

# **RAD Studio for Microsoft .NET**



# Table of Contents

<b>Concepts</b>	<b>1</b>
<b>Developing Database Applications with ADO.NET</b>	<b>3</b>
AdoDbx Client Overview	5
VCL for .NET Database Technologies	6
BDP Connection Pooling Overview	8
dbExpress Components overview	9
dbGo Components Overview	10
AdoDbx.NET Data Types	11
ADO.NET Overview	14
BDP Migration Overview	16
Blackfish SQL Overview	20
ADO.NET Component Designers	21
Deploying Database Applications for the .NET Framework	24
Data Providers for Microsoft .NET	27
Stored Procedure Overview	29
dbExpress Framework	30
dbExpress Framework Compatibility	31
Getting Started with InterBase Express	32
<b>Developing Applications with Unmanaged Code</b>	<b>38</b>
Using COM Interop in Managed Applications	38
Using DrInterop	43
Deploying COM Interop Applications	43
Using Platform Invoke with Delphi for .NET	44
Virtual Library Interfaces	52
<b>Modeling Concepts</b>	<b>54</b>
Code Visualization Overview	54
<b>Developing Reports for .NET Applications</b>	<b>56</b>
Using Rave Reports in RAD Studio	56
<b>Developing Applications with VCL.NET Components</b>	<b>57</b>
Changes Required Due to 64-bit .NET 2.0 Support	58
Language Issues in Porting VCL Applications to RAD Studio	59
Porting VCL Applications	69
VCL for .NET Overview	71
Porting Web Service Clients	74

<b>Developing Web Applications with ASP.NET</b>	<b>77</b>
ASP.NET Overview	79
CodeGear DB Web Controls Overview	81
Using DB Web Controls in Master-Detail Applications	83
DB Web Controls Navigation API Overview	85
DB Web Control Wizard Overview	86
Using XML Files with DB Web Controls	92
Working with DataViews	94
Deploying ASP.NET Applications	95
Working with WebDataLink Interfaces	96
<b>Developing Web Services with ASP.NET</b>	<b>97</b>
ASP.NET Web Services Overview	97
Web Services Protocol Stack	100
ASP.NET Web Services Support	102

## Procedures 105

<b>Database Procedures</b>	<b>106</b>
Adding a New Connection to the Data Explorer	107
Adding a BDP Reconcile Error dialog to your BDP Application	108
Browsing a Database in the Data Explorer	109
Connecting to the AdoDbx Client	110
Creating Database Projects from the Data Explorer	111
Creating Table Mappings	112
Executing SQL in the Data Explorer	113
Handling Errors in Table Mapping	114
Migrating Data Between Databases	115
Modifying Connections in the Data Explorer	116
Modifying Database Connections	117
Building a Database Application that Resolves to Multiple Tables	122
Passing Parameters in a Database Application	124
Using the Data Adapter Preview	126
Using the Command Text Editor	127
Using the Data Adapter Designer	128
Using the Connection Editor Designer	128
Using Standard DataSets	129
Using Typed DataSets	132
Connecting to a Database using the dbExpress Driver Framework	134
Building a Distributed Database Application	136

<b>Interoperable Applications Procedures</b>	<b>139</b>
Adding a J2EE Reference	139
Adding a Reference to a COM Server	139
<b>Modeling Procedures</b>	<b>141</b>
Exporting a Code Visualization Diagram to an Image	141
Importing and Exporting a Model Using XML Metadata Interchange (XMI)	142
Using the Model View Window and Code Visualization Diagram	143
Using the Overview Window	144
<b>VCL for .NET Procedures</b>	<b>145</b>
Building VCL Forms Applications With Graphics	147
Building a VCL.NET Forms ADO.NET Database Application	148
Building a VCL Forms Application	149
Creating Actions in a VCL Forms Application	150
Building a VCL Forms Hello World Application	151
Using ActionManager to Create Actions in a VCL Forms Application	152
Building a VCL Forms dbExpress.NET Database Application	153
Building an Application with XML Components	155
Making Changes Required Due to 64-bit .NET 2.0 Support	157
Creating a New VCL.NET Component	159
Displaying a Bitmap Image in a VCL Forms Application	160
Drawing Rectangles and Ellipses in a VCL Forms Application	161
Drawing a Rounded Rectangle in a VCL Forms Application	162
Drawing Straight Lines In a VCL Forms Application	163
Placing a Bitmap Image in a Control in a VCL Forms Application	164
Importing .NET Controls to VCL.NET	165
<b>ASP.NET Procedures</b>	<b>167</b>
Building an ASP.NET Application	170
Building an ASP.NET Database Application	171
Developing an ASP.NET Application with Database Controls, Part 1	174
Building an ASP.NET Application with Database Controls, Part 2	176
Building an ASP.NET Application with Database Controls, Part 3	177
Building an ASP.NET "Hello World" Application	178
Building an ASP.NET SiteMap	179
Creating a Briefcase Application with DB Web Controls	182
Building an Application with DB Web Controls	183
Converting HTML Elements to Server Controls	184
Creating an XML File for DB Web Controls	185

Creating Metadata for a DataSet	187
Creating a Virtual Directory	188
Adding Aggregate Values with DBWebAggregateControl	188
Debugging and Updating ASP.NET Applications	189
Deploying an ASP.NET Application using Blackfish SQL to a system without RAD Studio	190
Generating HTTP Messages in ASP.NET	191
Binding Columns in the DBWebGrid	191
Setting Permissions for XML File Use	192
Troubleshooting ASP.NET Applications	193
Using the DB Web Control Wizard	195
Using the ASP.NET Deployment Manager	196
Using the HTML Tag Editor	199
Working with ASP.NET User Controls	200
<b>Web Services Procedures</b>	<b>202</b>
Accessing an ASP.NET "Hello World" Web Services Application	202
Adding Web References in ASP.NET Projects	204
Building an ASP.NET "Hello World" Web Services Application	206
Porting a Delphi for Win32 Web Service Client Application to Delphi for .NET	207

## Index

**a**

# 1 Concepts

## Topics

Name	Description
Developing Database Applications with ADO.NET (see page 3)	<p>ADO.NET presents a coherent programming model for exposing data access within the .NET Framework. In addition to supporting MS SQL, Oracle, and OLE DB connection components within the .NET Framework, RAD Studio includes data providers for .NET (ADOdbClient Provider). ADOdb supports access to MS SQL, Oracle, DB2, and Interbase. ADOdb component designers ease the generation and configuration of ADOdb components.</p> <p>If you are developing new VCL Forms applications for the .NET Framework, or you are migrating existing Win32 VCL Forms applications to the .NET Framework, RAD Studio provides continued support for existing Delphi database technologies, such as dbExpress and dbGo.... more (see page 3)</p>
Developing Applications with Unmanaged Code (see page 38)	<p>RAD Studio provides the capability to work with the .NET features that support unmanaged code.</p> <p>If you have COM or ActiveX components that you want to use within the .NET Framework, you can use the .NET COM Interop capabilities from within RAD Studio while building your applications.</p>
Modeling Concepts (see page 54)	This section provides information on modeling and code visualization.
Developing Reports for .NET Applications (see page 56)	RAD Studio ships with Rave Reports from Nevrona. Using the report components, you can build full-featured reports for your applications. You can create solutions that include reporting capabilities which can be used and customized by your customers.
Developing Applications with VCL.NET Components (see page 57)	<p>VCL.NET is an extended set of the VCL components that provide a way to quickly build advanced applications in Delphi. With VCL.NET you can provide your Delphi VCL applications and components to Microsoft .NET Framework users. With RAD Studio you gain the benefit of the .NET Framework along with the ease-of-use and powerful component-driven application development of Delphi.</p> <p>RAD Studio provides distinct application types for your use: you can create VCL.NET form applications that run on the .NET Framework that use VCL.NET components and controls; you can create .NET applications that use the underlying .NET Framework and .NET controls while... more (see page 57)</p>
Developing Web Applications with ASP.NET (see page 77)	ASP.NET is the programming model for building Web applications using the .NET Framework. This section provides the conceptual background for building ASP.NET applications using RAD Studio. In addition to supporting data access components within the .NET Framework, RAD Studio includes DB Web Controls. DB Web Controls work with .NET Framework providers and Borland Data Providers for .NET (BDP.NET) to accelerate Web application development.

Developing Web Services with ASP.NET (see page 97)

Web Services is a programmable entity that provides a particular element of functionality, such as application logic. Web Services is accessible to any number of potentially disparate systems through the use of Internet standards, such as XML and HTTP. Applications built with ASP.NET Web Services can be either stand-alone applications or subcomponents of a larger web application and can provide application components to any number of distributed systems using XML-based messaging. RAD Studio provides a number of methods that can help you build, deploy, and use applications with ASP.NET Web Services. For more general information about Web Services, refer to... more (see page 97)



# 1.1 Developing Database Applications with ADO.NET

ADO.NET presents a coherent programming model for exposing data access within the .NET Framework. In addition to supporting MS SQL, Oracle, and OLE DB connection components within the .NET Framework, RAD Studio includes data providers for .NET (ADOdbxCClient Provider). ADOdbx supports access to MS SQL, Oracle, DB2, and Interbase. ADOdbx component designers ease the generation and configuration of ADOdbx components.

If you are developing new VCL Forms applications for the .NET Framework, or you are migrating existing Win32 VCL Forms applications to the .NET Framework, RAD Studio provides continued support for existing Delphi database technologies, such as dbExpress and dbGo.

This section includes conceptual information about how to use RAD Studio with the ADO.NET architecture, as well as the VCL for .NET database technologies. and how to build a simple ADO.NET project.

## Topics

Name	Description
ADOdbx Client Overview ( <a href="#">see page 5</a> )	<p>The ADOdbx Client implements an ADO.NET 2.0 provider for all dbExpress version 4 drivers that implement the newer extended metadata added to dbExpress 4. All dbExpress drivers shipped with Delphi implement the newer extended metadata.</p> <p>ADOdbx Client is an implementation of the ADO.NET 2.0 Provider classes. ADO.NET Provider is a set of classes that provide database services for .NET. It provides access to relational databases, XML and application data. You can use it to develop front end database applications as well as multi-tier business applications.</p> <p>See <a href="#">.NET Framework Developer's Guide ADO.NET</a> in the Microsoft documentation for more information.</p> <p>Here are... more (<a href="#">see page 5</a>)</p>
VCL for .NET Database Technologies ( <a href="#">see page 6</a> )	<p>In most cases, the ADOdbxCClient Provider provides the best database connectivity solution for your .NET applications. However, if you are developing new VCL Forms applications for the .NET Framework, or you are migrating existing Win32 VCL Forms applications to the .NET Framework, RAD Studio provides continued support for existing Delphi database technologies.</p> <p>RAD Studio provides a migration path from Delphi database technologies running strictly on Win32 clients to the .NET Framework. In addition to being able to build new database applications using ADO.NET, you can migrate existing database applications to take advantage of .NET capabilities. The Delphi database technologies now... more (<a href="#">see page 6</a>)</p>
ADOdbx.NET Data Types ( <a href="#">see page 11</a> )	<p>ADOdbx Client data types map to .NET logical types. Dependant upon the database, ADOdbx Client data types map to native data types. Where applicable, ADOdbx Client provides:</p> <ul style="list-style-type: none"> <li>• Consistent data type mapping across databases.</li> <li>• Logical data types mapped to .NET native types.</li> </ul>
ADO.NET Overview ( <a href="#">see page 14</a> )	<p>ADO.NET is the .NET programming environment for building database applications based on native database formats or XML data. ADO.NET is designed as a back-end data store for all Microsoft .NET programming models, including Web Forms and Web Services. Use ADO.NET to manage data in the .NET Framework.</p> <p><b>Note:</b> BDP.NET is based on ADO.NET 1.1. ADOdbx Client is based on .NET 2.0.</p> <p>CodeGear provides tools to simplify rapid ADO.NET development using ADOdbx Client and Borland Data Providers for .NET (BDP.NET). If you are familiar with rapid application development (RAD) and object oriented programming (OOP) using properties, methods, and events, you will... more (<a href="#">see page 14</a>)</p>

BDP Migration Overview (see page 16)	<p>BDP (Borland Data Provider) is being deprecated, and you should not use BDP for new development. Instead, you should use AdoDbx Client. This topic describes differences and equivalencies between BDP and AdoDbx Client. As a result of the deprecation of BDP:</p> <ul style="list-style-type: none"> <li>• BDP will be removed from the product in a future release.</li> <li>• There will be no further BDP development and minimal QA effort. Only critical bugs will be fixed.</li> <li>• No additional documentation will be provided, though documentation is not yet removed.</li> </ul> <p>BDP was based on ADO.NET 1.1. Many of the differentiating features of BDP, such as provider independence and extended... more (see page 16)</p>
Blackfish SQL Overview (see page 20)	<p>The design and implementation of Blackfish SQL emphasizes database performance, scalability, ease of use, and a strong adherence to industry standards. Blackfish SQL capabilities include the following:</p> <ul style="list-style-type: none"> <li>• Industry standards compliance</li> <li>• Entry level SQL-92</li> <li>• Unicode storage of character data</li> <li>• Unicode-based collation key support for sorting and indexing</li> <li>• dbExpress 4 drivers for win32 Delphi and C++</li> <li>• ADO.NET 2.0 providers for .NET</li> <li>• JDBC for Java</li> <li>• JavaBean data access components for Java</li> <li>• XA/JTA Distributed transactions for Java</li> <li>• High performance and scalability for demanding online transaction processing (OLTP) and decision support system (DSS) applications</li> <li>• Delphi, C#, and VB.NET stored procedures and triggers for Windows</li> <li>• Java-stored... more (see page 20)</li> </ul>
ADO.NET Component Designers (see page 21)	<p>Almost all distributed applications revolve around reading and updating information in databases. Different applications you develop using ADO.NET have different requirements for working with data. For instance, you might develop a simple application that displays data on a form. Or, you might develop an application that provides a way to share data information with another company. In any case, you need to have an understanding of certain fundamental concepts about the data approach in ADO.NET.</p> <p>Using these designers, you can work efficiently to access, expose, and edit data through database server-specific schema objects like tables, views, and indexes. These designers... more (see page 21)</p>

Deploying Database Applications for the .NET Framework (see page 24)	<p>When deploying database applications using RAD Studio, copy the necessary runtime assemblies and driver DLLs for deployment to a specified location. The following sections list the name of the assemblies and DLLs and the location of where each should reside.</p> <p><b>Note:</b> We strongly encourage you to use ADO.NET and the AdoDbx Client Provider for .NET database applications. The Borland Data Provider is being deprecated.</p>
Data Providers for Microsoft .NET (see page 27)	<p>In addition to supporting the providers included in the .NET Framework, RAD Studio includes AdoDbxClient Providers for Microsoft .NET. AdoDbx Client is an implementation of the .NET Provider and connects to a number of popular databases.</p> <p>This topic includes:</p> <ul style="list-style-type: none"> <li>• Data Provider Architecture</li> <li>• AdoDbx Client Advantages</li> <li>• AdoDbx Client and ADO.NET Components</li> <li>• Supported AdoDbx Client Providers</li> <li>• AdoDbx Client Data Types</li> <li>• AdoDbx Client Interfaces</li> </ul>
Stored Procedure Overview (see page 29)	<p>All relational databases have certain features in common that allow applications to store and manipulate data. A stored procedure is a self-contained program written in a language specific to the database system. A stored procedure typically handles frequently repeated database-related tasks, and is especially useful for operations that act on large numbers of records or that use aggregate or mathematical functions. Stored procedures are typically stored on the database server.</p> <p>Calling a stored procedure is similar to invoking a SQL command, and RAD Studio provides support for using stored procedures in much the same ways as it supports editing and... more (see page 29)</p>
dbExpress Framework (see page 30)	<p>The dbExpress framework (DBX framework) is a set of abstract classes provided in the unit DBXCommon. Applications can interface with the framework in several ways: using the framework directly for both native and managed applications, and using the dbExpress VCL components that are layered on top of the framework for both native and managed applications.</p> <p>Although many applications interface with dbExpress drivers via the dbExpress VCL components, the DBX framework offers a convenient, lighter weight option to communicate with a database driver. You can also create a database driver for dbExpress by extending the framework's DBXCommon abstract base classes. The... more (see page 30)</p>
dbExpress Framework Compatibility (see page 31)	<p>Some dbExpress software developed prior to the dbExpress driver framework (DBX driver framework) has been modified to work with the DBX driver framework. As a result of these changes, some compatibility issues arise.</p>
Getting Started with InterBase Express (see page 32)	<p>InterBase Express (IBX) is a set of data access components that provide a means of accessing data from InterBase databases. The InterBase Administration Components, which require InterBase 6, are described after the InterBase data access components.</p>

## 1.1.1 AdoDbx Client Overview

The AdoDbx Client implements an ADO.NET 2.0 provider for all dbExpress version 4 drivers that implement the newer extended metadata added to dbExpress 4. All dbExpress drivers shipped with Delphi implement the newer extended metadata.

AdoDbx Client is an implementation of the ADO.NET 2.0 Provider classes. ADO.NET Provider is a set of classes that provide database services for .NET. It provides access to relational databases, XML and application data. You can use it to develop front end database applications as well as multi-tier business applications.

See [.NET Framework Developer's Guide ADO.NET](#) in the Microsoft documentation for more information.

Here are the key classes in the AdoDbx Client ADO.NET implementation.

- TAdoDbxCommand. Represents a SQL statement or stored procedure to execute against a data source.

- TAdoDbxCommandBuilder. Generates single-table commands as part of the operation of the TAdoDbxDataAdapter.
- TAdoDbxConnection. Represents a connection to a database.
- TAdoDbxDataAdapter. Acts as a bridge between a DataSet and the underlying database.
- TAdoDbxDataReader. Class to read rows forward-only from a data source.
- TAdoDbxParameter. Represents a parameter that is passed to or from a command.
- TAdoDbxParameterCollection. Collects TAdoDbxParameters into a .NET collection object that can be read and manipulated.
- TAdoDbxProviderFactory. Base class for a provider's implementation of data source classes.
- TAdoDbxTransaction. A group of commands for a connection that can be committed or rolled back.

#### See Also

ADO.NET Overview (📖 see page 14)

[.NET Framework Developer's Guide ADO.NET \(MSDN\)](#)

Connecting to the AdoDbx Client (📖 see page 110)

[Deploying the AdoDbx Client](#)

TAdoDbxCommand

TAdoDbxCommandBuilder

TAdoDbxConnection

TAdoDbxDataAdapter

TAdoDbxDataReader

TAdoDbxParameter

TAdoDbxParameterCollection

TAdoDbxProviderFactory

TAdoDbxTransaction

---

## 1.1.2 VCL for .NET Database Technologies

In most cases, the AdoDbxClient Provider provides the best database connectivity solution for your .NET applications. However, if you are developing new VCL Forms applications for the .NET Framework, or you are migrating existing Win32 VCL Forms applications to the .NET Framework, RAD Studio provides continued support for existing Delphi database technologies.

RAD Studio provides a migration path from Delphi database technologies running strictly on Win32 clients to the .NET Framework. In addition to being able to build new database applications using ADO.NET, you can migrate existing database applications to take advantage of .NET capabilities. The Delphi database technologies now supported by RAD Studio include:

- dbExpress.NET
- DataSnap .NET Client (DCOM)
- IBX.NET (InterBase for .NET)
- ADO.NET
- dbGo

## Building .NET Applications with dbExpress.NET

RAD Studio includes a .NET version of dbExpress. This set of components provide comparable functionality as the dbExpress components for Win32, but updated to run on VCL Forms on the .NET Framework. dbExpress for .NET provides the same lightweight client capability and unidirectional dataset that is available in previous versions of the product.

## Building .NET Applications with the DataSnap .NET Client (DCOM)

RAD Studio provides the means to use the DataSnap (DCOM) client to connect to databases in three-tier applications.

## Building .NET Applications with IBX.NET

RAD Studio provides you with access to InterBase databases, by way of InterBase Express controls, in addition to the standard BDP.NET data adapter or the .NET Framework's ADO.NET providers. IBX.NET controls allow you to connect to InterBase databases, access tables, etcetera.

