal type data.

GetDouble

Format

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

Overview

Returns the floating-point data (8-byte size) at the offset specified in the Byte array as a .NET Framework Double data type.

Parameters

The array of Byte type data returned from the Btrieve API is specified in the first parameter. The second parameter sets the offset to the relevant data in the Byte array.

Returned value

Converted Double type data

GetInt16

Format

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

Overview

Returns the two-byte integer data at the offset specified in the Byte array as a .NET Framework Int16 data type.

Parameters

The array of Byte type data returned from the Btrieve API is specified in the first parameter. The second parameter sets the offset to the relevant data in the Byte array.

Returned value

The converted Int16 type data

GetInt32

Format

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

Overview

Returns the four-bytes integer data at the offset specified in the Byte array as a .NET Framework Int32 data type.

Parameters

The array of Byte type data returned from the Btrieve API is specified in the first parameter. The second parameter sets the offset to the relevant data in the Byte array.

Returned value

The converted Int32 type data

GetInt64

Format

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

Overview

Returns the eight-byte integer data at the offset specified in the Byte array as a .NET Framework Int64 data type.

Parameters

The array of Byte type data returned from the Btrieve API is specified in the first parameter. The second parameter sets the offset to the relevant data in the Byte array.

Returned value

The converted Int64 type data

GetSingle

Format

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

Overview

Returns the four-byte floating-point data at the offset specified in the Byte array as a .NET Framework Single data type.

Parameters

The array of Byte type data returned from the Btrieve API is specified in the first parameter. The second parameter sets the offset to the relevant data in the Byte array.

Returned value

The converted Single type data

GetTime

Format

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

Overview

Returns the Time data at the offset specified in the Byte array as a .NET Framework DateTime data type. The current system date is set into the date section of the DateTime data type.

Parameters

The array of Byte type data returned from the Btrieve API is specified in the first parameter. The second parameter sets the offset to the relevant data in the Byte array.

Returned value

The converted DateTime type data

Trim

Format

```
static string Trim(string src);
```

Overview

If for example, to acquire a character string from the record buffer code like the following is used:

```
data = new Byte[334];  
//  
// acquired through btrieve operation  
// ...  
string str = Encoder.GetString(data,9,16);
```

the size of the character string created by string is always the size of the area specified by GetString. However this area will also include trailing nulls (leaving only 14 bytes for the string itself in the above sample). If the trailing nulls are allowed to remain in a string like the one acquired here, the linking of the character string, etc., will not function correctly. By using this Trim method, the trailing nulls can be deleted and the character string is thus converted to one that has been confirmed to work correctly with the .NET Framework.

Returned value

Character string following conversion

Sample code

```
// get the appropriate encoder for the encoding of the data
// in your Pervasive.SQL database
Encoding asce = Encoding.GetEncoding("us-ascii");

// code reading record to data area(abbreviated)
// ...
// ...
string firstName = Native.Trim(asce.GetString(data,9,16)); //
string lastName = Native.Trim(asce.GetString(data,26,26)); //
```

Record Class

Overview

It is possible to access data by record with the Record class acquired from the Ddf class. The Record class is derived from System.Collections.HashTable, and access to the fields included in a record is made by accessing by column name as shown below.

```
Record rec = ddf.GetRecord("Person");// Get Record object
rec["First_Name"] = "Suzuki";
rec["Last_Name"] = "Papaya";
```

The data set in the HashTable collection is referenced to during data output-related Btrieve operations like Insert and Update, and output to the database.

Also, in case of Btrieve operations like GetEqual and GetGreater, etc., that require reference to a key, set the appropriate field values for the key before executing such operations.

When the Get and Step operations end normally, the values read from all records are set into the HashTable collection. The field values set before the Get and Set operations are replaced by the read-in values. The following is a code sample that references values obtained by executing a GetFirst.

