Borland®

# Creating Kylix™ Database Applications

## A New Standard In Productivity

*by Bill Todd, President, The Database Group, Inc.*

**Table of Contents**

## dbExpress™ – A New Vision

Past attempts to create a common API for multiple databases have all had one or more problems. Some have been large, slow and difficult to deploy because they tried to do too much. Others have offered a least common denominator approach that denied developers access to the specialized features of some databases. Others have suffered from the complexity of writing drivers making drivers either limited in functionality, slow or buggy. Borland® Kylix™ overcomes these problems by combining dbExpress™, a completely new approach to providing a common API for many databases, with Borland's proven provide/resolve architecture for managing the data editing and update process.

This paper examines the architecture of dbExpress and the provide/resolve mechanism. Subsequent sections describe the components that implement the Kylix data access strategy and lead you through the steps to build a simple database application demonstrating the power and productivity they provide.

Kylix™

white paper

## The dbExpress Architecture

dbExpress was designed to meet the following five goals.

1. Minimize size and system resource use.
2. Maximize speed.
3. Provide platform independence.
4. Provide easy deployment.
5. Make driver development easy.

dbExpress drivers are small and fast because they provide very limited functionality. A dbExpress driver implements five interfaces that support fetching metadata, executing SQL statements and stored procedures and returning a read only unidirectional cursor to the result set. However, when used with the DataSetProvider and ClientDataSet or the SQLClientDataSet to implement Borland's provide/resolve data access strategy, dbExpress gives you a full featured, high performance, high concurrency system for working with data in SQL databases.

## How Provide/Resolve Architecture Works

The provide/resolve architecture uses four components to provide data access and editing. The first is the SQLConnection component that provides a connection to the dbExpress driver for the database you are using. Next is one of the dbExpress dataset components that provides data by executing a SQL SELECT statement or calling a stored procedure. The third component is the DataSetProvider and the fourth is the ClientDataSet. When you open the ClientDataSet it requests data from the DataSetProvider. The DataSetProvider opens the query or stored procedure component, retrieves the records, closes the query or stored procedure component and supplies the records, with any required metadata, to the ClientDataSet. The ClientDataSet holds the records in memory while they are viewed or modified. As records are added, deleted or updated, either in code or via the user interface, the ClientDataSet logs all changes in memory.

To update the database you call the ClientDataSet's ApplyUpdates method. ApplyUpdates transmits the change log to the DataSetProvider. The provider starts a transaction then creates and executes SQL statements to apply the changes to the database. If all changes are applied successfully the provider commits the transaction; if not it rolls the transaction back. Database updates may fail if, for example, a change violates a business rule enforced by a trigger or if another user has changed a record you need to update since you read the record. If an error occurs the transaction is rolled back and the ClientDataSet's OnReconcileError event is fired giving you control of how the error is handled.

## Benefits of Provide/Resolve Architecture

### Short Transaction Life

Long transactions force the database server to hold locks, which reduces concurrency and consumes database server resources. With provide/resolve architecture transactions exist for a moment when records are read and again when updates are applied. This dramatically reduces resource consumption and improves concurrency on a busy database server.

### Make Any Rows Editable

Rows returned by multi-table joins, stored procedures or read-only views cannot be edited directly in database applications. The DataSetProvider gives you three tools to handle these situations. If the records include fields from a single table, for example records returned by a stored procedure, the only problem is that the DataSetProvider has no way to discover the name of the table. The solution is to create an OnGetTableName event handler for the DataSetProvider which returns the name of the table.

A second possibility is a multi-table join where fields from a single table must be updated. First, set the ProviderFlags property of the individual fields to identify the fields that should be updated. Then create an

OnGetTableName event handler to return the table name, and the provider will generate the SQL statements automatically.

If you need to update multiple tables for each record add a BeforeUpdateRecord event handler to the DataSetProvider. In the event handler you can generate the SQL statements for each table and execute them.

