Database Applications with VDBT Components

Ted Faison
96/4/23

Developing database applications has always been a very slow process, hindered by a myriad of low-level details imposed by the underlying database management system (DBMS). Recognizing this, Borland developed a new set of programming tools that make much of the development process visual. Programmers develop database user interfaces by creating windows, dropping visual components on them, and setting a number of properties. The new collection of components and tools created to support this programming style is known as *Visual Database Tools* (VDBT), and was initially developed for Borland Delphi using ObjectPascal. The VDBT components have now been ported to C++, bringing a new level of functionality to C++ programmers. This paper will discuss the architecture of VDBT components, showing how they are used in C++ applications.

Live Data

An important goal in rapid application development is to detect problems in screen layout and errors in database queries as soon as possible. Most database development environments require you to write code to pull data out of tables and populate dialog box fields. To see if your code is producing the desired results, you are compelled to go through a time-consuming process of edit-compile-link. Not with Borland C++ 5.0. The new VDBT components support the concept of *live-data*. When you place a VDBT control on a dialog box and set its database connection properties, you can use the **Test Dialog** command on the **Dialog** menu to see live data immediately. You can see how an entire form will appear to the user at runtime, allowing you to make corrections in the layout or database queries before you even compile your code. Queries can be used to produce complex result sets using table joins. The ability to test queries interactively enables you to switch your database development into high gear.

The Borland Database Engine

At the heart of VBDT is a database engine known as the *Borland Database Engine (BDE),* a DLL that sits between your database application and the DBMS. Your application's VDBT controls make calls to BDE in a database-independent manner. The BDE makes calls into DBMS-specific drivers, which translate them into DBMS calls, returning data to your application. The VDBT controls have no knowledge of the low-level DBMS calls needed to carry out a given task. The BDE plays a role equivalent to Microsoft's ODBC Driver Manager, insulating applications from low-level database issues, providing additional features to enhance performance, such as caching and cursor support. Using BDE you can access all standard desktop databases, ODBC data sources and client/server SQL databases. The overall relationship between your application and BDE is shown in Figure 1.

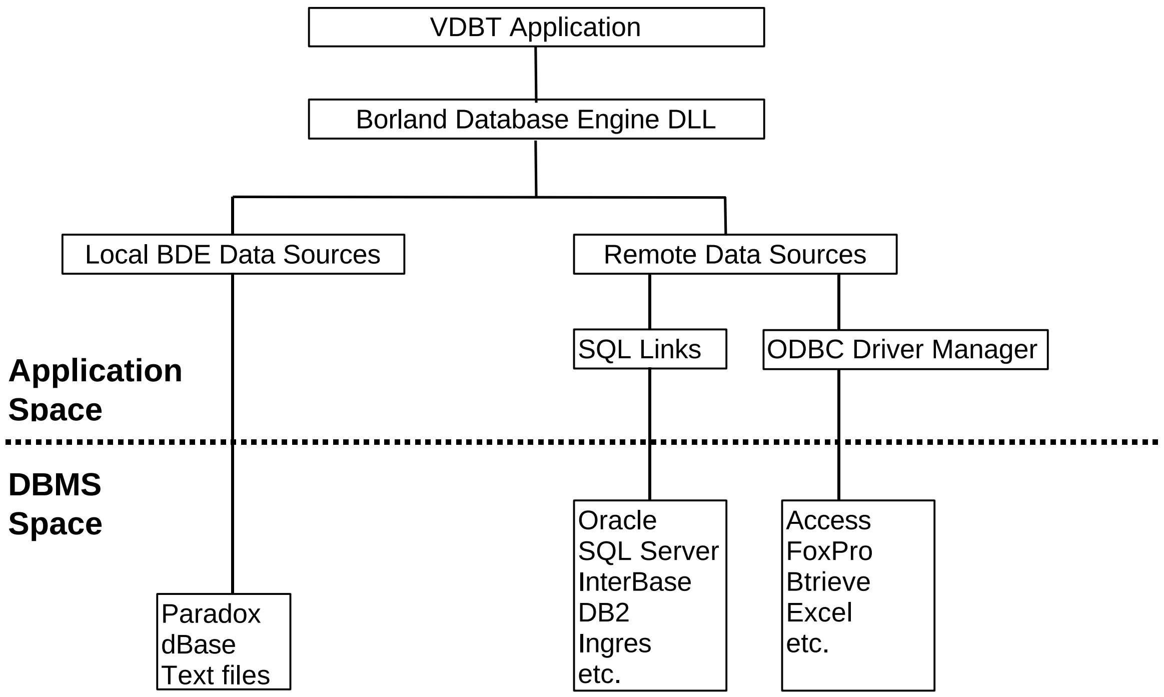
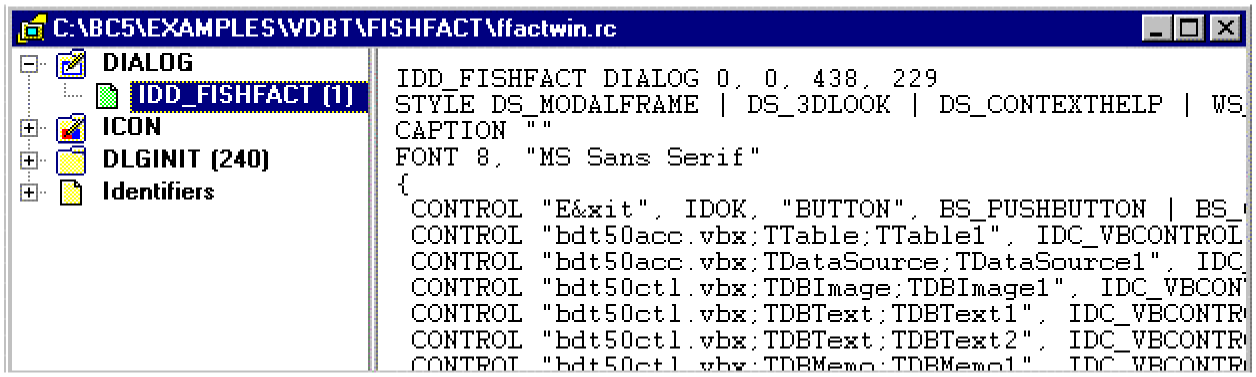