## Building .NET Applications with the AdoDbxClient Provider

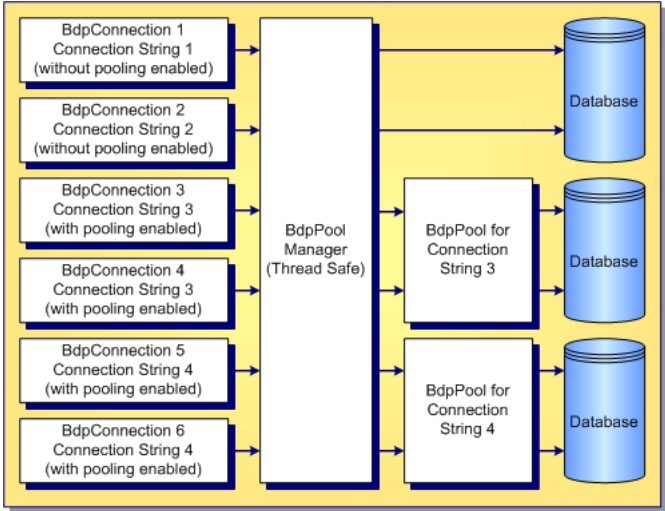
AdoDbx is a data-access mechanism that can be shared by several applications. AdoDbx defines a powerful library of API calls that can create, restructure, fetch data from, update, and otherwise manipulate local and remote database servers. Adoprovides a uniform interface to access a wide variety of database servers, using drivers to connect to different databases.

You can connect your RAD Studio database applications to BDE-supported databases, such as Paradox and dBase.

## Building .NET Applications with dbGo

RAD Studio includes a .NET version of dbGo. This set of components provides comparable functionality as the dbGo components for Win32, but updated to run on VCL Forms on the .NET Framework. dbGo for .NET provides the same powerful and logical object model that is available in previous versions of the product.

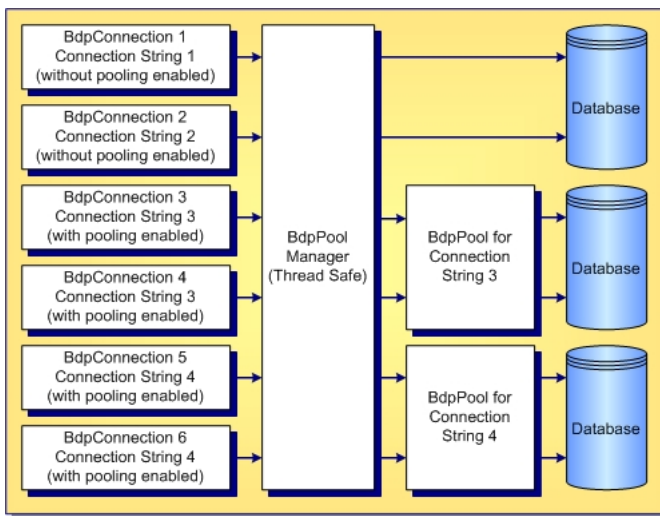
## Topics

Name	Description
BDP Connection Pooling Overview (📄 see page 8)	<p>You can use the connection pooling options to save connection time by using a connection from an existing pool. When you are using BDP, all connections go through the BDP Pool Manager, even if pooling is not enabled for your connection. For each connection, you can specify: Pooling (enabled or disabled), Minimum Pool Size, Maximum Pool Size, whether connection requests should Grow On Demand, and the number of seconds before a Connection Timeout (or number of seconds for Connection Lifetime).</p>  <p>As shown in the diagram above, the BDP Pool Manager creates a separate pool for each unique connection string.... more (📄 see page 8)</p>

dbExpress Components overview (see page 9)	dbExpress is a set of lightweight database drivers that provide fast access to SQL database servers. For each supported database, dbExpress provides a driver that adapts the server-specific software to a set of uniform dbExpress interfaces. When you deploy a database application that uses dbExpress, you might need to include a DLL (the server-specific driver) with the application files you build. For example, DbxClient is a 100% Delphi driver and needs no DLL. dbExpress lets you access databases using unidirectional datasets. Unidirectional datasets are designed for quick lightweight access to database information, with minimal overhead. Like other datasets, they can... more (see page 9)
dbGo Components Overview (see page 10)	dbGo provides the developers with a powerful and logical object model for programmatically accessing, editing, and updating data from a wide variety of data sources through Microsoft ADO system interfaces. The most common usage of dbGo is to query a table or tables in a relational database, retrieve and display the results in an application, and perhaps allow users to make and save changes to the data.  The ADO layer of an ADO-based application consists of the latest version of Microsoft ADO, an OLE DB provider or ODBC driver for the data store access, client software for the specific database... more (see page 10)

### 1.1.2.1 BDP Connection Pooling Overview

You can use the connection pooling options to save connection time by using a connection from an existing pool. When you are using BDP, all connections go through the BDP Pool Manager, even if pooling is not enabled for your connection. For each connection, you can specify: Pooling (enabled or disabled), Minimum Pool Size, Maximum Pool Size, whether connection requests should Grow On Demand, and the number of seconds before a Connection Timeout (or number of seconds for Connection Lifetime).



As shown in the diagram above, the BDP Pool Manager creates a separate pool for each unique connection string. The following connection options are available:

Options	Function
<b>MinPoolSize</b>	Specifies the minimum number of connections that will be maintained in the connection pool.
<b>MaxPoolSize</b>	Determines the maximum number of connections in the connection pool. The default maximum size is 100. If GrowOnDemand is False and MaxPoolSize is reached, subsequent connection requests will throw an exception.
<b>GrowOnDemand</b>	Specifies whether the new connection request should grow on demand after a pool reaches the MaxPool Size.  Connections that grow on demand will not be returned to the connection pool. Instead, they will be released on BdpConnection.Close().

<b>ConnectionLifetime (Timeout)</b>	Determines the life time of a pooled connection. When a connection returns to the pool, its lifetime is checked to see if it has expired. If it has, then the connection is released instead of returned to the pool. The ConnectionLifetime value is in seconds, and the default is 0.
-------------------------------------	---

## 1.1.2.2 dbExpress Components overview

dbExpress is a set of lightweight database drivers that provide fast access to SQL database servers. For each supported database, dbExpress provides a driver that adapts the server-specific software to a set of uniform dbExpress interfaces. When you deploy a database application that uses dbExpress, you might need to include a DLL (the server-specific driver) with the application files you build. For example, DbxClient is a 100% Delphi driver and needs no DLL.

dbExpress lets you access databases using unidirectional datasets. Unidirectional datasets are designed for quick lightweight access to database information, with minimal overhead. Like other datasets, they can send an SQL command to the database server, and if the command returns a set of records, obtain a reader for accessing those records. However, unidirectional datasets can only retrieve a unidirectional reader. They do not buffer data in memory, which makes them faster and less resource-intensive than other types of dataset. However, because there are no buffered records, unidirectional datasets are also less flexible than other datasets.

dbExpress connections, tables, views, and stored procedures that show up in a data tree view support drag and drop with native and managed VCL forms.

### Connection Strings

In dbExpress 4, all connection properties, including `ConnectionString`, are passed to the driver at connect time.

The `ConnectionString` property in dbExpress allows you to pass all database options and connection information (database, username, password) by means of a single connection string. This feature also allows you to introduce new properties to your drivers in the middle of a release by changing an interface.

You can load the `ConnectionProperties` in the `dbxconnections.ini` for the current `connectionName` by right clicking the connection and selecting the appropriate menu item. This creates a `Parameters` item (`Parameters['ConnectionString']`) that contains all of the connection properties in the inifile. This way you can add new properties to the `dbxconnections.ini` file, and you don't have type the whole string in yourself.

There is also a 'Clear Connection String' menu item off the `SqlConnection` right click menu, which appears whenever the `ConnectionString` property is set.

### dbExpress Components

The **dbExpress** section of the **Tool Palette** contains the following components that use **dbExpress** to access database information:

Component	Function
<code>TSQLConnection</code>	Encapsulates a dbExpress connection to a database server
<code>TSQLDataSet</code>	Represents any data available through <b>dbExpress</b> , or sends commands to a database accessed through <b>dbExpress</b>
<code>TSQLQuery</code>	A query-type dataset that encapsulates a SQL statement and enables applications to access the resulting records, if any
<code>TSQLTable</code>	A table-type dataset that represents all of the rows and columns of a single database table
<code>TSQLStoredProc</code>	A stored procedure-type dataset that executes a stored procedure defined on a database server
<code>TSQLMonitor</code>	Intercepts messages that pass between a SQL connection component and a database server and saves them in a string list

TSimpleDataSet	A client dataset that uses an internal <b>TSQLDataSet</b> and <b>TDataSetProvider</b> for fetching data and applying updates
----------------	--

**See Also**

VCL for .NET Overview (🔗 see page 71)

Porting VCL Applications (🔗 see page 69)

Deploying Database Applications for the .NET Framework

Building a VCL Forms dbExpress.NET Database Application (🔗 see page 153)

Configuring TSQL Connection

Using Data Explorer to get Connection Information

## 1.1.2.3 dbGo Components Overview

dbGo provides the developers with a powerful and logical object model for programmatically accessing, editing, and updating data from a wide variety of data sources through Microsoft ADO system interfaces. The most common usage of dbGo is to query a table or tables in a relational database, retrieve and display the results in an application, and perhaps allow users to make and save changes to the data.

The ADO layer of an ADO-based application consists of the latest version of Microsoft ADO, an OLE DB provider or ODBC driver for the data store access, client software for the specific database system used (in the case of SQL databases), a database back-end system accessible to the application (for SQL database systems), and a database. All of these must be accessible to the ADO-based application for it to be fully functional. Microsoft Data Access Components (MDAC) 2.1 or later contains these needed elements. RAD Studio supports MDAC 2.8.

The **dbGo** section of the **Tool Palette** contains the following components that use **dbGo** to access database information:

Component	Function
TADOConnection	Encapsulates a dbGo connection to a database server
TADODataSet	Represents any data available through <b>dbGo</b> , or sends commands to a database accessed through <b>dbGo</b>
TADOQuery	A query-type dataset that encapsulates an SQL statement and enables applications to access the resulting records, if any, from an ADO data store
TADOTable	A table-type dataset that represents all of the rows and columns of a single database table
TADOStoredProc	A stored procedure-type dataset that executes a stored procedure defined on a database server
TADOCommand	Represents the ADO Command object, which is used for issuing commands against a data store accessed through an ADO provider
TADODataSet	Represents a dataset retrieved from an ADO data store
TRDSCConnection	Exposes the functionality of the RDS DataSpace object

**See Also**

VCL for .NET Overview (🔗 see page 71)

Porting VCL Applications (🔗 see page 69)

Deploying Database Applications for the .NET Framework

Building a VCL Forms ADO.NET Database Application (🔗 see page 148)



## 1.1.3 AdoDbx.NET Data Types

AdoDbx Client data types map to .NET logical types. Dependant upon the database, AdoDbx Client data types map to native data types. Where applicable, AdoDbx Client provides:

- Consistent data type mapping across databases.
- Logical data types mapped to .NET native types.

### AdoDbx and .NET Framework

The DataSet class within ADO.NET uses .NET Framework data types. AdoDbx Client data types logically map .NET data types for supported databases. During designtime, you can use AdoDbx Client logical types, which will map to the appropriate native type.

### Data Types

The .NET Framework includes a wide range of logical data types. AdoDbx Client inherits logical data types, providing built-in mappings to supported databases. AdoDbx Client supports logical data type mappings for DB2, InterBase, MS SQL, MSDE, and Oracle.

### DB2

AdoDbx Client supports the following DB2 type mappings.

DB2 Type	Bdp Type	BdpSubType	System.Type
CHAR	String	stFixed	String
VARCHAR	String	NA	String
SMALLINT	Int16	NA	Int16
BIGINT	Int64	NA	Int64
INTEGER	Int32	NA	Int32
DOUBLE	Double	NA	Double
FLOAT	Float	NA	Single
REAL	Float	NA	Single
DATE	Date	NA	DateTime
TIME	Time	NA	DateTime
TIMESTAMP	Datetime	NA	DateTime
NUMERIC	Decimal	NA	Decimal
DECIMAL	Decimal	NA	Decimal
BLOB	Blob	stBinary	Byte[]
CLOB	Blob	stMemo	Char[]

### InterBase

AdoDbx Client supports the following InterBase type mappings.

InterBase Type	Bdp Type	BdpSubType	System.Type
CHAR	String	stFixed	String
VARCHAR	String	NA	String

SMALLINT	Int16	NA	Int16
INTEGER	Int32	NA	Int32
FLOAT	Float	NA	Single
DOUBLE	Double	NA	Double
BLOB Sub_Type 0	Blob	stBinary	Byte[]
BLOB Sub_Type 1	Blob	stMemo	Char[]
TIMESTAMP	Datetime	NA	DateTime

## MS SQL and MSDE

AdoDbx Client supports the following MS SQL and MSDE type mappings.

MSSQL Type	Bdp Type	BdpSubType	System.Type
BIGINT	Int64	NA	Int64
INT	Int32	NA	Int32
SMALLINT	Int16	NA	Int16
TINYINT	Int16	NA	Int16
BIT	Boolean	NA	Boolean
DECIMAL	Decimal	NA	Decimal
NUMERIC	Decimal	NA	Decimal
MONEY	Decimal	NA	Decimal
SMALLMONEY	Decimal	NA	Decimal
FLOAT	Double	NA	Double
REAL	Float	NA	Single
DATETIME	DateTime	NA	DateTime
SMALLDATETIME	DateTime	NA	DateTime
CHAR	String	stFixed	String
VARCHAR	String	NA	String
TEXT	Blob	stMemo	Char[]
BINARY	VarBytes	NA	Byte[]
VARBINARY	VarBytes	NA	Byte[]
IMAGE	Blob	stBinary	Byte[]
TIMESTAMP	VarBytes	NA	Byte[]
UNIQUEIDENTIFIER	Guid	NA	Guid

## Oracle

AdoDbx Client supports the following Oracle type mappings.

Oracle Type	Bdp Type	BdpSubType	System.Type
CHAR	String	stFixed	String
NCHAR	String	stFixed	String
VARCHAR	String	NA	String

NVARCHAR	String	NA	String
VARCHAR2	String	NA	String
NVARCHAR2	String	NA	String
NUMBER	Decimal	NA	Decimal
DATE	Date	NA	DateTime
BLOB	Blob	stHBinary	Byte[]
CLOB	Blob	stHMemo	Char[]
LONG	Blob	stMemo	Char[]
LONG RAW	Blob	stBinary	Byte[]
BFILE	Blob	stBFile	Char[]
ROWID	String	NA	String

### Sybase

AdoDbx Client supports the following Sybase type mappings.

Sybase Type	Bdp Type	BdpSubType	System.Type
CHAR	String	stFixed	String
VARCHAR	String	NA	String
INT	Int32	NA	Int32
SMALLINT	Int16	NA	Int16
TINYINT	Int16	NA	Int16
DOUBLE PRECISION	Float	NA	Single
FLOAT	Float	NA	Single
REAL	Float	NA	Single
NUMERIC	Decimal	NA	Decimal
DECIMAL	Decimal	NA	Decimal
SMALLMONEY	Decimal	NA	Decimal
MONEY	Decimal	NA	Decimal
SMALLDATETIME	DateTime	NA	DateTime
DATETIME	DateTime	NA	DateTime
IMAGE	Blob	stBinary	Byte[]
TEXT	Blob	stMemo	Char[]
BIT	Boolean	NA	Boolean
TIMESTAMP	VarBytes	NA	Byte[]
BINARY	Bytes	NA	Byte[]
VARBINARY	VarBytes	NA	Byte[]
SYSNAME	String	NA	String

### See Also

ADO.NET Overview ([see page 14](#))

## 1.1.4 ADO.NET Overview

ADO.NET is the .NET programming environment for building database applications based on native database formats or XML data. ADO.NET is designed as a back-end data store for all Microsoft .NET programming models, including Web Forms and Web Services. Use ADO.NET to manage data in the .NET Framework.

**Note:** BDP.NET is based on ADO.NET 1.1. AdoDbx Client is based on .NET 2.0.

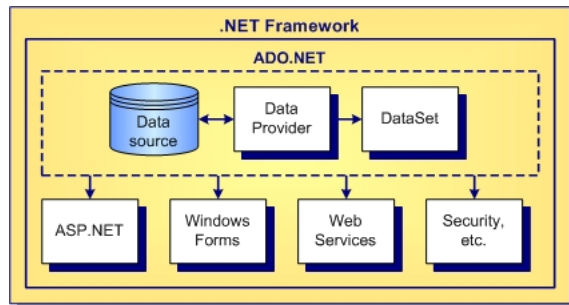
CodeGear provides tools to simplify rapid ADO.NET development using AdoDbx Client and Borland Data Providers for .NET (BDP.NET). If you are familiar with rapid application development (RAD) and object oriented programming (OOP) using properties, methods, and events, you will find the ADO.NET model for building applications familiar. If you are a traditional database programmer, ADO.NET provides familiar concepts, such as tables, rows, and columns with relational navigation. XML developers will appreciate navigating the same data with nodes, parents, siblings, and children.

This topic discusses the major components of the ADO.NET architecture, how ADO.NET integrates with other programming models in the .NET Framework, and key RAD Studio functionality to support ADO.NET.

This topic introduces:

- ADO.NET Architecture
- ADO.NET User Interfaces
- BDP.NET Namespace

### ADO.NET Architecture



The two major components of the ADO.NET architecture are the Data Provider and the DataSet. The data source represents the physical database or XML file, the Data Provider makes connections and passes commands, and the DataSet represents one or more data sources in memory. For more information about the general ADO.NET model, see the Microsoft .NET Framework SDK documentation.

### Data Source

The data source is the physical database, either local or remote, or an XML file. In traditional database programming, the developer typically works with the data source directly, often requiring complex, proprietary interfaces. With ADO.NET, the database developer works with a set of components to access the data source, to expose data, and to pass commands.

### Data Providers

Data Provider components connect to the physical databases or XML files, hiding implementation details. Providers can connect to one or more data sources, pass commands, and expose data to the DataSet.

The .NET Framework includes providers for MS SQL, OLE DB, and Oracle. In addition to supporting the .NET providers, this product includes AdoDbx Client and BDP.NET. These connect to a number of industry standard databases, providing a consistent programming environment. For more information, see the Borland Data Providers for Microsoft .NET topic.

## DataSet

The DataSet object represents in-memory tables and relations from one or more data sources. The DataSet provides a temporary work area or virtual scratch pad for manipulating data. ADO.NET applications manipulate tables in memory, not within the physical database. The DataSet provides additional flexibility over direct connections to physical databases. Much like a typical dataset object supported by many database systems, the DataSet can contain multiple DataTables, which are representations of tables or views from any number of data sources. The DataSet works in an asynchronous, non-connected mode, passing update commands through the Provider to the data source at a later time.

RAD Studio provides two kinds of DataSets for your use: standard DataSets and typed DataSets. A standard DataSet is the default DataSet that you get when you define a DataSet object implicitly. This type of DataSet is constructed based on the layout of the columns in your data source, as they are returned at runtime based on your Select statement.

Typed DataSets provide more control over the layout of the data you retrieve from a data source. A typed DataSet derives from a DataSet class. The typed DataSet lets you access tables and columns by name rather than collection methods. The typed DataSet feature provides better readability, improved code completion capabilities, and data type enforcement unavailable with standard DataSets. The compiler checks for type mismatches of typed DataSet elements at compile time rather than runtime. When you create a typed dataset, you see that some new objects are created for you and are accessible through the **Project Manager**. You will notice two files named after your dataset. One file is an XML `.xsd` file and the other is a code file in the language you are using. All of the data about your dataset, including the table and column data from the database connection, is stored in the `.xsd` file. The program code file is created based on the XML in the `.xsd` file. If you want to change the structure of the typed dataset, you can change items in the `.xsd` file. When you recompile, the program code file is regenerated based on the modified XML.

For more information about DataSets, see the Microsoft .NET Framework SDK documentation.

## ADO.NET User Interfaces

ADO.NET provides data access for the various programming models in .NET.

### Web Forms

Web Forms in ASP.NET provide a convenient interface for accessing databases over the web. ASP.NET uses ADO.NET to handle data access functions.

