

How to use Borland Visual Database Tools

Borland Visual Database Tools includes both visual and non-visual database components. Developers can use either the visual components, the non-visual components, or a combination of both to suit their needs. Also, the Borland Visual Database Tools provide database functionality to a broad range of developers: from spreadsheet macro programmers to C++ programmers.

Non-visual Database Components

The non-visual database components, which can be used from C++, provide an object oriented encapsulation of the Borland Database Engine, with the same objects, methods, properties, and events as the database components found in Delphi. The components are TDataSource, TDataSet, TDBDataSet, TTable, TQuery, TStoredProc, TDatabase, TBatchMove, TSession, TField, TBlobField, TFieldDef, TFieldDefs, TIndexDefs, TIndexDef, TParam, TParams, TStrings, TStream. The architecture used in building these non-visual database components is the Component Object Model (COM), the underlying architecture of OLE. Because all of the non-visual database components support the IDispatch interface, they can be used from an OLE Automation Controller like Excel. Also provided are OWL style classes which allow a C++ programmer to use the components without having to learn OLE or COM.

Here is an example of a C++ function which opens a table and enumerates through the fields in the first record using the OWL class TTable.

```
void EnumerateFields( void )
{
    // Create the TTable Object.
    PTable table = new TTable;
    if (table)
    {
        // Set the DatabaseName and TableName properties.
        table->DatabaseName = string( "DivePlan" );
        table->TableName = string( "biolife.db" );

        // Call the Open method.
        table->Open();

        // Call the First method. (This is actually unnecessary after the Open method.)
        table->First();

        // Get the FieldCount property.
        int count = table->FieldCount;

        // Enumerate through the fields.
        int f;           // iterator variable
        PField field;    // TField object

        for (f = 0; f < count; f++)
        {
            // Get the Field property.
            field = table->Fields[ f ];
            if (field)
            {
                // Get the FieldName property.
                PrintFieldName( field->FieldName->c_str() );
            }
        }
    }
}
```

```

        // Get the Text property.
        PrintFieldValue( field->Text->c_str() );
    }
}

// Call the Close method.
table->Close();

// When finished, delete.
delete table;
}
}

```

Here is an example of a C++ function which opens a table and enumerates through the fields in the first record using the COM interface ITTable.

```

void EnumerateFields( void )
{
    // Create the ITTable Object.
    PITTable table = CreateITTable( );
    if (table)
    {
        // Set the DatabaseName and TableName properties.
        PITAnyString d = CreateITAnyString();
        d->put_AsStringBuf( "DivePlan" );
        table->put_DatabaseName( d );
        d->Release();

        PITAnyString t = CreateITAnyString();
        t->put_AsStringBuf( "biolife.db" );
        table->put_TableName( t );
        t->Release();

        // Call the Open method.
        table->Open();

        // Call the First method. (This is actually unnecessary after the Open method.)
        table->First();

        // Get the FieldCount property.
        int count = table->get_FieldCount();

        // Enumerate through the fields.
        int f;           // iterator variable
        PITField field;  // ITField object
        char str[256];    // buffer used for field names and values
        for (f = 0; f < count; f++)
        {
            // Get the Field property.
            field = table->get_Fields( f );
            if (field)
            {
                // Get the FieldName property.
                PITAnyString n = field->get_FieldName();
            }
        }
    }
}

```

```

        n->get_AsStringBuf( str, sizeof(str)-1 );
        n->Release();
        PrintFieldName( str );

        // Get the Text property.

        PITAnyString t = field->get_Text();
        t->get_AsStringBuf( str, sizeof(str)-1 );
        t->Release();
        PrintFieldValue( str );

        // When finished, call the Release method.
        field->Release();
    }

    // Call the Close method.
    table->Close();

    // When finished, call the Release method.
    table->Release();
}

```

Here is an example of an Excel subroutine which opens a table and enumerates through the fields in the first record using the OLE Automation object BDT50.TTable.

```

Public table As Object
Public count As Integer

```

```

Sub EnumerateFields()

```

```

    Rem Create the TTable Object.
    set table = CreateObject( "BDT50.TTable" )

```

```

    Rem Set the DatabaseName and TableName properties.
    table.DatabaseName = "DivePlan"
    table.TableName = "biolife.db"