Figure 1 - The overall architecture of an application using BDE.

When you add VDBT controls to an application, the connection to BDE is automatic, and no programming is necessary. The process is entirely visual. At runtime, VDBT loads BDE and makes the necessary queries to obtain the data indicated by the properties of each VDBT control you use.

A New Visual Development Environment

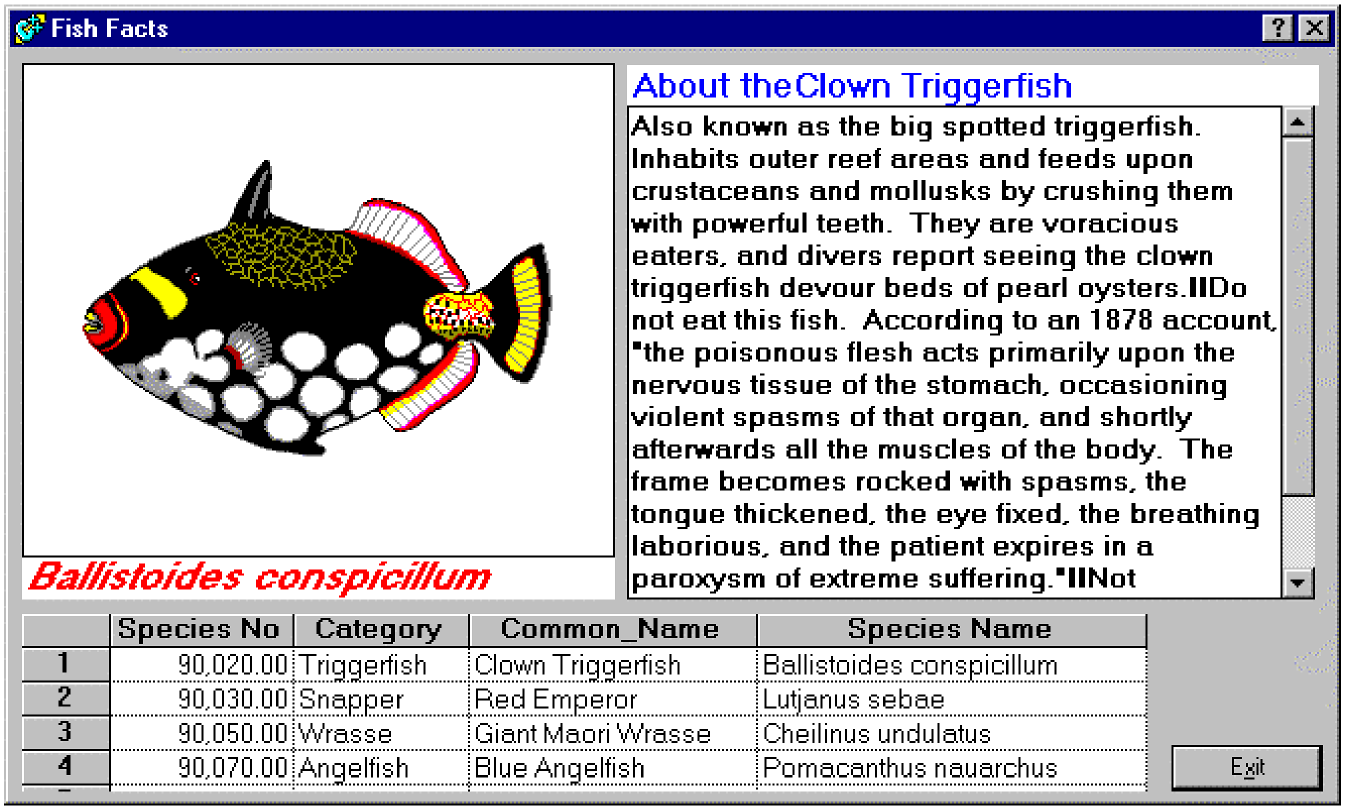
From an application developer’s perspective, all database connectivity is handled by placing VDBT tools on a dialog box, using the new Borland C++ Resource Editor, formerly known as Resource Workshop. The old Resource Workshop has been entirely integrated into the Borland C++ IDE, allowing seamless development of Windows resources and C++ code. Double clicking an RC node in the Project Window launches the Resource Editor, appearing as in Figure 2.



```
IDD_FISHFACT DIALOG 0, 0, 438, 229
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP | WS_
CAPTION ""
FONT 8, "MS Sans Serif"
{
 CONTROL "E&xit", IDOK, "BUTTON", BS_PUSHBUTTON | BS_
 CONTROL "bdt50acc.vbx:TTable:TTable1", IDC_VBCONTROL
 CONTROL "bdt50acc.vbx:TDataSource:TDataSource1", IDC_
 CONTROL "bdt50ctl.vbx:TDBImage:TDBImage1", IDC_VBCON
 CONTROL "bdt50ctl.vbx:TDBText:TDBText1", IDC_VBCONTR
 CONTROL "bdt50ctl.vbx:TDBText:TDBText2", IDC_VBCONTR
 CONTROL "bdt50ctl.vbx:TDBMemo:TDBMemo1", IDC_VBCONTR
```