.NET, AdoDbx Client and BDP.NET connection components ease integration between Web Forms and ADO.NET. DB Web Controls support both ADO.NET, AdoDbx Client and BDP.NET components, accelerating web application development.

### Windows Forms

Windows Forms are no longer supported.

### AdoDbx Client Namespace

AdoDbx Client classes are found under the `Borland.Data.AdoDbxClientProvider` namespace.

### BDP.NET Namespace

BDP.NET classes are found under the `Borland.Data` namespaces.

#### *BDP.NET Namespace*

Namespace	Description
Borland.Data.Common	Contains objects common to all Borland Data Providers, including Error and Exceptions classes, data type enumerations, provider options, and Interfaces for building your own Command, Connection, and Cursor classes.

Borland.Data.Provider	Contains key BDP.NET classes like BdpCommand, BdpConnection, BdpDataAdapter, and others that provide the means to interact with external data sources, such as Oracle, DB2, Interbase, and MS SQL Server databases.
Borland.Data.Schema	Contains Interfaces for building your own database schema manipulation classes, as well as a number of types and enumerators that define metadata.

**See Also**[Deploying Applications](#)

Building an ASP.NET Database Application (see page 171)

Building an ASP.NET Application with Database Controls 1 (see page 174)

Building an ASP.NET Application with Database Controls 2 (see page 176)

Building an ASP.NET Application with Database Controls 3 (see page 177)

Building an ASP.NET Database Application (see page 171)

Data Providers for Microsoft .NET (see page 27)

AdoDbx.NET Data Types (see page 11)

ADO.NET Component Designers (see page 21)

Creating and Using Typed DataSets (see page 132)

Creating Table Mappings (see page 112)

[.NET Framework Developer's Guide ADO.NET \(MSDN\)](#)

## 1.1.5 BDP Migration Overview

BDP (Borland Data Provider) is being deprecated, and you should not use BDP for new development. Instead, you should use AdoDbx Client. This topic describes differences and equivalencies between BDP and AdoDbx Client.

As a result of the deprecation of BDP:

- BDP will be removed from the product in a future release.
- There will be no further BDP development and minimal QA effort. Only critical bugs will be fixed.
- No additional documentation will be provided, though documentation is not yet removed.

BDP was based on ADO.NET 1.1. Many of the differentiating features of BDP, such as provider independence and extended metadata, were added to ADO.NET 2 using different approaches, incompatible with BDP. In addition, ADO.NET 2 uses abstract base classes and deprecated the ADO.NET 1.1 interfaces. This made extending BDP to ADO.NET 2.0 impractical.

AdoDbx Client is based on ADO.NET 2.0 and is intended to provide most of BDP's capabilities.

BDP consists of three namespaces:

**BDP Namespaces**

BDP Namespace	Description
Borland.Data.Common	Contains objects common to all BDP.NET, including Error and Exceptions classes, data type enumerations, provider options, and Interfaces for building your own Command, Connection, and Cursor classes.

Borland.Data.Provider	Contains the main BDP.NET classes such as BdpCommand, BdpConnection, BdpDataAdapter, BdpDataReader, and others that provide the means to interact with external data sources, such as Oracle, DB2, Interbase, and MS SQL Server databases.
Borland.Data.Schema	Contains Interfaces for building your own database schema manipulation classes, as well as a number of types and enumerators that define metadata.

This document describes migration for each of these namespaces.

### Borland.Data.Provider Migration

Two classes in this namespace provide data remoting and have not been deprecated, so they do not require migration:

- DataHub
- DataSync

### Corresponding classes in BDP and AdoDbx Client

Most BDP classes in this namespace are implementations of ADO.NET classes. These classes are also implemented in AdoDbx Client. Most source code using these classes should convert to AdoDbx Client with little effort.

The following table shows the correspondence between classes in ADO.NET, BDP and AdoDbx Client:

#### *Correspondence between classes in ADO.NET, BDP and AdoDbx Client*

ADO.NET	BDP.NET	AdoDbx Client
DbCommand	BdpCommand	TAdoDbxCommand
DbCommandBuilder	BdpCommandBuilder	TAdoDbxCommandBuilder
DbConnection	BdpConnection	TAdoDbxConnection
DbDataAdapter	BdpDataAdapter	TAdoDbxDataAdapter
DbDataReader	BdpDataReader	TAdoDbxDataReader
DbTransaction	BdpTransaction	TAdoDbxTransaction

Conversion of BDP classes in this group is fairly straightforward. Check the documentation to see if the method you've used is supported in the corresponding AdoDbx Client class. If it is, you probably don't need to do anything. If the method is not supported, then you need to modify your code to use methods that are supported in either AdoDbx Client or ADO.NET itself.

For example, the class BdpDataReader accesses database records. Most of its data access methods have corresponding TAdoDbxDataReader methods, as described in the ISQLCursor section. BdpDataReader.GetSchemaTable can be used to retrieve cursor metadata as a DataTable. See the ISQLExtendedMetaData and ISQLMetaData section for a description of metadata access for AdoDbx Client.

### BDP Classes Without Corresponding AdoDbx Client Classes

The BdpCopyTable class does not have a corresponding class in AdoDbx Client, so this capability is not available in AdoDbx Client.

### Borland.Data.Common Migration

This namespace has seven classes and three interfaces.

#### BdpConnectionString

The TAdoDbxConnection class has a ConnectionString property. This class also supports connection pooling.

## BdpError and BdpErrorCollection

All errors are handled as exceptions in AdoDbx Client in TAdoDbxException, so these classes are not needed.

## BdpException, BdpParameter and BdpParameterCollection

These classes are implementations of ADO.NET classes. These classes are also implemented in AdoDbx Client. Most source code using these classes should convert to AdoDbx Client with little effort.

Conversion of BDP classes in this group is fairly straightforward. Check the documentation to see if the method you've used is supported in the corresponding AdoDbx Client class. If it is, you probably don't need to do anything. If the method is not supported, then you need to modify your code to use methods that are supported in either AdoDbx Client or ADO.NET itself.

The following table shows the correspondence between classes in ADO.NET, BDP and AdoDbx Client:

*correspondence between classes in ADO.NET, BDP and AdoDbx Client*

ADO.NET	BDP.NET	AdoDbx Client
DbException	BdpException	TAdoDbxException
DbParameter	BdpParameter	TAdoDbxParameter
DbParameterCollection	BdpParameterCollection	TAdoDbxParameterCollection

## DbResolver

DbResolver is an implementation of the ISQLResolver interface in the Borland.Data.Schema namespace, described later in this topic.

## ISQLCommand, IAQLConnection and ISQLCursor

These interfaces are mainly used by driver writers and there is no AdoDbx Client equivalent. You should rewrite the driver using the dbExpress framework.

## Borland.Data.Schema Migration

This namespace contains five interfaces.

## ISQLDataSource

The GetProviders method returns a list of data providers. A similar capability is provided by TAdoDbxProviderFactory.CreateDataSourceEnumerator, which creates an enumerator for all providers. There is no analog for the methods GetConnections or GetDbObjectTypes in AdoDbx Client.

## ISQLExtendedMetaData and ISQLMetaData

Use the GetSchema method in TAdoDbxConnection to get a metadata collection. The Name parameter of GetSchema specifies the kind of metadata to get. The standard names supported by DbConnection.GetSchema are supported in AdoDbx Client.

In addition, you can specify one of the name constants in TDBXMetaDataCollectionName to get a particular metadata collection. You would then use the corresponding class in the DBXMetaDataNames namespace to get the column information you want for that metadata collection.

For instance, to get procedure source code in a database, use the constant ProcedureSources to get a DataTable with procedure source information. Use the DBXMetaDataNames classes TDBXProcedureSourcesColumns and TDBXProcedureSourcesIndex with the returned DataTable to access the procedure source information by name or ordinal.



## ISQLResolver

This class resolves SQL commands.

The closest analog is the TAdoDbxDataAdapter class for ISQLResolver methods. This table shows the properties that correspond to methods.

### *ISQLResolver-TAdoDbxDataAdapter correspondence*

ISQLResolver	TAdoDbxDataAdapter
GetDeleteSQL method	DeleteCommand property
GetInsertSQL method	InsertCommand property
GetSelectSQL	SelectCommand property
GetUpdateSQL	UpdateCommand property

For ISQLResolver properties, the closest analog are properties that TAdoDbxCommandBuilder inherits from the ADO.NET class DbCommandBuilder and a TAdoDbxDataReader property.

### *ISQLResolver-AdoDbx Client correspondence*

ISQLResolver	AdoDbx Client
QuotePrefix property	TAdoDbxCommandBuilder.QuotePrefix property
QuoteSuffix property	TAdoDbxCommandBuilder.QuoteSuffix property
Row property	TAdoDbxDataReader.Item property

## ISQLSchemaCreate

This interface allows you to create a database schema in your own provider. The AdoDbx Client does not provide this capability.

### See Also

ADO.NET Overview (see page 14)

[.NET Framework Developer's Guide ADO.NET \(MSDN\)](#)

TAdoDbxCommand

TAdoDbxCommandBuilder

TAdoDbxConnection

TAdoDbxDataAdapter

TAdoDbxDataReader

TAdoDbxException

TAdoDbxParameter

TAdoDbxParameterCollection

TAdoDbxProviderFactory

TAdoDbxTransaction

TDBXMetaDataCollectionName

## 1.1.6 Blackfish SQL Overview

The design and implementation of Blackfish SQL emphasizes database performance, scalability, ease of use, and a strong adherence to industry standards. Blackfish SQL capabilities include the following:

- Industry standards compliance
- Entry level SQL-92
- Unicode storage of character data
- Unicode-based collation key support for sorting and indexing
- dbExpress 4 drivers for win32 Delphi and C++
- ADO.NET 2.0 providers for .NET
- JDBC for Java
- JavaBean data access components for Java
- XA/JTA Distributed transactions for Java
- High performance and scalability for demanding online transaction processing (OLTP) and decision support system (DSS) applications
- Delphi, C#, and VB.NET stored procedures and triggers for Windows
- Java-stored procedures and triggers
- Zero-administration, single assembly or single-jar deployment
- Database incremental backup and failover

### Blackfish SQL DataStore

Blackfish SQL is the name of the product, its tools, and of the file format. Within this product, there is a datastore package that includes a DataStore class, as well as several additional classes that have DataStore as part of their names.

### Blackfish SQL Compatibility

Blackfish SQL for Windows and Blackfish SQL for Java are highly compatible with one another. The database file format is binary-compatible between Blackfish SQL for Windows and Blackfish SQL for Java. In addition, database clients and servers are interchangeable. Windows clients can connect to Java servers and Java clients can connect to Windows servers.

Because the Blackfish SQL for Windows implementation is more recent, some Blackfish SQL for Java features are not yet supported. The following features are not supported:

- ISQL SQL Command Line Interpreter
- High Availability features, including incremental backup and failover
- Graphical tooling for administrative capabilities
- Access to file and object streams
- Tracking and resolving of row-level insert, update and delete operations
- Access to the Blackfish SQL File System directory

### Blackfish SQL Connectivity

This section provides an overview of the connection drivers provided for Blackfish SQL for Windows and Blackfish SQL for Java, respectively. For instructions on using the drivers to connect to a Blackfish SQL database, see the Blackfish SQL Developer's Guide, Establishing Connections section

### Blackfish SQL for Windows Connectivity

Blackfish SQL for Windows provides the following connection drivers:

- **DBXClient:** This is a win32 dbExpress 4 database driver that enables win32 Delphi and C++ applications to connect to a remote Blackfish SQL for Windows or Blackfish SQL for Java server.
- **Local ADO.NET 2.0 Provider:** This is a 100% managed code driver that enables .NET applications to connect to a local Blackfish SQL for Windows server. The local ADO.NET driver executes in the same process as the BlackFishSQL database kernel, for greater performance.
- **Remote ADO.NET 2.0 Provider:** This is a 100% managed code driver that enables .NET applications to acquire a remote connection to either a Blackfish SQL for Windows or Blackfish SQL for Java server.

### Blackfish SQL for Java Connectivity

Blackfish SQL for Java provides the following JDBC connection drivers:

- **Local JDBC driver:** This is a 100% managed code driver that enables Java applications to connect to a local Blackfish SQL for Java server. The local JDBC driver executes in the same process as the BlackFishSQL database kernel, for greater performance.
- **Remote JDBC driver:** This is a 100% managed code driver that enables Java applications to acquire a remote connection to either a Blackfish SQL for Windows or Blackfish SQL for Java server.

#### See Also

[Blackfish SQL Developer's Guide: Preface](#)

[ConnectionProperties](#)

[DataStoreErrorCode](#)

---

## 1.1.7 ADO.NET Component Designers

Almost all distributed applications revolve around reading and updating information in databases. Different applications you develop using ADO.NET have different requirements for working with data. For instance, you might develop a simple application that displays data on a form. Or, you might develop an application that provides a way to share data information with another company. In any case, you need to have an understanding of certain fundamental concepts about the data approach in ADO.NET.

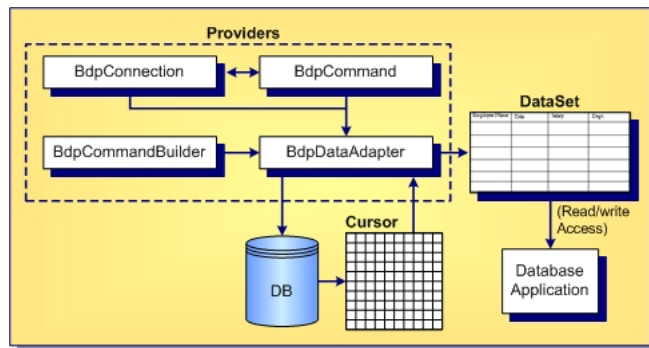
Using these designers, you can work efficiently to access, expose, and edit data through database server-specific schema objects like tables, views, and indexes. These designers allow you to use these schema objects to connect to a variety of popular databases, and perform database operations in a consistent and reliable way.

This topic includes:

- Component Designer Relationships
- Connection Editor
- Command Text Editor
- Stored Procedure Dialog Box
- Generate DataSets
- Configure Data Adapter
- Data Explorer

## Component Designer Relationships

OLD IMAGE



NEW IMAGE

The major elements of the database component designers include:

- The **Connection Editor** to define a live connection to a data source
- The **Command Text Editor** to construct command text for command components
- The **Configure Data Adapter** to set up commands for a data adapter
- The **Stored Procedure Dialog box** to view and specify values for Input or InputOutput parameters for use with command components
- The **Generate Dataset** to build custom datasets
- The **Data Explorer** to browse database server-specific schema objects and use drag-and-drop techniques to automatically populate data from a data source to your Delphi for .NET project

### Connections Editor

The **Connections Editor** manages connection strings and database-specific connection options. Using the **Connections Editor** you can add, remove, delete, rename, and test database connections. Changes to the connection information are saved into the [ADOdbxConnections.xml](#) file, where they are accessed whenever you need to create a new connection object. Once you have chosen a particular connection, the **Connections Editor** generates the connection string and any connection options, then assigns them to the `ConnectionString` and `ConnectionOptions` properties, respectively.

Display the **Connections Editor** dialog box by dragging the `TADOdbxConnection` component from the **Tool Palette** onto the form, and then clicking the component designer verb at the bottom of the **Object Inspector**.

### Command Text Editor

The **Command Text Editor** can be used to construct the command text for command components that have a `CommandText` property. A multi-line editing control in the editor lets you manually edit the command or build the command text by selecting tables and columns. Display the **Command Text Editor** dialog box by dragging a `TADOdbxCommand` component from the **Tool Palette** onto the form, and clicking the designer verb at the bottom of the **Object Inspector**.

The **Command Text Editor** is a simplified version of a SQL builder capable of generating SQL for a single table. The database objects are filtered by the `SchemaName` property set in `ISQLSchemaCreate` and only tables that are part of that schema are used. If there is no `SchemaName` listed, all of the available objects for the current login user are listed. The `QuoteObjects` setting for the `ConnectionOptions` property determines whether the objects are quoted with the database-specific quote character or not. This is important, for instance, when retrieving tables from databases that allow table names to include spaces.

To populate the Tables and Columns list boxes with items and build SQL statements, you must have defined a live `TADOdbxConnection`. Otherwise, data cannot be retrieved. The **Command Text Editor** allows you to choose table and column names from a list of available tables and columns. Using this information, the editor generates a SQL statement. To generate the

SQL, the editor uses an instance of the `TAdoDbxCommandBuilder`. When you request optimized SQL, the editor uses index information to generate the WHERE clause for SELECT, UPDATE, and DELETE statements; otherwise, non-BLOB columns and searchable columns form the WHERE clause.

When the SQL is generated, the `TAdoDbxCommand.CommandText` property is set to the generated SQL statement.

### Stored Procedure Dialog Box

The **Stored Procedure** dialog box is used to view and enter Input and InputOutput parameters for a stored procedure and to execute the stored procedure. Display the **Stored Procedure** dialog box by dragging a `TAdoDbxCommand` component from the **Tool Palette** onto the form, setting the `CommandType` property for the `TAdoDbxCommand` component to **StoredProcedure**, and clicking the **Command Text Editor** designer verb at the bottom of the **Object Inspector**.

The **Stored Procedure** dialog box lets you select a stored procedure from a list of available stored procedures, which is determined by the `TAdoDbxConnection` specified in the `Connection` property for the `TAdoDbxCommand` component. When you select a stored procedure, the dialog box displays the parameters associated with the stored procedure, and the parameter metadata for the selected parameter. You can specify values for Input or InputOutput parameters and execute the stored procedure. If the stored procedure returns results, such as Output parameters, InputOutput parameters, return values, cursor(s) returned, they are all populated into a `DataGrid` in the bottom of the dialog box when the stored procedure is executed. After the `CommandText` and `Parameters`, properties are all set for the `TAdoDbxCommand`, the stored procedure can be executed at runtime by making a single call to `ExecuteReader` or `ExecuteNonQuery`.

### Generate DataSets

The **Generate Dataset** designer is used to build a `DataSet`. Using this tool results in strong typing, cleaner code, and the ability to use code completion. A `DataSet` is first derived from the base `DataSet` class and then uses information in an XML Schema file (an `.xsd` file) to generate a new class. Information from the schema (tables, columns, and so on) is generated and compiled into this new dataset class as a set of first-class objects and properties. Display this dialog box by dragging a `TAdoDbxDataAdapter` component from the **Tool Palette** onto the form, and clicking the component designer verb at the bottom of the **Object Inspector**. If this component is not displayed, choose **Component ► Installed .NET Components** to add it to the **Tool Palette**.

### Configure Data Adapter

The **Configure Data Adapter** designer is used to generate SELECT, INSERT, UPDATE, and DELETE SQL statements. After successful SQL generation, the **Configure Data Adapter** designer creates new `TAdoDbxCommand` objects and adds them to the `TAdoDbxDataAdapter.SelectCommand`, `DeleteCommand`, `InsertCommand`, and `UpdateCommand` properties.

After successful SQL SELECT generation, you can preview data and generate a new `DataSet`. You can also use an existing `DataSet` to populate a new `DataTable`. If you create a new `DataSet`, it will be added automatically to the designer host. You can also generate Typed `DataSets`.

Data Adapters are an integral part of the ADO.NET managed providers. Essentially, Adapters are used to exchange data between a data source and a dataset. This means reading data from a database into a `DataSet`, and then writing changed data from the `DataSet` back to the database. A Data Adapter can move data between any source and a `DataSet`. Display the **Configure Data Adapter** dialog box by dragging a `TAdoDbxDataAdapter` component from the **Tool Palette** onto the form, and clicking the component designer verb at the bottom of the **Object Inspector**.