```

```

    Rem Call the Open method.
    table.Open

```

```

    Rem Call the First method. (This is actually unnecessary after the Open method.)
    table.First

```

```

    Rem Get the FieldCount property.
    count = table.FieldCount

```

```

    Rem Enumerate through the fields.
    For f = 0 to count-1
        Rem Get the FieldName property and put it in a cell.
        Cells( f+1, 1 ).Value = table.Fields( f ).FieldName

```

```

        Rem Get the Text property and put it in a cell.
        Cells( f+1, 2 ).Value = table.Fields( f ).Text
    Next f
End Sub

```

Next f

Rem Call the Close method.
table.Close

End Sub

Visual Database Components

The visual database components are VBX controls which mimic the visual database components found in Delphi. There are two sets of VBX controls. One set are data access components, also called data providers. The other set are data aware components. These VBX components provide the same properties as the corresponding Delphi components.

The data access components are DataSource, Table, Query, StoredProc, Database, and BatchMove. The data access components are built on top of the non-visual database components discussed earlier in this document. The data aware components are Navigator, Text, Edit, Memo, Image, ListBox, ComboBox, CheckBox, RadioGroup, LookupList, LookupCombo, and Grid. The data aware components are not dependent on the non-visual database components.

A user can use these VBX data access controls in dialog boxes by building the dialog boxes with Resource Workshop.

Here are steps to build a dialog with database components in Resource Workshop:

Create a new project by choosing the File | New | Resource Project... menu. You will be presented with a dialog box where you can choose what type of project you want to create. Select the Resource Script and press OK. Create a new dialog box by choosing the Resource | New... menu. You will be presented with a dialog box where you can choose what type of resource you want to create. Select DIALOG in the list box and press OK.

On the Controll Palette (right click within the Dialog window and select Control Palette if you don't see the Control Palette) you should see the data access and data aware components.

Choose the Table Component from the Data Access tab



and place it on the dialog box. This component is only visible while you are designing the form. When you test the form or use it from an application, this component will not be visible.

Using the Property Inspector (double click on the Table component if you don't see the Property Inspector), select the DatabaseName property. The property inspector will provide a combobox with all the databases that are currently installed on your system. (If you don't see any, you should make sure that the Borland Database Engine has been installed correctly and that you have at least one database alias by using the BDE configuration utility.) Choose the database that you would like to use.

Using the Property Inspector, select the TableName property. The property inspector will provide a combobox with all the tables that are currently in the database you selected in the previous step. Choose the table that you would like to use.

Using the Property Inspector, select the Active property. Choose True to "open" the table.

Choose the DataSource Component



and place it on the dialog box. This component is only visible while you are designing the form. When you test the form or use it from an application, this component will not be visible.

Using the Properties Palette, select the DataSet property. The property inspector will provide a combobox with the IDs of all the datasets (tables, queries, and storedprocs) that are currently on the dialog you are building. Choose the ID of the table that you placed previously.

Select the Table Component which you previously placed on the dialog.

Choose the Navigator Component from the Data Aware tab



and place it on the dialog box.

Using the Property Inspector, select the DataSource property. The property inspector will provide a combobox with the IDs of all the datasources that are currently on the dialog you are building. Choose the ID of the datasource that you placed previously.

Choose the Edit Component



and place it on the dialog box.

Using the Property Inspector, select the DataSource property. The property inspector will provide a combobox with the IDs of all the datasources that are currently on the dialog you are building. Choose the ID of the datasource that you placed previously.

Using the Property Inspector, select the DataField property. The property inspector will provide a combobox with the names of the fields in the table you are using. Choose the field that you would like to use.

If the table you are using has a Graphic field, choose the Image Component



and place it on the dialog box. Set the DataSource and DataField properties as you did with the Edit Component.

Select Dialog | Test Dialog from the main menu. You will see field values displayed in the Edit and Image components and you can to use the Navigator to move through the table.

Combining Visual and Non-Visual Database Components

After building a dialog box in Resource Workshop, you may want to enhance it by providing some functionality in C++. For example you might want to have a button which navigates forward in the table by 5 records.

Here is some code which gets the underlying non-visual database component from the visual database component and calls a method using the OWL Class TTable:

```
TTable* table = new TTable( hdlg, IDC_TVXDBTABLE1 );
if (table)
{
    table->MoveBy( 5 );
    delete table;
```

}