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

}

The accessor to the record object is a character string type. If the character string specified as accessor is not found in the DDF declaration, a null pointer exception is generated (the HashTable object's default exception). Also, the accessor character string is case sensitive. For example, for the DEMODATA database of Pervasive.SQL, a null pointer exception is generated if the "Comments" column of the Person table is specified as "comments" (all lower case) or "COMMENTS" (all upper case).

Constructor

```
Record();
```

For this class, use the GetRecord method of the Ddf class to generate instances. When an instance of this object is generated using "new", set the table information using the Attach method.

Properties

ColumnCount

Data type

Int32

Overview

Contains the number of columns in a record.

DataFileName

Data type

String

Overview

Referenced when the Create and Open methods are executed. If you want to set the data file name dynamically at execution time, change this property before executing the Create or Open methods.

FileFlag

Data type

short

Overview

Referenced when the Create method is executed. Specify the file flag when using the Create operation.

Index

Data type

String

Overview

Specifies the index name of the index to be used when a read using a key is executed in the Read method.

Sample code

The following is an example of the execution of a GET EQUAL operation with the Person table of Pervasive.SQL. The index name "Names" is set in the Index property. The Names index is a two segment key, so data should be set in both the First_Name and Last_Name columns.

```
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

Data type

Short

Overview

Sets the index number of the key to be used when read-in is executed using a key in the Read method. Setting the IndexNumber property causes the Index property to be set to Null.

Sample code

The following is an example of the execution of a GET EQUAL operation with the Person table of Pervasive.SQL. The index number for the “Names” key is set in the IndexNumber property. This index is a two segment key, so data should be set in both the First_Name and Last_Name columns.

```
BtLib.Ddf d = new BtLib.Ddf("c:¥¥pvsw¥¥demodata");
BtLib.Record r = d.GetRecord("person");
r.Open();
r.IndexNumber = 1;
r["First_Name"] = "Koichi";
r["Last_Name"] = "Adachi";
short rc = r.Read(BtLib.Operation.GetEqual);
```

IsOpen

Data type

Boolean

Overview

Returns the open state of the data file associated to the record. “True” is set when the Open method is called and is completed normally. “False” is set when the Close method is completed normally. It is recommended to ensure that related resources are released by calling the Close method in this class when the application is closed. Whether it is necessary to call the Close method when an exception occurs can be determined by referencing this property.

Lock

Data type

LockBias

Overview

Referenced by the Read and Step operations. To apply a lock at the time the record is read-in, set to any value other than NoLock. In addition to releasing

existing locks by the single record lock method, the lock can also be released by the Unlock method.

NullKeyValue

Data type

String

Overview

Referenced when the Create method is executed. Sets the null key value. The default value is 32.

OpenMode

Data type

short

Overview

Referenced by the Open method, this sets the open mode when the Btrieve data file is opened. For details, please refer to Open Mode in Pervasive.SQL manuals.

PageSize

Data type

short

Overview

Referenced by the Create method. Sets the page size for the Btrieve data file. The default is 4096. For details on the values of this property, please refer to Pervasive.SQL manuals.

Methods

ClearData

Format

```
short ClearData();
```

Overview

Initializes the contents of the data buffer. Numeric columns are set to zero, text strings of a set length are set to all spaces, text strings with no length specified are initialized.

Returned value

None.

Sample code

```
Dim ddf As BtLib.Ddf = New BtLib.Ddf("c:\pvsw\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

Format

```
short Close();
```

Overview

Closes the Btrieve data file and releases related resources. It is a prerequisite that the Open method has been called and completed normally before this method is called.

Returned value

Btrieve status code

Create

Format

```
short Create();
```

Overview

Generates the Btrieve data file as a new file. A data file subject to this must not be in the closed state. If resources cannot be reserved at the time of generation, the Btrieve Status is notified as an exception. Reference this value to diagnose the error.

Returned value

Btrieve status code

Delete

Format

```
short Delete();
```

Overview

Erases the current record. The current record must be set in advance using the Read or Step method. It is necessary to keep in mind that no current record exists, for example, immediately after the execution of the Open or Delete methods.