**Instantaneous Sorting and Searching**

Since the ClientDataSet holds records in memory, they can be sorted quickly. If an in-memory sort is too slow you can create indexes on the ClientDataSet's data at design time or runtime. These in-memory indexes let you change the viewing order of records or locate records virtually instantaneously without the overhead of maintaining indexes in the database.

**Automatic Summary Information**

ClientDataSets will automatically maintain complex summary calculations you define such as Sum(Price) – Sum(Cost). You can group summary calculations by any field or combination of fields to provide group totals. You can also use the Min, Max, Count and Avg (average) aggregates.

**View Subsets of Data**

Filter expressions using SQL WHERE syntax let you easily display a subset of the record in a ClientDataSet without the overhead of executing another query on the database server.

**Multiple Simultaneous Views of Data**

The ability to clone a ClientDataSet's cursor lets you view different subsets of the data in a ClientDataSet simultaneously. You can also view the same data sorted differently.

**Calculated Fields With No Overhead**

You can add calculated fields to a ClientDataSet at design time to make computed fields part of the in-memory dataset. Since the calculations are performed using compiled Object Pascal code, they are fast and can be far more complex than computed columns in a SQL statement or the calculations possible in triggers, yet they impose no storage or computational burden on the database server.

## The Limitation That Isn't There

Holding records in memory may seem like a limitation on the number of records you can work with. However, consider that traditional client/server application design has always been to select small sets of records to minimize network traffic and database server load. Even if you need to work with an unusually large number of records, remember that 10,000 records containing 100 bytes each requires only one megabyte of memory. In the unlikely event that you need to work with a very large number of records, the ClientDataSet and DataSetProvider include properties and events that let you fetch a portion of the records, edit them, remove them from memory, and then fetch the next group of records.

## Easy Deployment

An application using dbExpress requires just two shared object libraries to function. The first is the dbExpress driver, for example LIBSQLIB.SO in the case of Interbase®, and the second is LIBMIDAS.SO, the ClientDataSet support library. This minimizes application size and simplifies installation.

## Easy Driver Creation

dbExpress drivers must implement just five interfaces that are described in the on-line help. Borland also provides the source code for the MySQL™ and Interbase drivers as a model. This makes it easy for database vendors to create robust high performance drivers.  You can event create your own if you are working with an unusual or legacy database for which no driver is available.

## DataCLX™

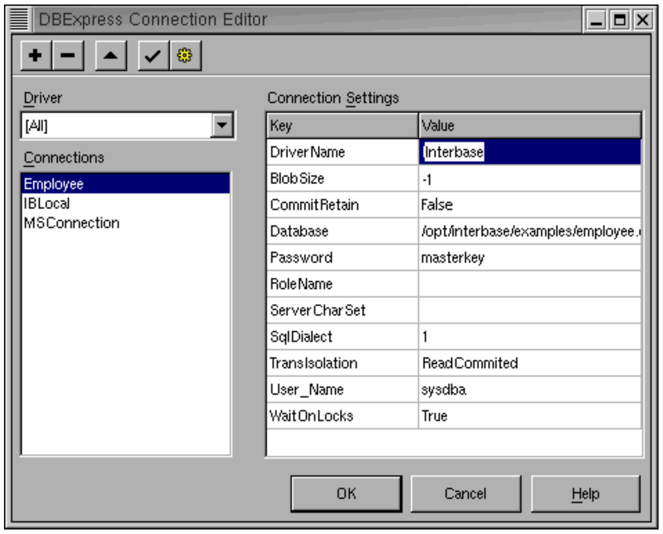
The cross-platform component library, CLX, includes two groups of components that provide access to data. The dbExpress components provide the basic connection and data retrieval functions. The data access components provide the ability to edit data using the provide/resolve architecture.

### The dbExpress Components

The dbExpress components include a SQLConnection component, several dataset components and a SQLMonitor component to provide easy access to data via dbExpress. Like all CLX components the data access components let you develop applications quickly by choosing components from the component palette and dropping them into your application.

