ACTIAN™

Zen Direct Access Components Guide

Zen v16

Activate Your Data™

Contents

Using Direct Access Components

The following topics introduce Zen Direct Access Components (PDAC) for Delphi and C++Builder:

- Overview of Zen Direct Access Components
- Using the Zen Direct Access Components
- Deploying an Application Based on PDAC

For details on implementing this feature, see Direct Access Components Reference.

Overview of Zen Direct Access Components

Zen Direct Access Components are a set of Visual Component Library (VCL) components that allow direct transactional and relational access to Zen database engines from within the Borland Delphi and C++ Builder Environments. This SDK was formerly known as Pervasive Direct Access Components (PDAC), and the short name PDAC is still found in use in the SDK. Before the Zen v15 release, these runtime components were integrated and installed as .bpl files in the 32- and 64-bit \bin directories of Zen databases. Beginning with Zen v15, Zen installations removed them, but they continue to be available in the downloaded SDK, which also includes design-time components for use in RAD Studio.

See the PDAC release notes for supported development environments.

All versions of PDAC contain the following nonvisual components:

- TPvSession
- TPvDatabase
- TPvTable
- TPvBatchMove
- TPvQuery
- TPvStoredProc
- TPvUpdateSQL
- TPvSqlDatabase

Supporting classes are also provided for these 32-bit components, which duplicate the properties, methods, and binding capabilities of the Embarcadero Data Access components without requiring the presence of the Borland Database Engine (BDE) at run time.

These components are provided in package format, and offer all the design time and runtime functionality of the built-in controls. They bind to the Embarcadero Data Aware controls in the same way as the BDE components, as well as to fully compatible third-party bound controls.

Special components include wwPvTable, wwPvQuery, and wwPvStoredProc and are provided for interoperability with the Woll2Woll InfoPower components.

Engine Version Checking

When opening a query or table, PDAC confirms the database engine version to avoid engine dependency errors. An insufficient engine version returns an exception indicating the required version.

TPvQuery.PassThrough

Allows SQL query text to be sent directly to the engine instead of being preparsed.

TPvDatabase.OEMConversion

This property indicates that the database contains characters encoded according to the OEM (DOS) code page, and that these characters should be converted to the ANSI (Windows) code page before use.

Where to Get More Information

Please see the SDK download for PDAC. You can find it at the Actian Zen website.

Using the Zen Direct Access Components

You install PDAC by extracting the components from a download archive to a directory (*installdir*). Be sure that any Delphi or C++Builder IDEs on your system can locate the extracted components.

Follow these steps to set C++Builder project options to reflect the INCLUDE and LIBRARY paths for PDAC. These can be added to the default options, so the steps are unnecessary for every project using the Zen components.

To add PDAC to the INCLUDE and LIBRARY paths in C++Builder

1. In C++Builder, select **Project** > **Options**, and choose the **Directories/Conditionals** tab.
2. Add *installdir*`\sdk\pdac\Delphi?\include` to the Include Path, where *installdir* represents the directory where you installed the SDK. Replace the question mark with the version of RAD Studio you are using.
3. Also add *installdir*`\sdk\pdac\Delphi?\lib` and *installdir*`\sdk\pdac\Delphi?\lib\dcu` to the Library Path, separated by a semicolon. Replace the question mark with the version of RAD Studio you are using.

After the Include and Library paths are set properly, you can use the Direct Access components in exactly the same way as the corresponding Embarcadero components.

Delphi and C++Builder

Zen provides support for all Delphi versions starting with Delphi 6. Use the following steps to incorporate Delphi or C++Builder into your IDE.

To add the Delphi or C++Builder packages and path information to your environment

1. In Delphi or C++Builder, select **Components > Install Packages > Add**.
2. Select the design packages.
3. Add the paths to the Delphi compiled units (DCUs) and interface files (INTs).

Once the components for your environment have been selected, a tab for Zen is visible on VCL projects. Applications using PDAC can now be built by adding PvTable or PvQuery.

Deploying an Application Based on PDAC

With a few variations for Embarcadero and other runtime libraries, you can build PDAC applications for deployment to end users in two ways: using runtime packages or statically linking the PDAC libraries in the application executable. C++Builder also offers dynamic or static linking of the Embarcadero library. Either a Zen Client or engine (Enterprise Server, Cloud Server, or Workgroup) must be properly installed on the target computer for a PDAC application to run successfully.

Deciding Whether to Use Runtime Packages

Using runtime packages has several advantages. It provides the smallest executable possible; this can be important where the executable or updates must be downloaded or transferred over slow links. This option allows sharing of the PDAC and Embarcadero runtime binaries and can save considerable disk space and memory in environments with many Embarcadero and PDAC applications.

A disadvantage is that more files must be distributed and kept current, and the shared files can be vulnerable to third party installations that install incorrect versions of them in shared subdirectories. If runtime packages are not used, only the application and drm.dll must be deployed. The PDAC libraries are statically linked into the executable, which will be correspondingly larger.

To use runtime packages, the developer must deploy both the application executable and the PDAC libraries (from the redist subdirectories in the PDAC installation), as well as any Embarcadero or other runtime libraries (borlndmm.dll, found in the compiler bin directory, is often needed) to end users.

Redist Subdirectory

Each Embarcadero compiler has a redist directory. This directory contains the .bpl libraries that must be distributed with an application compiled with **Build with runtime packages**.

Note: Each redist directory also includes .dcp or .bcp files that are not redistributable except within the licensee's organization to allow development with derived classes. Only the .dpl and .bpl packages may be distributed with applications.

Building and Deployment Instructions for Delphi or C++Builder

In the Delphi IDE, set the appropriate options as follows:

1. From the menu bar, select **Project** > **Options**
2. Click the **Packages** tab.

   To have the application dynamically link with the PDAC libraries at run time, select **Build with runtime packages**, and add the appropriate PDAC packages to the edit box list.

Note: If the PDAC packages were added to the Build list at installation, this step is not necessary.

3. Select the 32- or 64-bit runtime packages (.bpl files) required for the MicroKernel Engine depending on your version of Delphi.

Note: If the PDAC packages are not added to the list, they are compiled into the executable and the shared libraries are not used, even if the box is selected.

4. To statically link the PDAC libraries into your executable, clear the **Build with runtime packages** check box. If you select this setting at compile time, then you must install the libraries when you deploy the application and any other packages or DLLs required.

These files may be installed in the same subdirectory as the application executable, but we recommend placing them in the target machine path to be shared by other applications. The Zen installation \bin directory is a good choice on computers where it is the primary Zen executable directory.

In addition, the Delphi PDAC application will require an installation of either a Zen engine (server or workgroup) or a Zen Client and a remote server appropriately installed and configured.

Updated Package Names Starting with Delphi 6

Starting with Delphi 6, we have changed the package naming conventions to be more compatible with Embarcadero standards. Also, the runtime packages have been separated from the design time packages in order to comply with Embarcadero requirements.