Figure 2 - The new Integrated Resource Editor.

Double-clicking a resource launches a resource-specific editor. Figure 3 shows the Dialog Editor running on the dialog resource IDD_FISHFACT.



About the Clown Triggerfish

Also known as the big spotted triggerfish. Inhabits outer reef areas and feeds upon crustaceans and mollusks by crushing them with powerful teeth. They are voracious eaters, and divers report seeing the clown triggerfish devour beds of pearl oysters.IIDo not eat this fish. According to an 1878 account, "the poisonous flesh acts primarily upon the nervous tissue of the stomach, occasioning violent spasms of that organ, and shortly afterwards all the muscles of the body. The frame becomes rocked with spasms, the tongue thickened, the eye fixed, the breathing laborious, and the patient expires in a paroxysm of extreme suffering."IINot

Ballistoides conspicillum

	Species No	Category	Common_Name	Species Name
1	90,020.00	Triggerfish	Clown Triggerfish	Ballistoides conspicillum
2	90,030.00	Snapper	Red Emperor	Lutjanus sebae
3	90,050.00	Wrasse	Giant Maori Wrasse	Cheilinus undulatus
4	90,070.00	Angelfish	Blue Angelfish	Pomacanthus nauarchus

Figure 3 - The Integrated Dialog Resource Editor.

The dialog box shown contains a number of VDBT controls, such as a `TDBImage`, a `TDBText` and a `TDBGrid`. The tool palette contains pages of controls, and the VDBT controls are carried on two of the pages. By dropping the appropriate controls on a dialog, a database application can be created, often requiring little or even no programming.

The VDBT Controls

To make visual programming possible, it is obvious there is a lot going on behind the scenes. All VDBT controls are divided into 2 categories: data access controls and data-aware controls. The former handle the retrieval and manipulation of database data, the latter handle the presentation of data to the user. Figure 4 shows the overall grouping of the VDBT controls.