### Data Explorer

The **Data Explorer** is a hierarchical database browser and editing tool. The **Data Explorer** is integrated into the IDE and can also be run as a standalone executable. To access the **Data Explorer** within the IDE, choose **View ► Data Explorer**. Use the context menus in the **Data Explorer** to perform the following tasks:

- Manage database connections—add a new connection, modify, delete, or rename your existing connections
- Browse database structure and data—expand and open provider nodes to browse database server-specific schema objects including tables, views, stored procedure definitions, and indexes
- Add and modify tables—specify the data structure for a new table, or add or remove columns, and alter column information for

an existing table

- View and test stored procedure parameters—specify values for Input or InputOutput parameters and execute the selected stored procedure
- Migrate data—migrate table schema and data of one or more tables from one provider to another
- Drag-and-drop schema objects onto forms to simplify application development—drag tables or stored procedures onto your application form for the .NET Framework to add connection components and automatically generate connection strings

The **Data Explorer** provides connectivity to several industry-standard databases, and can be extended to connect to other popular databases. The **Data Explorer** uses the ISQLDataSource interface to get a list of available providers, database connections, and schema objects that are supported by different providers. The list of available providers is persisted in the TAdoDbxDataSources.xml file, and the available connections are persisted in the TAdoDbxConnections.xml file. Once you have chosen a provider the ISQLMetadata interface is used to retrieve metadata and display a read-only tree view of database objects. The current implementation provides a list of tables, views, and stored procedures for all AdoDbx Client-supported databases.

The **Data Explorer** lets you create new tables, alter or drop existing tables, migrate data from multiple tables from one provider to another, and copy and paste individual tables across ADO-supported databases. For all these operations, the **Data Explorer** calls into the ISQLSchemaCreate implementation of the provider.

Additionally, the **Data Explorer** can be used to drag data from a data source to any RAD Studio project for the .NET framework. Dragging a table onto a form adds TAdoDbxConnection and TAdoDbxDataAdapter components to your application and automatically configures the TAdoDbxDataAdapter for the given table. Dragging a stored procedure onto a form adds TAdoDbxConnection and TAdoDbxCommand components to your application, and sets the CommandType property of the TAdoDbxCommand object to StoredProcedure.

#### See Also

ADO.NET Overview (see page 14)

AdoDbx.NET Data Types (see page 11)

Using the Command Text Designer (see page 127)

Using the Connection Editor Designer (see page 128)

Using the Data Adapter Designer (see page 128)

Using the Data Adapter Preview (see page 126)

Using the Generate Dataset Designer (see page 132)

Migrating Data Between Databases (see page 115)

Creating Table Mappings (see page 112)

---

## 1.1.8 Deploying Database Applications for the .NET Framework

When deploying database applications using RAD Studio, copy the necessary runtime assemblies and driver DLLs for deployment to a specified location. The following sections list the name of the assemblies and DLLs and the location of where each should reside.

**Note:** We strongly encourage you to use ADO.NET and the AdoDbx Client Provider for .NET database applications. The Borland Data Provider is being deprecated.

## ADO.NET 2.0 Application Deployment

### Deploying by Updating machine.config

The microsoft .net machine.config file must have an entry added for the provider. Locate machine.config below your Windows\Microsoft.net directory tree and add the following entry in the <DbProviderFactories> section:

```
<c>
<add name="AdoDbx Data Provider" invariant="Borland.Data.AdoDbxClient"
description=".Net Framework Data Provider for dbExpress Drivers"
type="Borland.Data.TAdoDbxProviderFactory, Borland.Data.AdoDbxClient,
Version=11.0.5000.0,Culture=neutral,PublicKeyToken=91d62ebb5b0d1b1b" />
</c>
```

### Deploying Without Updating machine.config

An application can also deploy without updating the machine config by directly using the TAdoDbxProviderFactory class in the Borland.Data.AdoDbxClientProvider unit. Although this makes deployment easier, the TAdoDbxProviderFactory can only create ADO.NET 2.0 objects for the AdoDbxClient provider.

If your application links with assemblies, the [Borland.Data.AdoDbxClient.dll](#), and [Borland.Data.DbxCCommonDriver.dll](#) assembly must be either copied to the same directory as your executable or registered in the GAC.

If [dbxconnections.ini](#) and [dbxdrivers.ini](#) are used, they will have to also be deployed as well. If native dynalink drivers are being used, those drivers and their corresponding client libraries will need to be deployed.

See [Borland.Data.AdoDbxClientProvider.pas](#) source file for information.

## BDP.NET Application Deployment

Copy specific database runtime assemblies to the following location:

Managed Assemblies	Data Provider	Location
Borland.Data.Common.dll	All	GAC
Borland.Data.Provider.dll	All	GAC
Borland.Data.DB2.dll	DB2	GAC
Borland.Data.Interbase.dll	Interbase	GAC
Borland.Data.Mssql.dll	MS SQL/MSDE	GAC
Borland.Data.Oracle.dll	Oracle	GAC
Borland.Data.Msacc.dll	MS Access	GAC
Borland.Data.Sybase.dll	Sybase	GAC

**Note:** If you are deploying a distributed database application that uses the BDP.NET Remoting components, such as DataHub, DataSync, RemoteConnection, and RemoteServer, you must install Borland.Data.DataSync.dll

to the GAC. Copy unmanaged database driver DLLs to the following location:

DLLs	Data Provider	Location
bdpint20.dll	Interbase	search path
bdpdb220.dll	DB2	search path

bdpmss20.dll	MS SQL/MSDE	search path
bdpora20.dll	Oracle	search path
bdpmsa20.dll	MS Access	search path
bdpsyb20.dll	Sybase	search path

### dbExpress for .NET Application Deployment

Copy specific database runtime assemblies to the following location:

Managed Assemblies	Data Provider	Location
Borland.VclDbExpress.dll	All	GAC
Borland.VclDbCtrls.dll	All	GAC
Borland.VclDbxCds.dll	Required by database applications that use client datasets	GAC
Borland.Common.Driver.dll	All	GAC

You can deploy associated dbExpress.NET drivers and DataSnap DLLs with your executable. Copy unmanaged database driver DLLs to the following location:

DLLs	Data Provider	Location
dbxINT30.dll	InterBase 2007, 7.5.1, 7.1*, 7.0*, 6.5*	search path
dbxASA30.dll	Adaptive Sybase Anywhere 9, 8*	search path
dbxDB230.dll	DB2 UDB 8.x, 7.x*	search path
dbxINF30.dll	Informix 9.x	search path
dbxMSS30.dll	MSSQL 2005, 2000	search path
dbxMYSA30.dll	MySQL 4.0.24	search path
dbxMYS30.dll	MySQL 5.0.27, 4.1.22*	search path
dbxora30.dll	Oracle 10g, 9.2.0*, 9.1.0*	search path
dbxASE30.dll	Sybase 12.5	search path
Midas.dll	Required by database applications that use client datasets	search path

**Note:** \* Driver not fully certified with this version of the database.

### dbGo for .NET Application Deployment

There is no need to deploy runtime assemblies or database drivers for dbGo components used in VCL.NET applications. Microsoft Data Access Components (MDAC) version 2.1 or later is required to run applications with dbGo components outside of the IDE. This applies to Win32 VCL applications, as well as VCL.NET applications. RAD Studio supports MDAC 2.8.

### BDE for .NET Application Deployment

When deploying BDE-based applications, you must include the BDE with your application. While this increases the size of the application and the complexity of deployment, the BDE can be shared with other BDE-based applications and provides a broader range of support for database manipulation. Although you can use the API of the BDE directly in your application, the components on the **BDE** section of the **Tool Palette** wrap most of this functionality for you.

### See Also

[Borland Overview of Deploying Applications](#)

[Microsoft Overview of Deploying Applications](#)



## 1.1.9 Data Providers for Microsoft .NET

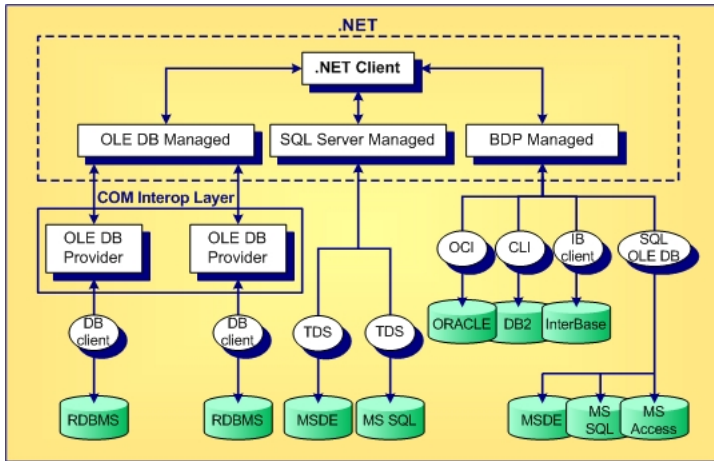
In addition to supporting the providers included in the .NET Framework, RAD Studio includes AdoDbxClient Providers for Microsoft .NET. AdoDbx Client is an implementation of the .NET Provider and connects to a number of popular databases.

This topic includes:

- Data Provider Architecture
- AdoDbx Client Advantages
- AdoDbx Client and ADO.NET Components
- Supported AdoDbx Client Providers
- AdoDbx Client Data Types
- AdoDbx Client Interfaces

### Data Provider Architecture

RAD Studio supports the .NET Framework providers and the AdoDbx Client providers.



AdoDbx.NET provides a high performance architecture for accessing data sources without a COM Interop layer.

The architecture exposes a set of interfaces for third-party integration. You can implement these interfaces for your own database to provide design-time, tools, and runtime data access integration into the CodeGear IDE. AdoDbx.NET -managed components communicate with these interfaces to accomplish all basic data access functionality. These interfaces were implemented to wrap database-specific native client libraries by way of Platform Invoke (P/Invoke) services. Depending on the availability of managed database clients, you can implement a fully-managed provider underneath AdoDbx.NET.

The database-specific implementation is wrapped into an assembly and the full name of the assembly is passed to the AdoDbxConnection component as part of the connection string. Depending on the Assembly entry in the ConnectionString property, AdoDbx.NET dynamically loads the database-specific provider and consumes the implementation for ISQLConnection, ISQLCommand, and ISQLCursor. This allows you to switch applications from one database to another just by changing the ConnectionString property to point to a different provider.

### AdoDbx.NET Advantages

AdoDbx.NET provides a number of advantages:

- Unified programming model applicable to multiple database platforms

- High performance data-access architecture
- Open architecture, which supports additional databases easily
- Portable code to write once and connect to any supported databases
- Consistent data type mapping across databases where applicable
- Logical data types mapped to .NET native types
- No need for a COM Interop layer, unlike OLE DB
- Lets you view live data as you design your application
- Extends ADO.NET to provide interfaces for metadata services, schema creation, and data migration
- Rich set of component designers and tools to speed database application development

RAD Studio extends .NET support to additional database platforms, providing a consistent connection architecture and data type mapping.

### AdoDbx.NET and ADO.NET Components

The DataSet is an in-memory representation of one or more DataTables. Each DataTable in a DataSet consists of DataColumnns and DataRowns. The DataSet is generated as a result of an SQL query that you supply to the provider. You can navigate the DataSet like you would any standard relational table. AdoDbx.NET providers encapsulate implementation details for each database type, yet allow you to customize your SQL statements and manage the result sets with complete flexibility.

AdoDbx.NET includes several designtime components that you can place onto a Windows Form or Web Form. A set of designers are also provided to help you build your data connections, DataSets, relations, and other elements.

The primary components that are most useful, particularly if you decide to implement your own database-specific provider, are:

- AdoDbxConnection—establishes a database connection
- AdoDbxCommand—includes a set of methods and properties for SQL and stored procedure execution
- AdoDbxDataReader—retrieves data
- AdoDbxParameter—supports runtime parameter binding
- AdoDbxTransaction—supports transaction control
- AdoDbxDataAdapter—provides and resolves data
- ISQLMetaData—retrieves metadata
- ISQLSchemaCreate—includes methods for creating, dropping, and altering database objects

For more information, click on the link for each component, or search for the components in the API reference documentation in this Help.

### Supported AdoDbx.NET Providers

AdoDbx.NET includes providers for a number of industry-standard databases. These are shown in the following table, along with their corresponding namespaces.

Database	Namespace
InterBase	Borland.Data.Interbase
Oracle	Borland.Data.Oracle
IBM DB2	Borland.Data.Db2
Microsoft SQL Server	Borland.Data.Mssql
Microsoft Access	Borland.Data.Msacc
Sybase	Borland.Data.Sybase

The AdoDbx.NET components, metadata access, and designers are defined under the following namespaces:

- `Borland.Data.AdoDbxClientProvider`
- `Borland.Data.Common`
- `Borland.Data.Schema`
- `Borland.Data.Design`

### AdoDbx.NET Data Types

AdoDbx.NET maps SQL data types to .NET Framework data types, eliminating the need for you to learn a database-specific type system. Every attempt has been made to implement consistent type mappings across database types, allowing you to write one set of source that you can run against multiple databases. You can achieve a similar effect with the .NET Framework data providers by communicating with their interfaces directly and by using untyped ancestors. However, once you use strongly typed accessors, your application becomes less portable. AdoDbx.NET does not support any database-specific typed accessors. For more information, see the AdoDbx.NET Data Types topic.

### AdoDbx.NET Interfaces

You can extend AdoDbx.NET to support other DBMSs by implementing a subset of the .NET Provider interface. AdoDbx.NET generalizes much of the functionality required to implement data providers. While the .NET Framework gives you the capabilities to create individual data providers for each data source, CodeGear has simplified the task by offering a generalized set of capabilities. Instead of building separate providers, along with corresponding DataAdapters, DataReaders, Connection objects, and other required objects, you can implement a set of AdoDbx.NET interfaces to build your own data source plug-ins to the AdoDbx Client Provider.

Building plug-ins is a much easier task than building a completely new data provider. You build an assembly that contains the namespace for your provider, as well as classes that encapsulate provider-specific functionality. Much of the functionality you need to connect to, execute commands against, and retrieve data from your data source has already been defined in the AdoDbx Client Provider interfaces.

### See Also

ADO.NET Overview (see page 14)

AdoDbx.NET Component Designers (see page 21)

AdoDbx.NET Data Types (see page 11)

---

## 1.1.10 Stored Procedure Overview

All relational databases have certain features in common that allow applications to store and manipulate data. A stored procedure is a self-contained program written in a language specific to the database system. A stored procedure typically handles frequently repeated database-related tasks, and is especially useful for operations that act on large numbers of records or that use aggregate or mathematical functions. Stored procedures are typically stored on the database server.

Calling a stored procedure is similar to invoking a SQL command, and RAD Studio provides support for using stored procedures in much the same ways as it supports editing and using SQL command text.

Stored procedures can enhance your database applications in the following ways: improve the performance, security, and reliability of your applications.

- **Performance**—stored procedures can improve the performance of a database application by taking advantage of the server's usually greater processing power and speed, and reducing network traffic by moving processing to the server. Also, the compiled SQL used in a stored procedure executes faster typically than standard SQL command text.
- **Security**—by creating a layer between clients and the database, stored procedures can enhance security for your data. You don't need to grant database permissions to individual users. Instead, you can grant users permission to execute a stored procedure independently of underlying table permissions.

- Reliability—stored procedures help to centralize code, which makes it easier to isolate and troubleshoot problems. Also, stored procedures allow you to move business logic which is inherent to the database into the database, thus making it available from all clients regardless of the language they are written in.

When you use the ADOdbx Client **Command Text Editor** and the **Data Explorer**, both provide the ability to view your stored procedure parameters, specify input parameters, and execute your stored procedures as you design your application.

#### See Also

ADO.NET Component Designers (see page 21)

Providers for Microsoft .NET (see page 27)

## 1.1.11 dbExpress Framework

The dbExpress framework (DBX framework) is a set of abstract classes provided in the unit DBXCommon. Applications can interface with the framework in several ways: using the framework directly for both native and managed applications, and using the dbExpress VCL components that are layered on top of the framework for both native and managed applications.

Although many applications interface with dbExpress drivers via the dbExpress VCL components, the DBX framework offers a convenient, lighter weight option to communicate with a database driver. You can also create a database driver for dbExpress by extending the framework's DBXCommon abstract base classes. The DBX framework provides most commonly needed database driver functionality for a "set" oriented database application, yet provides a simple interface.

Here are some of the key features of the DBX framework:

- The driver framework is written entirely in Delphi and allows drivers to be written in Delphi.
- It uses strongly typed data access instead of pointers. For instance, it uses String types rather than pointers to strings.
- The driver framework is single sourced. This means that a single copy of the source can be compiled with either the native DCC32 or managed DCCIL compilers.
- The framework has only Abstract base classes that are used for drivers, connections, commands, readers, and so on.
- The framework uses exception based error handling rather than returning error codes.

#### Capabilities

There are two categories of drivers that extend the classes in DBXCommon: DBXDynaLink and DBXDirect. These drivers differ from each other in the way they are loaded and the capabilities they provide to an application. These are described in greater detail later.

You can also extend the DBX framework to write *delegation drivers*, which provide an extra layer between the application and the actual driver. Delegate drivers are useful for connection pooling, driver profiling, tracing, and auditing. Another possible application of driver delegation is to create a thread safe driver delegate. Such a delegate could provide thread synchronized access to all public methods.

Absolute thread safety is left to applications using dbExpress. However, some thread safety issues are best handled by the dbExpress framework. dbExpress thread safe operations include loading and unloading drivers, and connection creation. As mentioned earlier, a delegate driver can be created to make the entire public interface of dbExpress thread safe if needed.

A dbExpress 4 driver can statically or dynamically link drivers built as Delphi packages. The easiest way to link a driver package is to just include it in the "uses" clause. The driver loader also loads packages specified in a config or ini file using the LoadPackage method. This allows dynamic loading of drivers that are never specified in a uses clause of any of the application's units. Note that the LoadPackage approach can only be employed for applications built to use packages.

dbExpress 4 driver writers should examine the initialization sections of the DBXDynalink and DBXTrace units in the source code provided with dbExpress. These sections register themselves with a singleton unit called the ClassRegistry. The ClassRegistry is

used by the dbExpress 4 driver loader to instantiate driver loader classes by name (a String). The ClassRegistry is a simple, lightweight mechanism for registering and instantiating a class by name.

### DBXDynalink Drivers

DBXDynalink is used for existing dbExpress 3 drivers as well as new drivers. It is compiled as a native Delphi package or as a managed .NET assembly. DBXDynalink loads native dbExpress drivers that implement a more primitive "native" interface called DBXExports. The DBXExports interface is a small collection of "flat" export methods. DBXExports's source is included with dbExpress. DBXExports provides a more strongly typed API than the dbExpress 3's COM based interface. This allows methods to be added in future product generations without breaking compatibility with older implementations of the DBXExports interface.

DBXAdapter is a dbExpress 4 compliant driver that adapts the DBXExports interface to the older dbExpress 3 COM interface. Newer native drivers can be written by implementing DBXExports directly.

Because the DBXExports interface is designed to be implemented using any native language (Delphi or C++), it uses more primitive, non-exception based error handling. DBXDynalink maps error codes to a DBXCommon exception.

The DBXDynalink unit contains a dbExpress 4 driver. This driver delegates to non-Delphi drivers that implement the DBXDynalinkExport flat export interface. DBXTrace is a delegate driver used for tracing. The dbExpress VCL uses DBXCommon, DBXDynalink and DbxTrace as "default" drivers. However, this can be changed for statically linked applications without modifying dbExpress VCL source code (SQLEXP.pas). SQLEXP.pas uses the unit DBXDefaultDrivers. The DBXDefaultDrivers unit only contains a uses clause. The DBXDefaultDrivers uses clause contains DBXCommon, DBXDynalink, and DBXTrace. DBXCommon must always be used. However, a statically linked application could remove DBXTrace and replace DBXDynalink with a different driver.

### DBXDirect Drivers

A DBXDirect driver is any driver that is implemented by extending the DBXCommon abstract base classes. These classes are written in Delphi for native implementations. For managed implementations, they can be written using any CLR compatible language such as Delphi, C#, or Visual Basic.NET.

Strictly speaking, all DBX framework drivers are a form of DBXDirect driver. However DBXDynalink and DBXRemote provide a more "indirect" linkage to driver implementations.

### See Also

dbExpress Framework Compatibility (see page 31)

Deploying dbExpress Database Applications

---

## 1.1.12 dbExpress Framework Compatibility

Some dbExpress software developed prior to the dbExpress driver framework (DBX driver framework) has been modified to work with the DBX driver framework. As a result of these changes, some compatibility issues arise.

### General

dbExpress 2.5 drivers *cannot* be used with the DBX framework.