#### SQLConnection

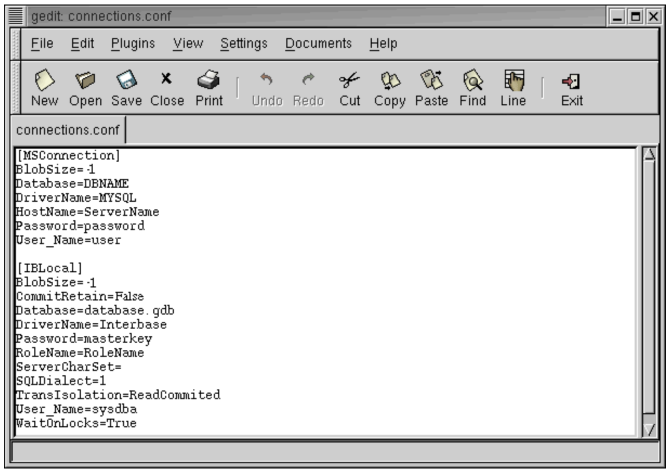
The SQLConnection component provides a database connection for any number of dataset components. You can use multiple SQLConnection components to connect to many databases simultaneously. There are three ways to define a connection to a database. You can use an existing named connection, create a new named connection or put the connection parameters in the Params property of the SQLConnection component. To use an existing named connection just set the ConnectionName property.



| Key | Value |
|---|---|
| DriverName | Interbase |
| BlobSize | -1 |
| CommitRetain | False |
| Database | /opt/interbase/examples/employee. |
| Password | masterkey |
| RoleName | |
| ServerCharSet | |
| SqlDialect | 1 |
| TransIsolation | ReadCommited |
| User_Name | sysdba |
| WaitOnLocks | True |

*The dbExpress Connection Editor*

To create a new named connection double click the SQLConnection component to open the dbExpress Connection Editor. The Connection Name list box on the left shows any connections that have already been defined. The Driver drop-down list lets you filter the Connection Names to show only the connections for the driver you select. The Connection Settings grid on the right shows the connection settings for the selected connection. All of the connections you create are stored in the dbxconnections.conf file. The accompanying screen shows the connections file with entries for a MySQL connection and an Interbase connection.


gedit: connections.conf
File Edit Plugins View Settings Documents Help
New Open Save Close Print Undo Redo Cut Copy Paste Find Line Exit
connections.conf

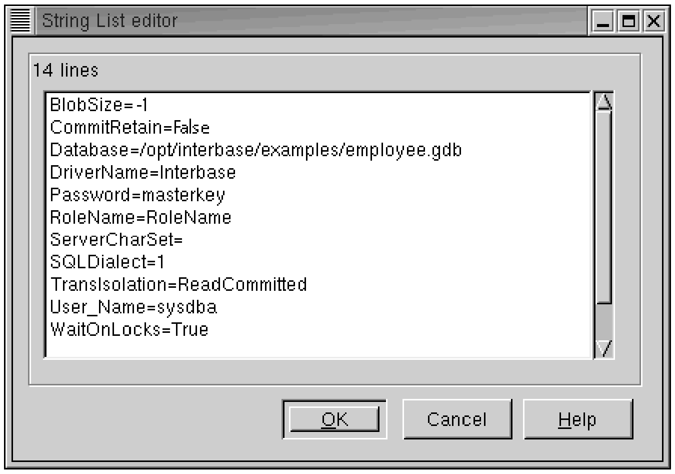
```
[MSConnection]
BlobSize=-1
Database=DBNAME
DriverName=MYSQL
HostName=ServerName
Password=password
User_Name=user

[IBLocal]
BlobSize=-1
CommitRetain=False
Database=database.gdb
DriverName=Interbase
Password=masterkey
RoleName=RoleName
ServerCharSet=
SQLDialect=1
TransIsolation=ReadCommited
User_Name=sysdba
WaitOnLocks=True
```

*The connections file*

After creating a named connection you can assign it to the SQLConnection component's ConnectionName property. If you use named connections, you will have to distribute a connections file with your application or locate an exiting connections file on the target computer and add your connection to it.