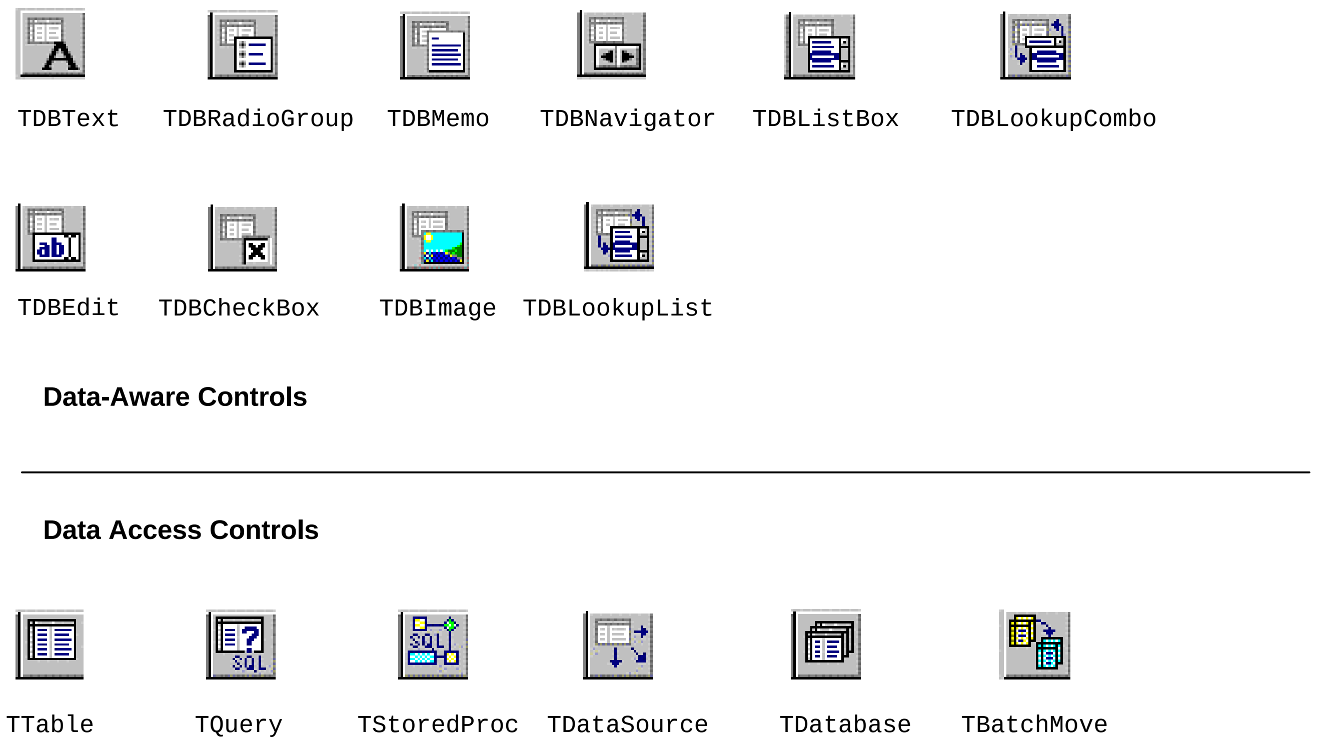
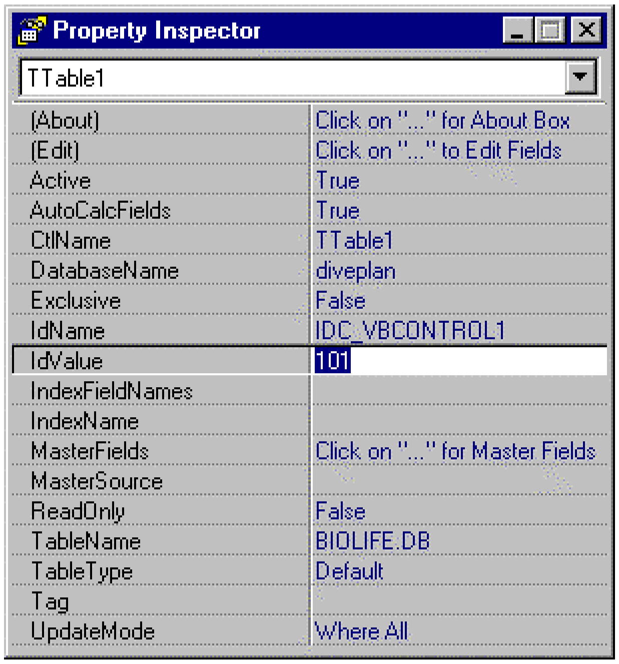


Figure 4 - The VDBT Controls.

VDBT controls are used in a standard way: You start with a control that manages the data set you want to work with. This control will usually be either a `TTable`, for the entire table or a `TQuery`, for a query. After placing a `TTable` on a dialog box, you setup its `DatabaseName`, `TableName` and `Active` properties. Figure 5 shows the **Property Inspector** associated with a `TTable` object.



