SYBASE iAnywhere

# Advantage TDataSet Descendant for Delphi/C++Builder

## Getting Started Guide

**TABLE OF CONTENTS**



**ADVANTAGE TECHNICAL SUPPORT (U.S. AND CANADA ONLY)**

iAnywhere delivers accurate and timely technical assistance to help you develop and deploy world-class solutions, quickly and cost-effectively. Electronic Support includes EBFs, Technical Documents, Product Manuals, Patches, and Newsgroups. Plus, if you are looking for a support plan, standard annual plans can include product updates, multiple customer contacts and an unlimited number of cases, while premium plans can also include extended hours, increased response times and Direct-to-Expert support.

**ADVANTAGE DEVELOPER ZONE**

Try our Advantage Developer Zone Web site for quick and easy access to the information you need. You can find product information, code samples and technical support notes to assist you.

To access the Advantage Developer Zone, go to: **http://DevZone.AdvantageDatabase.com**.

**ADVANTAGE TECHNICAL SUPPORT (OUTSIDE THE U.S. AND CANADA)**

For technical support outside the U.S. and Canada, contact your Advantage Dealer/Distributor in the country where you purchased the product. For a list of worldwide Advantage distributors, visit the Advantage Web site at **www.AdvantageDatabase.com.**

## ADVANTAGE TDATASET DESCENDANT DOCUMENTATION

### Complete Documentation Source—ADE.HLP or ade.htm

This guide is intended to provide an architecture overview of the Advantage TDataSet Descendant and to assist the developer, new to Advantage, in getting started using the product. For complete documentation about the Advantage TDataSet Descendant, see the Windows Help documentation (ADE.HLP) or Linux WebHelp (ade.htm) installed with the Advantage TDataSet Descendant.

### Database Knowledge Assumed in This Guide

The concepts, terminology, and examples found in this guide were developed based on the assumption that the reader has a basic understanding of database theory. This includes concepts such as databases, tables, and indexes. It is also assumed that the reader has developed a database application. For information on all these topics in relation to Delphi, see the Developing Database Applications sections as well as the sample database applications included with your Delphi development kit.

## CHAPTER 1 – INTRODUCTION

### Advantage TDataSet Descendant for DelphiC++Builder

The Advantage TDataSet Descendant is a programming tool developed specifically to eliminate the need for the BDE (Borland Database Engine) and dbExpress for database access. Using the Advantage TDataSet Descendant, developers can program as they always have using standard TTable, TQuery, and TStoredProc methods and properties. The Advantage TDataSet Descendant ties in seamlessly with the Advantage Database Server, a true client/server solution that adds performance and stability to multi-user applications. The Advantage TDataSet Descendant also provides access to the Advantage Local Server, a royalty-free local and peer-to-peer database engine-perfect for developing on a local workstation and ideal for customers who may later want the enhanced performance and security capabilities of a client/server RDBMS with the Advantage Database Server. The Advantage TDataSet Descendant consists of a set of native components that provide developers with easy access to the Advantage Database Server and Advantage Local Server. The Advantage Delphi/C++Builder TDataSet component does not require the BDE. The Advantage TDataSet Descendant uses the Advantage Client Engine directly, instead of the BDE or dbExpress, to retrieve data and provide client/server access to the Advantage Database Server or peer-to-peer access via the Advantage Local Server.

### System Requirements and Specifications

***Server Operating Systems (via the Advantage Database Server)***

- Novell NetWare 5.x or greater (IP, IPX)
- Microsoft Windows 2000/XP/2003/Vista/2008 (IP, IPX, IPC)
- Linux (glibc 2.3.2 or greater and kernel 2.4 and greater) (IP, IPC)

***Client Operating Systems***

- Windows 2000 or greater

***Supported File Formats***

- Advantage proprietary database (ADT tables, ADI index files, ADM memo files)
- FoxPro-compatible (DBF tables, CDX index files, FPT memo files)
- CA-Clipper compatible (DBF tables, NTX index files, DBT memo files)
- Visual FoxPro 9 compatible (DBF tables, CDX index files, FPT memo files)

***Delphi/C++Builder versions supported***

- Delphi 6, 7, 2006, 2007, and 2009
- C++Builder 5 and newer

## CHAPTER 2 – INSTALLATION

This chapter addresses how to install the Advantage TDataSet Descendant solution and contains information about using third-party components with Advantage.

After installing the Advantage TDataSet Descendant, it is recommended you install the Advantage Data Architect (ARC) utility to help with data creation and management tasks. ARC can be found as a separate installation on your Advantage product CD, or can be downloaded from the Advantage Web site (http://DevZone.AdvantageDatabase.com). For more information about Advantage Data Architect, see Chapter 5.

### Installing the Advantage TDataSet Descendant

- **To install the Advantage TDataSet Descendant on your computer**
  1. If you have an Advantage product CD, insert it in the CD-ROM drive. From the list of Advantage Client products, select **Advantage TDataSet Descendant**.
  2. If you have downloaded the **Advantage TDataSet Descendant** self-installing executable from the Advantage Web site, run that self-installing executable.
  3. The setup program prompts you through the installation process. Follow the instructions on the screen.
- **After the Advantage TDataSet Descendant has been installed**
  1. Verify the Advantage TDataSet Descendant was properly installed. When Delphi/C++Builder is started for the first time after the installation of the components, a new tab labeled **Advantage** should be visible in the components palette. If the **Advantage** tab is not visible in the components palette, perform the following steps.
     a. Add the Advantage components manually by selecting the **Components** menu item and choosing **Install Packages**. Clicking the **Add** button will bring up a dialog window looking for Borland Package Library (*.BPL) files. By default, the .BPL file is installed into the c:\Program Files\ Advantage 9.1\TDataSet\Delphix or c:\ Program Files\ Advantage 9.1\TDataSet\CBlderx, where x is the version of Delphi or C++Builder installed.
     b. Select this file and click **OK**. The Advantage package will be installed immediately into the Codegear components palette. Selecting **OK** again will complete the install of the package.
     c. To add the Advantage Components to the environment path, select the **Tools** menu item and choose **Environment Options**. Select the **Library** tab and enter the name of the directory where the Advantage Components are installed.
  2. It is recommended you install the Advantage Data Architect (ARC) utility to help with data creation and management tasks. ARC can be found as a separate installation on your Advantage product CD, or it can be downloaded from the Advantage web site (**http://DevZone.AdvantageDatabase.com**).

### Using Third-Party Components with Advantage

The Advantage TDataSet Descendant is compatible with all data-aware components that ship with Delphi/ C++Builder.

For third-party components to work with the Advantage TDataSet Descendant, the component must be engineered to work with a generic TDataSet descendant or standard TDataSource class. This means that the component must not typecast the dataset to a TTable or any other descendant other than TAdsDataSet, TAdsTable, TAdsQuery, or TAdsStoredProc. It also means that the component must not make any direct BDE API function calls (for example DbiGetRecordCount).

## CHAPTER 3 – ADVANTAGE TDATASET DESCENDANT ARCHITECTURE

The TDataSet component in Delphi 3 or greater, and C++Builder 3 or greater, is an abstract class that does not access the BDE nor dbExpress directly. Instead, this class relies on descendants to provide concrete methods. For example, TTable is a concrete descendant. TTable completely encapsulates access to the BDE. Advantage's TAdsTable component is also a concrete descendant, but does not require the BDE.

### Advantage TDataSet Descendant for Delphi and C++Builder

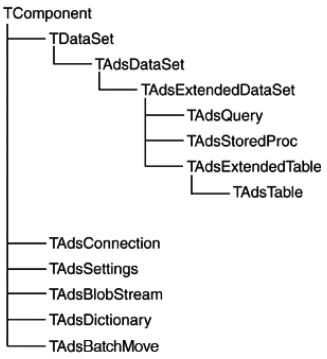
The Advantage TDataSet Descendant solution provides four key components, TAdsQuery, TAdsTable, TAdsStoredProc, and TAdsConnection. TAdsQuery, TAdsTable, TAdsStoredProc, and TAdsConnection are TDataSet descendants similar to Delphi's TQuery, TTable, TStoredProc, and TDatabase components. Because these components are similar, you can use TAdsQuery, TAdsTable, TAdsStoredProc, and TAdsConnection exactly as you would TQuery, TTable, TStoredProc, and TDatabase. In fact, your application will have to change very little except it will now have the benefit of using Advantage technology. Application deployment can be as simple as including the required Advantage DLLs in your application directory. The architecture of TAdsQuery, TAdsTable, TAdsStoredProc, and TAdsConnection allows the use of Delphi native data aware components plus any third-party components that are TDataSet descendant compatible.

The Advantage TDataSet Descendant solution also provides components such as the TAdsSettings component for environmental settings such as date format, and the TAdsDictionary component to provide administrative access to Advantage Data Dictionaries.

*Note: The Advantage TDataSet Descendant does not need a component equivalent to TSession. It is simply not required. The Advantage table and query components used in your applications can be referenced or created in the TThread.Execute body.*

An alias configuration utility is included in a stand-alone utility called the Advantage Data Architect (ARC). This functionality in ARC allows you to set up aliases similar to Delphi database aliases. The ARC utility is a separate install that is located on the Advantage Product CD and also available for download from the Advantage web site (http://DevZone.AdvantageDatabase.com).

**Advantage TDataSet Descendant Hierarchy**




#### *TAdsTable*

The Advantage TAdsTable component is equivalent to the Delphi TTable component. The Advantage TDataSet Descendant TAdsTable component provides most of the same methods, properties, and events that TTable provides. TAdsTable provides direct access to every record and field in an underlying table and provides access to extended Advantage functionality that is not available with TTable.

The TAdsTable component looks and behaves like the TTable component. One specific difference, however, is the TAdsTable.TableType property must be set to ttAdsADT, ttAdsVFP, ttAdsCDX, or ttAdsNTX for free connections (that is, Advantage connections not associated with an Advantage Data Dictionary). The TAdsTable.TableType property is ignored when using a TAdsConnection component that references an Advantage Data Dictionary. In this situation, the table type stored in the data dictionary is automatically used.