An alternative solution is to begin by setting the DriverName property of the SQLConnection component. The drop-down list for the DriverName property lists all of the drivers installed on your system. The driver information is contained in the dbxdrivers.conf file. Setting the DriverName property will also set the LibraryName and VendorLib properties using information from the drivers file. LibraryName contains the name of the dbExpress driver shared object file and VendorLib contains the name of the database vendor's client library file.



```
BlobSize=-1
CommitRetain=False
Database=/opt/interbase/examples/employee.gdb
DriverName=Interbase
Password=masterkey
RoleName=RoleName
ServerCharSet=
SQLDialect=1
TransIsolation=ReadCommitted
User_Name=sysdba
WaitOnLocks=True
```

*The SQLConnection component's Params property*

Enter the connection parameters from the Connections Editor in the SQLConnection component's Params property as shown in the preceding screen. Using this method the connection information is contained entirely within your application. If you want application users to be able to change any connection parameters you can store them in the application's own configuration file and provide a way to edit the file either within your application or with a separate configuration program. This makes each application totally self-contained.

The SQLConnection component also provides the StartTransaction, Commit and Rollback methods for explicit transaction control. If you need to execute SQL statements that do not return a result set, you can do so through the SQLConnection component using the Execute or ExecuteDirect methods. No dataset component is required. If you need access to the metadata of the database you are working with, SQLConnection provides the GetTableNames, GetFieldNames and GetIndexNames methods.

**The Dataset Components**

dbExpress provides four dataset components; SQLDataSet, SQLQuery, SQLStoredProc and SQLTable. SQLDataSet is the component of choice for any new application you write. By setting its CommandType property you can use it to execute SQL statements, call a stored procedure or access all of the rows and columns in a table. The other dataset components are provided mainly to make it easy to convert Windows® applications that use the Borland Database Engine to dbExpress.

To use a SQLDataSet set its SQLConnection property to the SQLConnection component you want to use. Next, set the CommandType property to ctQuery, ctStoredProc or ctTable. Most often you will use the default value of ctQuery. The value of the CommandText property depends on the value of CommandType. If CommandType is ctQuery, CommandText contains the SQL statement you want to execute. If CommandType is ctStoredProc, CommandText is the name of the stored procedure. If CommandType is ctTable, CommandText is the name of the table. You use the Params property to supply parameters for a parameterized query or a stored procedure and the DataSource property to link the SQLDataSet to another dataset component in a master-detail relationship.

The SQLDataSet provides a read only unidirectional cursor only. If this is the only access you need, for example for printing a report, you can use the SQLDataSet by itself or with a DataSource component depending on the requirements of your reporting tool. If you need to scroll back and forth through the records or edit the data, just add a DataSetProvider and ClientDataSet as described later in this paper.

If you need more detailed metadata information than the SQLConnection methods provide, use the SetSchemaInfo method of a SQLDataSet component. SetSchemaInfo takes three parameters, SchemaType, SchemaObject and SchemaPattern. SchemaType can be either stNone, stTables, stSysTables, stProcedures, stColumns, stProcedureParams or stIndexes. This parameter indicates the type of information that the SQLDataSet will contain when it is opened. The schema type is set to stNone when you are retrieving data from a table using a SQL statement or stored procedure. Each of the other schema types creates a dataset with a structure appropriate for the information being returned. SchemaObject is the name of the stored procedure or table when a stored procedure or table name is required. Schema pattern lets you provide a SQL pattern that will filter the result set. For example, if SchemaType is stTables and SchemaPattern is 'EMP%' the dataset will only contain tables that start with EMP.