(About)	Click on "..." for About Box
(Edit)	Click on "..." to Edit Fields
Active	True
AutoCalcFields	True
CtlName	TTable1
DatabaseName	diveplan
Exclusive	False
IdName	IDC_VBCONTROL1
IdValue	101
IndexFieldNames	
IndexName	
MasterFields	Click on "..." for Master Fields
MasterSource	
ReadOnly	False
TableName	BIOLIFE.DB
TableType	Default
Tag	
UpdateMode	Where All

Figure 5 - The Property Inspector for a `TTable` object.

Next you place a `TDataSource` control on the dialog box, and set its `DataSet` property to the `TTable` or `TQuery` component. Now you add the data-aware controls, setting their `DataSource` property to the `TDataSource` component and setting up the `DataField` property. No coding required. The following figure shows the relationship between the various parts, using a `TDBText` control. To see live data, choose **Dialog | Test Dialog** from the main menu.

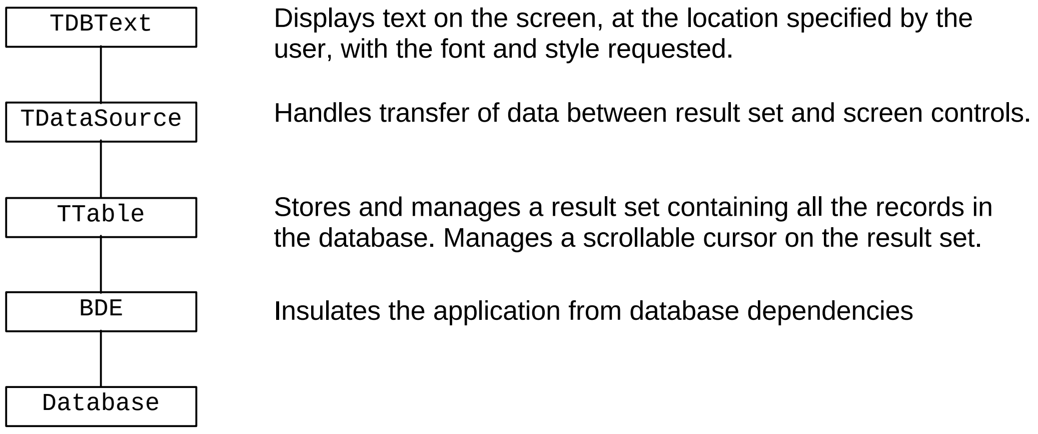

TDBText
Displays text on the screen, at the location specified by the user, with the font and style requested.
TDataSource
Handles transfer of data between result set and screen controls.
TTable
Stores and manages a result set containing all the records in the database. Manages a scrollable cursor on the result set.
BDE
Insulates the application from database dependencies
Database

Figure 6 - The hierarchy of VDBT components.

The following table describes the function of each of the VDBT data-aware controls.

Control	**Description**
`TDBText`	Displays simple text as static text. Doesn't allow users to change the text.
`TDBEdit`	Displays text in an edit box. Users can change text only if the database is not in `ReadOnly` mode, and they have update privileges on the database,
`TDBRadioGroup`	Displays a group box containing an array of radio buttons. Each radio button is associated with a value. Used for displaying enumerations, such as *Pizza, Beer, Milk Shake* for the values 0, 1, 2.
`TDBCheckBox`	Displays a normal checkbox, which is checked if the database field is non-zero, unchecked otherwise.
`TDBMemo`	Displays a multi-line edit control. Can manage up to 64 KB of text, unless the database has a lower limit.
`TDBImage`	Displays a picture stored as graphic data in the database. `TDBImage` can display BMP data formats.
`TDBNavigator`	Manages the scrollable cursor attached to a result set. All data-aware controls display fields obtained from the current record, which is the record in the result set on which the cursor is sitting. A `TDBNavigator` appears as a series of VCR-style controls.
`TDBGrid`	Appears as a spreadsheet containing multiple columns. By default a `TDBGrid` displays columns for all fields in the result set, but you can eliminate columns and rearrange them on the screen as necessary. By default, column titles match the database fields, but you can change them to anything you want.
`TDBListBox`	Displays a listbox of fixed selections. When the user makes a selection, the text is written to the database. Useful when the number of selections is large, and a `TDBRadioGroup` would need too many radio buttons to accomplish the same task.
`TDBLookupList`	Similar to a `TDBListBox`, except the listbox items are not fixed, but looked up in a database at runtime. You attach a separate `DataSource` object to govern how the lookup works.
`TDBLookupCombo`	Similar to a `TDBLookupList`, except the data is presented in a combo box.

Table 1 - The VDBT data-aware controls.

The next table describes the VDBT data access controls.