Returned value

Btrieve status code

GetBytes

Format

```
Byte [] GetBytes(String colName);
```

Overview

Returns the specified column data as a Byte array. Note that if Btrieve/Pervasive data is not read in by the Read or Step method before this method is called, undefined data in the internal buffer is returned. (Normally, the initial value is 0.) Also, note this version does not support variable length records.

Parameter

Column name

Returned value

Byte array

GetData

Format

```
void GetData(IntPtr pStructure);
```

Overview

Transfers the current record image to the memory area pointed to by IntPtr. This is used to transfer data to a structure that contains a record image. It is necessary to consider memory alignment when defining this structure. An example of this structure declaration can be found in the Appendix. Please refer to this.

Parameter

The memory area in which to store the record image, pointed to by IntPtr

Returned value

None

Sample code

```
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();
```

GetDataColumn

Format

```
DataColumn GetDataColumn(Int32 Index);
```

Overview

This is used to obtain the Microsoft .NET framework DataColumn object.

Parameter

Specify in the index parameter the zero-based number of the column for which you wish to obtain a DataColumn object. The order of the columns is that specified in the table definition (in the database's DDF files).

Returned value

Returns a DataColumn object.

Sample code

```
BtLib.Ddf d = new BtLib.Ddf("c:¥¥pvs¥¥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

Format

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

Overview

Returns the DataSet object of Microsoft .NET Framework. If called with no parameter specified, all columns and all records are returned to the DataSet. Only the specified columns are returned to the DataSet when called with the column names to be extracted specified in a String type array. If any number other than 0 is specified for an Int32 type value, a DataSet is created and returned with this value acting as the upper limit on the number of records.

Parameter

It is possible to set up to 2 parameters. "String [] *fields* " is a String type array of the column names to be extracted. "Int32 *maxRecords* " is the upper limit on the number of records to be returned.

Returned value

Returns the DataSet object.

GetDataTable

Format

```
DataTable GetDataTable ();
```

Overview

Returns the DataTable object of the Microsoft .NET Framework. The returned DataTable object includes the column information from the DDFs. Row is left in the empty state.

Returned value

DataTable object

GetNumOfRecords

Format

```
Int16 GetNumOfRecords(Int32 records);
```

Overview

Returns the number of records in the open data file.

Parameter

Number of records

Returned value

Btrieve status

GetPosition

Format

```
Int16 GetPosition(Int32 PhysicalPosition);
```

Overview

Returns the physical position of the current record. The physical position can then be used to read in the record again by specifying it in the Read method parameter.

Parameter

Physical position of the current record is returned. When using the C# language the Out attribute must be specified.

Returned value

Btrieve status

GetRecordLength

Format

```
Int32 GetRecordLength();
```

Overview

Returns the record length of the data file related to the Record class. Convenient for reserving a memory area with the same size as the record image.

Parameter

None

Returned value

Record length

Open

Format

```
short Open();
```

Overview

Opens Btrieve data files. The data file opened is the data file linked to the table specified as a parameter in the GetRecord method of the Ddf class.

Returned value

Btrieve status code

Query

Format

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

Overview

By specifying the same name as that of a column in a record, a generic list for that member can be obtained. The generic list thereby obtained can be used from LINQ.

Parameter

T

This is the class that receives the related table from the record class. The property name is made identical to that of the column in the record class. It is not necessary for all the columns in a table to be defined in the class, you can define the subset of columns from which you wish to pull out data.

KeyNo

This is the key number to be used when Btrieve read-type operations are to be done.

extendedConditions

These are the search conditions used as a filter on the results returned to the generic list. The same format is used as in this product's Extended class filtering conditions (i.e. the same format as the Extended class's SearchCond property is used).

In the current version of the product, there are 3 overloads provided for this method. In the first of the above formats, data is read using Btrieve Step type operations. In the second format, data is read into the list using Btrieve read with a key operations using the key specified in the KeyNo parameter. In the third format, Extended type operations ordered by a key are used to obtain the data.

Returned value

The specified generic list

Sample Code

The following is an example showing how to use LINQ to pull out data from a specified record. Data from the Person table in Pervasive PSQL's DEMODATA database is being used.