The dbExpress framework does not provide 100% compatibility with dbExpress 3.

dbExpress 3 drivers can be used with the DBX framework. The DBX framework driver loader automatically detects dbExpress 3 drivers and uses the DBXAdapter driver (dbxadapter30.dll) to make a dbExpress 3 driver look like a dbExpress 4 driver.

Here is a list of known compatibility issues:

- Static driver linkage. You cannot statically link dbExpress drivers into an executable.
- SqlExpr.TSQLConnection provided protected access to the Connection member that was of type TISQLConnection only in

the native version of `SqlExpr.pas`. This was omitted from the managed version due to the complexity of how `PlInvoke` was used in the managed version of the `dbExpress VCL`. `SqlExpr.TSQLConnection` now provides protected access to a `TDBXConnection` instead. This protected connection is accessible to both native and managed applications.

- The event for trace monitoring is slightly different because it is based on the `DBX` driver framework.
- The `DBXadapter` driver can adapt `dbExpress 3` drivers to `dbExpress 4`, but not `dbExpress 2.5` drivers.

### VCL issues

Most applications using `dbExpress VCL` components should work without modification. However, there are some localized changes to `VCL` components due to `VCL` now interfacing to the more object oriented `DBX` driver framework instead of the C-like COM-based `dbExpress 3` driver interface.

In addition, the API has changed slightly for two of the `VCL` components: `TSQLConnection` and `TSQLDataSet`. Some data structures have also changed. A summary of the API changes follows.

**Note:** Because of API changes, you must recompile `SqlExpr.pas`, which is provided with the product. The `DBXpress` unit has been deprecated.

- `TSQLConnection`. The `Commit` method has been deprecated in favor of the new `CommitFreeAndNil` method. The `Rollback` method has been deprecated in favor of the new `RollbackFreeAndNil` and `RollbackIncompleteFreeAndNil` methods. The `SetTraceCallbackEvent` method has been replaced by `SetTraceEvent`. The `StartTransaction` method has been deprecated in favor of the new `BeginTransaction` method. The `MetaData` property contains an instance of the new class `TDBXDatabaseMetaData` instead of `TISQLMetaData`. The `SQLConnection` property has been replaced by `DBXConnection`, which contains an instance of the new class `TDBXConnection`. The `TraceCallbackEvent` property now contains a `TDBXTraceEvent`.
- `TSQLDataSet`. A new property `DbxCommandType` has been added, which contains one of the constant strings in the `TDBXCommandTypes` class.
- Data structures. `TTransactionItem` has been deprecated, replaced by the new `TDBXTransaction` class. `TSQLDriverOption`, `TSQLConnectionOption`, `TSQLCommandOption`, `TSQLCursorOption`, `TSQLMetaDataOption`, and `TSQLObjectType` are obsolete. `TSTMTParamType` has been replaced by the `TDBXParameterDirections` class. `TSQLTraceFlag` has been replaced by `TDBXTraceFlags`. `SQLTRACEDesc` is replaced by `TDBXTraceInfo`.

### See Also

[dbExpress Framework](#)





[Deploying the dbExpress Framework](#)









## 1.1.13 Getting Started with InterBase Express

InterBase Express (IBX) is a set of data access components that provide a means of accessing data from InterBase databases. The InterBase Administration Components, which require InterBase 6, are described after the InterBase data access components.

### IBX components

The following components are located on the InterBase tab of the component palette.

Icon	Component Name	Description
	TIBTable	A dataset component that encapsulates a database table.
	TIBQuery	Executes an InterBase SQL statement.
	TIBStoredProc	Encapsulates a stored procedure on a database server.
	TIBDatabase	Encapsulates an InterBase database connection.

	TIBTransaction	Provides discrete transaction control over a one or more database connections in a database application.
	TIBUpdateSQL	Provides an object for updating read-only datasets when cached updates are enabled.
	TIBDataSet	Executes InterBase SQL statements.
	TIBSQL	Provides an object for executing an InterBase SQL statement with minimal overhead.
	TIBDatabaseInfo	Returns information about the attached database.
	TIBSQLMonitor	Monitors dynamic SQL passed to the InterBase server.
	TIBExtract	Fetches metadata from an InterBase server.
	TIBCustomDataSet	The base class for all datasets that represent data fetched using InterBase Express.

Though they are similar to BDE components in name, the IBX components are somewhat different. For each component with a BDE counterpart, the sections below give a discussion of these differences.

There is no simple migration from BDE to IBX applications. Generally, you must replace BDE components with the comparable IBX components, and then recompile your applications. However, the speed you gain, along with the access you get to the powerful InterBase features make migration well worth your time.

### IBDatabase

Use a TIBDatabase component to establish connections to databases, which can involve one or more concurrent transactions. Unlike BDE, IBX has a separate transaction component, which allows you to separate transactions and database connections.

To set up a database connection:

1. Drop an IBDatabase component onto a form or data module.
2. Fill out the DatabaseName property. For a local connection, this is the drive, path, and filename of the database file. Set the Connected property to true.
3. Enter a valid username and password and click OK to establish the database connection.

**Warning:** Tip: You can store the username and password in the IBDatabase component's Params property by setting the LoginPrompt property to false after logging in. For example, after logging in as the system administrator and setting the LoginPrompt property to false, you may see the following when editing the Params property:

```
user_name=sysdba
password=masterkey
```

### IBTransaction

Unlike the Borland Database Engine, IBX controls transactions with a separate component, TIBTransaction. This powerful feature allows you to separate transactions and database connections, so you can take advantage of the InterBase two-phase commit functionality (transactions that span multiple connections) and multiple concurrent transactions using the same connection.

Use an IBTransaction component to handle transaction contexts, which might involve one or more database connections. In most cases, a simple one database/one transaction model will do.

To set up a transaction:

1. Set up an IBDatabase connection as described above.

2. Drop an IBTransaction component onto the form or data module
3. Set the DefaultDatabase property to the name of your IBDatabase component.
4. Set the Active property to true to start the transaction.

#### **IBX dataset components**

There are a variety of dataset components from which to choose with IBX, each having their own characteristics and task suitability:

##### **IBTable**

Use an IBTable component to set up a live dataset on a table or view without having to enter any SQL statements.

IBTable components are easy to configure:

1. Add an IBTable component to your form or data module.
2. Specify the associated database and transaction components.
3. Specify the name of the relation from the TableName drop-down list.
4. Set the Active property to true.

##### **IBQuery**

Use an IBQuery component to execute any InterBase DSQL statement, restrict your result set to only particular columns and rows, use aggregate functions, and join multiple tables.

IBQuery components provide a read-only dataset, and adapt well to the InterBase client/server environment. To set up an IBQuery component:

1. Set up an IBDatabase connection as described above.
2. Set up an IBTransaction connection as described above.
3. Add an IBQuery component to your form or data module.
4. Specify the associated database and transaction components.
5. Enter a valid SQL statement for the IBQuery's SQL property in the String list editor.
6. Set the Active property to true

##### **IBDataSet**

Use an IBDataSet component to execute any InterBase DSQL statement, restrict your result set to only particular columns and rows, use aggregate functions, and join multiple tables. IBDataSet components are similar to IBQuery components, except that they support live datasets without the need of an IBUpdateSQL component.

The following is an example that provides a live dataset for the COUNTRY table in [employee.gdb](#):

1. Set up an IBDatabase connection as described above.
2. Specify the associated database and transaction components.
3. Add an IBDataSet component to your form or data module.
4. Enter SQL statements for the following properties: SelectSQL, RefreshSQL, ModifySQL, DeleteSQL, InsertSQL. See the following table for example SQL statements.
5. Set the Active property to true.

#### **Sample SQL statements**

Property	SQL Statement
SelectSQL	SELECT Country, Currency FROM Country
RefreshSQL	SELECT Country, Currency FROM Country WHERE Country = :Country



ModifySQL	UPDATE Country SET Country = :Country, Currency = :Currency WHERE Country = :Old_Country
DeleteSQL	DELETE FROM Country WHERE Country = :Old_Country
InsertSQL	INSERT INTO Country (Country, Currency) VALUES (:Country, :Currency)

**Note: Note:** Parameters and fields passed to functions are case-sensitive in dialect 3. For example,

```
FieldByName(EmpNo)
```

would return nothing in dialect 3 if the field was 'EMPNO'.

### IBStoredProc

Use TIBStoredProc for InterBase executable procedures: procedures that return, at most, one row of information. For stored procedures that return more than one row of data, or "Select" procedures, use either IBQuery or IBDataSet components.

### IBSQL

Use an TIBSQL component for data operations that need to be fast and lightweight. Operations such as data definition and pumping data from one database to another are suitable for IBSQL components.

In the following example, an IBSQL component is used to return the next value from a generator:

1. Set up an IBDatabase connection as described above.
2. Put an IBSQL component on the form or data module and set its Database property to the name of the database.
3. Add an SQL statement to the SQL property string list editor, for example:

```
SELECT GEN_ID(MyGenerator, 1) FROM RDB$DATABASE
```

### IBUpdateSQL

Use an TIBUpdateSQL component to update read-only datasets. You can update IBQuery output with an IBUpdateSQL component:

1. Set up an IBQuery component as described above.
2. Add an IBUpdateSQL component to your form or data module.
3. Enter SQL statements for the following properties: DeleteSQL, InsertSQL, ModifySQL, and RefreshSQL.
4. Set the IBQuery component's UpdateObject property to the name of the IBUpdateSQL component.
5. Set the IBQuery component's Active property to true.

### IBSQLMonitor

Use an TIBSQLMonitor component to develop diagnostic tools to monitor the communications between your application and the InterBase server. When the TraceFlags properties of an IBDatabase component are turned on, active IBSQLMonitor components can keep track of the connection's activity and send the output to a file or control.

A good example would be to create a separate application that has an IBSQLMonitor component and a Memo control. Run this secondary application, and on the primary application, activate the TraceFlags of the IBDatabase component. Interact with the primary application, and watch the second's memo control fill with data.

### IBDatabaseInfo

Use an TIBDatabaseInfo component to retrieve information about a particular database, such as the sweep interval, ODS version, and the user names of those currently attached to this database.

For example, to set up an IBDatabaseInfo component that displays the users currently connected to the database:

1. Set up an IBDatabase connection as described above.
2. Put an IBDatabaseInfo component on the form or data module and set its Database property to the name of the database.
3. Put a Memo component on the form.
4. Put a Timer component on the form and set its interval.
5. Double click on the Timer's OnTimer event field and enter code similar to the following:

```
Memo1.Text := IBDatabaseInfo.UserNames.Text; // Delphi example
Memo1->Text = IBDatabaseInfo->UserNames->Text; // C++ example
```

**IBEvents**

Use an *IBEvents* component to register interest in, and asynchronously handle, events posted by an InterBase server.












To set up an IBEvents component:

1. Set up an IBDatabase connection as described above.
2. Put an IBEvents component on the form or data module and set its Database property to the name of the database.
3. Enter events in the Events property string list editor, for example: `IBEvents.Events.Add( 'EVENT_NAME' );` (for Delphi) or `IBEvents->Events->Add( "EVENT_NAME" );` (for C++).
4. Set the Registered property to true.

**InterBase Administration Components**

If you have InterBase 6 installed, you can use the InterBase 6 Administration components, which allow you to use access the powerful InterBase Services API calls.

The components are located on the InterBase Admin tab of the IDE and include:

	TIBConfigService
	TIBBackupService
	TIBRestoreService
	TIBValidationService
	TIBStatisticalService
	TIBLogService
	TIBSecurityService
	TIBLicensingService
	TIBServerProperties
	TIBInstall
	TIBUnInstall

Note: You must install InterBase 6 to use these features.

**IBConfigService**

Use an TIBConfigService object to configure database parameters, including page buffers, async mode, reserve space, and sweep interval.

**IBBackupService**

Use an TIBBackupService object to back up your database. With IBBackupService, you can set such parameters as the blocking factor, backup file name, and database backup options.

**IBRestoreService**

Use an `TIBRestoreService` object to restore your database. With `IBRestoreService`, you can set such options as page buffers, page size, and database restore options.

**IBValidationService**

Use an `TIBValidationService` object to validate your database and reconcile your database transactions. With the `IBValidationService`, you can set the default transaction action, return limbo transaction information, and set other database validation options.

**IBStatisticalService**

Use an `TIBStatisticalService` object to view database statistics, such as data pages, database log, header pages, index pages, and system relations.

**IBLogService**

Use an `TIBLogService` object to create a log file.

**IBSecurityService**

Use an `TIBSecurityService` object to manage user access to the InterBase server. With the `IBSecurityService`, you can create, delete, and modify user accounts, display all users, and set up work groups using SQL roles.

**IBLicensingService**

Use an `TIBLicensingService` component to add or remove InterBase software activation certificates.

**IBServerProperties**

Use an `TIBServerProperties` component to return database server information, including configuration parameters, and version and license information.

**IBInstall**

Use an `TIBInstall` component to set up an InterBase installation component, including the installation source and destination directories, and the components to be installed.

**IBUnInstall**

Use an `TIBUnInstall` component to set up an uninstall component.

## 1.2 Developing Applications with Unmanaged Code

RAD Studio provides the capability to work with the .NET features that support unmanaged code.

If you have COM or ActiveX components that you want to use within the .NET Framework, you can use the .NET COM Interop capabilities from within RAD Studio while building your applications.

### Topics

Name	Description
Using COM Interop in Managed Applications ( <a href="#">see page 38</a> )	COM Interop is a .NET service that allows seamless interoperability between managed and unmanaged code. The COM Interop service is a two-way bridge: It allows you to leverage existing COM servers and ActiveX Controls in new .NET applications, as well as to expose .NET components in legacy, unmanaged applications.  The RAD Studio IDE features tools that will help you integrate your legacy COM servers and ActiveX Controls into managed applications. Within the IDE, you can add references to unmanaged DLLs to your project, and then browse the types contained in them, just as you can with managed assemblies. You can... more ( <a href="#">see page 38</a> )
Using DllInterop ( <a href="#">see page 43</a> )	The <code>drinterop</code> command line tool examines an assembly and produces a set of diagnostic messages that help you prepare the assembly for use with COM/Interop.  The <code>drinterop</code> tool is located in the <code>bin</code> directory of the product installation. It is invoked by typing
Deploying COM Interop Applications ( <a href="#">see page 43</a> )	Two things are important to keep in mind when working with unmanaged components. First, remember that an interop assembly is not a replacement for the COM server; it is a stand-in, or proxy for it. The interop assemblies produced by <code>tlbimp</code> and RAD Studio are not transformations of the component's unmanaged code into managed code. Every file required by the component in an unmanaged deployment environment, must also be deployed in a managed environment <i>in addition to</i> the interop assemblies. Second, the .NET Framework's interop services do not circumvent the requirement of registering the COM server on the end-user's machine.... more ( <a href="#">see page 43</a> )
Using Platform Invoke with Delphi for .NET ( <a href="#">see page 44</a> )	This topic describes the basic techniques of using unmanaged APIs. Some of the common mistakes and pitfalls are pointed out, and a quick reference for translating Delphi data types is provided. This topic does not attempt to explain the basics of platform invoke or marshaling data. Please refer to the links at the end of this topic for more information on platform invoke and marshaling. Understanding attributes and how they are used is also highly recommended before reading this document.  The Win32 API is used for several examples. For further details on the API functions mentioned, please see the Windows... more ( <a href="#">see page 44</a> )
Virtual Library Interfaces ( <a href="#">see page 52</a> )	This topic describes how to use a feature of Delphi called Virtual Library Interfaces. Virtual Library Interfaces allows you to discover, load, and call unmanaged code at runtime, without the use of the <code>DllImport</code> attribute.

### 1.2.1 Using COM Interop in Managed Applications

COM Interop is a .NET service that allows seamless interoperability between managed and unmanaged code. The COM Interop service is a two-way bridge: It allows you to leverage existing COM servers and ActiveX Controls in new .NET applications, as well as to expose .NET components in legacy, unmanaged applications.

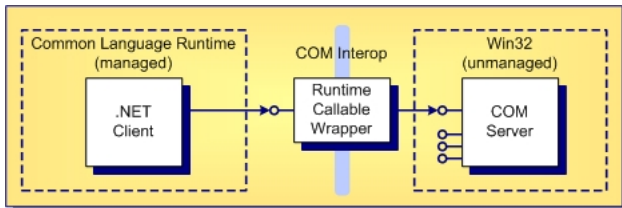
The RAD Studio IDE features tools that will help you integrate your legacy COM servers and ActiveX Controls into managed applications. Within the IDE, you can add references to unmanaged DLLs to your project, and then browse the types contained in them, just as you can with managed assemblies. You can add ActiveX Controls to the **Tool Palette**, and then drop them on

your forms as you would with any .NET component.

The following topics are covered in this overview:

- Introduction to the terminology of COM Interop. If you are already familiar with these concepts, you can skip directly to the section on RAD Studio IDE features and tools for COM/Interop.
- Introduction to some of the .NET Framework SDK tools for working with COM/Interop.
- Using COM Interop Assemblies in the IDE.

## COM Interop Overview



Seamless interoperability is achieved through stand-in objects called Runtime Callable Wrappers (RCW). The RCW is a layer of communication between your managed application, and the actual unmanaged COM server.

## COM Interop Terminology

The .NET Framework has a rich collection of terms and three-letter acronyms. This section will help you understand the terminology you will encounter when reading other COM Interop literature.

### Metadata

In the context of .NET and COM, metadata is a term used to mean type information. In COM, type information can be stored in a variety of ways. For instance, a C++ header file is a language-specific container for type information. A type library is also a container for type information, but being a binary format, type libraries are language neutral. Unlike the COM development model where type libraries are not required, language neutral metadata is mandatory for all .NET assemblies. Every assembly is self-describing; its metadata contains complete type information, including private types and private class members.

### Custom Attributes

Developers often tag program entities (such as classes and their methods) with descriptive attributes such as static, private, protected, and public. In the .NET Framework, you can tag any entity, including classes, properties, methods, and even assemblies themselves, with an attribute of your own design and meaning. Custom attributes are expressed in source code, and are processed by the compiler. At the end of the build process, custom attributes are emitted into the output assembly just like all metadata.

### Reflection

A unique characteristic of the .NET Framework is that type information is not lost during the compilation process. Instead, all metadata, including custom attributes, is emitted by the compiler into the final output assembly. Metadata is available at runtime, through .NET Reflection services. The .NET Framework SDK provides a reflection tool called `ildasm` that allows the developer to open any .NET assembly, and inspect the types declared therein. Such reflection tools often allow programmers to directly view the IL code generated by the compiler. The RAD Studio IDE contains its own integrated reflection tool, in the form of the meta data explorer tool that appears when you open a .NET assembly.

### Global Assembly Cache

In COM, components can be deployed anywhere on the user's machine. Usually, a component's installation script records its location in the system registry. Command-line tools such as `regsvr32` and `treregsvr` can also add and remove COM components from the registry. Registration of components is required in COM programming, even if the components are not intended to be shared by multiple applications.

The .NET programming model drastically simplifies deployment of applications and components. On the .NET platform, non-shared components are deployed directly into the application's local installation directory; no registration is required. Alternatively, a non-shared component can be deployed in a directory specified in the application's configuration file. Again, registration is not required for this deployment scenario.

Shared components are installed into a special location called the Global Assembly Cache (GAC). The GAC is an evolution of the system registry (though it is a completely separate mechanism and is not associated with the registry at all). The GAC exists in the file system in a folder called `\Windows\Assembly`. The .NET Framework supports simultaneous, or "side-by-side" deployment of different versions of the same component. When you view the Global Assembly Cache folder using Windows Explorer, you are actually looking at the GAC through a special shell extension. The shell extension presents all of the assemblies that have been installed into the GAC, with their version, culture, and public key information.

There are three ways to install a .NET component into the GAC. The first way is to use the Framework SDK command-line tool called `gacutil`, which is discussed below. Another way is to install a component into the GAC is to navigate to the `\Windows\Assembly` folder using Windows Explorer, and then simply drag and drop the assembly into the directory listing pane. Finally, you can also use the .NET Configuration management tool, which is accessible through the Windows Control Panel.