Control	**Description**
`TTable`	Makes direct calls into the BDE DLL. Creates and manages a result set containing all the records in the attached database table.
`TQuery`	Similar to a `TTable`, except the result set contains records that satisfy the SQL SELECT statement attached to the control.
`TStoredProc`	Similar to a `TTable`, except the result set is built using records returned by a given stored procedure. The name of the stored procedure is set using the property `StoredProcName`. Not all databases support the use of stored procedures, which are pre-compiled SQL procedures normally stored on the server side of a client/server DBMS.

`TDataSource`	Manages the transfer of data into and out of individual fields in the result set of an attached `TTable`, `TQuery` or `TStoredProc` result set.
`TDatabase`	A low-level encapsulation of a generic database. You use a `TDatabase` when you want to manipulate items that are at the database, not the table, level. For example, to issue commit or rollback commands you need to use a `TDatabase`. Use a `TDatabase` to create a temporary local Alias to a database.
`TBatchMove`	This component is used generally to support database administrator tools that require copying result sets, or even entire databases, from one place to another. You specify a source result set and a destination table. The destination is created if necessary.

Table 2 - The VDBT data-access components.

Handling VDBT Controls at runtime

Internally, the VDBT controls are implemented as VBX controls. This allows them to offer design-time features, and work at runtime as resources requiring no C++ code. You can manipulate the VDBT controls at runtime by adding the appropriate C++ object to your code. For example, to directly manipulate a `TDBText` control you added to the dialog `TMyDialog` in the Resource Editor, you would first add a data member to `TMyDialog`, like this:

```
class TMyDialog : public TDialog, public TVbxEventHandler {
protected:
      TDBText* myDBTextControl;
 // …

};
```

Objects of type `TVbxControl` are used to create C++ encapsulations of VBX controls, allowing you to use C++ notation to interface with the underlying controls. Because VDBT controls are VBX controls, any dialog that uses them must have `TVbxEventHandler` as a base class. Having declared a pointer to the VDBT control, you actually instantiate the control in the constructor body of the parent dialog box, like any other VBX control:

```
TTestDialog::TTestDialog(TWindow* parent, const char* name, TModule* module)
           : TDialog(parent, name, module)
{
   myDBTextControl = new TDBText(this, IDC_MYTEXTID);
}
```

The code assumes you gave the `TDBText` control the ID `IDC_MYTEXTID` in the Resource Editor. You don't need to delete the controls in the dialog destructor, since OWL takes care of that for you. To manipulate the `TDBText` control you use standard OWL VBX notation. For example, to change the text's font size to 14, you would use the code:

```
myDBTextControl-> FontSize = 14;
```

The property name is the same as the one shown in the Property Inspector. To get the value of the font size, you would use the code:

```
int fontSize = myDBTextControl->FontSize;
```

Handling VDBT Control Events

Most Windows controls have the ability to generate notification messages, and VDBT controls are no different. When the user does something to a control, that control will often fire off a notification message to the parent dialog. For example, when the user edits the data in a `TDBEdit` control, an `OnChange` event is fired. To handle it, you setup a VBX notification handler. First you declare the handler:

```
class TMyDialog : public TDialog, public TVbxEventHandler {
protected:
  TDBEdit *MyDBEdit;
  void ChangeHandler(TDBEditNotifySink& Sink, TDBEdit& Sender);
  TDBEditNotifySink ChangeSink;
// …

};
```

then you modify the dialog constructor to initialize the event sink and link the event source with the event sink:

```
TMyDialog::TMyDialog( TWindow* parent, const char* name, TModule* module )
  : TDialog( parent, name, module )
  , MyDBEdit( new TDBEdit( this, IDC_MYEDITID ) )
  , ChangeSink( TDBEditNotify_MFUNCTOR( *this, &TMyDialog::ChangeHandler ) )
{
   MyDBEdit->OnChangeSource += OnEditChangeSink; // Attach Sink to Source
}
```

and finally you write the body of the handler:

```
void TMyDialog::ChangedHandler( TDBEditNotifySink& Sink, TDBEdit& Sender )
{
  // do something
  MessageBeep(0);
  Sink.Pass( Sender );  // Foreward the event to next handler (optional)
}
```

The list of possible notifications is available for each control in the online help.

Handling complex queries

`TTable` objects are used to obtain all the records in a table. Most non-trivial database applications work with a subset of all the possible records, using queries. Joins are typically used to get results from multiple tables simultaneously. The VDBT `TQuery` Component supports this kind of functionality. By setting its `SQL` property, you can specify an arbitrarily complex SQL SELECT statement. The limits in complexity are generally a function of the underlying database drivers and DBMS, rather than of the VDBT components.