```
try
{
    BtLib.Ddf d = new BtLib.Ddf("c:¥¥pvsww¥¥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());
    }
}

```

The above sample code is referencing a Person class of the following form. Note that the each property name must match its corresponding column in the Person table.

```

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

Format

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

Overview

When the first of the above overloaded formats is used, this method reads in the record in accordance with the specified read using a key operation code.

When the second of the above overloaded formats is used, this method reads in the record at the physical position specified in the parameter. The read-in record data is retained as a Record object collection. If the Lock property is set the specified lock is also set when this method is executed.

With the third overloaded format the data resulting from the read operation specified in the first parameter is set into the class instance specified in the second parameter. The specified class property names must match the names of the columns in the table being read. Even if the properties are only a subset of all the columns in the table, data can still be read in.

Parameter

The Read Btrieve operation code that uses an index or the physical location of the record as obtained by the GetPosition method.

Returned value

Btrieve status code

SetBytes

Format

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

Overview

Sets the specified column data as a Byte array. After calling this method, the change is actually reflected in the Btrieve/Pervasive.SQL database when Write is called. The data specified as a Byte array must be set with the correct size in the Byte array and in the correct format in accordance with the specifications

of the data type of the specified column. Since wrong data being written may lead to unexpected results in the application that reads this data, please be especially careful. (For example, if improper data is set for the date format, this will generate an “unable to read” error when read-in by the Pervasive Control Center and it may not be possible to display any following data.) Also, this version does not support variable length records.

Parameter

Column name

Returned value

None

SetData

Format

```
void SetData(IntPtr pStructure);
```

Overview

Transfers the memory area pointed to by IntPtr to the current record image. Used for outputting a structure that contains a record image. The structure must be set while considering memory alignment. The Appendix contains an example of such a structure declaration; please refer to this.

Parameter

The memory area pointed to by the IntPtr holding a record image

Returned value

None

Step

Format

```
short Step(Operation op);
```

Overview

Reads in a record by means of the specified Step-related operation code. The data of the read-in records is held as a collection of the Record object.

Parameter

Step-related Btrieve operation code

Returned value

Btrieve status code

Write

Format

```
short Write(Operation op);
```

Overview

Executes the writing of the record by means of the specified Insert or Update operation code. The value of the written-in fields are set in the Record collection before executing this method.

Parameter

Insert or Update Btrieve operation code

Returned value

Btrieve status code

Transaction Class

Overview

Allow the use of Btrieve/Pervasive.SQL transaction management for connections in the Ddf Class. Transaction control calls are made using the static methods declared in this class.

Sample code

```
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();
```

Constructor

Overview

There is no constructor in the Transaction Class.

Property

Lock

Data type

LockBias

Overview

A property that holds the LockBias value. This property value is referenced when the parameters of the Begin() or BeginConCurrent() methods are omitted.

Method

Abort

Format

```
static short Abort();
```

Overview

Aborts the transaction in progress.

Begin

Format

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

Overview

Starts the transaction.

BeginConCurrent

Format

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

Overview

Starts a concurrent transaction.

End

Format

```
static short End();
```

Overview

Commits the transaction or concurrent transaction.

Reset

Format

```
static short Reset();
```

Overview

Executes a Btrieve/Pervasive.SQL Reset.

Appendix-A: Code Samples

Visual Basic.NET Sample Code

This is the Visual Basic.NET version of the C# sample code shown in the Tutorial.

```
Private Sub FillTable()  
    Dim rc As Integer  
    Dim demodata As New BtLib.Ddf("C:\pvs\w\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
```


Sample Code showing the use of IDictionaryEnumerator

The following Visual C# sample shows a method for acquiring all the data about the columns in the Record class. The Record class is derived from the HashTable, and it is therefore possible to get all the column names and their values directly by using IDictionaryEnumerator.

```
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();
```

GetDataSet C# Sample

This sample is for displaying data in the DataGrid of a Windows Form using Visual C#. The column names and the number of records to be displayed can be specified in GetDataSet of the Record class.

```
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());
    }

```

C# Sample Showing DataBind to DataGrid in WebForm

It is simple to bind data to a DataGrid of .NET Framework, which is frequently used in WebForm. The following is a sample code in C#.

```

private void Page_Load(object sender, 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# Record Scan Sample

This is a sample code for calling the Btrieve API using Native Class. The language is Visual C#. Records are read-in from the start of the key.