**SQLMonitor**
The last dbExpress component, the SQLMonitor, is provided to help you debug your application. SQLMonitor monitors all of the SQL statements passed between a SQLConnection component and the database server it is connected to. The SQL statements can be logged to a file or you can write event handlers to process them in any way you wish.

## The Data Access Components.

**DataSetProvider**
A DataSetProvider is linked to one of the dbExpress dataset components through its DataSet property. The DataSetProvider supplies data to the ClientDataSet on request and generates the SQL DML statements to update the database from the change log supplied by the ClientDataSet.

**ClientDataSet**
The ClientDataSet is connected to a DataSetProvider via its ProviderName property. It receives data from its DataSetProvider, buffers the data in memory, logs all changes to the data and sends the change log to the DataSetProvider when the ClientDataSet's ApplyUpdates method is called.

**DataSource**
A DataSource component is connected to a dataset via its DataSet property. The dataset may be a ClientDataSet or one of the dbExpress dataset components. The DataSource provides common functionality and a connection point for the data aware user interface components used to display and edit data interactively. It is also used to link dataset components to model one-to-many or one-to-one relationships.

**SQLClientDataSet**
This component is a ClientDataSet with a built-in DataSetProvider and SQLDataSet. Just connect it to an SQLConnection component; set the DataSet, CommandType and CommandText properties and you are ready to go. While SQLClientDataSet hides some of the less commonly used features of the SQLDataSet and DataSetProvider components it saves development time and provides all of the functionality you normally need for accessing data from a single table.

## MyBase™ – The Local XML Database

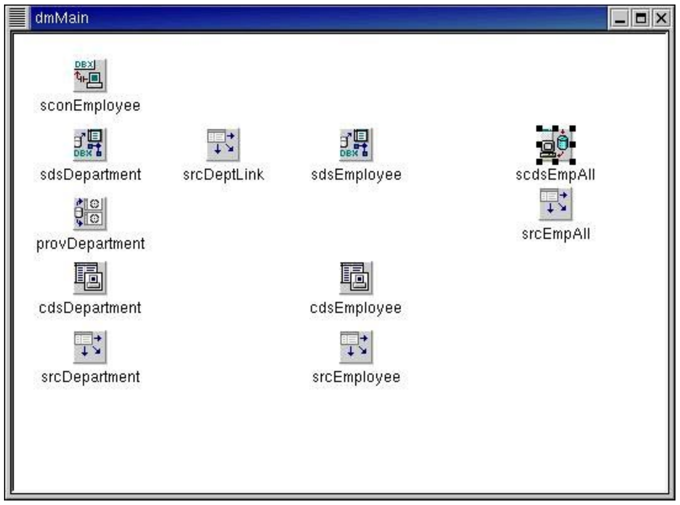
The data contained in a ClientDataSet can be saved to and loaded from a disk file in either binary or XML format. This allows a ClientDataSet to function as a single user relational database system. The only limitation is that the data must fit in memory while it is being accessed.

This capability can be used in many ways. It lets you build briefcase model applications where a user with a notebook computer selects data from the database server and saves it locally. The user then disconnects from the network; inserts, deletes and updates records; saves the data and change log; reconnects to the network and applies the updates to the database. You can also use this capability to import data from, or export data to, XML. Finally, you can use a ClientDataSet as a high performance temporary in-memory table that can be created and destroyed on-the-fly.

## Building a Database Application

The real power of dbExpress and DataCLX™ is the speed with which you can build complex database applications. Consider a simple application that lets users see and edit a one-to-many relationship between departments and employees and see a list of all employees in descending order by job grade. The following screen shows the data module that contains the required components. The following paragraphs describe the steps you would take to build this application.

Start with a new Kylix project then choose File | New from the menu and double click Data Module in the Object Repository to add a data module to your project.



*The sample data module*

### The Database Connection

After dropping a SQLConnection component in the upper left corner of the data module, right click and choose Edit Connection Properties. Add a new connection, choose the driver for your database, set the path to the database and set any other properties you need to change from their default values. Set the Name property to sconEmployee.