The TAdsTable contains many methods that are classified as "extended methods". Each method name is prefaced with "Ads". These are wrappers of most Advantage Client Engine API's. Often, these methods closely match a method from the TTable equivalent methods. For example, the TAdsTable.AdsSkip method and the TAdsTable.Next method provide the same functionality. The TAdsTable.AdsSkip method is merely a wrapper for the Advantage Client Engine API named AdsSkip, which sets the current table position to the next subsequent row. The TAdsTable.Next component is identical to standard Delphi TTable.Next method. They both set the current table position to the next row. In many cases, the TTable equivalent function (TAdsTable.Next in this case), is more efficient. For every extended method that is not as efficient as the TTable equivalent function, there exists a special warning in the method's documentation. For those methods, the code necessary to call the more efficient TTable equivalent function is mentioned, so that you can you use it instead. As a general rule, do not use Advantage TAdsTable extended methods unless absolutely necessary (such as when no Delphi equivalent method exists).

***TAdsQuery***

The TAdsQuery component allows developers to submit queries and other SQL commands just as they would with the TQuery component. The TAdsQuery component bypasses the BDE and uses Advantage SQL technology, which is built into the Advantage Database Server and Advantage Local Server to provide the ability to execute SQL statements, thereby adding powerful capabilities to a Delphi/C++Builder application. For a full description of the SQL functionality supported by Advantage, refer to the Advantage SQL book located in your Advantage Help file (Advantage.hlp).

The Advantage SQL engine documentation provides descriptions and examples of the supported statement types as well as providing optimization details and descriptions of the different cursor types. In general, the Advantage SQL engine is a subset of SQL-92 with extensions. When using the Advantage Database Server, the server performs all the processing of the SQL statements. If you are using Advantage Local Server, the processing of SQL statements occurs at the client workstation, but the ability to manipulate your data with SQL statements is still very useful and powerful. If you use Advantage Database Server, the Advantage SQL engine can provide even more power because the SQL statement is executed "closer" to the data. For example, an UPDATE statement can update multiple records in a table with a single network request.

At the simplest level, you can use SQL SELECT statements to retrieve and filter data from a single table. For example, using TAdsQuery with the SQL statement "SELECT * FROM testdata WHERE id < 100" would effectively provide the same results as using TAdsTable with the TableName property set to "testdata" and the Filter property set to "id < 100". SQL, though, provides many additional capabilities that allow you to summarize, group, order, and join tables together.

When you use SELECT statements, the rowset will be one of two types of cursors: dynamic (live) or static. Advantage produces a live cursor when it can filter and order the base table in the query without using temporary relations. If it is not possible to do this, then Advantage produces a static cursor. The primary difference between these two cursor types is that live cursors can be edited, while static cursors are read-only. A more detailed description of the cursor types is provided in the Advantage SQL book located in your Advantage Help documentation (Advantage.hlp or advantage.htm). In general, however, a static cursor is produced any time a SELECT statement has more than one table or does any summarization of the data.

An application can also use the TAdsQuery component to modify data in tables through UPDATE, DELETE, and INSERT statements. UPDATE statements can be used to modify one or more records in a table; DELETE statements can be used to delete one or more records in a table; and INSERT statements can be used to insert records into an existing table. CREATE and DROP statements are also available for creating and deleting tables and indexes.

The TAdsQuery component is intended to supplement TAdsTable rather than replace it. The two components can be used together very effectively in applications. As you develop your application, you should evaluate each component and choose the best one (or both) on a case-by-case basis.

Like the TAdsTable component, TAdsQuery has a source table type property that must be set to either ttAdsADT, ttAdsVFP or ttAdsCDX prior to executing the SQL query when dealing with a free connection. The ttAdsNTX source table type property is not supported with TAdsQuery on a free connection. The source table type property is ignored when the TAdsQuery instance is associated with a database connection. With a database connection, the table type is stored in the data dictionary and is automatically used. For example, if a database has been created and defined (in a data dictionary), and that database contains DBF tables and the associated NTX index files, those NTX index files will then get automatically used and updated by the TAdsQuery SQL statements.

The TAdsQuery component also contains nearly all of the TAdsTable functionality as well as several of the Advantage Extended Methods.

In addition to run simple SQL statements you can also pass in a complete SQL script. ADS supports a subset of ANSI SQL 2003 PSM (persistent stored modules) not only for Stored Procedures, Triggers and User Defined Functions, but also anywhere where you can pass in a SQL statement to ADS. If the last statement in the script returns a cursor (e.g. a SELECT statement or an EXECUTE PROCEDURE statement on a Stored Procedure with output parameters), the cursor is returned to the TAdsQuery component. Following example creates a temporary table if it doesn't exist and displays it in a TDBGrid.

```
AdsQuery1.Close;
AdsQuery1.SQL.Clear;
AdsQuery1.SQL.Add('TRY');
AdsQuery1.SQL.Add('  CREATE TABLE #mytable(id AUTOINC, text CICHAR(30));');
AdsQuery1.SQL.Add('CATCH ALL');
AdsQuery1.SQL.Add('END TRY;');
AdsQuery1.SQL.Add('SELECT * FROM #mytable');
AdsQuery1.Open;
DataSource1.DataSet:=AdsQuery1;
DBGrid1.DataSource:=DataSource1;
```

***TAdsStoredProc***

Use the TAdsStoredProc component to simplify the execution of Stored Procedures and Advantage Extended Procedures and the management of input and output parameters.

**Stored Procedures**

Stored Procedures allow you to execute a set of code at the server where the data resides. This allows you to remove data intensive tasks from the workstations and reduces your network traffic to a single send and receive operation. Advantage supports a subset of ANSI SQL 2003 PSM to write your own set of Stored Procedures.

**Advantage Extended Procedures**

Advantage Extended Procedures are stored procedures that are easy to develop and easy to use. Unlike traditional stored procedures, however, Advantage Extended Procedures allow developers to write, store, and execute stored procedures on the server using their preferred application development tool. No database administrator is required to develop Advantage Extended Procedures.

***TAdsConnection***

TAdsConnection is a component that is similar to the Delphi TDatabase component. TAdsConnection defines a connection to the Advantage Database Server. This component is used primarily for connection management and performing connection-specific functionality such as transaction processing.

To use the TAdsConnection component, create a new instance by placing a TAdsConnection component on the form. Then, modify every TAdsTable and TAdsQuery component to be associated with this connection by setting its AdsConnection property to reference this new TAdsConnection component. You may create multiple TAdsConnection components to force multiple server connections. This is valuable when you want to connect to two different servers within your application or if you need to have some tables/cursors open via one Advantage server type and others open via a different Advantage server type. Using one TAdsConnection component per thread also allows for the highest level of concurrency in a multi-threaded Advantage application.

When the TAdsConnection.BeginTransaction method is called, every TAdsTable, TAdsQuery and TAdsStoredProc associated with the connection component will enter the transaction. Since more than one connection component may be used, multiple independent transactions can be processed concurrently.

*Note: Advantage Database Server licensing, unlike many DBMS products, is based on concurrent users, not concurrent connections. Therefore, using two connection components to the same server only uses one license.*

***TAdsSettings***

The TAdsSettings component exposes Advantage-oriented, application-wide settings. The properties associated with this component will affect the behavior of every Advantage component in the entire application.

***TAdsDictionary***

The TAdsDictionary component is used to gain administrative access to an Advantage Data Dictionary. TAdsDictionary gives you the ability to add and delete tables, set table, and record level constraints, construct Referential Integrity, create views, and get all available information about the data dictionary. The majority of the functionality this component provides can be retrieved in an easier fashion through the use of the system tables (system.dictionary, system.tables, etc.) and canned stored procedures (sp_ModifyDatabase, sp_ModifyTableProperty, etc).

***TAdsBlobStream***

TAdsBlobStream is similar to Delphi's TBlobStream. Because of implementation details, the new component with a different name was created.

Use TAdsBlobStream to access or modify the value of a Blob field object. Blob field objects are TBlobField objects and descendants of TBlobField such as TGraphicField and TMemoField. Blob fields use Blob streams to implement many of their data access properties and methods.

TAdsBlobStream allows objects that have no specialized knowledge of how data is stored in a Blob field to read or write such data by employing the uniform stream interface.

To use a Blob stream, create an instance of TAdsBlobStream, use the methods of the stream to read or write the data, and then free the Blob stream. Do not use the same instance of TAdsBlobStream to access data from more than one record. Instead, create a new TAdsBlobStream object every time you need to read or write Blob data on a new record.

You can also access binary data using the following:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   ImageField: TBlobField;
begin
   ImageField := AdsTable1.FieldByName( 'graphic' ) as TBlobField;

   //Load binary image into table.
   AdsTable1.Edit();
   ImageField.LoadFromFile( 'c:\temp\winnt.bmp' );
   AdsTable1.Post();