The following table shows how to interpret package names if you are coming from a previous version of Delphi and PDAC:

File name digit position	Possible values	Meaning
1	p	PSQL (former name for Zen product)
2	c, b, or s	Common, Btrieve, or SQL
3–5	depends on release	Major and minor version of PDAC components
6	r, d	Run time or design time
7–9	depends on release	Delphi version to which these components apply

Note: In Delphi 6, InfoPower merges into the main packages, with Woll2Woll no longer separate.

Direct Access Components Reference

The following topics provide detailed information for Delphi and C++Builder developers:

- MicroKernel Engine Components
- Relational Engine Components
- Differences Between PDAC and Embarcadero Components
- PDAC Classes, Properties, Events, and Methods
- Zen and Embarcadero Data Types

MicroKernel Engine Components

The set of Btrieve components includes the following:

- TPvSession
- TPvDatabase
- TPvTable
- TPvBatchMove
- TwwPvTable

These components use no relational functionality and do not require the Relational Engine or ODBC at run time.

The components replace Embarcadero ones as listed in the following table.

Zen Direct Access Component	Replaces This Embarcadero Component
TPvBatchMove	TBatchMove
TPvDatabase	TDatabase
TPvSession	TSession
TPvTable, TwwPvTable	TTable

The components can be used standalone (that is, without the Borland Database Engine (BDE) installed) or coincident with the BDE within a single application. Use of the components is the same as the corresponding BDE components, with exceptions noted under specific components where applicable and in PDAC Classes, Properties, Events, and Methods.

TPvSession

Provides thread safety and Client ID support. Its functionality is similar to the TSession VCL component.

Related Information

See the following in this chapter pertaining to TPvSession:

- Class TPvSessionList/TPvSqlSessionList (under Supporting Classes)
- Global Variables (under General Differences from VCL)
- TPvSession and TPvSqlSession (under Specific Class Differences from VCL)
- TPvSession Specific (under Specific Class Differences from VCL)

TPvDatabase

Provides database connection-specific functionality for non-SQL databases, such as login control, transaction support, persistent database connections. Its functionality is similar to the TDatabase VCL component.

Functional Differences Between TPvDatabase and Embarcadero Components

- Property DriverName, Locale, and TTraceFlags have been dropped
- Handle type is changed to DRM_DBID
- *IsSQLBased* always returns False
- TransIsolation has only two possible values: tiDirtyRead and tiReadCommitted (the default).

Secure Tables and Prompting for User Name and Password

If you are connecting to a secure table, PDAC prompts you for the user name and password. If you want suppress these prompts, set up connection parameters in TPvDatabase using the following as an example. (See also Table Security under TPvTable.)

```
PvSession.ServerName:='ServerName';
PvSession.SessionName:='session1';
PvSession.Active:=True;

PvDatabase.AliasName:= 'DatabaseName';
PvDatabase.DatabaseName:='DB';
PvDatabase.SessionName:='session1';
PvDatabase.Params.Clear();
```

```
// here you specify user name and password to
// connect to remote database.

PvDatabase.Params.Add('User Name=UserName');
PvDatabase.Params.Add('password=Password');
PvDatabase.Connected:=True;
PvTable.DatabaseName:='DB';
PvTable.SessionName:='session1';
PvTable.TableName:='person';
PvTable.Active:=True;
```

Related Information

See the following in this chapter pertaining to TPvDatabase:

- TransIsolation Property Differences (under Differences Between PDAC and Embarcadero Components
- TPvDatabase and TPvSqlDatabase (under Specific Class Differences from VCL)
- TPvDatabase Specific (under Specific Class Differences from VCL)

TPvTable

Provides single-table access and manipulation. Its functionality is similar to the TTable VCL component.

Functional Differences Between TPvTable and Embarcadero Components

- All alias management functions perform "named database" management.
- Inherits from TPvDataSet rather than TDBDataSet.
- Properties TableLevel, UpdateObject, UnlockTable, OpenIndexFile, CloseIndexFile have been dropped. TableType is ignored.
- The Database Name property can be provided at both design time and runtime as a DSN, a Zen Named Database, or as a fully qualified path to the database.

Table Security

PDAC security for the MicroKernel Engine uses owner names for individual tables. Using this security model, you provide the owner name for the table. See the operation Set Owner (29) in the *Btrieve API Guide* for more information.

The TPvTable has the property `Owner: string` and the following method:

```
SetOwnerOnTable(AOwner: string; AccessMode: integer).
```

With the *Owner* property you can set the owner name, and with the *SetOwnerOnTable* method you can set or clear the owner name on the table.

Code	Description
0	Requires an owner name for any access mode (no data encryption)
1	Permits read-only access without an owner name (no data encryption)
2	Requires an owner name for any access mode (with data encryption)
3	Permits read-only access without an owner name (with data encryption)

As controlled by these PDAC access modes, to access the Btrieve table, provide a valid owner name.

Table Creation

Call the CreateTable method at run time to create a table using current definition in this dataset.

If the FieldDefs property contains values, these values are used to create field definitions. Otherwise the Fields property is used. One or both of these properties must contain values in order to recreate a dataset.

Use the Add method to assign field properties

```
procedure Add(const Name: string; DataType: TFieldType; Size: Word; Required: Boolean);
```

Add is provided for backward compatibility. The recommended way to add new field definitions to the Items property array is using the AddFieldDef method. You should also use it to specify precision and scale for the ftBCD data type.

Add uses the values passed in the Name, DataType, Size, and Required parameters and assigns them to the respective properties of the new field definition object.

Note: Set **Required** to False if the field is nullable.

Note: To activate the autoincrement property for an AutoInc field, create a unique index on this field. Also, in the current release, PDAC does not support the BIGIDENTITY data type.

Delphi Example

```
PvTable1.DatabaseName := 'TestData';
PvTable1.TableName := 'TestData1';
with PvTable1.FieldDefs do
begin
```

```
  Clear;
  Add('F_autoinc',  ftAutoInc, 0, True);
  Add('F_currency', ftCurrency, 0, False);
  Add('F_integer',  ftInteger, 0, False);
  Add('F_word',     ftWord, 0, False);
  Add('F_fixchar',  ftFixedchar, 30, False);
  Add('F_varbin',   tString, 25, False);
  Add('F_blob',     ftBlob, 60, False);
end;
with PvTable1.FieldDefs.AddFieldDef do
begin
  Name := 'F_BCD';
  DataType := ftBCD;
  Size:=2;     //Scale
  Precision := 10;  //Precision
  Required := false;
end;


with PvTable1.IndexDefs do
begin
  Clear;
  Add('Index1', 'F_autoinc', [ixPrimary, ixUnique]);
  Add('Index2', 'F_integer', [ixCaseInsensitive]);
end;
PvTable1.CreateTable;
```