### Add the DataSet Components

Add two SQLDataSet components to the data module. Set the SQLConnection property of both to sconEmployee. Set the Name property of the first to sdsDepatement and its SQL property to SELECT * FROM DEPARTMENT. Add a DataSource component, set it Name to srcDeptLink and set its DataSet property to sdsDepartment to connect it to the sdsDepartment SQLDataSet. Select the second SQLDataSet, set its name to sdsEmployee and set its SQL property to SELECT * FROM EMPLOYEE WHERE DEPT_NO = :DEPT_NO. Next, set its DataSource property to srcDeptLink. This tells the sdsEmployee dataset to get the value of the DEPT_NO parameter from the DEPT_NO field of the current record in sdsDepartment.

Add a single DataSetProvider component, set its name to provDepartment and its DataSet to sdsDepartment. Add two ClientDataSets. Name the first cdsDepartment and the second cdsEmployee. Select cdsDepartment and set its ProviderName to provDepartment. Right click cdsDepartment, choose Fields Editor, then right click in the Fields Editor and choose Add All Fields. Note that the last field is named sdsEmployee. It is a nested dataset field and it contains the employee records for the department record. Now select cdsEmployee and set its DataSetField property to sdsEmployee so it will displsy the data from the sdsEmployee field.

Next, add two DataSource components to the data module and name them srcDepartment and srcEmployee. Set the DataSet property of srcDepartment to cdsDepartment and the DataSet property of srcEmployee to cdsEmployee. All of the components for the one-to-many relationship between Department and Employee are now in place.

## Viewing All Employees

The next step is to add the components that will provide a view of all employee records. Drop a SQLClientDataSet on the data module and set its Name property to scdsEmpAll, its DBConnection property to sconEmployee and its CommandText property to SELECT * FROM EMPLOYEE ORDER BY JOB_GRADE DESC. Since users will not be able to edit this view of the data set the ReadOnly property to true.

Add a DataSource component, name it srcEmpAll and set its DataSet property to scdsEmpAll. That's all there is to it. Since this dataset is not linked to another you can take advantage of the SQLClientDataSet which combines a SQLDataSet, DataSetProvider and ClientDataSet in a single component.

## Handling Errors

The easiest way to handle database update errors is to use the Reconcile Error Dialog from the Object Repository. Choose  File | New from the menu, click the Dialogs tab then double click the Reconcile Error Dialog to add it to your project. Click the Save button on the toolbar and save the new unit file as RecErrF.pas.

## Adding the Code

Click an open area of the data module then click the Events tab of the Object Inspector™. Double click the edit box for the OnCreate event to create the shell of the event handler procedure and add the following three lines of code to open the three ClientDataSets:

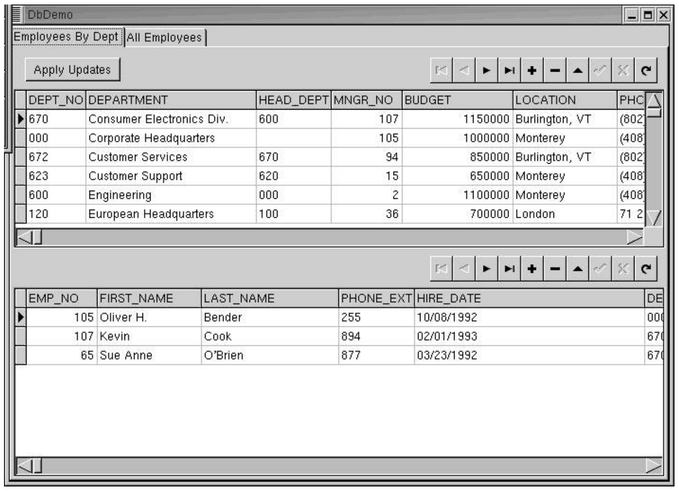
```
cdsDepartment.Open;
cdsEmployee.Open;
scdsEmpAll.Open;
```

Choose File | Use Unit  from the menu and select the RecErrF unit. This lets you call the Reconcile Error Dialog's methods. Select the cdsDepartment ClientDataSet and double client the edit area of the OnReconcileError event. Add the following line to the event handler procedure:

```
Action := HandleReconcileError(DataSet, UpdateKind,
E);
```

Repeat this process for the cdsEmployee ClientDataSet. This code will call the Reconcile Error Dialog if an error occurs when applying updates to the database.