```

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 the appropriate encoder for the encoding of the data
in your Pervasive.SQL database
Encoding asce = Encoding.GetEncoding("us-ascii");
// 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(asce.GetString(data,9,16));
    //
    string lastName = Native.Trim(asce.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 Record Scan Sample

This is a sample code for calling the Btrieve API using Native Class. The language is VB.NET. Records are read-in from a starting position set by a Btrieve GetEqual operation.

```
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 encoder with the encoding set to that of the data in
the Pervasive.SQL database
sje = System.Text.Encoding.GetEncoding("us-ascii")

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# Insert Sample Code

This is a sample code for calling the Btrieve API using the Native Class. A record is inserted. The point to note is the part of the code in which data is converted from .NET data types to Byte arrays and then set. As data types like Double type are thought to be difficult to convert into a Byte array using the normal .NET Framework functions, the Native Class provides the GetBytes method as a helper method for converting each of the .NET Framework data types into a Byte array. Even in the following example, character strings, etc., are acquired as Byte arrays by using the GetBytes method of the Encoding class, but in the case of the Double type, the GetBytes method of the Native class is used.

```

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 encoder with the encoding set to that of the data in the
Pervasive.SQL database
Encoding sje = Encoding.GetEncoding("us-ascii");

// 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);
```

Sample for Inputting the Changed parts of the DataSet into the Database

The DataSet class can acquire the changed parts of the dataset by calling the GetChanges method.

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

if(uds.HasErrors) // DataSet error check
{
    // error display code(omitted)
}
try
{
    BtLib.Ddf d = new BtLib.Ddf("c:¥¥pvs¥¥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++) // loop over the changed
records
    {
        // only key is transferred
        r[dt.Columns[0].ColumnName.ToString()]
        =dt.Rows[i][0].ToString();
        // execution of get equal
        rc = r.Read(BtLib.Operation.GetEqual);
        // transfer of data
        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());
}

```

Native Class C# Sample for Specifying the Data Area in the Structure

This is a C# sample for reading data in the Native Class by specifying the structure as a data area.

```

byte [] pb = new Byte[128];
byte [] data = new byte[1];
Int16 dataLength = 0;
byte [] keyBuf =
Encoding.ASCII.GetBytes("c:¥¥pvs¥¥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# Structure Declaring Sample

An example of the declaration of the structure used in the sample described above. This is the structure declared for the Department sample data in the DEMODATA database of Pervasive.SQL. Automatic generation is possible with the structure builder add-in utility.

```

[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
}

```

Native Class VB.NET Sample for Specifying the Data Area in the Structure

A VB.NET sample specifying the structure as a data area and showing the read-in of data using the Native Class.

```

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 Structure Declaring Sample

An example of the declaration of the structure used in the sample described above. This is the structure declared for the Department sample data in the DEMODATA database of Pervasive.SQL. Automatic generation is possible with the structure builder add-in utility.

```
<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  ' unsigned 8
End Class
```

Appendix-B: FAQs (Frequently Asked Questions)

This chapter explains problems and questions common to application programming and system setup.

Status 94 at the time of Web execution

The Btrieve status 94 exception may be generated at the time that the first Btrieve access occurs by the Open method, etc., of the Record class when an ASP.NET Web application is executed. The reason for this is that Pervasive.SQL and Btrieve recognize the user executing ASP.NET as an Administrator. This problem can be avoided by changing the user executing ASP.NET. Specifically speaking, this means that a new ASP.NET executing user is registered and this user is specified in the processModel tag of the machine.config file. The procedure is as follows.