C++Builder Example

```
PvTable1->DatabaseName="TestData";
PvTable1->TableName="Test1";
PvTable1->FieldDefs->Clear();
PvTable1->FieldDefs->Add("F_autoinc", ftAutoInc, 0, True);
PvTable1->FieldDefs->Add("F_integer", ftInteger, 0, False);
PvTable1->FieldDefs->Add("F_Curr", ftCurrency, 0, False);
PvTable1->FieldDefs->Add("F_Word", ftWord, 0, False);
PvTable1->FieldDefs->Add("F_fixchar", ftFixedChar, 0, False);
PvTable1->FieldDefs->Add("F_String", ftString, 20, False);
PvTable1->FieldDefs->Add("F_blob", ftBlob, 60, False);

TFieldDef *FieldDef = PvTable1->FieldDefs->AddFieldDef();
FieldDef->Name="F_BCD";
FieldDef->DataType=ftBCD;
FieldDef->Size=2;
FieldDef->Precision=10;
FieldDef->Required=False;
PvTable1->IndexDefs->Clear();
PvTable1->IndexDefs-> Add("Index1","F_autoinc",TIndexOptions() <<ixPrimary << ixUnique);
PvTable1->CreateTable();
```

Related Information

See the following topics pertaining to TPvTable:

- TransIsolation Property Differences (under Differences Between PDAC and Embarcadero Components
- TPvTable, TPvQuery, and TPvStoredProc (under Specific Class Differences from VCL)
- TPvTable Specific (under Specific Class Differences from VCL)

- Zen and Embarcadero Data Types

TPvBatchMove

Enables applications to perform database operations on groups of records or entire tables. Its functionality is similar to the TBatchMove VCL component.

Related Information

See the following in this chapter pertaining to TPvTable:

- TPvBatchMove (under Specific Class Differences from VCL)

TwwPvTable

Included with PDAC for InfoPower compatibility. It is directly derived from TPvTable and has an extra property, "Control Type."

Relational Engine Components

The set of Relational Engine components includes the following:

- TPvSQLSession
- TPvSQLDatabase
- TPvQuery
- TPvUpdateSQL
- TPvStoredProc
- TwwPvQuery
- TwwPvStoredProc

The components replace Embarcadero ones as listed in the following table.

Zen Direct Access Component	Replaces This Embarcadero Component
TPvQuery, TwwPvQuery	TQuery
TPvSqlDatabase	TDatabase
TPvSqlSession	TSession

Zen Direct Access Component	Replaces This Embarcadero Component
TPvStoredProc, TwwPvStoredProc	TStoredProc
TPvUpdateSQL	TUpdateSQL

The components can be used standalone (that is, without the Borland Database Engine (BDE) installed) or coincident with the BDE within a single application. Use of the components is the same as the corresponding BDE components, with exceptions noted under specific components where applicable and in PDAC Classes, Properties, Events, and Methods.

TPvSQLSession

Provides thread safety and Client ID support for Zen. Functionality is similar to the TSession VCL component.

Functional Differences Between TPvSQLSession and Embarcadero Components

- All alias management functions perform DSN management.
- The type of Property 'Handle' has been changed to SQLHENV.
- Method 'AddStandardAlias' adds an Engine DSN with default settings.
- Property 'UpdateDsnType: TDsnTypes' has been added. 'TDsnTypes = (dsnSystem, dsnUser)'. 'UpdateDsnType' means:
  - In case of 'DeleteAlias' function - which DSN user is going to delete: System or User DSN?
  - In case of 'AddAlias' and 'GetAliasParameters' functions - hints which DSN type is preferred.

Related Information

See the following in this chapter pertaining to TPvSQLSession:

- Class TPvSessionList/TPvSqlSessionList (under Supporting Classes)
- Global Variables (under General Differences from VCL)
- TPvSession and TPvSqlSession (under Specific Class Differences from VCL)
- TPvSqlSession Specific (under Specific Class Differences from VCL)

TPvSQLDatabase

Provides for Zen database connection-specific functionality, such as login control, transaction support and persistent database connections. Functionality is similar to the TDatabase VCL component.

Functional Differences Between TPvDatabase and Embarcadero Components

- The type of Property 'Handle' has been changed to SQLHDBC.
- Property 'IsSQLBased' is always true.
- Properties 'Tables' and 'StoredProcs' have type 'TSqlTablesInfoCollection' and 'TSqlStoredProcInfoCollection' correspondingly.

Secure Databases and Prompting for User Name and Password

If you are connecting to a secure database, PDAC prompts you for the user name and password. If you want suppress these prompts, set up connection parameters in TPvSQLDatabase using the following as an example.

```
PvSession.ServerName:='ServerName';
PvSession.SessionName:='session1';
PvSession.Active:=True;

PvDatabase.AliasName:= 'DatabaseName';
PvDatabase.DatabaseName:='DB';
PvDatabase.SessionName:='session1';
PvDatabase.Params.Clear();

// here you specify user name and password to
// connect to remote database.

PvDatabase.Params.Add('User Name=UserName');
PvDatabase.Params.Add('password=Password');
PvDatabase.Connected:=True;
PvTable.DatabaseName:='DB';
PvTable.SessionName:='session1';
PvTable.TableName:='person';
PvTable.Active:=True;
```

DSN-Less Connections from a Client

PDAC's relational components are able to connect from a client machine without a DSN or Named Database to a remote server database. The server must have a DSN for the database.

This feature works through the use of the property, **AliasNameIsConnectionString**, which is available with the TPvSqlDatabase component. For more information, see Specific Class Differences from VCL.

1. Drop a PvSQLDatabase on a Delphi form.
2. Supply a fictitious DatabaseName—it can be anything.
3. Set the property A**liasNameIsConnectionString** to True.
4. Set the **AliasName** property (or the **DatabaseName**, leaving **AliasName** blank) to the Connection String.

   The Connection String is the complete connection string for ODBC, including the DSN on the server and the name of the server, as shown in the following example:

   ```
   DRIVER={Pervasive ODBC Client Interface};ServerName=DSLINUX2; ServerDSN=DEMODATA;
   UID=myuserid;PWD=mypassword
   ```

Note: There are no quotation marks or line breaks in the connection string. If your editor wraps the preceding line, make it a single line in the property editor.

   If a user name and password are required, they may be supplied as part of the Connection String; if they are not in the Connection String, the standard database login dialog displays if the LoginPrompt property is True.

5. Set the database to Connected and use as usual.

All these steps may be performed at design time or through code at run time.

Related Information

See the following in this chapter pertaining to TPvSQLDatabase:

- TPvDatabase and TPvSqlDatabase (under Specific Class Differences from VCL)
- TPvSqlDatabase Specific (under Specific Class Differences from VCL)

TPvQuery

Encapsulates a rowset based on a SQL statement, which provides full relational access including joins and cached updates to multiple tables. Functionality is similar to the TQuery VCL component.

Functional Differences Between TPvQuery and Embarcadero Components

- Inherited from TQuery
- The type of Property 'Handle' has been changed to SQLHSTMT.