### Strong Names

The concept of a strong name is similar to that of the 128-bit Globally Unique Identifier (GUID) in COM programming. A GUID is a name that is guaranteed to be globally unique. Every .NET assembly has a basic name, which consists of a text string, a version number, and optional culture information. For shared assemblies installed into the GAC, the basic name alone is not enough to guarantee the assembly is uniquely identified. To generate a globally unique name, an encryption key with public and private components is used to generate a digital signature. The signature is then applied to the assembly using the .NET Framework SDK Assembly Linker (`al.exe`), or by using assembly attributes in source code.

### Runtime Callable Wrappers and COM Callable Wrappers

Accessing a component, be it a .NET component or a COM server, is largely transparent. That is, when you are using a COM server in a .NET application, the COM server looks like any other .NET component. Similarly a .NET component, when exposed to an unmanaged application through COM Interop, looks like a COM server. This transparency is accomplished by behind-the-scenes proxies, or wrapper objects.

When you use a COM object in a managed application, the Common Language Runtime (CLR) creates an RCW, which is the interface between managed and unmanaged code. The complexities of data marshaling and reference counting are handled by the RCW. In fact the RCW does not even expose the *IUnknown* and *IDispatch* interfaces.

When you use a .NET component in an unmanaged application, the system creates a stand-in called a COM Callable Wrapper (CCW).

### Primary Interop Assembly

In the COM programming model, once a GUID is assigned to a type, the GUID always refers to that specific type no matter where the type appears. For example, a common interface might be defined in many different type libraries, but each separate type library would have to define the interface with the same GUID, so the duplication is not a problem. However, if you generate COM Interop assemblies for these separate type libraries, a new and distinct assembly would be created for each type library. Each of these separate assemblies would contain distinct types (as far as the CLR is concerned). The strong identity and self-describing nature of .NET assemblies is actually working against you in this case. Here, it is leading to a GAC that is cluttered with interop assemblies that all contain RCWs for the same type library. Worse yet, to the CLR each assembly contains distinct and incompatible types, because each one has a different strong name.

To avoid this proliferation of assemblies and potential type incompatibilities, the framework gives you the ability to designate one assembly as the primary interop assembly for a type library. A primary interop assembly is always signed with a strong name, by the original publisher of the type library.

## COM Interop Tools in the .NET Framework SDK

Some of the functionality provided by the .NET Framework SDK tools is exposed in the development environment. This section is not intended to be a complete reference for these tools; it is merely a starting point for more exploration of the .NET Framework SDK, and hopefully will give you a bit more understanding of how to work with COM Interop technology in the IDE.

### Importing and Exporting Type Libraries

`Tlbimp` is a command-line tool that you can use to generate a .NET assembly from a type library. `Tlbimp` will operate on a type library directly, or on an unmanaged DLL that contains a type library as an embedded resource. Note the assembly produced by `tlbimp` contains code for only the RCW, not for the original COM object itself. Therefore you must still deploy and register the COM object on the end-user's machine. The assembly also contains the types described in the type library, expressed as metadata. `Tlbimp` uses a command line switch to produce a primary interop assembly.

The .NET Framework SDK contains another command-line tool called `tlbexp` that is used to create a type library from a .NET assembly. Such an exported type library would then be used to expose the .NET component as a COM server, for use within an unmanaged application.

### Importing ActiveX Control Libraries

`Aximp` is a command-line tool that generates an ActiveX Control wrapper assembly. This assembly is required so that the ActiveX Control can be used on a Windows Form. A special utility is required, because a Windows Form can only host controls that are derived from the `System.Windows.Forms.Control` class, and the `tlbimp` utility does not create a wrapper derived from that class.

The `aximp` tool will generate both interop assemblies (as with `tlbimp`, this includes dependent assemblies), and the ActiveX wrapper assembly. Like `tlbimp`, `aximp` has command-line switches to sign the assemblies produced with a strong name. Unlike `tlbimp`, `aximp` cannot generate a primary interop assembly.

### Generating Strong Names

If you are deploying a .NET component into the GAC, you will need to sign your assembly with a strong name key. This is done by using a .NET Framework SDK command-line tool called `sn`. The assembly is signed with the strong name in one of three ways:

- By specifying the strong name key file in the assembly linker (`al`) command line
- By tagging the assembly with the `AssemblyKeyFile` attribute
- By using a technique called "delay signing"

When using delay signing, the assembly is signed with the public portion of the key file at build time. Before shipping the assembly, the `sn` tool is used again to sign the assembly with the private key.

### Deploying a .NET Component to the Global Assembly Cache

The .NET Framework SDK utility called `gacutil` is a command-line program that is used to install, remove, and view components in the GAC. The `gacutil` command is usable from installation scripts as well as from batch files. The `gacutil` command supports installation and removal of shared assemblies, with and without the use of reference counting. It is recommended that the non-reference counted command switches be used only during development. Installation scripts that use `gacutil` to install shared components should always use the reference counted command line switches.

### Using COM Interop Assemblies in the IDE

All of the functionality encompassed by the .NET Framework SDK command-line tools is in fact exposed by the .NET Framework Class Library itself. The RAD Studio IDE also takes advantage of the .NET Framework classes to expose interoperability features. The IDE goes beyond the capabilities of the command-line tools, however, making interoperation with unmanaged components even easier.

## Type Libraries and Interop Assemblies

The IDE initiates the creation of interop assemblies through the **Project Manager**. When you add a reference to a DLL to your project, you can select from registered type libraries and unmanaged DLLs, or you can browse to an unregistered component.

The IDE creates one interop assembly for each imported type library or DLL. The assemblies are named `Interop.LibraryName.dll`, where *LibraryName* is the name of the type library. The name of the library is specified in the library statement in IDL source code, so the file name of the generated assembly might be different from that of the original DLL or type library. Each interop assembly (and all of its dependent assemblies) are added to your project as referenced assemblies. The types contained in the interop assembly are added to a namespace with the same name as the type library; Again, this is derived from the library statement in IDL source code.

If the assembly you reference has a primary interop assembly, the IDE will recognize this and avoid generating a new interop assembly. In this case, the IDE will add a reference to the primary interop assembly in the GAC, and it will not copy the assembly to your local project directory.

## Importing ActiveX Controls

To use an ActiveX Control in your managed application, you must first add the control to the tool palette. This will create both an interop assembly, and an ActiveX assembly with a wrapper class derived from `System.Windows.Forms.AxHost`. The ActiveX wrapper assembly will be named `AxInterop.LibraryName.dll`, where *LibraryName* is the name of the type library. Dragging the control from the palette onto a Windows Form will automatically add references to both assemblies to your project.

Once on your form, the ActiveX Control can be treated as any other .NET component. You can select the control, and set its properties and event handlers in the **Object Inspector**. The ActiveX Control wrapper will expose the properties of the `Windows.Forms.Control` class, while properties exposed by the ActiveX Control will be grouped under the *Misc* category.

## Interop Assemblies and the Project Manager

Interop assemblies (including ActiveX Control wrapper assemblies) generated by the IDE are kept in a separate folder called `COMImports`, underneath your project. Each generated assembly will have its 'Copy Local' property set, meaning that when the project is built, the assembly will be copied to the folder where the final build target of the project is kept. The exceptions to this rule are primary interop assemblies, which are deployed in the GAC. When you add a reference to a primary interop assembly, the IDE will not copy the assembly to the `COMImports` folder. The assembly will still be shown in the **Project Manager**, however, if you right click on it to display its properties, you will notice that the 'Copy Local' setting is turned off.

The list of referenced assemblies (including those that are not interop assemblies) is an attribute of your project. If the `COMImports` folder (or one of the interop assemblies contained therein) does not exist when you open a project, the IDE will attempt to recreate it. If the IDE cannot create an interop assembly, it will still be shown as a referenced assembly in the **Project Manager**; the IDE will highlight such an assembly so that you know it currently does not exist (or is not registered) on the machine.

## See Also

Adding a Reference to a COM Server (🔗 see page 139)

Adding an ActiveX Control to the Tool Palette

Using Platform Invoke with Delphi (🔗 see page 44)

Virtual Library Interfaces (🔗 see page 52)



## 1.2.2 Using DrInterop

The `drinterop` command line tool examines an assembly and produces a set of diagnostic messages that help you prepare the assembly for use with COM/Interop.

The `drinterop` tool is located in the `bin` directory of the product installation. It is invoked by typing

```
drinterop assembly
```

Message	Cause
Assembly <code>ComVisible</code> attribute is true when it should be false.	The <code>[assembly:ComVisible(bool)]</code> attribute is set to true, or is not present. Assemblies should be hidden from COM to reduce registry clutter. Set the <code>ComVisible</code> attribute to false, and selectively expose classes and interfaces.
Assembly, class, or interface is exposed to COM but does not contain the <code>Guid</code> attribute.	The assembly, class, or interface has the <code>ComVisible</code> attribute set to true but does not contain a <code>Guid</code> attribute.
A type library should be generated and registered for assembly.	This message is generated when a type library is not found in the same directory as the assembly.
Assembly does not contain the <code>TypeLibVersion</code> attribute.	The assembly does not contain the <code>[assembly:TypeLibVersion(x,y)]</code> attribute. By default type library version numbers are generated using only the first two numbers of the assembly version. Using the <code>TypeLibVersion</code> attribute can help avoid problems where two assemblies would produce the same type library because the first two digits of their version number are the same.
Reduce registry size by adding attribute <code>[ClassInterface(ClassInterfaceType.None)]</code> to class.	The class does not contain the <code>ClassInterface</code> attribute. By default, each class will cause the creation of a corresponding interface with the class name prefixed with an underscore character. This interface has no methods associated with it. You can reduce registry size and clutter by putting the <code>[ClassInterface(ClassInterfaceType.None)]</code> attribute on the class.

**Note:** The `drinterop`

tool will not print any messages if it does not find any of the above conditions.

## 1.2.3 Deploying COM Interop Applications

Two things are important to keep in mind when working with unmanaged components. First, remember that an interop assembly is not a replacement for the COM server; it is a stand-in, or proxy for it. The interop assemblies produced by `tlbimp` and RAD Studio are not transformations of the component's unmanaged code into managed code. Every file required by the component in an unmanaged deployment environment, must also be deployed in a managed environment *in addition to* the interop assemblies. Second, the .NET Framework's interop services do not circumvent the requirement of registering the COM server on

the end-user's machine. Note the registration requirement also applies during the development of your managed application.

As with any other .NET assembly, an interop assembly can be deployed alongside the managed executable in the installation folder, or it can be deployed in the GAC. If you deploy the interop assembly into the GAC, you must give it a strong name during development. Primary interop assemblies are always deployed into the GAC; however, just because an assembly is deployed to the GAC, does not automatically make it a primary interop assembly. An interop assembly is designated as a primary interop assembly by using the `/primary` command-line option of the `tlbimp` utility. The IDE currently has no built-in support for creating primary interop assemblies. Unmanaged COM servers can be deployed anywhere on the end-user's machine, however, as noted previously, you must still register unmanaged components when your application is installed.

#### See Also

[CodeGear Overview of Deploying Applications](#)

[Microsoft Overview of Deploying Applications](#)

## 1.2.4 Using Platform Invoke with Delphi for .NET

This topic describes the basic techniques of using unmanaged APIs. Some of the common mistakes and pitfalls are pointed out, and a quick reference for translating Delphi data types is provided. This topic does not attempt to explain the basics of platform invoke or marshaling data. Please refer to the links at the end of this topic for more information on platform invoke and marshaling. Understanding attributes and how they are used is also highly recommended before reading this document.

The Win32 API is used for several examples. For further details on the API functions mentioned, please see the Windows Platform SDK documentation.

The following topics are discussed in this section:

- Calling unmanaged functions
- Structures
- Callback functions
- Passing Object References
- Using COM Interfaces

#### Calling Unmanaged Functions

When calling unmanaged functions, a managed declaration of the function must be created that represents the unmanaged types. In many cases functions take pointers to data that can be of variable types. One example of such a function is the Win32 API function `SystemParametersInfo` that is declared as follows:

```
BOOL SystemParametersInfo(  
    UINT uiAction, // system parameter to retrieve or set  
    UINT uiParam,  // depends on action to be taken  
    PVOID pvParam, // depends on action to be taken  
    UINT fWinIni   // user profile update option  
);
```

Depending on the value of `uiAction`, `pvParam` can be one of dozens of different structures or simple data types. Since there is no way to represent this with one single managed declaration, multiple overloaded versions of the function must be declared (see [Borland.Vcl.Windows.pas](#)), where each overload covers one specific case. The parameter `pvParam` can also be given the generic declaration `IntPtr`. This places the burden of marshaling on the caller, rather than the built in marshaler. Note that the data types used in a managed declaration of an unmanaged function must be types that the default marshaler supports. Otherwise, the caller must declare the parameter as `IntPtr` and be responsible for marshaling the data.

## Data Types

Most data types do not need to be changed, except for pointer and string types. The following table shows commonly used data types, and how to translate them for managed code:

Unmanaged Data Type	Managed Data Type	
	Input Parameter	Output Parameter
Pointer to string (PChar)	String	StringBuilder
Untyped parameter/buffer	TBytes	TBytes
Pointer to structure (PRect)	const TRect	var TRect
Pointer to simple type (PByte)	const Byte	var Byte
Pointer to array (PInteger)	array of Integer	array of Integer
Pointer to pointer type (^PInteger)	IntPtr	IntPtr

IntPtr can also represent all pointer and string types, in which case you need to manually marshal data using the Marshal class. When working with functions that receive a text buffer, the StringBuilder class provides the easiest solution. The following example shows how to use a StringBuilder to receive a text buffer:

```
function GetText(Window: HWND; BufSize: Integer = 1024): string;
var
  Buffer: StringBuilder;
begin
  Buffer := StringBuilder.Create(BufSize);
  GetWindowText(Window, Buffer, Buffer.Capacity);
  Result := Buffer.ToString;
end;
```

The StringBuilder class is automatically marshaled into an unmanaged buffer and back. In some cases it may not be practical, or possible, to use a StringBuilder. The following examples show how to marshal data to send and retrieve strings using SendMessage:

```
procedure SetText(Window: HWND; Text: string);
var
  Buffer: IntPtr;
begin
  Buffer := Marshal.StringToHGlobalAuto(Text);
  try
    Result := SendMessage(Window, WM_SETTEXT, 0, Buffer);
  finally
    Marshal.FreeHGlobal(Buffer);
  end;
end;
```

An unmanaged buffer is allocated, and the string copied into it by calling StringToHGlobalAuto. The buffer must be freed once it's no longer needed. To marshal a pointer to a structure, use the Marshal.StructureToPtr method to copy the contents of the structure into the unmanaged memory buffer.

The following example shows how to receive a text buffer and marshal the data into a string:

```
function GetText(Window: HWND; BufSize: Integer = 1024): string;
var
  Buffer: IntPtr;
begin
  Buffer := Marshal.AllocHGlobal(BufSize * Marshal.SystemDefaultCharSize);
  try
    SendMessage(Window, WM_GETTEXT, BufSize, Buffer);
    Result := Marshal.PtrToStringAuto(Buffer);
  finally
    Marshal.FreeHGlobal(Buffer);
  end;
```

```
end;
end;
```

It is important to ensure the buffer is large enough, and by using the `SystemDefaultCharSize` method, the buffer is guaranteed to hold `BufSize` characters on any system.

## Advanced Techniques

When working with unmanaged API's, it is common to pass parameters as either a pointer to something, or `NULL`. Since the managed API translations don't use pointer types, it might be necessary to create an additional overloaded version of the function with the parameter that can be `NULL` declared as `IntPtr`.

## Special Cases

There are cases where a `StringBuilder` and even the `Marshal` class will be unable to correctly handle the data that needs to be passed to an unmanaged function. An example of such a case is when the string you need to pass, or receive, contains multiple strings separated by `NULL` characters. Since the default marshaler will consider the first `NULL` to be the end of the string, the data will be truncated (this also applies to the `StringToHGlobalXXX` and `PtrToStringXXX` methods). In this situation `TBytes` can be used (using the **PlatformStringOf** and **PlatformBytesOf** functions in `Borland.Delphi.System` to convert the byte array to/from a string). Note that these utility functions do not add or remove terminating `NULL` characters.

When working with COM interfaces, the `UnmanagedType` enumeration (used by the `MarshalAsAttribute` class) has a special value, `LPStruct`. This is only valid in combination with a `System.Guid` class, causing the marshaler to convert the parameter into a Win32 GUID structure. The function `CoCreateInstance` that is declared in Delphi 7 as:

```
function CoCreateInstance([MarshalAs(UnmanagedType.LPStruct)] clsid: TCLSID;
  [MarshalAs(UnmanagedType.IUnknown)] unkOuter: TObject;
  dwClsContext: Longint;
  [MarshalAs(UnmanagedType.LPStruct)] iid: TIID;
  [MarshalAs(UnmanagedType.Interface)] out pv
): HRESULT;
```

This is currently the only documented use for `UnmanagedType.LPStruct`.

## Structures

The biggest difference between calling unmanaged functions and passing structures to unmanaged functions is that the default marshaler has some major restrictions when working with structures. The most important are that dynamic arrays, arrays of structures and the `StringBuilder` class cannot be used in structures. For these cases `IntPtr` is required (although in some cases **string** paired with various marshaling attributes can be used for strings).

## Data Types

The following table shows commonly used data types, and how to "translate" them for managed code:

Unmanaged Data Type	Managed Data Type	
	Input Parameter	Output Parameter
Pointer to string ( <b>PChar</b> )	String	IntPtr
Character array (array[a..b] of Char)	String	String
Array of value type (array[a..b] of Byte)	array[a..b] of Byte	array[a..b] of Byte
Dynamic array (array[0..0] of type)	IntPtr	IntPtr
Array of struct (array[1..2] of TRect)	IntPtr or flatten	IntPtr or flatten
Pointer to structure (PRect)	IntPtr	IntPtr
Pointer to simple type (PByte)	IntPtr	IntPtr
Pointer to array (PInteger)	IntPtr	IntPtr

Pointer to pointer type (^PInteger)	IntPtr	IntPtr
-------------------------------------	--------	--------

When working with arrays and strings in structures, the MarshalAs attribute is used to describe additional information to the default marshaler about the data type. A record declared in Delphi 7, for example:

```
type
  TMyRecord = record
    IntBuffer: array[0..31] of Integer;
    CharBuffer: array[0..127] of Char;
    lpszInput: LPTSTR;
    lpszOutput: LPTSTR;
  end;
```

Would be declared as follows in RAD Studio:

```
type
  [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]
  TMyRecord = record
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 32)]
    IntBuffer: array[0..31] of Integer;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 128)]
    CharBuffer: string;
    [MarshalAs(UnmanagedType.LPTStr)]
    lpszInput: string;
    lpszOutput: IntPtr;
  end;
```

The above declarations assume that the strings contain platform dependant **TChar**'s (as commonly used by the Win32 API). It is important to note that in order to receive text in `lpszOutput`, the Marshal. `AllocHGlobal` method needs to be called before passing the structure to an API function.

A structure can contain structures, but not pointers to structures. For such cases an `IntPtr` must be declared, and the Marshal. `StructureToPtr` method used to move data from the managed structure into unmanaged memory. Note that `StructureToPtr` does not allocate the memory needed (this must be done separately). Be sure to use Marshal. `SizeOf` to determine the amount of memory required, as Delphi's **SizeOf** is not aware of the MarshalAs attribute (in the example above, `CharBuffer` would be 4 bytes using Delphi's **SizeOf** when it in fact should occupies 128 bytes on a single byte system). The following examples show how to send messages that pass pointers to a structure:

```
procedure SetRect(Handle: HWND; const Rect: TRect);
var
  Buffer: IntPtr;
begin
  Buffer := Marshal.AllocHGlobal(Marshal.SizeOf(KindOf(TRect)));
  try
    Marshal.StructureToPtr(TObject(Rect), Buffer, False);
    SendMessage(Handle, EM_SETRECT, 0, Buffer);
  finally
    Marshal.DestroyStructure(Buffer, KindOf(TRect));
  end;
end;

procedure GetRect(Handle: HWND; var Rect: TRect);
var
  Buffer: IntPtr;
begin
  Buffer := Marshal.AllocHGlobal(Marshal.SizeOf(KindOf(TRect)));
  try
    SendMessage(Handle, EM_GETRECT, 0, Buffer);
    Rect := TRect(Marshal.PtrToStructure(Buffer, KindOf(TRect)));
  finally
    Marshal.DestroyStructure(Buffer, KindOf(TRect));
  end;
end;
```

It is important to call `DestroyStructure` rather than `FreeHGlobal` if the structure contains fields where the marshaling layer needs to free additional buffers (see the documentation for `DestroyStructure` for more details).