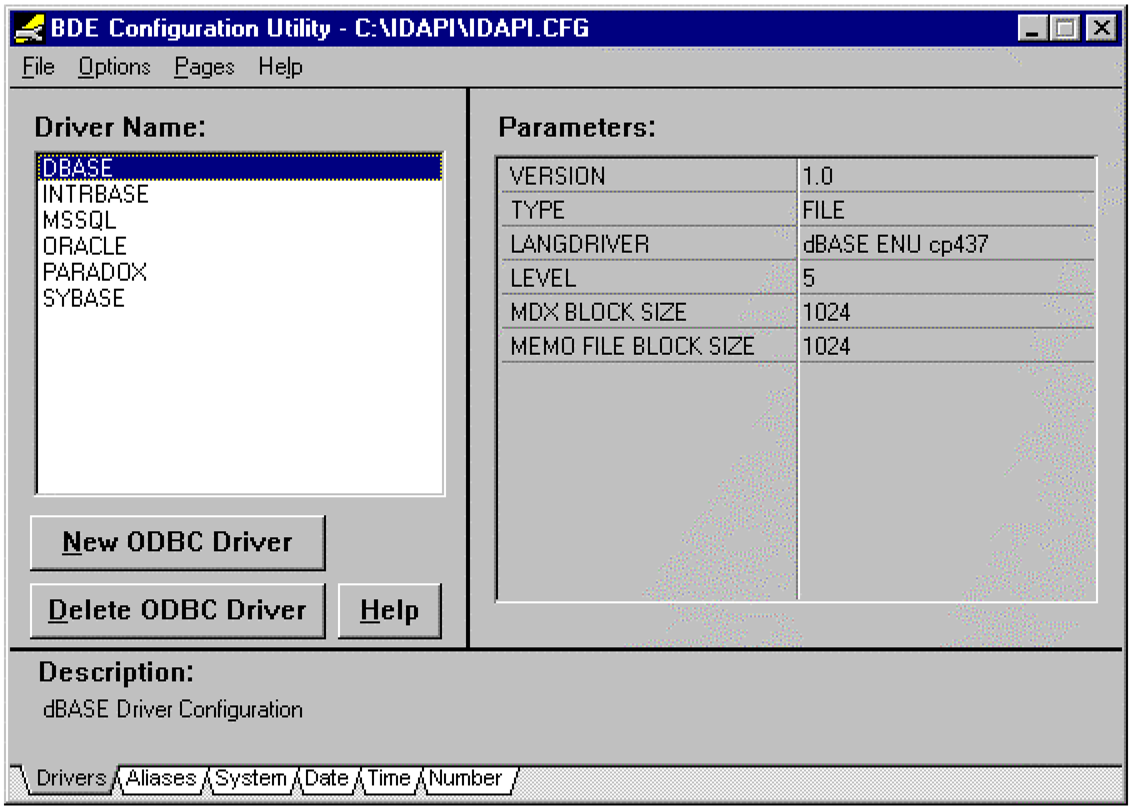
It is easy to create joins using a `TQuery` component. The returned result set can be handled with the same simplicity as a `TTable` result set. As an example of a join, you might use a SQL expression like this:

```
select Customer.Company, Orders.OrderNo, Orders.SaleDate
  from Customer, Orders
  where Customer.CustNo = Orders.CustNo
```

You can also produce updateable joins by setting the `RequestLive` property of `TQuery` to `True`. Users must obviously have sufficient database access privileges to update database records.

ODBC Support

The BDE has built-in support for common desktop databases, such as Paradox and dBase, and native access to SQL servers such as InterBase, Oracle, Sybase and Microsoft SQL Server. To access ODBC data sources, such as a Microsoft Access database, you must first create an ODBC Data Source for it, then use the **BDE Configuration Utility** to create an ODBC connection to that data source. The main window of the **BDE Configuration Utility** is shown in Figure 7.