- The type of Property 'DBHandle' has been changed to SQLHDBC.
- Property 'StmtHandle' has been dropped.
- Property 'Text' always returns text from property 'SQL'.
- Method 'CheckOpen' has been dropped.
- All index and key related properties and methods (like 'GetIndexInfo') have been dropped.
- If 'LoadBlobOnOpen' property is true, then all BLOBs will be cached in memory on query open. If 'LoadBlobOnOpen' is false, then BLOBs will be read as required.

BookMarksEnabled Property

TPvQuery contains a Boolean property BookmarksEnabled. If you do not use Bookmarks in your application, set this property to False to increase TPvQuery performance. The default value is True.

```
PvQuery.BookmarksEnabled :=False
```

Cursor Management

TPvQuery has a property, CursorType, that can be set to *ctCached* or *ctDynamic*. The following table describes the behavior of this property in more detail.

Cursor type	Behavior
ctCached	This setting corresponds to the previously available fully cached, fully static Cursor Manager. It reads every record in the ResultSet object before returning, which can be very slow for large tables, yet fast after it has opened for operations such as look-ups.
ctDynamic	This setting is the default. It uses the database engine's dynamic cursors and offers improved performance for most operations when compared with ctCached, especially with regard to large tables.

Note: Dynamic cursors can see the Inserts/Updates/Deletes of their own or other clients.

You can change CursorType on a PvQuery by changing the Property, but only at run time if Active is False. If you change it in Design mode, and the query is Active, it will deactivate the query and change the cursor type but not reactivate. At run time, changing CursorType on an Active PvQuery results in the exception "Cannot perform this operation on an open dataset."

Case Where Dynamic Cursors Change to Static

If you request a dynamic cursor (ctDynamic), but your SQL statement contains a construct that a dynamic cursor cannot process, the engine will connect and return data, but with a static cursor. For example, such a construct could be:

- A view
- A join
- An ORDER BY on a column with no index defined

In the event the cursor is transformed into a static one, this static cursor will be engine-based one that cannot see inserts, updates, or deletes made by other clients. It will perform better than the static cursors on previous releases (that is, ones using *ctCached* as the cursor type.).

When the dynamic to static transformation occurs, the database engine notifies the PDAC component, and sets a read-only public property named **EngineCursor**.

This EngineCursor property is not published, so it is not visible in the Object Inspector during design. It can have two values:

- *ecDynamic*
- *ecStatic*

You cannot modify this property, but you can check its value after opening a query. You might want to check this property for example if it is important that your application know whether your cursor includes updates made by other clients.

Related Information

See the following items pertaining to TPvQuery:

- TPvTable, TPvQuery, and TPvStoredProc (under Specific Class Differences from VCL)
- TPvQuery and TPvStoredProc Specific (under Specific Class Differences from VCL)
- TPvQuery Specific (under Specific Class Differences from VCL)

TPvUpdateSQL

Allows transparent updating (including cached updates) of SQL row sets not otherwise updatable (multiple-table joins, and so forth). Functionality is similar to TUpdateSQL VCL component.

Related Information

See the following item pertaining to TPvTable:

- TPvUpdateSQL (under Specific Class Differences from VCL)

TPvStoredProc

Provides access to Zen stored procedures; it allows parameterized use and can return rowsets the same as the TPvQuery component. Functionality is similar to TStoredProc VCL component.

Related Information

See the following items pertaining to TPvQuery:

- TPvTable, TPvQuery, and TPvStoredProc (under Specific Class Differences from VCL)
- TPvQuery and TPvStoredProc Specific (under Specific Class Differences from VCL)
- TPvStoredProc Specific (under Specific Class Differences from VCL)

TwwPvQuery

Included with PDAC for InfoPower compatibility. It is directly derived from TPvQuery and has an extra property, "Control Type."

TwwPvStoredProc

Included with PDAC for InfoPower compatibility. It is directly derived from TPvStoredProc and has an extra property, "Control Type."

Database Security for ODBC and SQL

ODBC and SQL security is database security on the DDF level. There are no special methods in PDAC to set up SQL security. You should use external programs such as Zen Control Center or any ODBC tool.

If the database has security, a dialog displays in which you enter the user name and password each time you open a table or connect to a TPvDatabase component. To verify if security is set up on the database, you can check the public property *TPvDatabase.IsSecured*. See also Secure Tables and Prompting for User Name and Password.

Note: The MicroKernel Engine security uses owner names (see Table Security). If the database has SQL security enabled, owner name security is ignored.

Differences Between PDAC and Embarcadero Components

The following table summarizes the functional differences between Zen and Embarcadero components.

TPvTable	All alias management functions perform "named database" management. Inherits from TPvDataSet rather than TDBDataSet; Properties TableLevel, UpdateObject, UnlockTable, OpenIndexFile, CloseIndexFile have been dropped, TableType is ignored. The Database Name property can be provided at both design time and runtime as a DSN, a Zen Named Database, or as a fully qualified path to the database.
TPvDatabase	Property DriverName, Locale, and TTraceFlags have been dropped; Handle's type is changed to DRM_DBID *IsSQLBased* always returns False; TransIsolation has only two possible values, 'tiDirtyRead' and 'tiReadCommitted' (the default).
EPvDBEngineError (see Class EPvDBEngineError)	Replaces EDBEngineError
TPvDBError (see Class TPvDBError)	Replaces TDBError
TPvSQLSession	All alias management functions perform DSN management. The type of Property 'Handle' has been changed to SQLHENV. Method 'AddStandardAlias' adds an Engine DSN with default settings. Property 'UpdateDsnType: TDsnTypes' has been added. 'TDsnTypes = (dsnSystem, dsnUser)'. 'UpdateDsnType' means: • In case of 'DeleteAlias' function - which DSN user is going to delete: System or User DSN? • In case of 'AddAlias' and 'GetAliasParameters' functions - hints which DSN type is preferred.

TPvSQLDatabase	The type of Property 'Handle' has been changed to SQLHDBC. Property 'IsSQLBased' is always true. Properties 'Tables' and 'StoredProcs' have type 'TSqlTablesInfoCollection' and 'TSqlStoredProcInfoCollection' correspondingly.
TPvQuery	Inherited from TQuery The type of Property 'Handle' has been changed to SQLHSTMT. The type of Property 'DBHandle' has been changed to SQLHDBC. Property 'StmtHandle' has been dropped. Property 'Text' always returns text from property 'SQL'. Method 'CheckOpen' has been dropped. All index and key related properties and methods (like 'GetIndexInfo') have been dropped. If 'LoadBlobOnOpen' property is true, then all BLOBs will be cached in memory on query open. If 'LoadBlobOnOpen' is false, then BLOBs will be read as required.

TransIsolation Property Differences

The **TransIsolation** property behavior of PDAC is different from that of their Embarcadero equivalents.

Transaction isolation level determines how a transaction interacts with other simultaneous transactions when they work with the same tables, and how much a transaction sees of operations performed by other transactions.