1. Declare the new user.
2. Assign the user to the following groups.
Administrators
<Machine name> admins
<Machine name> authors
<Machine name> browsers
VS Developers
3. Edit the machine.config file found in
<windir>\Microsoft.NET\Framework\v1.0.3705\config. Edit and save the userName and password items in the processModel tag using the user ID declared above.
4. Restart the computer.

The following is an example of setting of the machine.config file.

```
<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"/>
```

The user setting information is referenced as follows.

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

Sample does not operate on a domain controller

The reason for this is that the service is operated in an account unique to the domain controller, and there are cases when the Web samples of this product do not operate on a domain controller. This problem is not a problem unique to TechKnowledge products and it can be solved by re-configuring IIS. Refer to the following link on how to reconfigure IIS in this case.

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

How to delete Structure Builder from VS.NET Add-in Manager

Although changing registry entries is required, if the following registry entries are deleted, it is possible to remove the Structure Builder utility from the Add-In Manager. After closing down Visual Studio, delete the following registry keys if they exist:-

When using Visual Studio 2005

HKEY_LOCAL_MACHINE\Software\Microsoft\Visual Studio\8.0\AddIns\SbAddIn.Connect

HKEY_CURRENT_USER\Software\Microsoft\Visual Studio\8.0\AddIns\SbAddIn.Connect

When using Visual Studio 2008

HKEY_LOCAL_MACHINE\Software\Microsoft\Visual Studio\9.0\AddIns\SbAddIn.Connect

HKEY_CURRENT_USER\Software\Microsoft\Visual Studio\9.0\AddIns\SbAddIn.Connect

Structure Builder does not display on Visual Studio 2008 Startup

Sometimes if the Structure Builder add-in is set as a Dockable Window or a Floating Window, then Structure Builder is not seen on Visual Studio 2008 startup, despite it being set to appear on startup in Add-in Manager.

This problem can be resolved by performing the following:-

- 1) Open up Add-in Manager and de-select all the checkboxes for the Btrieve Classes for .NET structure builder. Hit OK to close the Add-In Manager.
- 2) Open up Add-in Manager once again and re-select the Btrieve Classes for .NET structure builder addin and check its Startup column too. Hit OK to close the Add-in Manager.
- 3) The structure builder add-in will then appear.

To avoid this happening again, please set Structure Builder as a tabbed document window.

On re-installing Structure Builder does not display on the Tool Menu

After Structure Builder is deleted from Visual Studio's tool menu during an uninstall of Btrieve Classes for .NET, sometimes even after a re-install of Btrieve Classes for .NET structure builder does not re-appear. If this happens, please first uninstall Btrieve Classes for .NET and with the product uninstalled delete the following registry keys.

When using Visual Studio 2005

```
CURRENT_USER\Software\Microsoft\VisualStudio\8.0\PreloadAddinState\SbAddin.Connect
```

When using Visual Studio 2008

```
CURRENT_USER\Software\Microsoft\VisualStudio\9.0\PreloadAddinState\SbAddin.Connect
```

Because these registry keys are created not by the Btrieve Classes for .NET installer, but by the Visual Studio Add-In Manager, they are not removed during the Btrieve Classes for .NET uninstall.

What is a DDF file?

These are files that hold information on Pervasive/Btrieve databases. Specifically speaking, there are at least three files FILE.DDF, FIELD.DDF, and INDEX.DDF which must reside in the same directory. These files are themselves Btrieve data files. DDF files created by an old version of the Btrieve engine can be read by a higher version of Pervasive.SQL. Conversely, please note that status 30 may be returned when DDF file formats created in newer versions of Pervasive.SQL are read by an older Btrieve engine. In this case, the DDF file information will fail to be read-in.

I want to place the DDF files and the data files in separate folders

In general, this is possible by including the path name in the data file name. However, the DDF specifications only feature an area of 64 bytes for storing data file names, and caution must therefore be exercised when using the long file names or paths commonly used nowadays. .