   //Read binary image and save to disk.
   ImageField.SaveToFile( 'c:\images\image1.bmp' );
end;
```

***Recommended Best Usage for the Advantage Native Components***

When programming with the Advantage TDataSet Descendant, our goal is for the developer to use standard Delphi TTable, TQuery, and TStoredProc programming techniques. There are extended TAdsTable methods provided as a wrapper to the Advantage Client Engine API, however these should be used only when the equivalent functionality is not available with TTable methods. For example, if when using a standard TTable you included the statement 'Table.First', when accessing an Advantage table you would use the same method, 'AdvantageTable.First'.

The Advantage Technical Services team maintains an application available from the download area of the Advantage Developer Zone Web site (http://DevZone.AdvantageDatabase.com), that covers the most common topics in Advantage/Delphi programming. The application is intended to be an example of best programming techniques and is updated on a regular basis. Some of the topics addressed by this example application are filters and how they can be used for extremely fast data manipulation, creating tables and indexes programmatically, ranges versus scopes, master/detail relationships, transaction processing, encryption, finding data via FindKey, FindNearest, GotoKey, GotoNearest, Locate and Lookup methods, how to gracefully handling record locking, and multi-threaded Advantage database applications.

**Borland Delphi Architecture—Overview**

***Database Applications***

When developing database applications using Delphi (this applies to C++Builder too), you have most likely used the Delphi data access components and the Delphi data-aware components. The data access components appear on the Data Access page of the Component Palette within Delphi. Some examples of the components are TTable, TQuery, TDataSource, and TDatabase. The data-aware components appear on the Data Controls component page. Examples of these components include DBGrid, DBEdit, and DBMemo. The components on the Data Access page encapsulate all data retrieval and manipulation. The components on the Data Controls page provide functionality to visually display and edit data such as edit boxes and grids.

***Delphi Data Access Components***

The most commonly used Data access components are TTable, TQuery, and TDataSet. The TTable and TQuery Delphi components are data providers. The TTable component is a descendant of the TDataSet component and provides all functionality associated with a database table. The database table exists on a hard drive. Program access and table manipulation is performed by the TTable component. The TQuery component is also a descendant of the TDataSet component. It encapsulates a result set based on an SQL query. All reads and writes to the database are handled by the TQuery component.

In Windows, both the TTable component and the TQuery component traditionally employ the Borland Database Engine (BDE). The BDE is the native server for many Borland products such as dBASE, Delphi, and C++Builder. It is the database engine that handles all of the low-level database request routines. The TTable and TQuery components traditionally use the BDE to gain access to the actual data or data server. When using native Delphi components, the data can reside in a file, such as a dBASE style DBF file or a Paradox DB file, or can be provided by a third-party database server product such as Oracle or InterBase.

### *Delphi Data Control Components*

The set of Delphi components that provide a visual presentation of data are the data-aware components. These components appear on the Data Controls page of the Component Palette and retrieve data by making requests to a TDataSet component. Some of these commonly used components are TDBGrid, TDBMemo, TDBImage, and TDBEdit. For example, the TDBGrid component displays the data from the TDataSet in a two dimensional grid. It allows the user to visually navigate and edit the data within the grid.

When utilizing data-aware controls, your TDataSet component must be associated with a TDataSource component. The TDataSource component references the TDataSet and provides an interface to the TDataSet that data-aware components use to retrieve their data. The data-aware components are limited to using only the database functionality exposed by TDataSet through this interface. Many methods of TTable and TQuery such as TTable.ApplyRange and TQuery.ExecSQL are not accessible via this interface and are therefore never accessed by any Delphi data-aware components.

To summarize, the data-aware components such as TDBGrid and TDBEdit link to a TDataSource. The TDataSource component provides an interface to a particular TDataSet component. The TDataSet component is an abstract component that provides functionality used to access data. The TTable or TQuery components descend from the abstract TDataSet component. These components are concrete. They provide all functionality that the abstract ancestor components require. The native TTable and the TQuery components encapsulate all database functionality as far as the Delphi environment is concerned, but use the BDE to actually retrieve the data.

## Differences Between Advantage and Native Delphi Functionality

Specific differences exist between some Advantage functionality and native Delphi/C++Builder expected behaviors. In addition to the information below, the How the Advantage SQL Engine Differs from the BDE topic in the Advantage Help file (Advantage.hlp) contains specific differences in the SQL engines.

The following sections discuss the functionality in terms of properties, methods, and events that are currently not supported, and what can be done to achieve equivalent functionality using the Advantage TDataSet Descendant. In a few cases the equivalent functionality must be coded specifically by the developer.

### *Cached Updates*

Cached updates enable you to retrieve data from a database, cache and edit it locally, and then apply the cached updates to the database as a unit. When cached updates are enabled, updates to a dataset (such as posting changes or deleting records) are stored in an internal cache instead of being written directly to the dataset's underlying table. When changes are complete, your application calls a method that writes the cached changes to the database and clears the cache.

Below are the properties, methods, and events related to CachedUpdates that are NOT supported by the Advantage TDataSet Descendant:

| | |
|---|---|
| CachedUpdates | property |
| UpdateObject | property |
| UpdatesPending | property |
| UpdateRecordTypes | property |
| UpdateStatus | method |
| OnUpdateError | event |
| OnUpdateRecord | event |
| ApplyUpdates | method |
| CancelUpdates | method |
| CommitUpdates | method |
| FetchAll | method |
| RevertRecord | method |

In general, cached updates have two unique purposes. First, they provide a way for a pseudo transaction to take place by not applying the updates immediately to the database. This provides an "undo" functionality that would discard changes because they were cached local to the workstation. Second, they typically improve performance by controlling network traffic and timing the transactions to occur during non-critical times.

The Advantage Database Server provides transaction processing capability. With transaction processing, you can rollback or commit any changes that occurred inside a transaction. This will be similar to the ability provided with cached updates, yet more robust because the changed records will be locked during the transaction until a commit or rollback takes place. Another advantage is that it is a true transaction because the server is handling the commit. If anything happens during a commit, the entire transaction is rolled back. This is not the case when the cache is flushed with the cached update. If something happens during the update of the cached changes, the results will be indeterminate. Some may have been updated, and others not. For more information on the functionality of Advantage transactions see the Transaction Processing System topic in the Advantage Help documentation (Advantage.hlp or Advantage.htm).

The Advantage Database Server, by nature, performs updates very quickly. Although Advantage does not support Cached Updates, network traffic is dramatically reduced when using the Advantage Database Server due to its client/server processing. Thus, the need to use Cached Updates to improve performance is eliminated when using the Advantage Database Server.

Transaction processing is available with the Advantage Database Server, but not the Advantage Local Server. Using the Advantage Local Server would require a developer to create a caching scheme to accomplish the functionality of a cached update.

An example is available on the Advantage Developer Zone (http://devzone.advantagedatabase.com), which shows how to implement CachedUpdates and how to update static cursors. The download file is called updatestatic.exe and can be found in the Delphi Tools download area.

***Constraints***

Constraints are used to read or add record-level constraints to the dataset. Record-level constraints usually impose relationships between the fields in a single record. BDE dataset components support check constraints that define the values that can be entered into a single record. The TCheckConstraints object maintains the collection of check constraints that apply to a given dataset. This functionality is not implemented with the Advantage TDataSet Descendant. In order for an application to support an enforced constraint to a user, the developer will need to configure the constraints using an Advantage Data Dictionary. See the Advantage Data Dictionary topic in the Advantage Help file (Advantage.hlp).

***DefaultIndex***

The data dictionary can store a default index for each table. The default index does not have to be the primary key; it can be any existing index. When new table objects are created (in Delphi, OLE DB, etc.) they will automatically be configured to use the default index from the data dictionary. If the default index is changed in the dictionary at a later time, all tables will automatically begin using the new default index. For optimized performance, the default index is initially not set to reduce unnecessary index page reads.

***Filters***

The BDE supports brackets around field names in filters. For example: [Home State] = 'CA'. Advantage does not support these brackets; therefore, they must be removed.

***ObjectView***

The ObjectView property specifies whether fields are to be stored hierarchically or flattened out in the Fields property. If this property is True, there can be a hierarchy to the objects in the TFields.Fields array. If this property is False, the fields are stored sequentially in the Fields property. With Advantage, all fields are stored sequentially in the Fields property.

***SessionName***

The Advantage TDataSet Descendant does not support the concept of a session, nor does it need one. With a BDE dataset, the application's database connections, drivers, cursors, queries, and so on, are maintained within the context of one or more BDE sessions. This provides the ability to cross the multi-threaded boundaries with a BDE dataset. In Advantage, no "session" is necessary in order for all connections, tables, and index handles to be available across all threads.

***Master/Detail Relationships***

Depending on the table type being used, native Delphi functionality will sometimes allow the following code to run without raising any exceptions, even though the MasterFields property is set before an index is active:

```
table2.MasterSource := datasource1;
table2.MasterFields := 'id';
table2.IndexName := 'idind';
```

This works on certain table types (like Paradox) because they already have a default index (for Paradox tables, the 'Primary' index).

In Advantage, if an index is not already active, the code above will raise an exception, as there is no index to use in the master/detail relationship. To work around this problem, re-arrange your code as shown in the example below:

```
table2.IndexName := 'idind';
table2.MasterSource := datasource1;
table2.MasterFields := 'id';
```

***Recno***

The Recno property is affected by active indexes, scopes, filters, etc. A variety of default settings have been optimized to provide the best performance when retrieving the record number. If you find some of these settings do not meet your needs you may need to change them.

See the following properties for details:

- TAdsDataSet.Sequenced
- TAdsDataSet.SequencedLevel
- AdsTableOptions.AdsFreshRecordCount
- AdsTableOptions.AdsFilterOptions
- TAdsTable.AdsGetRecordNum

**How the Advantage SQL Engine Differs from the BDE**

The following table shows how the Advantage SQL engine differs from Borland Database Engine (BDE) Local SQL.

| SQL Construct | BDE Local SQL | Advantage SQL Engine |
|---|---|---|
| Concatenation Operator | Supported by the "\|\|" operator. Strings are automatically trimmed. | Supported by the "+" operator. Strings are not automatically trimmed. |
| SOME keyword | Supported | Not Supported |
| FROM | Support BDE alias in the FROM clause. For example: FROM ":DBDemos:Orders". | Does not support database alias in the FROM clause. |
| ORDER BY | Support ORDER BY using ASCENDING/DESCENDING keywords. | Support ORDER BY using ASC/DESC keywords. |
| Quoted Strings | Supports single and double quotes interchangeably. | Single quotes are used for character string literals. Double quotes and square brackets are used for delimiting table and field names. |
| Comparison to NULL | "field = NULL" is supported. | Supported through the standard syntax "field IS NULL". |
| Not Equal Comparison | Supports "!=" operator | Does not support "!=" operator. Only "<>" operator is supported. |
| Named parameter | Any character except white spaces. | If the name is not quoted with ", only characters A-Za-z and _ is allowed in the name. If other character is to appear in the parameter name, the name must be quoted with ". For example, :"Last+First" |
| Numeric value | Can be enclosed in quotes. For example: IntField = '2' or IntField = 3 | Must not be enclosed in quotes. For example: IntField = 2 or IntField = 3 |

## CHAPTER 4 – DEVELOPING A SAMPLE APPLICATION

For consistency, the following notation will be used in instructions to modify properties in the Delphi Object Inspector:

*Object Inspector:*

*Property - new value for property*
*Property - new value for property*
*etc...*

After adding a component in any of the instructions, press **[F11]** to activate the Delphi Object Inspector and change the specified properties. Only properties that need to be changed from their default value will be included in the property list.

### Task 1 – Convert Data Using the Advantage Data Architect

The first task in creating this sample application is to convert a paradox table to an Advantage Database Table (ADT table). It is possible to write your own Delphi application for this purpose, but you will most likely find it easier to use the Advantage Data Architect.

1. Close Delphi if it is currently running.
2. Create a new data directory to store your Advantage files. For this example we will use X:\ADS\DATA.
3. Open Advantage Data Architect.
4. Select Connection, Create New Connection from the main menu.
5. In the ConnectionPath property, enter the path you created in Step 2.
6. In the DatabaseName property, enter a name for this new connection. We will use "testdata".
7. Click the OK button.
8. Choose Import Data from the Tools menu.
9. From the Import Data Type combo box, select Paradox / dBASE.
10. In the source file name edit box, enter the path to your data, or click the browse button to search. By default, Delphi installs demo data to C:\PROGRAM FILES\COMMON FILES\BORLAND SHARED\DATA. We will convert all of the demo tables to the Advantage file format for this example, so use C:\PROGRAM FILES\COMMON FILES\BORLAND SHARED\DATA*.DB*.
    *Note: If you cannot find the data directory, reference your Delphi installation documentation for details on how to re-install with this option. There is also the possibility that you will already have the files from a previous Delphi 1 installation, in this case the default directory is C:\DELPHI\DEMOS\DATA.*
11. Click the Next button.
12. Enter the destination connection. This should be the connection created in Step 2 above. This is where all of your new Advantage tables will be created.
13. Click the Finish button to begin the data conversion.
14. When the conversion is finished, click Close and exit Advantage Data Architect.
    *Note: You will encounter a Paradox Import Note when converting Paradox tables that have primary indexes defined. By default, the PRIMARY index is not the default index with an Advantage application. The PRIMARY index can be set as the default index in the Advantage Data Dictionary or otherwise, the PRIMARY indexes will need to be explicitly activated in your application. Click OK to continue.*

**Task 2 – Create a Simple Browser**

This task involves creating a simple application that can browse a database table. This browser will be the foundation for this sample application, so leave extra room on the form for components that will be added later.

***Start a New Application***

1. Open Delphi and select New Application from the File the menu.

***Set Project Search Path***

1. Select Options from the Project menu.
2. Select the Directories/Conditionals tab.
3. If the installation program has not modified the search path to include your Advantage directory, enter the path to your TDataSet Descendant files in the Search Path edit box (the default directory is C:\Program Files\Advantage ?\TDataSet\Delphi?\Win32).
   *Note If you wish to set the same path to your TDataSet Descendant files for all projects, enter the path in the Delphi Environment Library. To access the Delphi Environment Library, choose Environment Options from the Tools menu and click the Library tab.*

***Change Window Caption***

1. Select Form1.
   *Object Inspector:*
   *Caption - My First Advantage-Enabled Application*

***Add DataSource***

1. Choose the Data Access tab on the Delphi Component Palette.
2. Select a DataSource component and place it on Form1.
   *Object Inspector:*
   *Name - dsCustomer*

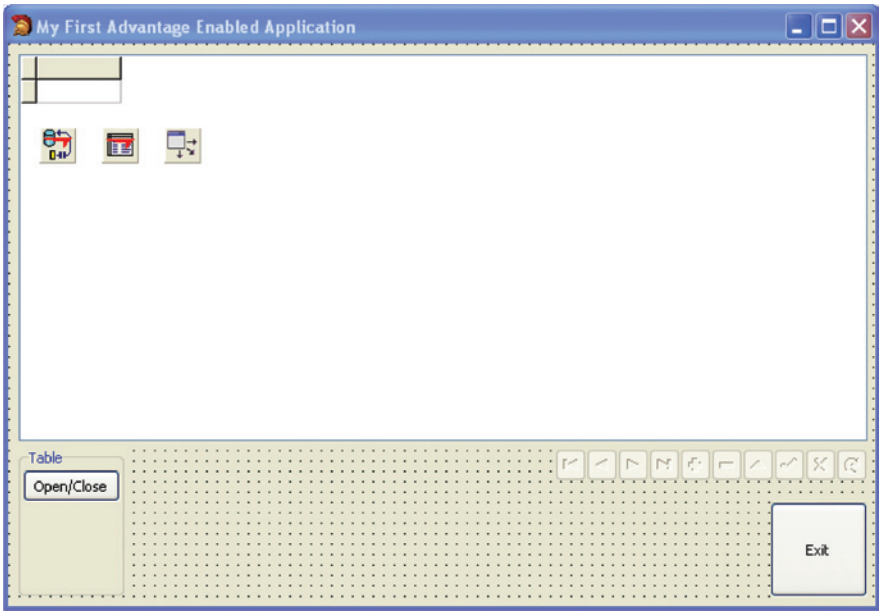



**Figure 1** – Browser Layout

*Note: Be sure to leave room for components that will be added in future steps.*

***Add Data-Aware Controls***

1. Choose the **Data Controls** tab on the Delphi Component Palette.
2. Select a DBGrid component and place it on Form1.
   *Object Inspector:*
   *DataSource - dsCustomer*
3. Select a DBNavigator component and place it on Form1.
   *Object Inspector:*
   *DataSource - dsCustomer*

***Add DataSet Component***

1. Choose the **Advantage** tab from the Delphi Component Palette.
2. Select an AdsConnection component and place it on Form1
   *Object Inspector:*
   *Name - cnCustomer*
   *ConnectPath - X:\Ads\Data*
3. (Optional) If you want to use the Advantage Local Server instead of the Advantage Database Server, you will need to configure your application to attempt an Advantage Local Server connection. This is most easily accomplished by modifying the AdsServerTypes property accordingly. For more information on the possible AdsServerTypes combinations, see the topic Advantage Server Types in the Advantage Help documentation (Advantage.hlp or advantage.htm).
   *Object Inspector:*
   *AdsServerTypes - stADS_LOCAL - True*
4. Select an AdsTable component and place it on Form1.
   *Object Inspector:*
   *Name - tblCustomer*
   *AdsConnection - cnCustomer*
   *TableName - Customer.adt*

***Link DataSource to DataSet***

1. Select the dsCustomer DataSource object in Form1.
   *Object Inspector:*
   *DataSet - tblCustomer*

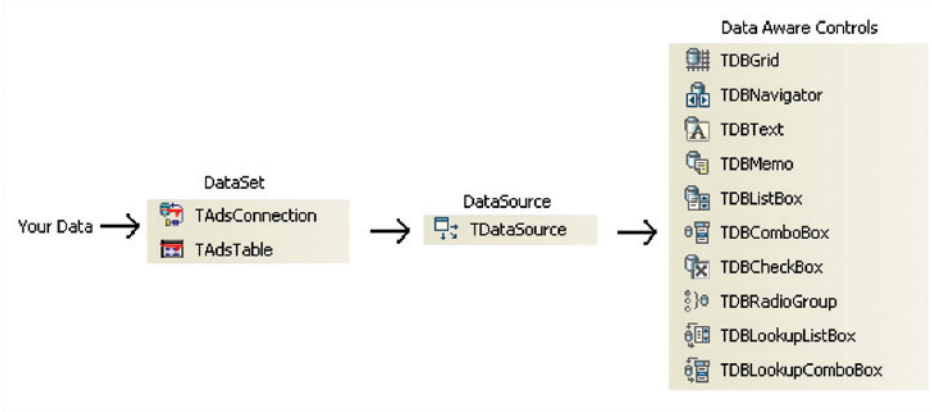



**Figure 2** – Data Component Hierarchy

It is now possible, at design-time, to open the table and fill the grid. To do this, select **tblCustomer** and change its Active property to True. The table will be opened and your grid will fill with data. If you do not see any records in your grid, or if you receive an error, carefully re-trace each step above. After verifying your table setup is correct change the Active property back to False.

*Add Open/Close Functionality*

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a GroupBox component and place it on Form1.

   *Object Inspector:*

   *Name - grpbxTable*

   *Caption - Table*
3. Choose the **Standard** tab on the Delphi Component Palette.
4. Select a Button component and place it in the **Table** groupbox on Form1.

   *Object Inspector:*

   *Name - btnOpenClose*

   *Caption - Open/Close*
5. Double-click the **Open/Close** button and add the following code to the OnClick event handler:

```
procedure TForm1.btnOpenCloseClick(Sender: TObject);
begin
   tblCustomer.Active := not tblCustomer.Active;
end;
```

*Add Exit Button*

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a Button component and place it in the bottom right corner of Form1.

   *Object Inspector:*

   *Name - btnExit*

   *Caption - Exit*
3. Double-click the **Exit** button and add the following code to the OnClick event handler:

```
procedure TForm1.btnExitClick(Sender: TObject);
begin
   Close;
end;
```

*Save and Run*

1. Select **Save All** from the **File** menu.
2. Select **Run** from the **Run** menu, or press **[F9]** to build and run your application.

**Task 3 – Increase Insert/Edit Functionality**

In this task you will add a separate form to support single record edits and/or insertions. This task will introduce you to the concept of linking data-aware controls to individual fields in your table.

*Create Insert/Edit Form (Form2)*

1. Select **New Form** from the **File** menu.

   *Object Inspector:*

   *Caption - Insert/Edit*
2. Select **Use Unit** from the **File** menu and double-click **Unit1**. This step allows Form2 to access objects and methods that belong to Form1. This link must be established to access the table component on Form1.

*Add Data-Aware Controls*

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a GroupBox component and place it on Form2.

   *Object Inspector:*

   *Name - grpbxCustInfo*

   *Caption - Customer Information*

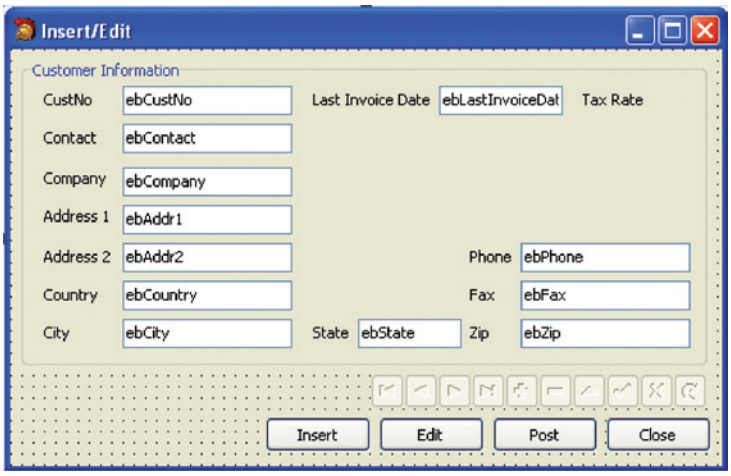



**Figure 3** – Single Record Entry Screen

For each column in the table that is displayed do the following (Note: the columns in the CUSTOMER.ADT file are CustNo, Company, Addr1, Addr2, City, State, Zip, Country, Phone, Fax, TaxRate, Contact, and LastInvoiceDate):

a) Choose the Standard tab on the Delphi Component Palette.

b) Select a Label component and place it in the Customer Info groupbox.

   *Object Inspector:*

   *Caption -<name of field> (e.g., Contact, Phone, etc.)*

c) Align the labels from top to bottom by column position.

d) Choose the **Data Controls** tab on the Delphi Component Palette.

e) Select a DBEdit component and place it in the Customer Info groupbox next to the label created in the previous step.

   *Object Inspector:*

   *Name - eb<name of field> (e.g., ebContact)*

   *DataSource - Form1.dsCustomer*

   *DataField - <name of field> (e.g., Contact, Phone, etc.)*

f) Align the edit boxes next to the corresponding label.

4. Choose the **Data Controls** tab from the Delphi Component Palette.
5. Select a DBNavigator component and place it on Form2.

   *Object Inspector:*

   *DataSource - Form1.dsCustomer*

***Add Insert, Edit, Post, and Close Buttons***

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select 4 Button components and place them on Form2.

   *Object Inspector:*

   *Name - btnInsert, btnEdit, btnPost, btnClose*

   *Caption - Insert, Edit, Post, Close*

   *Default - True (for the Close button only)*
3. Double-click the **Insert** button and add the following code to the OnClick event handler:

```
procedure TForm2.btnInsertClick(Sender: TObject);
begin
   Form1.tblCustomer.Insert;
end;
```

4. Double-click the **Edit** button and add the following code to the OnClick event handler:

```
procedure TForm2.btnEditClick(Sender: TObject);
begin
   Form1.tblCustomer.Edit;
end;
```

5. Double-click the **Post** button and add the following code to the OnClick event handler:

```
procedure TForm2.btnPostClick(Sender: TObject);
begin
   Form1.tblCustomer.Post;
end;
```

6. Double-click the **Close** button and add the following code to the OnClick event handler:

```
procedure TForm2.btnCloseClick(Sender: TObject);
begin
   Close;
end;
```

*Note: The functionality added with the Insert, Edit, and Post buttons already exists in the DBNavigator control (see your Delphi DBNavigator documentation for details). These buttons were included to demonstrate an alternate way of controlling record manipulation.*

***Modify Form1 to Display Form2***

1. Select **Forms** from the **View** menu and double-click **Form1**.
2. Choose the **Standard** tab on the Delphi Component Palette.
3. Select a Button component and place it below the **Open/Close** button in the **Table** groupbox on Form1.

   *Object Inspector:*

   *Name - btnInsertEdit*

   *Caption - Insert/Edit*
4. Double-click the Insert/Edit button and add the following code to the OnClick event handler:

```
procedure TForm1.btnInsertEditClick(Sender: TObject);
begin
   Form2.ShowModal;
end;
```

5. Select **Use Unit** from the **File** menu and double-click **Unit2**. This step allows Form1 to access objects and methods that belong to Form2. This link must be established in order for Form1 to display Form2.

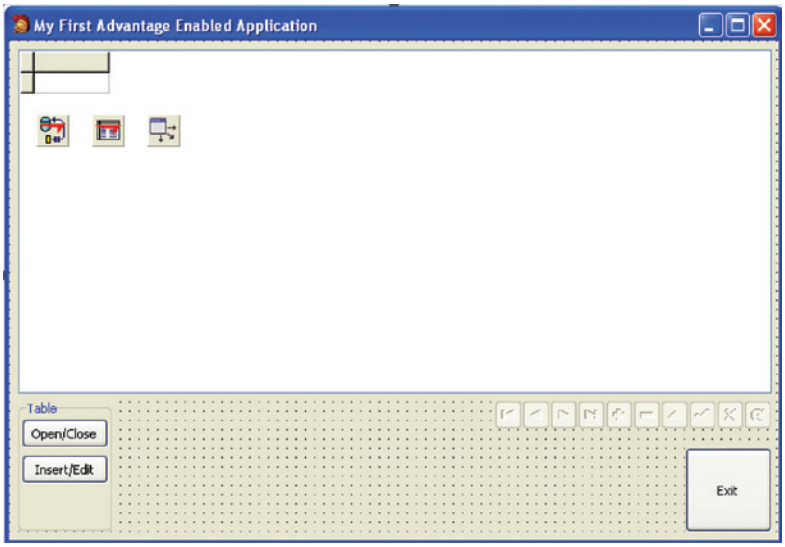



**Figure 4** – Browser with the Insert/Edit Button in Place

***Save and Run***

1. Select **Save All** from the **File** menu.
2. Select **Run** from the **Run** menu, or press **[F9]** to build and run your application.

**Task 4 – Add Filter Functionality**

In this task you will add the ability to set filters on the table using an expression provided by the user.

***Add Filter GroupBox***

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a GroupBox component and place it on Form1.

*Object Inspector:*

*Name - grpbxFilter*

*Caption – Filter*

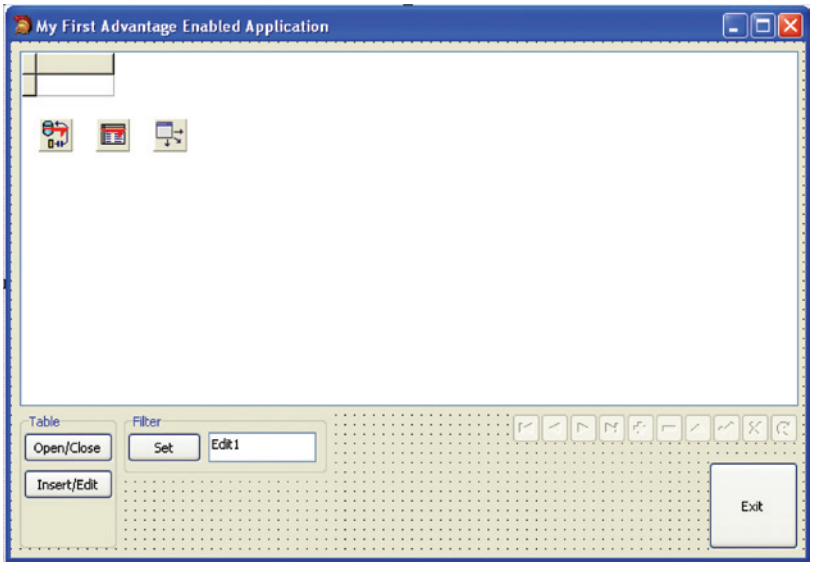



**Figure 5** – Browser with the New Filter GroupBox

***Add Filter Expression Edit Box***

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select an Edit component and place it in the **Filter** groupbox.

   *Object Inspector:*

   *Name - ebFilter*

   *Text - '' (empty string)*

***Add Filter Button and Code***

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a Button component and place it in the Filter groupbox.

   *Object Inspector:*

   *Name - btnSetFilter*

   *Caption - Set*

3. Double-click the **Set** button and add the following code to the OnClick event handler:

```
procedure TForm1.btnSetFilterClick(Sender: TObject);
begin
   if btnSetFilter.Caption = 'Set' then
   begin
      tblCustomer.Filter := ebFilter.Text;
      try
         tblCustomer.Filtered := TRUE;
      except
         on E: Exception do
         begin
           Application.MessageBox( Pchar(E.message), 'Filter Error', 0 );
            exit;
         end;
      end;
      btnSetFilter.Caption := 'Clear';
   end
   else {* btnSetFilter.Caption = 'Clear' *}
   begin
      tblCustomer.Filtered := FALSE;
      btnSetFilter.Caption := 'Set';
   end;
end;
```

***Save and Run***

1. Select **Save All** from the **File** menu.
2. Select Run from the Run menu, or press **[F9]** to build and run your application.

### Task 5 – Add Search Functionality

In this task you will add search functionality to your application using the Locate function.

#### *Add Search GroupBox*

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a GroupBox component and place it on Form1.

   *Object Inspector:*

   *Name - grpbxSearch*

   *Caption - Search*

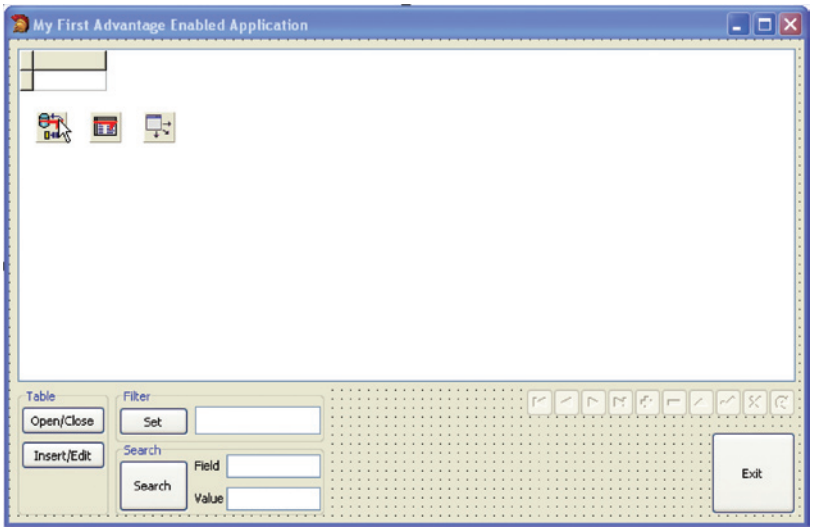



**Figure 6** – Browser with the New Search GroupBox

#### *Add Search Edit Boxes and Labels*

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a Label component and place it in the **Search** groupbox.

   *Object Inspector:*

   *Caption - Field*

3. Select an Edit component and place it in the **Search** groupbox next to the Field label.

   *Object Inspector:*

   *Name - ebSearchField*

   *Text - " (empty string)*

4. Select a Label component and place it in the **Search** groupbox.

   *Object Inspector:*

   *Caption - Value*

5. Select an Edit component and place it in the **Search** groupbox next to the **Value** label.

   *Object Inspector:*

   *Name - ebSearchValue*

   *Text - " (empty string)*

***Add Search Button and Code***

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a Button component and place it in the **Search** groupbox.

   *Object Inspector:*

   *Name - btnSearch*

   *Caption - Search*
3. Double-click the **Search** button and add the following code to the OnClick event handler:

```
procedure TForm1.btnSearchClick(Sender: TObject);
begin
   tblCustomer.Locate( ebSearchField.text,
                       Variant(ebSearchValue.text),
                       [loPartialKey] );
end;
```

*Note: For more information on the Locate method, reference your Borland Delphi documentation.*

***Save and Run***

1. Select **Save All** from the **File** menu.
2. Select **Run** from the **Run** menu, or press **[F9]** to build and run your application.

## TASK 6 – ADD ACTIVE INDEX SELECTION

In this task you will add the ability to set the active index on the table. This will allow you to choose between the table's available indexes and view the table in its indexed order.

***Add OrderBy GroupBox***

1. Choose the Standard tab on the Delphi Component Palette.
2. Select a GroupBox component and place it on Form1.

   *Object Inspector:*

   *Name - grpbxOrder*

   *Caption - Order By:*
3. Select a ComboBox component and place it in the Order By: groupbox.

   *Object Inspector:*

   *Name - cbActiveIndex*

   *Style – csDropDownList*

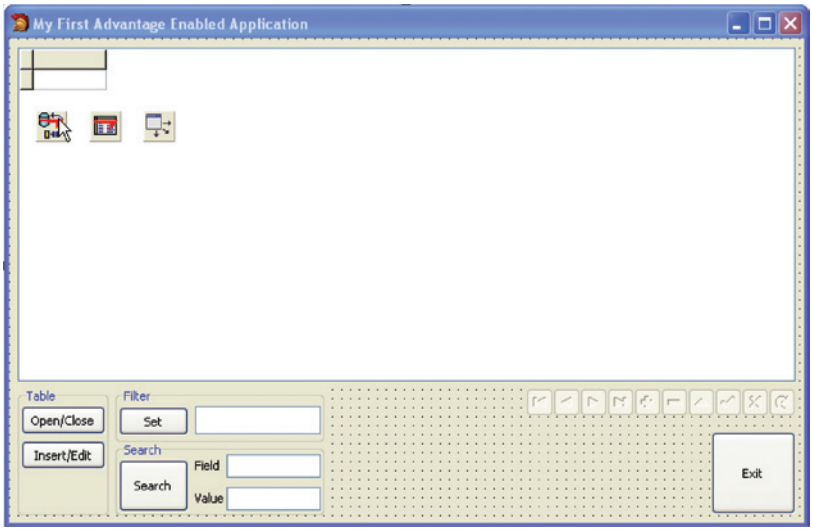



**Figure 7** – Browser with the New OrderBy GroupBox

***Add RefreshIndexList Procedure***

The RefreshIndexList procedure is used to retrieve a list of indexes that belong to the table and place them in the cbActiveIndex combobox. In a later task you will add index creation capabilities to your application and include a call to RefreshIndexList to update the cbActiveIndex combobox with the names of any new indexes that may have been created.

1. Add the "procedure RefreshIndexList" line to the end of the procedure declarations as shown below in the TForm1 class declaration in UNIT1.PAS:

```
    procedure btnOpenCloseClick(Sender: TObject);
    procedure btnExitClick(Sender: TObject);
    procedure btnInsertEditClick(Sender: TObject);
    procedure btnSetFilterClick(Sender: TObject);
    procedure btnFindClick(Sender: TObject);
    procedure RefreshIndexList;
private
    { Private declarations }
public
    { Public declarations }
end;
```

2. Add the procedure TForm1.RefreshIndexList and its code to the implementation section as shown below in UNIT1.PAS:

```
var
  Form1: TForm1;
implementation
uses Unit2;
{$R *.DFM}
procedure TForm1.RefreshIndexList;
begin
   with cbActiveIndex do
   begin
      Items.Clear;
      tblCustomer.GetIndexNames( Items );
      Items.Add( 'Natural Order' );
      ItemIndex := Items.Count - 1;
   end;
end;
```

3. Select **Forms** from the **View** menu and double-click **Form1**.
4. Double-click in a blank area inside Form1 to access the Form1 OnCreate event handler.
5. Add the following code to the OnCreate event handler:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   RefreshIndexList;
end;
```

***Implement cbActiveIndex***

1. Select the **cbActiveIndex** combobox from Form1 and press **[F11]** to activate the Delphi Object Inspector.
2. Select the **Events** tab in the Delphi Object Inspector.
3. Double-click inside the **OnChange** edit box to access the OnChange event handler for **cbActiveIndex**.
4. Add the following code to the OnChange event handler:

```
procedure TForm1.cbActiveIndexChange(Sender: TObject);
begin
   with cbActiveIndex do
   begin
      if Items[itemindex] = 'Natural Order' then
         tblCustomer.IndexName := ''
      else
         tblCustomer.IndexName := Items[itemindex];
   end;
end;
```

***Save and Run***

1. Select **Save All** from the **File** menu.
2. Select **Run** from the **Run** menu, or press the **[F9]** key to build and run your application.

**Task 7 – Add Index Creation Capability**

In this task you will add the ability to create new indexes to your application.

***Create a New Form (Form3)***

1. Select **New Form** from the **File** menu.

   *Object Inspector:*

   *Caption - Create Index*

2. Choose **Use Unit** from the **File** menu and double-click **Unit1**. This step allows Form3 to access objects and methods that belong to Form1. This link must be established to access the table component on Form1.

***Add Indexname and Expression Edit Boxes***

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a Label component and place it on Form3.

   *Object Inspector:*

   *Caption - Index Name*

3. Select an Edit component and place it next to the **Index Name** label.

   *Object Inspector:*

   *Name - ebIndexName*

   *Text - " (empty string)*

4. Select a Label component and place it on Form3 below the **Index Name** label.

   *Object Inspector:*

   *Caption - Expression*

5. Select an Edit component and place it next to the Expression label.

   *Object Inspector:*

   *Name - ebExpression*

   *Text - " (empty string)*

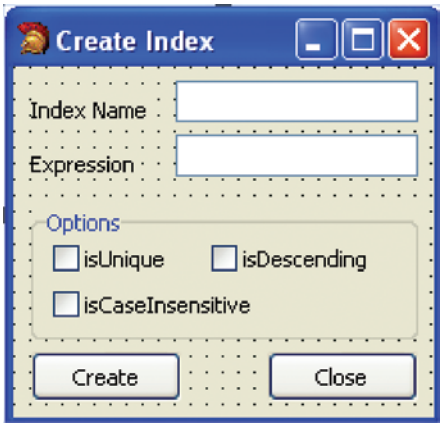



**Figure 8** – Index Creation Form Layout

***Add Options GroupBox***

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a GroupBox component and place it on Form3.

   *Object Inspector:*

   *Name - grpbxOptions*

   *Caption - Options*

3. Select a CheckBox component and place it in the **Options** groupbox.

   *Object Inspector:*

   *Name - chkbxUnique*

   *Caption - ixUnique*

4. Select a CheckBox component and place it in the **Options** groupbox.

   *Object Inspector:*

   *Name - chkbxDescending*

   *Caption - ixDescending*

5. Select a CheckBox component and place it in the **Options** groupbox.

   *Object Inspector:*

   *Name - chkbxCaseInsensitive*

   *Caption - ixCaseInsensitive*

***Add Create and Close Buttons***

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a Button component and place it on Form3.

   *Object Inspector:*

   *Name - btnCreate*

   *Caption - Create*

3. Double-click the **Create** button and add the following code to the OnClick event handler:

```
procedure TForm3.btnCreateClick(Sender: TObject);
var
   setOptions : TIndexOptions;
begin
   setOptions := [];
   if chkbxUnique.Checked = TRUE then
      setOptions := setOptions + [ixUnique];
   if chkbxDescending.Checked = TRUE then
      setOptions := setOptions + [ixDescending];
   if chkbxCaseInsensitive.Checked = TRUE then
      setOptions := setOptions + [ixCaseInsensitive];
   {* close the table and open it for exclusive use *}
   with Form1.tblCustomer do
   begin
      Close;
      Exclusive := TRUE;
      Open;
      AddIndex( ebIndexName.text, ebExpression.text, setOptions );
      {* re-open shared *}
      Close;
      Exclusive := FALSE;
      Open;
   end;
end;
```

4. To use the setOptions variable of TIndexOptions type in the **btnCreate** OnCreate event you will need to add db.pas to the uses list in UNIT3.PAS:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, DB;
```

5. Select a Button component and place it on Form3.

   *Object Inspector:*

   *Name - btnClose*

   *Caption - Close*

   *Default - TRUE*

6. Double-click the **Close** button and add the following code to the OnClick event handler:

```
procedure TForm3.btnCloseClick(Sender: TObject);
begin
   Form1.RefreshIndexList;
   Close;
end;
```

***Modify Form1 to Display Form3***

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a Button component and place it in the **Table** groupbox in Form1.

   *Object Inspector:*

   *Name - btnCreate*

   *Caption - Create Index*

3. Double-click the **Create Index** button and add the following code to the OnClick event handler:

```
procedure TForm1.btnCreateClick(Sender: TObject);
begin
   Form3.ShowModal;
end;
```

4. Select **Use Unit** from the **File** menu and double-click **Unit3**. This step allows Form1 to access objects and methods that belong to Form3. This link must be established in order for Form1 to display Form3.

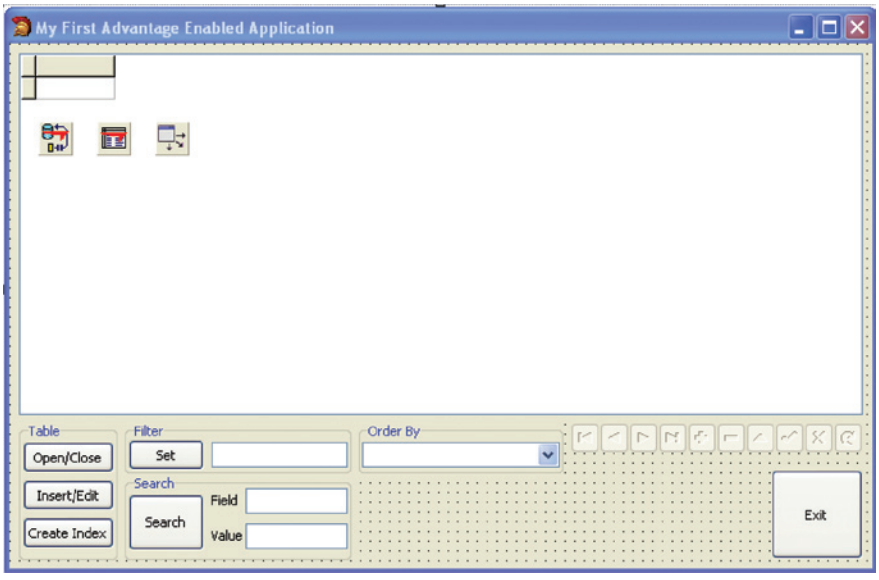



**Figure 9** – Browser with the New "Create Index" Button

***Save and Run***

1. Select Save All from the File menu.
2. Select Run from the Run menu, or press [F9] to build and run your application.

*Note: If you receive a 7008 error when trying to create an index, you have left the Active property on* ***tblCustomer*** *set to True. If this is the case then the instance of your program in the Delphi IDE has the table open shared, and when your application tries to get exclusive use of the table the open will fail, returning the 7008 error. To remedy this situation close your application, change the Active property on* ***tblCustomer*** *to False, and re-run your application.*

**Task 8 – Add Range Functionality**

In this task you will add the ability to set a range on your table using the active index. For more information on ranges, see the topic *Index Scopes (Ranges)* in the Advantage TDataSet Descendant Help file (ADE.HLP) or reference your Delphi documentation.

***Add Range GroupBox***

1. Choose the Standard tab on the Delphi Component Palette.
2. Select a GroupBox component and place it on Form1.

   *Object Inspector:*

   *Name - grpbxRange*

   *Caption – Range*

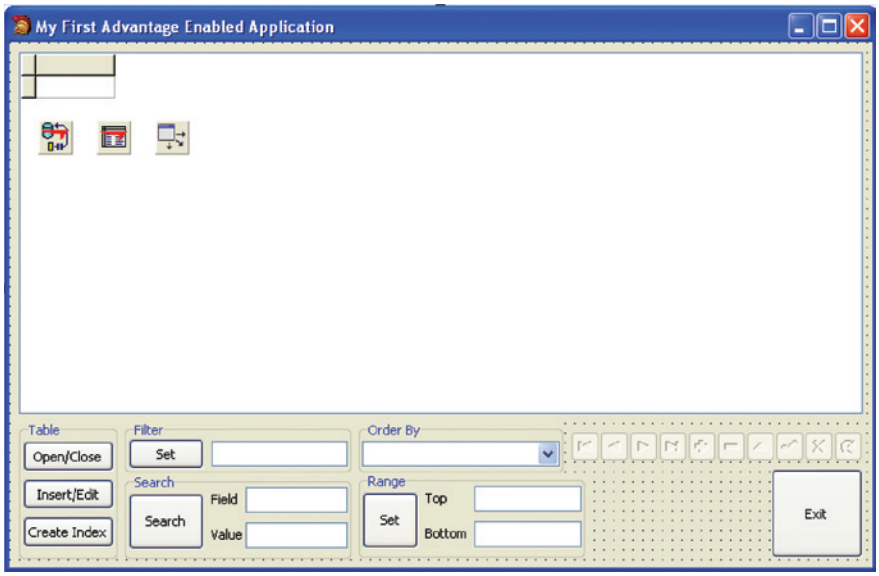



**Figure 10** – Browser with the New Range GroupBox

***Add RangeTop and RangeBottom Edit Boxes***

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a Label component and place it in the **Range** groupbox.

   *Object Inspector:*

   *Caption - Top*

3. Select an Edit component and place it in the **Range** groupbox next to the **Top** label from the previous step.

   *Object Inspector:*

   *Name - ebRangeTop*

   *Text - " (empty string)*

4. Select a Label component and place it in the **Range** groupbox.
   *Object Inspector:*
   *Caption - Bottom*
5. Select an Edit component and place it in the **Range** groupbox next to the **Bottom** label from the previous step.
   *Object Inspector:*
   *Name - ebRangeBottom*
   *Text - " (empty string)*
6. Select a Button component and place it in the **Range** groupbox.
   *Object Inspector:*
   *Name - btnSetRange*
   *Caption - Set*
7. Double-click the **Set** button and add the following code to the OnClick event handler:

```
procedure TForm1.btnSetRangeClick(Sender: TObject);
var
   def: TIndexDef;
begin
   {* make sure there is an active index *}
   with cbActiveIndex do
   begin
     if Items[ItemIndex] = 'Natural Order' then
     begin
        Application.MessageBox( 'No Active Index', 'Error', 0 );
        exit;
     end;
   end;

   if btnSetRange.Caption = 'Set' then
   begin
      {* set the range *}
      with tblCustomer do
      begin
         try
            SetRangeStart;
            def := IndexDefs.
                       Find(cbActiveIndex.Items[cbActiveIndex.ItemIndex]);
            FieldByName(def.fields).asstring := ebRangeTop.Text;
            SetRangeEnd;
            FieldByName(def.fields).asstring := ebRangeBottom.Text;
            ApplyRange;
         except
            on E: Exception do
            begin
               Application.MessageBox( PChar(E.message), 'Error', 0 );
               exit;
            end;
```

```
            end;
            btnSetRange.Caption := 'Clear';
        end;
    end {* if setting range*}
    else
    begin
        {* clear the range *}
        tblCustomer.CancelRange;
        btnSetRange.Caption := 'Set';
    end;
end;
```

*Note: For more information on SetRangeStart, SetRangeEnd, and ApplyRange reference your Delphi documentation.*

***Save and Run***

1. Select **Save All** from the **File** menu.
2. Select **Run** from the **Run** menu, or press **[F9]** to build and run your application.

**Task 9 – Add FindKey/FindNearest Functionality**

In this task you will add the ability to your application to do a FindKey or FindNearest. FindKey/FindNearest are in general very fast, but they require an index on the key in question. See the topic Seek (FindKey, FindNearest, GotoKey, GotoNearest) in the Advantage TDataSet Descendant Help file (ADE.HLP) for more information.

***Add Find GroupBox***

1. Choose the **Standard** tab on the Delphi Component Palette.
2. Select a GroupBox component and place it on Form1.

    *Object Inspector:*

        *Name - grpbxFind*

        *Caption – Find*

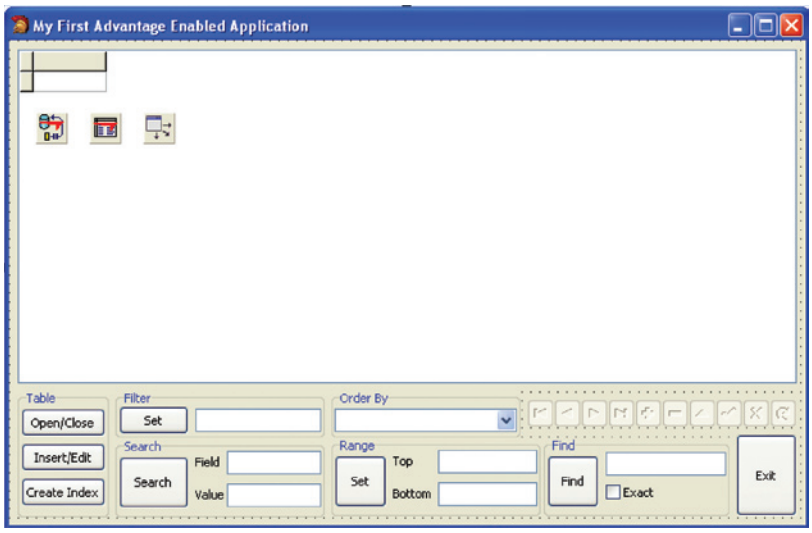



**Figure 11** – Browser with the New Find GroupBox Completing Your Sample Application

***Add Find Edit Box and Exact Checkbox***

1. Choose the Standard tab on the Delphi Component Palette.
2. Select an Edit component and place it in the Find groupbox.

   Object Inspector:

   Name - ebFindValue

   Text - " (empty string)

3. Select a CheckBox component and place it in the Find groupbox.

   Object Inspector:

   Name - chkbxExact

   Caption - Exact

***Add Find Button and Code***

1. Choose the Standard tab on the Delphi Component Palette.
2. Select a Button component and place it in the Find groupbox.

   *Object Inspector:*

   *Name - btnFind*

   *Caption - Find*

3. Double-click the **Find** button and add the following code to the OnClick event handler:

```
procedure TForm1.btnFindClick(Sender: TObject);
begin
   if chkbxExact.Checked then
      tblCustomer.FindKey( [ebFindValue.Text] )
   else
      tblCustomer.FindNearest( [ebFindValue.Text] );
end;
```

***Save and Run***

1. Select **Save All** from the **File** menu.
2. Select **Run** from the **Run** menu, or press **[F9]** to build and run your application.

### Advantage Data Architect

Advantage Data Architect (ARC32.EXE) is a fully integrated tool to assist you in efficiently developing and maintaining Advantage database applications. Advantage Data Architect is free of charge and ships with source code. The Advantage Data Architect product can be downloaded from the Downloads page on the Advantage Database Server Web site, **www.AdvantageDatabase.com**.

While developing your database applications, the Advantage Data Architect allows you to:

- Import and convert other table types such as Paradox, dBASE, Access, and SQL Server tables to Advantage ADT tables.
- Create Advantage data dictionaries and define Referential Integrity rules, record and field level constraints, etc.
- Create tables and indexes.
- Generate and test Advantage SQL queries using a visual query designer.
- Generate code to automatically create tables and indexes using the Advantage Tables to Code Generator.

When setting up your database application, Advantage Data Architect allows you to:

- Check the client workstation's environment to test and diagnose connection issues with the Advantage Database Server.
- Setup Access Control Lists to define user access rights for users who wish to connect to the Advantage Database Server.
- Create aliases similar to those used by the Borland Database Engine via the Connection Repository.

After the application is up and running, Advantage Data Architect allows you to:

- Manage the database with functionality for rebuilding indexes, packing tables, restructuring tables, and repairing tables.
- Manage Advantage data dictionaries with the Connection Repository.
- Observe Advantage Database Server activity with the Advantage Management Utility, which includes verifying files are actually opened by the Advantage Database Server and determining which index files are opened, which users are connected, what installation parameters where selected, and what files or records are locked.

### Converting Data Using Advantage Data Architect

The first step in any conversion is to move existing data to the Advantage data format. It is possible to write your own Delphi/C++Builder application for this purpose, but you will most likely find it easier to use the Advantage Data Architect.

1. Open the Advantage Data Architect.
2. Create a new data directory or data dictionary to store your Advantage files (file->new connection wizard).
3. Choose **Import** from the **Tools** menu.
4. Under the **Select Import Type** tab, select the type of data you will be importing
5. In the Select Import Data tab, enter the path to your data in the **Import Filename** text box, or click the browse button to search
6. Select the desired **Import Table Type**
7. Under the **Select Destination** tab, name of the new connection. This should be the connection you created in Step 2 above. This is where all of your new Advantage tables will be created.
8. Click the **Import** button to begin the data conversion.
9. When the conversion is finished, click **Close**.

*Note: You will encounter a **Paradox Import Note** when converting Paradox tables that have primary indexes defined. By default, the PRIMARY index is not the default index with an Advantage application. The PRIMARY index can be set as the default index in the Advantage Data Dictionary or otherwise, the PRIMARY indexes will need to be explicitly activated in your application. Click **OK** to continue.*

***Paradox/Advantage Field Data Type Conversion***

This table shows how field types are converted when Paradox tables are imported to the Advantage ADT table format.

| Paradox Data Type | Advantage Data Type | Comment |
|---|---|---|
| Alpha | Character | |
| Number | Double | |
| Money | Money | This data type applies to Advantage ADT tables only. It does not apply to DBF tables. |
| Short | Integer | The Paradox Short type is unsigned. Since Advantage has no unsigned 2-byte data types, an Integer type is used. |
| Long Integer | Integer | |
| BCD | Money or Double | If the precision of the BCD field contains four or less decimal digits, the result data type will be Money. Otherwise, it will be double. Note: The Money data type applies to ADT tables only. |
| Date | Date | |
| Time | Time | This data type applies to Advantage ADT tables only. It does not apply to DBF tables. |
| Timestamp | TimeStamp | This data type applies to Advantage ADT tables only. It does not apply to DBF tables. |
| Memo | Memo | |
| Formatted Memo | Memo | |
| Graphic | Image | |
| OLE | Binary | |
| Logical | Logical | |
| AutoIncrement | AutoIncrement | This data type applies to Advantage ADT tables only. It does not apply to DBF tables. |
| Binary | Binary | |
| Bytes | Raw | This data type applies to Advantage ADT tables only. It does not apply to DBF tables. |

***Primary Indexes***

An index named PRIMARY is created for each of the Paradox tables during the import process. By default, the PRIMARY index is not the default index with an Advantage application. The PRIMARY index can be set as the default index in the Advantage Data Dictionary or otherwise, the PRIMARY index name will need to be specified in the application wherever needed.

### *Index Structures*

Advantage places all individual indexes into a single file called a compound index file. Specific indexes within a compound index file are differentiated via "tag" names. Tag names are a way to specify a unique name to set an active index. If the base file name of the compound index file is the same as the base file name of the associated table, the index file is called an auto-open or structural index file and is automatically opened when the table is opened. The maximum number of tags inside a single compound index file is limited to 50.

### *Index Expression*

You can use an index expression and an index condition with Advantage proprietary ADI index files. The Advantage Expression Engine is used to evaluate expression index statements. See the Advantage Expression Engine topic in the Advantage Help file (Advantage.hlp) for more information.

## Converting Existing Delphi/C++Builder Applications to Use the Advantage TDataSet Descendant

This section describes the steps necessary to convert an existing Delphi application to use the Advantage TDataSet Descendant. The information shows how to convert TTable instances to TAdsTable instances without having to reconfigure persistent fields, datasource links, or table properties.

### *Converting TTable Instances to TAdsTable Instances*

This section includes generic instruction for converting an application to use the Advantage TDataSet Descendant. Specific instructions using the Delphi demo applications will be provided in the next section. The easiest way to convert TTable instances to TAdsTable instances and preserve your current table configuration is to view your data form as text and then make only the necessary changes.

*Note: TQuery and TStoredProc instances can be converted to TAdsQuery and TAdsStoredProc instances using this same kind of process.*

1. Right-Click on your form and choose View as Text from the quick menu. You will now see the text representation of your form and all of the objects on the form.
2. Search for TTable
3. For each TTable object you find do the following:
   - Change TTable to TAdsTable. For example the line object tblMaster: TTable would be changed to object tblMaster: TAdsTable
   - Below the object declaration you will find the DatabaseName property. If this property does not already include the path to your data then replace it with the path (e.g., X:\ADS\DATA) or the ALIAS name defined in the ads.ini file.
   - Below the DatabaseName property you will find the TableName property. Under most circumstances your table name will change after converting to the Advantage table format (e.g., MYTABLE.DB will change to MYTABLE.ADT). If this is the case then change the TableName property to reflect this new extension.
4. Once you are finished changing all of your table declarations right-click in the edit window and select **View as Form** from the quick menu. You should now see TAdsTable objects in place of your original TTable objects.
5. Select **Save** from the **File** menu.
6. After making these changes the next time you build the project Delphi will warn you that some declarations have changed and offer to correct them for you. Select **yes** and let Delphi correct the rest of the declarations.

***Converting Delphi Demos to Use Advantage TDataSet Descendant***

The following instructions apply the generic method of converting your Delphi applications with a few existing applications from the Delphi demo directory.

**CtrlGrid Demo**

1. Select **Open Project** from the File menu
2. Browse to your Delphi database demo directory, the default directory is C:\Program Files\Borland\Delphi?\Demos\DB
3. Browse into the CtrlGrid directory and open CtrlGrid.dpr
   *Note If you cannot find the demo directory, reference your Delphi installation documentation for details on how to re-install with this option.*
4. Press **[F9]** to build and run the application. Use this time to get familiar with the application and verify that it is working correctly with the BDE.
5. Select **Forms** from the **View** menu and select **DM1**, the data module for this application.
6. Right-click on the data module (DM1) and select **View as Text**. In this text file you will find the declarations for three tables; tblMaster, tblIndustry, and tblHoldings.
7. In the tblMaster declaration change the object type from **TTable** to **TAdsTable**. On the next line you will find the databasename property, change it to contain the path to your data (e.g., X:\ADS\DATA) or the ALIAS name defined in the ads.ini file (e.g. DBDEMOS). Finally you will see the tablename property, which will need to be changed from MASTER.DBF to MASTER.ADT.
8. Scroll down until you find the tblIndustry declaration. Change the object type from **TTable** to **TAdsTable**. Change the databasename property to your data path or ALIAS, and the tablename from **industry.dbf** to **industry.adt**.
9. Scroll down until you find the tblHoldings declaration. As before change the object type from **TTable** to **TAdsTable**, the databasename to your data path or ALIAS, and the tablename from **holdings.dbf** to **holdings.adt**.
10. Right-click in the editor window and select **View as Form** from the menu. Note the TTable objects will now appear as TAdsTable objects.
11. Highlight each of the three tables and change their active properties in the Delphi Object Inspector to True.
12. Select **Options** from the **Project** menu.
13. Select the **Directories/Conditionals** tab.
14. In the **Search Path** field enter the path to your Advantage TDataSet Descendant files so that Delphi can find the necessary source files when building your project. The default is C:\Program Files\ Advantage ?\TDataSet\Delphi?\Win32
15. Select **Save All** from the File menu to save these new changes.
16. After making these changes the next time you build the project Delphi will warn you that some declarations have changed and offer to correct them for you. Select **yes** and let Delphi correct the rest of the declarations.
17. Press **[F9]** to build and run the application. Ctrlgrid is now using the Advantage TDataSet Descendant for all database access.

**FishFact**

1. Select **Open Project** from the **File** menu
2. Browse to your Delphi database demo directory, the default directory is C:\Program Files\Borland\Delphi?\Demos\DB
3. Browse into the FishFact directory and open FISHFACT.DPR
   *Note: If you cannot find the demo directory reference your Delphi installation documentation for details on how to re-install with this option.*
4. Press **[F9]** to build and run the application. Use this time to familiarize yourself with the application and verify that it is working correctly with the BDE.
5. Select **Forms** from the **View** and select **Form1**
6. Right-click on the form and select **View as Text** from the quick menu.
7. Select **Find** from the Search menu and search for TTable.
8. Change Table1's object type from 'TTable' to 'TAdsTable'.
9. Change the databasename property from 'DBDEMOS' to your data path (e.g., X:\ADS\DATA) or ALIAS.
10. Right-click in the editor window and select **View as Form** from the menu. Note the TTable objects will now appear as TAdsTable objects.
11. Select **Options** from the **Project** menu.
12. Select the **Directories/Conditionals** tab.
13. In the **Search Path** field enter the path to your Advantage TDataSet Descendant files so that Delphi can find the necessary source files when building your project. The default is C:\Program Files\ Advantage ?\TDataSet\Delphi?\Win32
14. Select **Save All** from the **File** menu to save these new changes.
15. After making these changes the next time you build the project Delphi will warn you that some declarations have changed and offer to correct them for you. Select **yes** and let Delphi correct the rest of the declarations.
16. Press **[F9]** to build and run the application. FishFact is now using the Advantage TDataSet Descendant for all database access.

**GDSDemo**

1. Select **Open Project** from the **File** menu
2. Browse to your Delphi database demo directory, the default directory is C:\Program Files\Borland\Delphi?\Demos\DB
3. Browse into the GDSDemo directory and open GDSDEMO.DPR
   *Note: If you cannot locate the demo directory, reference your Delphi installation documentation for details on how to re-install with this option.*
4. Press **[F9]** to build and run the application. Use this time to familiarize yourself with the application and verify that it is working correctly with the BDE.
5. Select **Forms** from the **View** and select **StdDataForm**
6. Right-click on the form and select **View as Text** from the quick menu.
7. Select **Find** from the **Search** menu and search for **TTable**.
8. Change the Orders table object type from TTable to TAdsTable.
9. Change the DatabaseName property from 'DBDEMOS' to your data path (e.g., X:\ADS\DATA) or ALIAS.
10. Right-click in the editor window and select **View as Form** from the menu. Note the TTable objects will now appear as TAdsTable objects.
11. Select **Options** from the **Project** menu.
12. Select the **Directories/Conditionals** tab.
13. In the **Search Path** field enter the path to your Advantage TDataSet Descendant files so that Delphi can find the necessary source files when building your project. The default is C:\Program Files\ Advantage ?\TDataSet\Delphi?\Win32
14. Select **Save All** from the **File** menu to save these new changes.
15. After making these changes the next time you build the project Delphi will warn you that some declarations have changed and offer to correct them for you. Select **yes** and let Delphi correct the rest of the declarations.
16. Press **[F9]** to build and run the application. GDSDemo is now using the Advantage TDataSet Descendant for all database access.

Contact Us:
North America
Advantageinfo@Sybase.com
1 800 235 7576

Germany
ADS-team@Sybase.com
+49 (0) 7032 / 798 - 200

United Kingdom
AdvantageUK@Sybase.com
+44 (0) 117 315 3957

Sybase, Inc.
Worldwide Headquarters
One Sybase Drive
Dublin, CA 94568-7902
U.S.A
1 800 8 sybase

**www.AdvantageDatabase.com**



SYBASE® iAnywhere®