Use the property **TPvDatabase.TransIsolation** to specify the transaction isolation level for the database.

PDAC supports only the *tiReadCommitted* mode. This means that other users cannot see the changes made to a file until the transaction ends. This is the only setting supported by the database engine.

PDAC Classes, Properties, Events, and Methods

This topic lists all classes, properties, events, and methods in PDAC. All are described in Pascal notation and show differences with their Embarcadero counterpart components.

- Exception Classes
- Supporting Classes

Exception Classes

Class EPvDatabaseError

This class is an ancestor for all exception classes in PDAC. It has the property `Owner` that contains a reference to the class that generated the exception.

Class EPvDBEngineError

This is an abstract base class for all DB engine error related classes. Property `Errors` lists the errors (in classes, derived from the TPvDBError class) and property `ErrorCount` indicates the total number of errors contained in the `Errors` property.

Class TPvDBError

TPvDBError is an abstract base class for all classes that represent database engine errors for the EPvDBEngineError exception class. It contains follow properties:

- Property `Message` specifies the text of the error message.
- Property `NativeError` indicates the status code returned from the engine (the Btrieve status code).

Class EPvDrmEngineError

Exceptions of this class are raised by the Btrieve subset of PDAC. Property `ErrorCode` is a DRM error code. The `Errors` array of EPvDrmEngineError contains objects with type TPvDrmError.

Class TPvDrmError

Class TPvDrmError describes a DRM error. In `ErrorCode` it contains a DRM error code, and in `NativeError` it contains a Btrieve status code.

Class EPvSqlEngineError

Exceptions of this class are raised by the SQL (relational) subset of PDAC. Property `ErrorCode` is the return value of the last ODBC call. The `Errors` array of EPvSqlEngineError contains objects with type TPvSqlError.

Class TPvSqlError

Class TPvSqlError describes ODBC errors.

Class EPvDbAdminEngineError

Exceptions of this class are raised during execution of DBNames and DSN management functions, such as TPvSqlSession.AddAlias, on local and remote servers. It has an additional property ErrorType: TPvDbAdminEngineErrorTypes. If ErrorType = dbmeDTI then NativeError contains a DTI (Distributed Tuning Interface) error code, otherwise NativeError contains local error codes of DBNames and DSN related function (see EPvSqlInstallerEngineError and EPvOwnSqlInstallerEngineError).

Class EPvSqlInstallerEngineError

Exceptions of this class are raised during execution of DNSs management functions, such as TPvSqlSession.AddAlias, on local servers, i.e. if TPvSqlSession.ServerName property is empty or has name of the local server. `Errors` contains the value, returned by SQLInstallerError() function.

Class EPvOwnSqlInstallerEngineError

Because DSN management functions in some cases lack meaningful errors, exceptions of this class are provided to raised in order to introduce new types of errors. In the `NativeError` the following values are possible:

- cPvOwnSqlInstallerEngineErrorDsnAlreadyExist – user tried to create a preexisting DSN.
- cPvOwnSqlInstallerEngineErrorDsnNotFound – user tried to delete non-existent DSN.
- cPvOwnSqlInstallerEngineErrorInvalidOpenMode – value of prmOPEN_MODE ('OPEN_MODE') parameter is invalid. See prmOPEN_MODE description below for details.
- cPvOwnSqlInstallerEngineErrorClientDSNsAreNotSupported – client DSNs are not supported in remote server mode (TPvSqlSession.ServerName is not empty).

Supporting Classes

Supporting classes are required by the higher-level components and encapsulate BDE-specific functionality. These classes are "cloned" in PDAC with as few changes as possible.

Class TPvSessionList/TPvSqlSessionList

Classes TPvSessionList and TPvSqlSessionList manage session components in applications that provide multiple sessions. In this class only the types of contained objects (TSession to TPvSession or TPvSqlSession) have been changed.

Class TPvBlobStream/TPvSQLBlobStream

Classes TPvBlobStream and TPvSQLBlobStream are stream objects that let applications read from or write to field objects that represent Binary large object (BLOB) fields. They function in the same way as the TBlobStream VCL class.

Classes TParam/TParams

Class TParam represents a field parameter. Properties of a TParam are used to set the value of a parameter that represents the value of a field. TParams is list of the TParam objects. TParam and TParams in PDAC have no changes in their interface sections; they are only moved to the new file.

Class TMasterDataLink

Note: For Delphi/C++Builder 3 and 4 only

TMasterDataLink allows a dataset to establish a master/detail relationship. It has no changes in its interface section, and is only moved to a new file.

General Differences from VCL

Interfaces exposed by PDAC components correspond almost exactly to the appropriate VCL components, which work via the BDE. More detailed info about related VCL components is contained in the Delphi/C++Builder help system (file del?vcl.hlp or bcbvcl?.hlp, where ? is 3 or 4 or 5). However, since some BDE features do not exist in Zen and some Zen features do not exist in the BDE, Zen has corrected these interfaces (dropped or added properties/methods/events). The listing below enumerates only those interfaces that have been changed. All other interfaces are cloned as-is.

Global Variables

Instead of the BDE Session and Sessions global variables, PDAC has its own global variables BtvSession: TPvSession and BtvSessions: TPvSessionList for the Btrieve subset and PvSqlSession: TPvSqlSession and PvSqlSessions: TPvSqlSessionList for the relational subset. They behave the same as the Session and Sessions variables and are created automatically on application startup and destroyed automatically on application shutdown.

Btrieve Subset

In the Transactional subset, PDAC uses Zen Named Databases as aliases.

SQL Subset

In the Relational subset, PDAC uses Data Source Names (DSNs) as aliases.

Specific Class Differences from VCL

The following sections contain specific differences from the Embarcadero VCL.

TPvSession and TPvSqlSession

- TraceFlags property was dropped, since tracing of Zen API calls is carried out via an external utility or the Distributed Tuning Interface.
- Zen does not use the following properties (for compatibility purposes they are present only as storage for strings):
  - NetFileDir
  - PrivateDir
- Locale property was dropped, as there is no analog in Zen for this BDE-specific feature.
- Zen has no driver. All interfaces items relating to drivers were dropped (methods AddDriver, DeleteDriver, GetAliasDriverName, GetDriverNames, GetDriverParams and ModifyDriver).
- Password-related interfaces only store and retrieve strings (AddPassword, RemovePassword and RemoveAllPasswords). Event OnPassword never fires, except in GetPassword method.
- GetConfigParams method does nothing.
- New published properties have been added: ServerName, ServerAdminUser, ServerAdminPassword and ServerAdminLoginPrompt. They provide the way to connect to remote servers. ServerAdminUser and ServerAdminPassword are DTI (Distributed Tuning Interface) user name and password.

Tip... For more information see DTI documentation.

- Property ServerAdminLoginPrompt is analog to TPv(Sql)Database.LoginPrompt, except that in design time if ServerAdminLoginPrompt=false and there are problems to get list of DBNames from server, you will be prompted for user name/password after unsuccessful attempt to login.