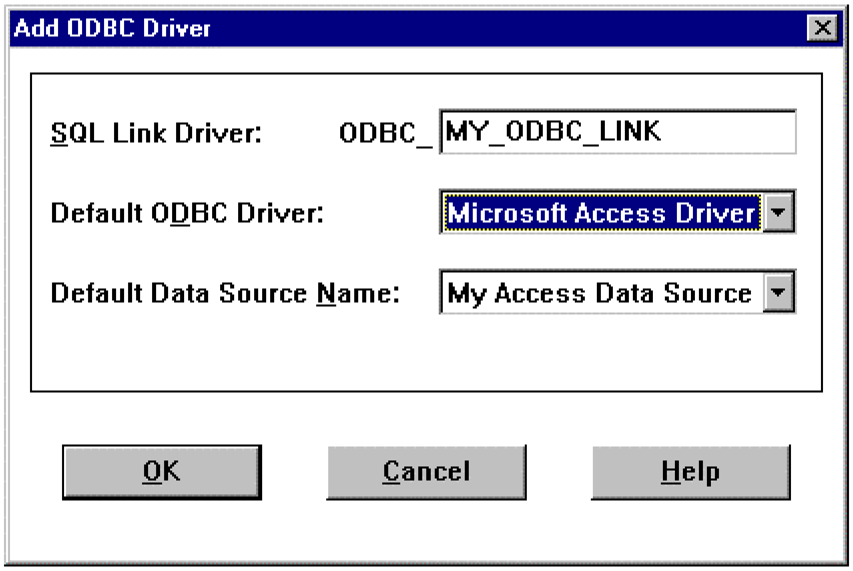
DBASE
INTRBASE
MSSQL
ORACLE
PARADOX
SYBASE

VERSION	1.0
TYPE	FILE
LANGDRIVER	dBASE ENU cp437
LEVEL	5
MDX BLOCK SIZE	1024
MEMO FILE BLOCK SIZE	1024

dBASE Driver Configuration

Figure 7 - The main window of the BDE Configuration Utility.

By clicking the **New ODBC Driver** button, the **Add ODBC Driver** dialog is displayed, as shown in Figure 8.



SQL Link Driver: ODBC_ MY_ODBC_LINK

Default ODBC Driver: Microsoft Access Driver

Default Data Source Name: My Access Data Source

Figure 8 - The BDE Configuration Utility's dialog box for adding a link to an ODBC data source.

Using the **Aliases** page in the **BDE Configuration Utility**, click the **New Alias** button to access the **Add New Alias** dialog box. Create a new alias. From the **Alias Type** dropdown list, select the ODBC driver you just added. You later use this name as the database name in the `DatabaseName` property of `TTable` or `TQuery` components. The **Default ODBC Driver** field is set to specify the type of database associated with the given ODBC data source. The **Default Data Source Name** is the ODBC Data Source name assigned to the data source using the Microsoft ODBC Administrator utility.

Client/Server Database Support

Using VDBT controls, you gain automatic access to all the popular desktop databases, either directly, through IDAPI drivers, or through ODBC drivers. But corporate developers often require database power that exceeds the capabilities of products like Paradox, Access or dBase. VDBT components also allow you to develop applications for large client/server database systems like Oracle, Informix, Sybase SQL Server, Microsoft SQL Server and others, without affecting your application source code. As described earlier, your application uses VDBT controls, which talk to the BDE, which dispatches your requests to a driver that handles a specific database. Using the **BDE Configuration Utility**, shown in Figure 7, you setup a database type using the **Driver Name** list. The **Parameters** pane shows the bindings and settings used by BDE. All entries have defaults, but you can change items to suit your specific needs. For example, assume you install version 7.3 of Oracle on your system. The BDE may have been configured to use the older 7.2 drivers. Using the **BDE Configuration Utility**, you can easily specify your new drivers, and nothing in your application is affected at all.

Because BDE offers local caching, it has the capability to let your applications run even faster than if you had used direct database-dependent SQL calls. The BDE uses the Borland **SQL Links** DLL to interface with the client-side drivers provided by SQL database vendors. SQL Links completely insulates the BDE from dependencies on networking or driver DLL idiosyncrasies. Whether your client/server networking uses a Windows NT network, Novell or TCP/IP, it makes no difference to you. It is only SQL Links that talks to the client-side driver of your DBMS, and only this driver has anything to do with the network. This degree of insulation between application code and DBMS makes it easy to develop client/server systems initially using a local server. After debugging the system, you can then deploy the full client/server system by simply changing a few database parameters.

Conclusion

If you develop any kind of database applications using C++, version 5.0 of the Borland C++ compiler will definitely turbo-charge your work. Simple database user interfaces can be built visually in a matter of hours or even minutes. Complex systems requiring queries are just as easy. Applications requiring queries with variable parameters are still much easier with the VDBT components than ever before. Because everything has evolved to a visual environment, almost everything you do will be easier to accomplish and take less time.

The simple hierarchy of the VDBT components allows you to programmatically make changes to the system at any level. You can control the low level database using a `TDatabase` object. Individual tables can be manipulated using `TTable` objects. The ability to handle complex queries and updateable joins is a really powerful feature of Borland C++ 5.0, allowing you to develop arbitrarily complex database applications. Advanced features in the Borland Database Engine – such as enhanced record locking, local caching and cursors – ensure your applications achieve outstanding performance levels.