Running Btrieve 6.15 under Windows 2000 or Windows XP

Windows 2000 and Windows XP are not included in the operating environments supported by Btrieve 6.15. To use Btrieve in an operating environment guaranteed by the manufacturer, Pervasive.SQL 2000i, Pervasive.SQL V8 or Pervasive PSQL v9 should be used.

Is operation possible with Btrieve 6.10?

The Btrieve 6.10 bundled with NetWare does not support 32-bit programming interfaces, and it can therefore not be used from a 32-bit module such as this product. 32-bit programming interfaces are supported from Btrieve 6.15 onwards. However, sales of Btrieve 6.15 are currently stopped, so please obtain a newer version like Pervasive.SQL V8, Pervasive PSQL v9 or Pervasive PSQL v10.

The character string field has been declared, but the leading 1 byte seems to be displaced

In versions of Pervasive.SQL from 2000i onwards, when a column is specified as true nullable a 1-byte area for the NULL indicator byte is reserved at the head of the record. The old DDF formats, like Btrieve 6.15, etc., do not

possess this specification. Even when issuing an operation referencing a key like GET_EQUAL, etc., it is necessary to allocate and set correctly a 1-byte area for the NULL indicator byte at the head of each such defined column in the key buffer.

Moving from a Btrieve 5.x system

A Btrieve database engine that supports a 32-bit programming environment is necessary in order to create applications to run under 32-bit operating systems. Pervasive PSQL v9 and Pervasive PSQL v10 are the currently available versions. These versions of Pervasive PSQL support the reading of Btrieve data file format 5.x, but no longer support writing to this data format. Accordingly, any Btrieve 5.x format data must be rebuilt in a later format before use with Pervasive PSQL v9 and Pervasive PSQL v10.

Appendix-C: About Data Types

The data types found in Pervasive.SQL and the mapping of these to Btrieve data types, and the mapping to .NET Framework data types in the Record and Extended classes of Btrieve Classes for Btrieve are shown in the following table.

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

(*) The operation of Btrieve Classes may cause conversions to String, so please convert to the target data type using the appropriate Convert class.

Appendix-D: Exception Class Error Code Table

This section describes the error codes of the Exception Class. Please note that error codes may be added or modified without prior notice for product improvements.

OutOfMemory	100	<p>“Temporary memory could not be reserved”</p> <p>This condition may be avoided by increasing the memory in the system, increasing swap, or stopping applications or services running concurrently.</p>
DdfOpenError	101	<p>“Couldn’t open DDF file”</p> <p>Check the path to the DDF file specified in the DDF class. In the case of C#, it is possible that a backwards slash mark has not been entered twice. If you think the path to the DDF file is correct, there may be problems with the Btrieve/Pervasive.SQL environment settings. Because the status from the Btrieve engine is stored in the BtrieveStatus property of the thrown exception, please refer to this value to solve problems related to Btrieve settings.</p>
DdfReadError	102	<p>“DDF read error”</p> <p>The DDF file has become mismatched for some reason. Check again whether the DDF file can be correctly read by the Pervasive Control Center. Like in the case of the DdfOpenError, if a value other than 0 is returned from the Btrieve/Pervasive.SQL engine, detailed information on the error can be obtained by referencing the BtrieveStatus property of the thrown exception instance.</p>
DdfCloseError	103	<p>“DDF close error”</p> <p>A status other than 0 is returned from Btrieve at the time closing of the DDF file is attempted.</p>
DdfInvalid	104	<p>“DDF is invalid”</p> <p>The DdfDir property had not been specified at the time that the Load method was executed.</p>
DdfAlreadyLoaded	105	<p>“DDF is already loaded”</p> <p>The Load method was called a second time even though the DDF is in the loaded state.</p>