Tip... DTI is supported only by Pervasive.SQL 2000 and above.

- LocalSystem public property.
  For remote servers LocalSystem=false, for local – true.

TPvSession Specific

- All alias management functions perform Named Database management in PDAC. TPvSession allows creating local and remote DBNames and getting parameters for local and remote DBNames. To establish a connection with a database on the server, the user should create a DBName using the path to the database on the server. The developer can use the following values on entrance:
  - prmDDF_PATH ('DDF_PATH') – path to data dictionary files.

- prmPATH ('PATH') – path to data files.
- prmBOUND ('BOUND') – is database bound? Default is false.
- prmINTEGRITY ('INTEGRITY') – is database has integrity constraints? Default is true.
- prmCREATE_DDF ('CREATE_DDF') – if this parameter is true, then empty database will be created.

- AddStandardAlias method adds DB name with standard settings: INTEGRITY – True, BOUND – False.
- Handle property type was changed to DRM_SESID.
- Example: Create DBName.

```
var MyList: TStringList;
begin
    MyList := TStringList.Create;
try
  with MyList do
  begin
      Add('DDF_PATH=D:\MyDemoData');
      Add('PATH=D:\MyDemodata');
      Add('BOUND=False');
      Add('INTEGRITY=False');
      Add('Create_DDF=False');
  end;
  PvSession1.AddAlias('TestAlias', MyList);
finally
        MyList.Free;
      end;
```

TPvSqlSession Specific

- All alias management functions do DSN management. User can use follow values on entrance
  - prmDB_NAME (DB) – DB name for database. Engine DSN specific.
  - prmDSN_DESCRIPTION (DESCRIPTION) – description for DSN.
  - prmIS_ENGINE_DSN ('IS_ENGINE_DSN') – determines, is the given DSN Engine DSN (True) or Client DSN (False).

    - prmIS_SYSTEM_DSN ('IS_SYSTEM_DSN') – determines, if the given DSN is System DSN (True) or User DSN (False).
    - prmOPEN_MODE (OPEN_MODE) – determines open mode for DSN. Possible values are: prmOPEN_MODE_normal (Normal), prmOPEN_MODE_accelerated (Accelerated), prmOPEN_MODE_readonly (ReadOnly) and prmOPEN_MODE_exclusive (Exclusive). Engine DSN specific.
    - prmSERVER_NAME ('SERVER_NAME') – the address of the server or host name and the port number where the data resides. Client DSN specific.
    - prmTCP_PORT ('TCP_PORT') – TCP port on server. Client DSN specific.
    - prmSERVER_DSN ('SERVER_DSN') – the name of an Engine DSN on server. Client DSN specific.
    - prmTRANSPORT_HINT ('TRANSPORT_HINT') – contains transport hint. Client DSN specific. Possible values: TCP.
    - prmARRAY_FETCH_ON ('ARRAY_FETCH_ON') – enables array fetching (True/False). Client DSN specific.
    - prmARRAY_BUFFER_SIZE ('ARRAY_BUFFER_SIZE') – size of the array buffer. Values between 1 and 64KB are acceptable. Client DSN specific.
- Handle property type was changed to SQLHENV.
- Remote DSNs management is supported only by Pervasive.SQL 2000 SP2a or above. Remote client and user DSNs are not supported.
- AddStandardAlias method adds a Engine DSN with default settings. Second parameter is DB name.
- UpdateDsnType: TDsnTypes property has been added. TDsnTypes = (dsnSystem, dsnUser). UpdateDsnType means:
    - In case of DeleteAlias function – which DSN user is going to delete: System or User DSN.
    - In case of AddAlias and GetAliasParameters functions – hints which DSN type we prefer.
- Sample program how to create system client DSN:

```
var MyStringList: TStringList;
begin
  MyStringList := TStringList.Create;
  try
    MyStringList.Clear();
    MyStringList.Add('IS_ENGINE_DSN=False');
```

```
        MyStringList.Add('IS_SYSTEM_DSN=True');
        MyStringList.Add('SERVER_NAME=ServerName');
        MyStringList.Add('SERVER_DSN=DEMO1');
              // DSN on the Server
        PvSqlSession1.AddAlias('ATest', MyStringList);
      finally
        MyStringList.Free;
      end;
```

TPvDatabase and TPvSqlDatabase

- ODBCPacketSize property is an SQLUINTEGER value that specifies the network packet size in bytes. It is used in SQLSetConnectAttr() to set the SQL_ATTR_PACKET_SIZE attribute that governs aspects of connections.

Tip... For more information on SQLSetConnectAttr(), please refer to Microsoft ODBC Programmer's Reference.

- The following properties were dropped:
  - DriverName
  - Locale
  - TraceFlags
- TransIsolation property has only 1 value (tiReadCommitted), because Zen does not support other isolation levels.
- Tables property was added. It contains list of tables in database.
- StoredProcs property was added. It contains list of stored procedures in database.
- LoginPromptOnlyIfNecessary was added. Following situations are possible:
  - If LoginPrompt=true, LoginPromptIfNecessary=true. Login dialog is displayed only for secured database (after unsuccessful login attempt). This is the default behavior.
  - If LoginPrompt=true, LoginPromptIfNecessary=false. Login dialog is displayed always before login attempt. This is VCL's TDatabase behavior. The most suitable for secured databases. Provides fastest way to login.
  - If LoginPrompt=false, LoginPromptIfNecessary=true/false (not matter). Login dialog is never displayed. User can implement own login dialog.

TPvDatabase Specific

- Handle property has been changed to DRM_DBID.
- IsSQLBased property is always false.
- Directory property on local servers (that is, if TPvSession.ServerName property is empty or has name of the local server) contains path to database's data dictionary files. For remote servers it is always empty. In both cases attempt to set it will cause exception. You could know if you work with local or remote server by examining property TPvSession.LocalSystem.
- The following properties have type TDRMTableCollection:
  - Tables
  - StoredProcs
- IsSecured property added boolean. It is read-only property. IsSecured = True if database has security turned on. In this case the MicroKernel Engine owner name in property TPvTable.Owner is ignored, and user must authorize in order to login in the DB.
- OEMConversion property are added.

  This property indicates that the database contains characters encoded according to the OEM (DOS) code page, and that these characters should be converted to the ANSI (Windows) code page before use. The database remains in the OEM code page, but all reads and writes of character data are translated by PDAC.

  This conversion uses the mapping provided by the Windows OemToCharBuff and CharToOemBuff functions. It is important to note that not all characters are round-trip convertible. Only the characters present in both the OEM and ANSI code pages will be preserved in an update. As a rule of thumb, most of the alphabetic characters are preserved, but other types of characters, such as the box-drawing characters, may not be. For characters that cannot be preserved exactly, the closest look-alike character is chosen. For example, the box-drawing characters are replaced by plus (+), minus (-), and pipe (|).