## Building the User Interface



| DEPT_NO | DEPARTMENT | HEAD_DEPT | MNGR_NO | BUDGET | LOCATION | PHC |
|---|---|---|---|---|---|---|
| 670 | Consumer Electronics Div. | 600 | 107 | 1150000 | Burlington, VT | (802 |
| 000 | Corporate Headquarters | | 105 | 1000000 | Monterey | (408 |
| 672 | Customer Services | 670 | 94 | 850000 | Burlington, VT | (802 |
| 623 | Customer Support | 620 | 15 | 650000 | Monterey | (408 |
| 600 | Engineering | 000 | 2 | 1100000 | Monterey | (408 |
| 120 | European Headquarters | 100 | 36 | 700000 | London | 71 2 |

| EMP_NO | FIRST_NAME | LAST_NAME | PHONE_EXT | HIRE_DATE | DE |
|---|---|---|---|---|---|
| 105 | Oliver H. | Bender | 255 | 10/08/1992 | 00 |
| 107 | Kevin | Cook | 894 | 02/01/1993 | 67 |
| 65 | Sue Anne | O'Brien | 877 | 03/23/1992 | 67 |

*The main form*

To build the user interface shown above begin by dropping a PageControl on the form and setting its Align property to alClient. Right click the PageControl and choose New Page twice to add two TabSheets to the PageControl. Select the first TabSheet and set its Caption property to Employees By Dept. Select the second and set its Caption to All Employees.

Return to the Employees By Dept TabSheet and add a Button, a DBNavigator, a DBGrid, another DBNavigator and a second DBGrid. Choose File | Use Unit from the grid and select the data module. Click the first DBNavigator to select it then shift+click the first DBGrid so both are selected. In the Object Inspector set the DataSource properties to srcDepartment. Select the second DBNavigator and DBGrid and set their DataSource properties to srcEmployee.

## Adding the Apply Updates Code

Select the Button and set its Caption property to Apply Updates. Double click the button to create its OnClick event handler and add the following code.

```
with dmMain.cdsDepartment do
begin
  if ChangeCount > 0 then
    ApplyUpdates(0);
end;
```

This code checks the ClientDataSet's ChangeCount property to see if there are any changes to apply to the database. If there are, the ClientDataSet's ApplyUpdates method is called. The parameter indicates the number of errors that are allowed before the update process is halted.

It is very important to note that you do not need to call the ApplyUpdates method of cdsEmployee. Since the employee records are embedded in the department ClientDataSet, its DataSetProvider handles updates to both tables. It also handles referential integrity correctly by applying inserts, first to the department table and second to the employee table, while applying deletes, first to the employee table and second to the department table.

## What Does It Mean?

The significant thing about this application is not what it does but rather that a moderately experienced Kylix developer can build a complete client/server database application in less than 30 minutes. This is a dramatic increase in programmer productivity for Linux®.

Bill Todd is President of The Database Group, Inc., a database consulting and development firm based near Phoenix. He is co-author of four database programming books, the author of over 80 articles, and is a member of Team Borland, a developer group that provides technical support on the Borland Internet newsgroups. He is a frequent speaker at Borland Developer Conferences in the U.S. and Europe. Bill is also a nationally known trainer and has taught Delphi programming classes across the country and overseas. He can be reached at billtodd_az@qwest.net

Borland®
100 Enterprise Way
Scotts Valley, CA 95066-3249
www.borland.com | 831-431-1000