DdfNotLoaded	106	<p>“DDF is not loaded”</p> <p>In the state in which the DDF is not loaded, a method is called that requires DDF information, like the GetRecord method, or property referencing is executed. Using the Load method, load the DDF.</p>
OwnerNameTooLong	107	<p>“Owner name length is too long”</p> <p>The owner name is set with a character string having a size that is longer than permitted by the Btrieve/ Pervasive DDF specifications.</p>
InvalidKey	108	<p>“Key is invalid”</p> <p>A key name that is not found in the table declaration of the database is specified.</p>
InvalidTableName	109	<p>“Table name is invalid”</p> <p>A table name that is not found in the table declaration of the database is specified.</p>
InvalidFieldName	110	<p>The specified column is not in the specified table.</p>
InvalidRecordSize	111	<p>“Record length is invalid”</p> <p>The DDF declaration and the record length of the actual Btrieve data do not match. Confirm the record declaration once more.</p>
InvalidOperation	112	<p>“Operation code is invalid”</p> <p>The specified operation code cannot be used with the specified method.</p>
ExpandFileName	113	<p>“File name cannot be converted”</p> <p>Error is returned from Win32 API when the Btrieve data file name is converted to a short file name.</p>
OpenDataFile	114	<p>“Data file could not be opened”</p> <p>Btrieve data file could not be opened. Btrieve Open operation failed. Check the Btrieve status code and take remedial action to interpret the status. In general, it is often the case that the data file is already placed in the open state in exclusive mode by another process.</p>
StringConversion	115	<p>“String conversion failed”</p> <p>An error is generated when an attempt is made to change a character string in the Managed code of the called language to unmanaged.</p>

DataTypeNotSupported	116	<p>“Data type is not supported”</p> <p>The data type declared in DDF is not supported by this product.</p>
InvalidSearchCond	117	<p>“Invalid search condition”</p> <p>The character string specified in the SearchCond property at the time of execution of an extended operation is incorrect. Confirm the string again.</p>
DuplicateFieldName	118	<p>“Field name is duplicated”</p> <p>A field having the same name has already been added by the AddField method, etc.</p>
VariableLengthNotSupported	119	<p>“Variable length fields are not supported in Extended operations”</p>
InvalidOperator,	120	<p>“Operator is invalid”</p> <p>The operator was not specified correctly at the time the search conditions for the Extended operation were specified. Specify the correct operator.</p>
NoFieldSpecified	121	<p>“No field is specified”</p> <p>The field to be acquired when the Extended operation is executed has not been specified.</p>
InvalidMaxRecords	122	<p>“Maximum number of records is invalid”</p> <p>MaxRecord is not specified correctly when the Extended operation is executed.</p>
DataConversion	123	<p>Data type conversion could not be performed. The data type related to the conversion is not supported.</p>
InvalidParameter	124	<p>The parameter value specified in the method is invalid. In the case of a parameter with a range specification, confirm this once again with the manual.</p>
InvalidDataTable	125	<p>The DataSet specified with the Fill method is returned as an Extended object. An object other than DataSet may have been specified.</p>
BtrieveError	126	<p>From Btrieve a fatal error preventing continued operation has been returned. In the exception class the BtrieveStatus property, containing the Btrieve status code returned, is available, so please refer to Pervasive documentation on this status code, to resolve this problem.</p>

LogInFail	127	Login to the secure Btrieve database returned a status code indicating failure. In the exception class the BtrieveStatus property, containing the Btrieve status code returned, is available, so please refer to Pervasive documentation on this status code, to resolve this problem.
LogOutFail	128	Logout from the secure Btrieve database returned a status code indicating failure. In the exception class the BtrieveStatus property, containing the Btrieve status code returned, is available, so please refer to Pervasive documentation on this status code, to resolve this problem.
InvalidLockBias	129	On starting the transaction, the specified lock bias was invalid. Please set the lock bias to a valid value.

Appendix-E: Btrieve Status Codes

The following is a summary and description of the Btrieve status codes. Details can be found in the Pervasive PSQL manuals.

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	Insufficient 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

Btrieve Classes for .NET Version 4.00
Last Updated
May 19, 2010

Copyright TechKnowledge Inc.