  Currently, only characters stored in user tables are converted. Metadata (stored in DDF files) such as table, column, and file names are not.

TPvSqlDatabase Specific

- Handle property type was changed to SQLHDBC.
- IsSQLBased property is always true.
- Directory property is always empty. Attempt to set it will cause an exception.
- The following properties have type TSqlTablesInfoCollection and TSqlStoredProcInfoCollection correspondingly.

    - Tables
    - StoredProcs
- Exclusive property means nothing. It is provided for VCL compatibility only.
- AliasNameIsConnectionString property was added. This property provides possibility for DSN-less connections. If AliasNameIsConnectionString=true, then AliasName (or DatabaseName, if AliasName is empty) is connection string.

TPvTable, TPvQuery, and TPvStoredProc

- TDBDataSet properties/methods/events types have changed to TPvDataSet or *derived*.
- TSession properties/methods/events type have changed to TPvAbsSession or *derived*.
- TDatabase properties/methods/events type have changed to TPvAbsDatabase or *derived*.
- ExpIndex property is always false.
- The following properties were dropped.
    - Locale
    - DBLocale
    - ObjectView
- Database property with type TPvDatabase or TPvSqlDatabase was added.
- ConstraintCallBack was dropped.
- CheckOpen method parameter type was changed to DRM_ERR.

TPvTable Specific

- Handle property type was changed to DRM_TABLEID.
- TDBHandle property type was changed to DRM_DBID.
- Owner property was added, representing the Btrieve Owner Name for the table.
- BtrHandle property: TBtrieveInfo has been added. This is helper property for function DirectBtrCall. TBtrieveInfo contains the following fields:
    - pKeyBuf – pointer to key buffer;
    - KeyLen – key length;
    - KeyNum – key number;
    - CurFilter – pointer to formed input data buffer structure for extended operations (see GetNextExtended in the *Btrieve API Guide*). This buffer also contains the current filter.

- CurFilterLen – length of CurFilter buffer. CurFilterLen is useful when developer want to copy CurFilter data from the buffer to another location in memory.

- DirectBtrCall (Op: Smallint function; pDataBuf: Pointer; var DataLen: Word; pKeyBuf: Pointer; KeyLen: Byte; KeyNum: Shortint): integer has been added. It allows developers to call Btrieve directly, using the TpvTable Position Block. It has the following parameters:
  - Op – Btrieve operation. See *Btrieve API Guide*;
  - pDataBuf – analog for the data buffer parameter of BTRCALL;
  - DataLen – analog for the data buffer length parameter of BTRCALL;
  - pKeyBuf – analog for the key buffer parameter of BTRCALL;
  - KeyLen –length of the key buffer;
  - KeyNum – analog for the key number parameter of BTRCALL.

  DirectBtrCall returns the Btrieve status code. pKeyBuf, KeyLen and KeyNum parameters should be taken from PvTable.BtrHandle property.

  Small samples below demonstrate how to lock and unlock the current record via direct calls to Btrieve:

```
procedure TForm1.Lock(Sender: TObject);
var    b: TBookmark;
DataLen: word;
Res: integer;
begin
b := PvTable1.GetBookmark();
try
     DataLen := 4;
      Res := PvTable1.DirectBtrCall(B_GET_DIRECT + 300, b, DataLen, PvTable1.BtrHandle.pKeyBuf,
        PvTable1.BtrHandle.KeyLen, PvTable1.BtrHandle.KeyNum);
    finally
    PvTable1.FreeBookmark(b);
    end;
    end

procedure TForm1.Unlock(Sender: TObject);
var   Res: integer;
vr: Word;
    begin
      vr := 0;
      Res := PvTable1.DirectBtrCall(B_UNLOCK, @vr, vr, @vr, vr, -2);
    end
```

- IndexFiles property has been dropped.
- TableType property has been dropped.
- TableLevel property is ignored.
- UpdateObject property has been dropped.
- The following methods have been dropped.

    - CloseIndexFile
    - OpenIndexFile
    - LockTable
    - UnlockTable
- PvCreateTable(PvFieldDefs: TPvFieldDefs) method has been added. It allows the developer to tune the table creation process somewhat. See the appropriate section for additional details.
- SetOwnerOnTable(AOwner: string; AccessMode: integer) method has been added. It allows the developer to set the Btrieve owner name on a table. AccessMode can be:
    - B_ACCESS_RWOWNER – requires an owner name for any access mode (no data encryption).
    - B_ACCESS_WOWNER – permits read-only access without an owner name (no data encryption).
    - B_ACCESS_RWOWNERENCRYPT – requires an owner name for any access mode (with data encryption).
    - B_ACCESS_WOWNERENCRYPT – permits read-only access without an owner name (with data encryption).

TPvQuery and TPvStoredProc Specific

- Handle property type was changed to SQLHSTMT.
- DBHandle property type was changed to SQLHDBC.
- StmtHandle was dropped.
- Text property always returns text from property SQL.
- CheckOpen method was dropped.
- All index and key related properties and methods (like GetIndexInfo) were dropped.

TPvQuery Specific

The following properties were added:

- LoadBlobOnOpen

  If LoadBlobOnOpen is true, then all BLOBs will be cached in memory on query open. If LoadBlobOnOpen is false, then BLOBs will be read as required.
- PassThrough

Setting this property true will force PDAC to pass the SQL Text directly to the engine, without the pre-parsing that is ordinarily done to bind parameters. This is necessary when, for instance, creating Stored Procedures with parameters. Use the property as follows:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  PvQuery1.SQL.Clear;

  PvQuery1.SQL.Add('CREATE PROCEDURE TestPr(IN :A INTEGER) AS');
  PvQuery1.SQL.Add('BEGIN');
  PvQuery1.SQL.Add('PRINT :A;');
  PvQuery1.SQL.Add('END');
  PvQuery1.PassThrough := True;
  PvQuery1.ExecSQL;
  PvQuery1.PassThrough := False;
end;
```

The PassThrough Property is available at Design Time in the IDE, as well.

TPvStoredProc Specific

- Overload property was dropped.

TPvUpdateSQL

- TQuery property type (for all properties) has been changed on TPvQuery.

TPvBatchMove

- Transliterate property was dropped.
- Destination property type was changed to TPvTable.

Zen and Embarcadero Data Types

The following topics detail data type mappings:

- Zen and Embarcadero Data Type Mappings
- Btrieve and Embarcadero Data Type Mappings

Zen and Embarcadero Data Type Mappings

The following table shows data type mappings from Zen column types to Delphi data types. Types on the left stored in Zen databases are exposed by PDAC components as those listed on the right.

Zen Data Type	Delphi Data Type
BigInt	ftBCD
Binary	ftBytes
Bit	ftBoolean
Char	ftString
Currency	ftCurrency
Date	ftDate
Decimal	ftBCD
Double	ftFloat
Float	ftFloat
Identity	ftAutoInc
Integer	ftInteger
Longvarbinary	ftBlob
Longvarchar	ftMemo
Numeric	ftBCD
Real	ftFloat
Smallidentity	ftAutoInc
Smallint	ftSmallInteger
Time	ftTime
TimeStamp	ftDateTime
Tinyint	ftSmallInteger
Ubigint	ftBCD
Uint	ftInteger
Usmallint	ftWord

Zen Data Type	Delphi Data Type
Utinyint	ftWord
Varbinary	ftVarBytes
Varchar	ftString

Note: In the current release, PDAC does not support the BIGIDENTITY data type.

The following table shows data type mappings from Delphi to Zen data types. When new database tables are created using PDAC, columns defined as the field types listed in the left hand column will be stored by Zen using the types listed on the right.

Delphi Data Type	Zen Data Type
ftAutoInc	Identity
ftBCD	Numeric
ftBlob,	Longvarbinary
ftBoolean	Bit
ftBytes	Binary
ftCurrency	Currency
ftDate	Date
ftDateTime	DateTime
ftFixedChar	Char
ftFloat	Double
ftFmtMemo	LongVarChar
ftGraphic	Blob
ftInteger	Integer
ftLargeInt	BigInt
ftMemo	Longvarchar
ftSmallInteger	Smallint
ftString	Varchar
ftTime	Time
ftTypedBinary	Binary

Delphi Data Type	Zen Data Type
ftVarBytes	VarChar
ftWord	Smallint

Btrieve and Embarcadero Data Type Mappings

The following table shows data type mappings from Btrieve to VCL.

Note: Binary flags refer to a flag in X$Fields.Xe$Flags.

Btrieve Data Types	Binary flag	Length in bytes	VCL Data Types
AUTOINCREMENT (15)			ftAutoInc
BFLOAT (9)			ftFloat
BIT (16)			ftBoolean
BLOB (21)	+		ftBlob
BLOB (21)	-		ftMemo
CURRENCY (19)			ftBCD
DATE (3)			ftDate
DECIMAL (5)			ftBCD
FLOAT (2)			ftFloat
INTEGER (1)		1	ftSmallint
INTEGER (1)		2	ftSmallint
INTEGER (1)		4	ftInteger
INTEGER (1)		8	ftBCD
LOGICAL (7)		1	ftBoolean
LOGICAL (7)		2	ftSmallint
LSTRING (10)	+		ftVarBytes
LSTRING (10)	-		ftString
LVAR (13)	+		ftBCD
LVAR (13)	-		ftMemo
MONEY (6)			ftBCD

Btrieve Data Types	Binary flag	Length in bytes	VCL Data Types
NOTE (12)	+		ftBlob
NOTE (12)	-		ftMemo
NUMERIC (8)			ftBCD
NUMERICSA (18)			ftBCD
NUMERICSTS (17)			ftBCD
STRING (0)	+		ftBytes
STRING (0)	-		ftString
TIME (4)			ftTime
TIMESTAMP (20)			ftDateTime
UNSIGNED BINARY (14)		1	ftWord
UNSIGNED BINARY (14)		2	ftWord
UNSIGNED BINARY (14)		4	ftInteger
UNSIGNED BINARY (14)		8	ftBCD
ZSTRING (11)			ftString

The following table shows data type mappings from VCL to Btrieve.

VCL Data Types	Btrieve Type	Binary flag	Length, bytes
ftAutoInc	AUTOINCREMENT (15)		4
ftBCD	NUMERIC (8)		
ftBlob	BLOB (21)	+	FieldDefs[].Size
ftBoolean	LOGICAL (7)		1
ftBytes	STRING (0)	+	FieldDefs[].Size
ftCurrency	CURRENCY (19)		
ftDate	DATE (3)		
ftDateTime	TIMESTAMP (20)		
ftFixedChar	STRING (0)	-	FieldDefs[].Size
ftFloat	FLOAT (2)		8

VCL Data Types	Btrieve Type	Binary flag	Length, bytes
ftFmtMemo	BLOB (21)	-	FieldDefs[].Size
ftGraphic	BLOB (21)	+	FieldDefs[].Size
ftInteger	INTEGER (1)		4
ftLargeint	INTEGER (1)		8
ftMemo	BLOB (21)	-	FieldDefs[].Size
ftSmallint	INTEGER (1)		2
ftString	ZSTRING (11)	-	FieldDefs[].Size
ftTime	TIME (4)		
ftTypedBynary	STRING (0)	+	FieldDefs[].Size
ftVarBytes	LSTRING (10)	+	FieldDefs[].Size
ftWord	UNSIGNED BINARY (14)		2

Note: Binary flags refer to a flag in X$Fields.Xe$Flags

Additional Field Type Information

Zen (TPvTable.PvCreateTable) has a specific table create method that allows you to tune additional parameters related to field types. It has the following definition:

```
Procedure PvCreateTable(PvFieldDefs: TPvFieldDefs)
```

In PvFieldDefs you can adjust several parameters:

```
TPvFieldDef = class(TCollectionItem)
public
  FieldNum: integer;
  BtrType: integer;
  DrmType: word;
  ColumnSize: integer;
  DefaultValue: string;
  IsColumnCaseInsensitive: boolean;
  ACS_FileName: string;
  ACS_Name: string;
  ACS_ID: string;
```

```
end;
```

Where:

- FieldNum – field number in primary FieldDefs.
- BtrType – Btrieve type.
- DrmType – data record manager type. Developer does not need to use this field. It was added for use in TPvBatchMove only.
- ColumnSize – Column size in bytes.
- DefaultValue – Default value for column in string format.
- IsColumnCaseInsensitive – Set it in true if indexes for field are case insensitive.
- ACS_FileName – Set file name with ACS data without .alt extension.
- ACS_Name – Set name of ACS data.
- ACS_ID – Set ID of ACS data.

If you do not want to set a particular field, it can be set to 0 (DrmType, ColumnSize), -1 (BtrType), false (IsColumnCaseInsensitive) or (all string fields). In this case, default values will be used. Field FieldNum is required.

Fields IsColumnCaseInsensitive, ACS_FileName, ACS_Name, ACS_ID are mutually exclusive. That is, you can set only one of them.

To match Btrieve behavior, index option ixCaseInsensitive is ignored.

The following is an example of using the PvCreateTable method:

```
with PvTable1.FieldDefs do
begin
  Clear;
  Add('F_AutoInc', ftAutoInc, 0, true);
  Add('F_Bytes', ftBytes, 10, False);
end;

PvFieldDefs := TPvFieldDefs.Create(TPvFieldDef);
try
  PvFieldDef := PvFieldDefs.Add();
  PvFieldDef.FieldNum := 1; // F_Bytes
  PvFieldDef.BtrType := 10;
```

```
  PvFieldDef.DrmType := DRM_coltypVarText;
  PvFieldDef.ColumnSize := 20;
  PvFieldDef.IsColumnCaseInsensitive := true;

  PvTable1.PvCreateTable(PvFieldDefs);
finally
  PvFieldDefs.Free();
end;
```