### Advanced topics

Working with unmanaged API's it is not uncommon to need to convert a byte array into a structure (or retrieve one or more fields from a structure held in a byte array), or vice versa. Although the `Marshal` class contains a method to retrieve the offset of a given field, it is extremely slow and should be avoided in most situations. Informal performance tests show that for a structure with eight or nine numeric fields, it is much faster to allocate a block of unmanaged memory, copy the byte array to the unmanaged memory and call `PtrToStructure` than finding the position of just one field using `Marshal.OffsetOf` and converting the data using the `BitConverter` class. `Borland.Vcl.WinUtils` contains helper functions to perform conversions between byte arrays and structures (see `StructureToBytes` and `BytesToStructure`).

### Special cases

There are cases where custom processing is required, such as sending a message with a pointer to an array of integers. For situations like this, the `Marshal` class provides methods to copy data directly to the unmanaged buffer, at specified offsets (so you can construct an array of a custom data type after allocating a buffer). The following example shows how to send a message where the `LParam` is a pointer to an array of `Integer`:

```
function SendArrayMessage(Handle: HWND; Msg: UINT; WParam: WPARAM;
    LParam: TIntegerDynArray): LRESULT;
var
    Buffer: IntPtr;
begin
    Buffer := Marshal.AllocHGlobal(Length(LParam) * SizeOf(Integer));
    try
        Marshal.Copy(LParam, 0, Buffer, Length(LParam));
        Result := SendMessage(Handle, Msg, WParam, Buffer);
    finally
        Marshal.FreeHGlobal(Buffer);
    end;
end;
```

### Callback Functions

When passing a function pointer for a managed function to an unmanaged API, a reference must be maintained to the delegate or it will be garbage collected. If you pass a pointer to your managed function directly, a temporary delegate will be created, and as soon as it goes out of scope (at the end of `MyFunction` in the example below), it is subject to garbage collection. Consider the following Delphi 7 code:

```
function MyFunction: Integer;
begin
    ...
    RegisterCallback(@MyCallback);
    ...
end;
```

In order for this to work in a managed environment, the code needs to be changed to the following:

```
const
    MyCallbackDelegate: TFNMyCallback = @MyCallback;

function MyFunction: Integer;
begin
    ...
    RegisterCallback(MyCallbackDelegate);
    ...
end;
```

This will ensure that the callback can be called as long as `MyCallbackDelegate` is in scope.

## Data types

The same rules apply for callbacks as any other unmanaged API function.

## Special cases

Any parameters used in an asynchronous process must be declared as `IntPtr`. The marshaler will free any memory it has allocated for unmanaged types when it returns from the function call. When using an `IntPtr`, it is your responsibility to free any memory that has been allocated.

## Passing Object References

When working with for example the Windows API, object references are sometimes passed to the API where they are stored and later passed back to the application for processing usually associated with a given event. This can still be accomplished in .NET, but special care needs to be taken to ensure a reference is kept to all objects (otherwise they can and will be garbage collected).

## Data types

Unmanaged Data Types	Managed Data Type	
	Supply Data	Receive Data
Pointer (Object reference, user data)	GCHandle	GCHandle

The `GCHandle` provides the primary means of passing an object references to unmanaged code, and ensuring garbage collection does not happen. A `GCHandle` needs to be allocated, and later freed when no longer needed. There are several types of `GCHandle`, `GCHandleType.Normal` being the most useful when an unmanaged client holds the only reference. In order pass a `GCHandle` to an API function once it is allocated, type cast it to `IntPtr` (and optionally onwards to **`LongInt`**, depending on the unmanaged declaration). The `IntPtr` can later be cast back to a `GCHandle`. Note that `IsAllocated` must be called before accessing the `Target` property, as shown below:

```
procedure MyProcedure;
var
  Ptr: IntPtr;
  Handle: GCHandle;
begin
  ...
  if Ptr <> nil then
  begin
    Handle := GCHandle(Ptr);
    if Handle.IsAllocated then
      DoSomething(Handle.Target);
  end;
  ...
end;
```

## Advanced techniques

The use of a `GCHandle`, although relatively easy, is fairly expensive in terms of performance. It also has the possibility of resource leaks if handles aren't freed correctly. If object references are maintained in the managed code, it is possible to pass a unique index, for example the hash code returned by the `GetHashCode` method, to the unmanaged API instead of an object reference. A hash table can be maintained on the managed side to facilitate retrieving an object instance from a hash value if needed. An example of using this technique can be found in the `TTreeNode`s class (in `Borland.Vcl.ComCtrls`).

## Using COM Interfaces

When using COM interfaces, a similar approach is taken as when using unmanaged API's. The interface needs to be declared,

using custom attributes to describe the type interface and the GUID. Next the methods are declared; using the same approach as for unmanaged API's. The following example uses the **IAutoComplete** interface, defined as follows in Delphi 7:

```
IAutoComplete = interface(IUnknown)
  ['{00bb2762-6a77-11d0-a535-00c04fd7d062}']
  function Init(hwndEdit: HWND; punkACL: IUnknown;
    pwszRegKeyPath: LPCWSTR; pwszQuickComplete: LPCWSTR): HRESULT; stdcall;
  function Enable(fEnable: BOOL): HRESULT; stdcall;
end;
```

In RAD Studio it is declared as follows:

```
[ComImport, GuidAttribute('00BB2762-6A77-11D0-A535-00C04FD7D062'),
InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)]
IAutoComplete = interface
  function Init(hwndEdit: HWND; punkACL: IEnumString;
    pwszRegKeyPath: IntPtr; pwszQuickComplete: IntPtr): HRESULT;
  function Enable(fEnable: BOOL): HRESULT;
end;
```

Note the custom attributes used to describe the GUID and type of interface. It is also essential to use the `ComImportAttribute` class. There are some important notes when importing COM interfaces. You do not need to implement the `IUnknown/IDispatch` methods, and inheritance is not supported.

## Data types

The same rules as unmanaged functions apply for most data types, with the following additions:

Unmanaged Data Type	Managed Data Type	
	Supply Data	Receive Data
GUID	System.Guid	System.Guid
IUnknown	TObject	TObject
IDispatch	TObject	TObject
Interface	TObject	TObject
Variant	TObject	TObject
SafeArray (of type)	array of <type>	array of <type>
BSTR	String	String

Using the `MarshalAsAttribute` custom attribute is required for some of the above uses of `TObject`, specifying the exact unmanaged type (such as `UnmanagedType.IUnknown`, `UnmanagedType.IDispatch` or `UnmanagedType.Interface`). This is also true for certain array types. An example of explicitly specifying the unmanaged type is the `Next` method of the `IEnumString` interface. The Win32 API declares `Next` as follows:

```
HRESULT Next(
  ULONG celt,
  LPOLESTR * rgelt,
  ULONG * pceltFetched
);
```

In RAD Studio the declaration would be:

```
function Next(celt: Longint;
  [out, MarshalAs(UnmanagedType.LPArray, ArraySubType = UnmanagedType.LPWSTR, SizeParamIndex =
0)]
  rgelt: array of string;
  out pceltFetched: Longint
): Integer;
```



## Advanced techniques

When working with safearrays, the marshal layer automatically converts (for example) an array of bytes into the corresponding safearray type. The marshal layer is very sensitive to type mismatches when converting safearrays. If the type of the safearray does not exactly match the type of the managed array, an exception is thrown. Some of the Win32 safearray API's do not set the type of the safearray correctly when the array is created, which will lead to a type mismatch in the marshal layer when used from .NET. The solutions are to either ensure that the safearray is created correctly, or to bypass the marshal layer's automatic conversion. The latter choice may be risky (but could be the only alternative if you don't have the ability to change the COM server that is providing the data). Consider the following declaration:

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer;
  out RecsOut: Integer; Options: Integer; const CommandText: WideString;
  var Params: OleVariant; var OwnerData: OleVariant): OleVariant;
```

If the return value is known to always be a safearray (that doesn't describe its type correctly) wrapped in a variant, we can change the declaration to the following:

```
type
  TSafeByteArrayData = packed record
    VType: Word;
    Reserved1: Word;
    Reserved2: Word;
    Reserved3: Word;
    VArray: IntPtr; { This is a pointer to the actual SafeArray }
  end;

function AS_GetRecords(const ProviderName: WideString; Count: Integer;
  out RecsOut: Integer; Options: Integer; const CommandText: WideString;
  var Params: OleVariant; var OwnerData: OleVariant): TSafeByteArrayData;
```

Knowing that an OleVariant is a record, the TSafeByteArrayData record can be extracted from Delphi 7's TVarData (equivalent to the case where the data type is varArray). The record will provide access to the raw pointer to the safearray, from which data can be extracted. By using a structure instead of an OleVariant, the marshal layer will not try to interpret the type of data in the array. You will however be burdened with extracting the data from the actual safearray.

## Special cases

Although it is preferred to use Activator.CreateInstance when creating an instance, it is not fully compatible with CoCreateInstanceEx. When working with remote servers, CreateInstance will always try to invoke the server locally, before attempting to invoke the server on the remote machine. Currently the only known work-around is to use CoCreateInstanceEx.

Since inheritance isn't supported, a descendant interface needs to declare the ancestor's methods. Below is the IAutoComplete2 interface, which extends IAutoComplete.

```
[ComImport, GuidAttribute('EAC04BC0-3791-11d2-BB95-0060977B464C'),
InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIUnknown)]
IAutoComplete2 = interface(IAutoComplete)
  // IAutoComplete methods
  function Init(hwndEdit: HWND; punkACL: IEnumString;
    pwszRegKeyPath: IntPtr; pwszQuickComplete: IntPtr): HRESULT;
  function Enable(fEnable: BOOL): HRESULT;
  //
  function SetOptions(dwFlag: DWORD): HRESULT;
  function GetOptions(var dwFlag: DWORD): HRESULT;
end;
```

## See Also

[Marshaling Data with Platform Invoke](#)

Using COM Interop in Managed Applications (see page 38)

Virtual Library Interfaces (see page 52)

## 1.2.5 Virtual Library Interfaces

This topic describes how to use a feature of Delphi called Virtual Library Interfaces. Virtual Library Interfaces allows you to discover, load, and call unmanaged code at runtime, without the use of the `DllImport` attribute.

### Standard PInvoke

To call an unmanaged function from managed code, you must use a .NET service called Platform Invoke, or PInvoke. The Platform Invoke service requires you to declare in source code, a prototype for each unmanaged function you wish to call. You can do this either within an existing .NET class, or you can create an entirely new class to organize the prototypes. You must also tag each unmanaged prototype declaration with the `DllImport` attribute.

The `DllImport` attribute requires you to specify the name of the DLL in which the unmanaged function resides. Since the unmanaged prototype is tagged with the `DllImport` attribute at compile-time, dynamic discovery of DLLs and their exported unmanaged functions is difficult. Furthermore, if the unmanaged function is not actually exported from the DLL named in the `DllImport` attribute, a runtime failure will result. To avoid a runtime failure, you would have to use `LoadLibrary` to load the exact DLL you require, and then call `GetProcAddress` to verify the existence of the unmanaged function. Even so, you would not be able to directly call the function using the pointer returned from `GetProcAddress`. Instead you would have to pass the pointer along to a function in another unmanaged DLL. That function would then use the pointer to make the call.

### Using Virtual Library Interfaces

Virtual Library Interfaces still must use the Platform Invoke service to call unmanaged code. However, instead of using the `DllImport` attribute, Virtual Library Interfaces creates an interface on the unmanaged DLL at runtime, using methods of the .NET `System.Reflection.Emit` namespace.

Using Virtual Library Interfaces requires that you do three things:

- Add `Borland.Vcl.Win32` to the **uses** clause.
- Declare an interface containing the exported, unmanaged functions you wish to call.
- Call the `Supports` function to ensure that the unmanaged DLL exists and that the functions in the interface declaration are actually exported.

If the `Supports` function returns `True`, then the DLL supports all of the functions named in the interface declaration, so you know it is safe to call them. Within the interface declaration, you do not need to use the `DllImport` attribute on the prototypes.

For example, if you have a DLL called `MyFunctions.dll`, that contains the following exported functions:

```
function AFunction      : Boolean;
function AnotherFunction : Boolean;
```

To call these functions from managed code, add the `Borland.Vcl.Win32` unit to the **uses** clause and declare an interface in Delphi:

```
uses Borland.Vcl.Win32, ...;
...
type
IMyFunctions = interface
['Your GUID'] // Not strictly required, but good practice
function AFunction      : Boolean;
function AnotherFunction : Boolean;
end;
```

The signature of the `Supports` function is:

```
function Supports(ModuleName: string; Source: System.Type; var Instance) : Boolean;
```


To call the unmanaged functions, first call Supports to load the DLL, and create the interface on the DLL:

```
var
MyFunctions : IMyFunctions;
begin
  if Supports("MyFunctions.dll", IMyFunctions, MyFunctions) then
    if MyFunctions.AFunction then
      begin
        ...
      end;
    end;
  end;
end;
```

Virtual Library Interfaces have the same limitations in terms of compatible native parameter types and their mapping to .NET types. In addition, all unmanaged functions are expected to use the stdcall calling convention.

**See Also**

Using COM Interop in Managed Applications ( see page 38)

Using Platform Invoke with Delphi ( see page 44)

## 1.3 Modeling Concepts

This section provides information on modeling and code visualization.

### Topics

Name	Description
Code Visualization Overview (see page 54)	The Code Visualization feature is available in both the Enterprise and Architect versions of RAD Studio. All other modeling tools and information related to modeling relates only to the Architect version of RAD Studio.

### 1.3.1 Code Visualization Overview

The Code Visualization feature is available in both the Enterprise and Architect versions of RAD Studio. All other modeling tools and information related to modeling relates only to the Architect version of RAD Studio.

#### Code Visualization and UML Static Structure Diagrams

RAD Studio's code visualization diagram presents a graphical view of your source code, which is reflected directly from the code itself. When you make changes in source code, the graphical depiction on the diagram is updated automatically. The code visualization diagram corresponds to a UML *static structure* diagram. A structural view of your project focuses on UML packages, data types such as classes and interfaces, and their attributes, properties, and operations. A static structure diagram also shows the relationships that exist between these entities.

This section will explain the relationship between source code and the code visualization diagram.

**Note:** code visualization, and the integrated UML modeling tools are two separate and distinct features of RAD Studio. Code visualization refers to the ability to scan an arbitrary set of source code, and map the declarations within that code onto UML notation. The resulting diagram is "live", in the sense that it always reflects the current state of the source code, but you cannot make changes directly to the code visualization diagram itself. RAD Studio's model driven UML tools go a step further, giving you the ability to design your application on the diagramming surface. While both product features are based on CodeGear Together technologies, they each use different underlying mechanisms to produce and manipulate the diagram. The integrated model design surface is additionally based on the designtime capabilities of CodeGear's Enterprise Core Objects (ECO) framework. This document covers only the code visualization. For more information about building applications with the ECO framework and using the wizards and modeling tools with ECO-enabled applications, see the separate ECO online help.

#### Understanding the Relationship between Source Code and Code Visualization

RAD Studio's code visualization tools use the UML notation and conventions to graphically depict the elements declared in source code. The **Code Visualization diagram** shows you the logical relationships, or *static structure* in UML terms, of the classes, interfaces and other types defined in your project. The IDE creates the **Code Visualization diagram** by mapping certain source code constructs (such as class declarations, and implementation of interfaces) onto their UML counterparts, which are then displayed on the diagram.

#### Top-Level Organization: Projects, UML Packages, and .NET Namespaces

To begin, code visualization consists of two parts of the IDE working together: The **Model View window**, and the **Code Visualization diagram**. The **Model View window** shows you the logical structure of your projects in a tree, as opposed to the file-centric view of the **Project Manager window**. Each project in a project group is a top-level node in the **Model View tree**.

Nested within each project tree-node, you will find UML packages. Each UML package corresponds to a .NET namespace

declaration in your source code (.NET namespaces can span multiple source files). You can expand the UML package to reveal the types declared within.

### Inheritance and Interface Implementation

The UML term for the relationship formed when one class inherits from a superclass is *generalization*. When the IDE sees an inheritance relationship in your source code, it creates a generalization link within the child class node in the **Model View tree**. On the **Code Visualization diagram**, the generalization link will be shown the using standard UML notation of a solid line with a hollow arrowhead pointing at the superclass.

The UML term for interface implementation is *realization*. Similar to the case of inheritance, the IDE creates a realization link when it sees a class declaration that implements an interface. The realization link appears within the implementor class in the **Model View tree**, and on the diagram as a dotted line with a hollow arrowhead pointing at the interface. There will be one such realization link for every interface implemented by the class.

### Associations

In the UML, an *association* is a navigational link produced when one class holds a reference to another class (for example, as an attribute or property). Code visualization creates association links when one class contains an attribute or property that is a non-primitive data type. On the diagram the association link exists between the class containing the non-primitive member, and the data type of that member.

### Class Members: Attributes, Operations, Properties, and Nested Types

Code visualization can also map class and interface member declarations to their UML equivalents. Within the elements on the **Code Visualization diagram**, members are grouped into four distinct categories:

- Fields: Contains field declarations. The type, and optional default value assignment are shown on the diagram.
- Methods: Contains method declarations. Visibility, scope, and return value are shown.
- Properties: Contains Delphi property declarations. The type of the property is shown.
- Classes: Contains nested class type declarations.

Standard UML syntax is used to display the UML declaration of attributes, operations, and properties. Each of the four categories can be independently expanded or collapsed to show or hide the members within.

### See Also

UML Features in Delphi for .NET (Architect SKU only)

Integrated Modeling Tools Overview (Architect SKU only)

Importing and Exporting a Model Using XML (Architect SKU only) (🔗 see page 142)

Using the Model View Window and Code Visualization Diagram (🔗 see page 143)

Using the Overview Window (🔗 see page 144)

# 1.4 Developing Reports for .NET Applications

RAD Studio ships with Rave Reports from Nevrona. Using the report components, you can build full-featured reports for your applications. You can create solutions that include reporting capabilities which can be used and customized by your customers.

## Topics

Name	Description
Using Rave Reports in RAD Studio (see page 56)	<p>The RAD Studio environment supports the integration of report objects in your applications. This integration allows you to create a report using the Rave Reports Designer or to add Rave Reports ActiveX components directly onto your Web Forms in the RAD Studio <b>Designer</b>. Your application users can create and display their own reports, or display existing reports. The RAD Studio integration with Rave Reports allows you to:</p> <ul style="list-style-type: none"><li>• Include new report objects in projects.</li><li>• Add Rave Reports ActiveX objects onto Web Forms.</li></ul>

## 1.4.1 Using Rave Reports in RAD Studio

The RAD Studio environment supports the integration of report objects in your applications. This integration allows you to create a report using the Rave Reports Designer or to add Rave Reports ActiveX components directly onto your Web Forms in the RAD Studio **Designer**. Your application users can create and display their own reports, or display existing reports. The RAD Studio integration with Rave Reports allows you to:

- Include new report objects in projects.
- Add Rave Reports ActiveX objects onto Web Forms.

### Creating New Reports in RAD Studio

You can include Rave reports in RAD Studio just as you would other third-party components. The report is stored as a separate Rave Report object. You can reference the report in other applications that need to call or generate that report. When you create a new application, you can include the report object by adding a reference to it in the **Project Manager**. Rave Reports also provide the capability to connect your report object to a datasource, which allows your application to build the report dynamically, based on current database information.

### Using Rave Reports ActiveX Components

You can add any Rave Reports ActiveX objects to your applications. The RAD Studio**Tool Palette** provides a list of any available ActiveX objects. Just drag the objects you want onto a Windows Form or a Web Form during design. Fill in the appropriate properties and modify any code in the **Code Editor**. You may need to reset your .NET components and select the ActiveX components from the **Installed .NET Components** dialog.

### See Also

[Design Report Method](#)

# 1.5 Developing Applications with VCL.NET Components

VCL.NET is an extended set of the VCL components that provide a way to quickly build advanced applications in Delphi. With VCL.NET you can provide your Delphi VCL applications and components to Microsoft .NET Framework users. With RAD Studio you gain the benefit of the .NET Framework along with the ease-of-use and powerful component-driven application development of Delphi.

RAD Studio provides distinct application types for your use: you can create VCL.NET form applications that run on the .NET Framework that use VCL.NET components and controls; you can create .NET applications that use the underlying .NET Framework and .NET controls while offering RAD Studio code-behind; you can create powerful ASP.NET applications that use the underlying .NET Framework, ASP.NET controls, and also offer RAD Studio code-behind. The following topics provide more information on how to take advantage of the new VCL.NET provisions in RAD Studio.

## Topics

Name	Description
Changes Required Due to 64-bit .NET 2.0 Support (🔗 see page 58)	Changes were made to support 64-bit .NET 2.0. These changes might require minor code changes so that existing applications work correctly. See Making Changes Required Due to 64-bit .NET 2.0 Support (🔗 see page 157) for detailed information on specific changes required.
Language Issues in Porting VCL Applications to RAD Studio (🔗 see page 59)	The VCL in RAD Studio was created with backward compatibility as the primary goal. However, there are some ways in which the managed environment of .NET imposes differences in the way VCL applications must work. This document describes most of these differences, and indicates some of the steps you should take to port a VCL application to the .NET environment.  This document does not attempt to describe the new extensions to the Delphi language. It is limited to the way existing Delphi code maps to the new RAD Studio language and VCL framework. This document does contain links into specific... more (🔗 see page 59)
Porting VCL Applications (🔗 see page 69)	When porting VCL applications from Delphi 7 to RAD Studio, there are issues you need to consider. Along with basic language elements that need to be replaced or modified, there are strategies that you should follow to make sure that you port your applications fully and reliably.  This topic includes <ul style="list-style-type: none"> <li>• General Language Issues</li> <li>• Renaming Packages</li> <li>• New Language Features</li> <li>• Porting Web Service Client Applications</li> </ul>
VCL for .NET Overview (🔗 see page 71)	VCL for .NET is the programming framework for building RAD Studio applications using VCL components. RAD Studio and VCL for .NET are intended to help users leverage the power of Delphi when writing new applications, as well as for migrating existing Win32 applications to the .NET Framework.  These technologies allow a Delphi developer to migrate to .NET, taking their Delphi skills and much of their current Delphi source code with them. RAD Studio supports Microsoft .NET Framework development with the Delphi language and VCL for .NET controls. RAD Studio ASP.NET also supports WebForms, and SOAP and XML Web Services application... more (🔗 see page 71)
Porting Web Service Clients (🔗 see page 74)	RAD Studio web services use the .NET Framework as the services layer. As a consequence, any existing Delphi 7 or earlier web service client applications need to be modified to use the .NET Framework.  This topic includes: <ul style="list-style-type: none"> <li>• Changes and Additions to Your Applications</li> <li>• Implementation Notes</li> </ul>

## 1.5.1 Changes Required Due to 64-bit .NET 2.0 Support

Changes were made to support 64-bit .NET 2.0. These changes might require minor code changes so that existing applications work correctly. See Making Changes Required Due to 64-bit .NET 2.0 Support (see page 157) for detailed information on specific changes required.

### Changes to support 64-bit

This document describes changes that might be required to existing VCL.NET applications to function correctly with the changes made to support 64-bit .NET 2.0.

To provide single source, single executable support for both 32 and 64-bit platforms, handle types have been changed from `LongWord` to `IntPtr`. `IntPtr` is an integer type the size of a pointer on the underlying platform, which is implemented as a structure in .NET. It doesn't support the assignment, comparison and arithmetic operations commonly performed on handles in the VCL. To supply backwards compatibility, a record helper has been created to support commonly used operations through operator overloading.

Most code does not need modification to continue running on both x86 and x64 platforms, but there are some cases that require minor changes.

**Warning:** These changes are required due to changes made to support 64-bit systems. They must be made for applications intended for both 32-bit and 64-bit systems.

### Changed Code Patterns

Code patterns may need to change in the following areas:

- Case statements
- Casting enumerations
- Using handles with sets
- Indexing into arrays
- Assuming a specific handle size
- Casting enumerations

### Windows API declarations

A number of declarations have been changed to properly support single executables for both x86 and x64 platforms. In most of these cases, the type of a function's parameter or a structure's field has been changed, and the compiler provides information to easily fix affected code. Changes where the fix is less obvious include type changes to callback functions and modifications to variant records. In these cases, a list of the affected callbacks and records has been provided in Making Changes Required Due to 64-bit .NET 2.0 Support (see page 157).

### Modified Callbacks

A few callbacks have changed parameters, so each of their occurrences needs to be modified.

### Modified Variant Records

Variant records are implemented in Delphi.NET as records with explicit field layouts (as opposed to sequential layouts, which is the .NET Framework default). This allows multiple fields to be "overlaid" at the same field offset, which results in incorrect declarations when running on 64-bit platforms when the structure contains any variable size field (such as `IntPtr`). Variant records with such fields have been modified to contain only one of the "variant" cases. The remaining fields have been changed to properties to provide backwards compatibility. In most cases, source code is not affected by these changes. However, passing a field as a `var` parameter that is now declared as a property does require a code change.



## See Also

Making Changes Required Due to 64-bit .NET 2.0 Support (see page 157)

## 1.5.2 Language Issues in Porting VCL Applications to RAD Studio

The VCL in RAD Studio was created with backward compatibility as the primary goal. However, there are some ways in which the managed environment of .NET imposes differences in the way VCL applications must work. This document describes most of these differences, and indicates some of the steps you should take to port a VCL application to the .NET environment.

This document does not attempt to describe the new extensions to the Delphi language. It is limited to the way existing Delphi code maps to the new RAD Studio language and VCL framework. This document does contain links into specific topics within the Delphi Language Guide, where new language features are explained in detail.

This topic covers the following material:

- Migrating Pointer types
- Migrating **Char** and **string** types
- Creating and destroying objects
- Calling the Win32 API
- Migrating Variants
- Working with resources
- Change to `OnCompare`

### Migrating Pointer Types

Pointer types are not CLS compliant, and are not considered "safe" in the context of the .NET Common Language Runtime environment. The port of the VCL has, therefore, eliminated pointers, replacing them with appropriate alternatives such as dynamic arrays, indexes into an array or string, class references, and so on. When porting a VCL application, one of the first steps is to locate where you use pointer types and replace them as appropriate.

### Untyped Pointers

Untyped pointers are considered unsafe code. If your code includes untyped pointers, the .NET utility `PEVerify` will fail to verify it. Code that cannot be verified for type safety cannot be executed in a secured environment, such as a web server, SQL database server, web browser client, or a machine with restricted security policies.

In the VCL, untyped pointers have been replaced with more strongly-typed values. In most cases, where you used to find an untyped pointer, you will now find `TObject`. For example, the elements of `TList` are now of type `TObject`, rather than of type **Pointer**. Your code can cast any type to an object, and cast a `TObject` to any other type (even value types such as **Integer**, **Double**, and so on). Casting `TObject` to another type will generate a runtime error if the object is not, in fact, an instance of the type to which you are casting it. That is, this cast has the same semantics as using the **as** operator.

In some cases, the **Pointer** type has been replaced with a more precise type. For example, on `TObject`, the `ClassInfo` function returns a value of type `Type` rather than an untyped pointer.

Untyped pointers that were used for parameters whose type varied depending on context have typically been replaced by overloading the routine and using **var** parameters with the possible types. In the case of untyped pointers that are used with API calls to unmanaged code (such as the Windows API or calls to a data access layer such as the BDE) the untyped pointer is replaced with `System.IntPtr`. Thus, for example, the `TBookmark` type, defined in the `Db` unit, now maps to `IntPtr`.

Code that used the address operator (@) to convert a value to an untyped pointer must now change. When the untyped pointer has changed to TObject, usually all you need to do is eliminate the @ operator. On value types, you may need to replace the @ operator with a typecast to TObject, so that the value is "boxed". Thus, the following code

```
var
  P: Pointer;
  I: Integer;
begin
  I := 5;
  P := @I;
```

could be converted to

```
var
  P: TObject;
  I: Integer;
begin
  I := 5;
  P := TObject(I);
```

When the untyped pointer has changed to IntPtr, you need to use the Marshal class to allocate a chunk of unmanaged memory and copy a value to it, rather than just using the @ operator. Thus the following code:

```
var
  P: Pointer;
  R: TRect;
begin
  R := Rect(0, 0, 100, 100);
  P := @R;
  CallSomeAPI(P);
```

would be converted to

```
var
  P: IntPtr;
  R: TRect;
begin
  R := Rect(0, 0, 100, 100);
  P := Marshal.AllocHGlobal(Marshal.SizeOf(TypeOf(TRect)));
  try
    Marshal.StructureToPtr(TObject(R), P, False);
    CallSomeAPI(P);
  finally
    Marshal.FreeHGlobal(P);
  end;
```

**Note:** All unmanaged memory that you allocate using the Marshal class must be explicitly freed. The .NET garbage collector does not clean up unmanaged memory.

## Procedure Pointers

A special case for untyped pointers is when they represent procedure pointers. In managed code, procedure pointers are replaced by .NET delegates, which are more strongly typed. Declarations of procedural types are delegate declarations in RAD Studio. You can obtain a delegate for a method or global routine using the @ operator. The code looks the same as obtaining a procedure pointer on the Win32 platform, so in many cases there is nothing you need to change when porting code. However, it is important to keep in mind that when you use the @ operator, you get a newly-created delegate, not a pointer.

If you are passing a procedure pointer to an unmanaged API using the @ operator, for example,

```
Handle := SetTimer(0, 0, 1, @TimerProc);
```

the only reference to the delegate is the one passed to the API call because the delegate is created on the fly. This means that the garbage collector will eventually dispose of the delegate after the return of the unmanaged API. If, as in this case, the unmanaged code may call the procedure after the return of the API call, you will encounter a runtime exception because the delegate no longer exists. You can work around this situation by assigning the delegate to a global variable, and passing the

global variable to the unmanaged API.

When you call the Windows API `GetProcAddress` to obtain a procedure pointer, it is returned as an `IntPtr`. This value is not a delegate. You can't cast it to a delegate and call it. Instead, typically such code is translated to use Platform Invoke to call an unmanaged API. `GetProcAddress` is useful to determine whether the API is available so that you do not get a runtime exception when you use Platform Invoke. Thus, code such as the following:

```
type
  TAnimateWindowProc = function(hWnd: HWND; dwTime: DWORD; dwFlags: DWORD): BOOL; stdcall;
var
  AnimateWindowProc: TAnimateWindowProc = nil;
  UserHandle: HMODULE;
begin
  UserHandle := GetModuleHandle('USER32');
  if UserHandle <> 0 then
    @AnimateWindowProc := GetProcAddress(UserHandle, 'AnimateWindow');
    ...
  if AnimateWindowProc <> nil then
    AnimateWindowProc(Handle, 100, AW_BLEND or AW_SLIDE);
```

Would be translated to the .NET platform as follows

```
[DllImport('user32.dll', CharSet = CharSet.Ansi, SetLastError = True, EntryPoint =
'AnimateWindow')]
function AnimateWindow(hWnd: HWND; dwTime: DWORD; dwFlags: DWORD): BOOL; external;
var
  UserHandle: HMODULE;
  CanAnimate: Boolean;
begin
  UserHandle := GetModuleHandle('USER32');
  if UserHandle <> 0 then
    CanAnimate := GetProcAddress(UserHandle, 'AnimateWindow') <> nil
  else
    CanAnimate := False;
    ...
  if CanAnimate then
    AnimateWindow(Handle, 100, AW_BLEND or AW_SLIDE);
```

**Note:** The .NET example above is still late bound to the `AnimateWindow` API. An exception will not be generated when this code is loaded, if the DLL or function aren't available. The function call is resolved only when the code is executed for the first time.

## String Pointers

Code that uses the **PChar** type usually serves one of three purposes:

- The type refers to a null-terminated string (especially when it is used with a Windows API call or an older RTL function).
- The type is used to navigate through a string when processing its value.
- The type is used to reference a block of bytes, relying on the fact that in Delphi for Win32, the **Char** type is a byte (the **Char** type is two bytes on the .NET platform).

In the first case, you can usually replace the **PChar** type with the type **string**. In the case of Windows API calls, the managed versions of the APIs now use a string or `StringBuilder` rather than a **PChar**, with the marshaling layer handling the conversions implicitly. Note that many of the RTL functions that supported the **PChar** type have been eliminated from the RTL, and you must replace them with corresponding versions that use the **string** type. The following table lists functions from the `SysUtils` units that have been eliminated because they relied on the **PChar** type, and the corresponding functions that use the **string** type:

PChar version	String version
<code>AnsiExtractQuotedStr</code>	<code>AnsiDequotedStr</code> or <code>DequotedStr</code>
<code>AnsiLastChar</code> , <code>AnsiStrLastChar</code>	(use index operator and string length)

AnsiStrComp, StrComp	CompareStr, AnsiCompareStr, WideCompareStr
AnsiStrIComp, StrIComp	CompareText, AnsiCompareText, WideCompareText
AnsiStrLComp, StrLComp	System.String.Compare (StartsStr)
AnsiStrLIComp, StrLIComp	System.String.Compare (StartsText)
AnsiStrLower, StrLower	AnsiLowerCase, WideLowerCase,
AnsiStrUpper, StrUpper	UpCase, AnsiUpperCase, WideUpperCase
AnsiStrPos, StrPos, AnsiStrScan, StrScan	Pos
AnsiStrRScan, StrRScan	LastDelimiter
StrLen	Length
StrEnd, StrECopy	(no equivalent)
StrMove, StrCopy, StrLCopy, StrPCopy, StrPLCopy	Copy
StrCat, StrLCat	<ul style="list-style-type: none"> <li>• operator, Concat</li> </ul>
StrFmt	Format, FmtStr
StrLFmt	FormatBuf
FloatToText	FloatToStrF
FloatToTextFmt	FormatFloat
TextToFloat	FloatToStr

When a **PChar** type is used to navigate through a string, you must rewrite the code, replacing the **PChar** with an **Integer** that represents an index into the string. When rewriting such code, you must recognize when you have reached the end of the string. When using the **PChar** type, there is a null character at the end of the string, and the code typically recognizes the end of the string by finding this null character. With a string-and-index approach, there is no such null character and you must use the string length to identify the end of the string. Be careful to check that the index is not past the end of the string before reading a character or you will get a runtime error.

**Note:** String data is immutable, so you can't write a single character into an existing string using a **PChar**

. You can accomplish this using string indexing (e.g. `s[5]`), however. When a **PChar** is used to reference a block of bytes, it is typically replaced by either an `IntPtr` or a dynamic array of bytes ( `TBytes`). If replaced by an `IntPtr`, the issues in translating are the same as when replacing an untyped pointer. When replaced by `TBytes`, you may need to replace some **PChar** values with an index into the byte array if it is used to navigate the block of bytes. This is like replacing **PChar** with `Integer` to navigate through a string, except that indexes into `TBytes` are 0-based while indexes into strings are 1-based.

## Writing Strings to Streams

In Delphi for Win32, it is common to find code similar to the following:

```
S1 := 'This is a test string';
Stream.WriteBuffer(S1[1], Length(S1));
```

On the Win32 platform, this code results in the entire string being written to the stream. On the .NET platform however, this same code produces a quite different result. On the .NET platform, the compiler generates a call to the **Char** overloaded version of `WriteBuffer`, with the result being only a single character (`S1[1]`) being written to the stream.

## Other Pointer Types

Other typed pointers have been eliminated from the VCL. Typically, they are replaced by the type to which the original pointed. If the pointer type was the parameter to a procedure call, it is typically converted to a **var** parameter so that the resulting code still passes a reference rather than a copy of the argument. Sometimes, it is useful to change a value type into a class type so that

rather than passing a typed pointer, your code passes an object reference.

### Migrating Char and String Types

In RAD Studio, the string type maps to the .NET String type, and you can freely access the members of String using a Delphi **string** type, as demonstrated in the following example:

```
var
  S: string;
begin

  S := 'This is a string';

  // Note the typecast is not necessary.
  // S := System.String(S).PadRight(25);

  // Direct access to string class members
  S := S.PadRight(25);
  S := ('This is a new string').PadRight(25);
```

### ANSI Strings and Wide Strings

The biggest difference for strings in RAD Studio is that the string type is now a Unicode wide string rather than an **AnsiString**. This simplifies code for some locales, because you no longer need to worry about multibyte character sets. However, you must examine your code for any assumptions about the size of a **Char**, because it is now two bytes rather than one. You can still use strings with one-byte characters, but you must now declare them as **AnsiString** rather than **string**. The compiler converts between wide and narrow strings if you use an explicit typecast or if you implicitly cast them by assigning to a variable or parameter of the other type.

If your code calls any of the `AnsiXXX` routines for manipulating strings, you may want to change these to the corresponding wide string version of the routine. The `AnsiXXX` routines have (deprecated) overloads that map to the wide versions, and the overloaded routines accept wide strings for their parameters; this avoids implicit conversion back and forth between wide and single-byte strings.

**Note:** Information can be lost when converting from wide to single-byte characters, therefore, you should avoid downcasting as much as possible.

### String Operations

Following the CLR value-type semantics, typically operations on strings return a copy of the string rather than alter the existing string. This may make some code less efficient, because there is more copying going on. For example, consider the following:

```
var
  S: string;
begin
  S := 'This is a string';
  S[3] := 'a';
  S[4] := 't';
```

When compiled using on the Win32 platform, the character substitutions only require a single byte of memory to change each time. In RAD Studio, each substitution results in a copy of the entire string. Because of this, it is a good idea to use a `StringBuilder` instance when you are manipulating string values. `StringBuilder` allocates a chunk of unmanaged memory and manipulates the string the way you expect. When you are finished, you can convert the result to a string by calling the `ToString` method.

**Note:** The conversion to string

from a `StringBuilder` is a low-cost operation. The string data is not copied again.

## Uninitialized Strings

In RAD Studio, an uninitialized string has the value of **nil**. The compiler will automatically compensate if you compare an uninitialized string with an empty string. That is, if you have a line such as

```
if S <> '' then ...
```

The compiler handles the comparison and treats the uninitialized string as an empty string. However, unlike code compiled on the Win32 platform, other string operations do not automatically treat an uninitialized string like an empty string. This can lead to Null Object exceptions at runtime.

## Typecasts

Unlike Delphi for Win32, in RAD Studio, there is no distinction between an explicit typecast and the **as** operator. In both cases, the cast only succeeds if the variable being cast is really an instance of the type to which you cast it. This means that code which used to work (by casting between incompatible data types) may now generate a runtime exception.

## Message Crackers

Perhaps the most common situation where the change to typecasts causes a problem is in the use of the message cracker types. In the VCL on Win32, the `Messages` unit defined a number of record types to represent the parameters of a Windows message. These records were all the same size, with the fields laid out to extract the information from the Windows message. Thus, you could have the message parameters in one form (say, `TMessage`), and typecast it to another (say `TWMMouse`), and extract the information you wanted. This worked because the two types were the same size, and an explicit typecast did not raise an exception when you reinterpreted the type with the cast. Such a reinterpret cast is not allowed in .NET, and the same code would lead to an invalid cast exception in RAD Studio.

To work around this situation, the message cracker types in RAD Studio are not records at all, but classes. Instead of casting a `TMessage` value to another type such as `TWMMouse`, you must instantiate the other type, passing the original `TMessage` as a parameter. That is, instead of

```
procedure MyFunction(Msg: TMessage);
var
  MouseMsg: TWMMouse;
begin
  if Msg.Msg = WM_MOUSE then
    with Msg as TWMMouse do
      ...
end;
```

you would do something like the following:

```
procedure MyFunction(Msg: TMessage);
var
  MouseMsg: TWMMouse;
begin
  if Msg.Msg = WM_MOUSE then
    with TWMMouse.Create(Msg) do
      ...
end;
```

To convert in the other direction (from a specialized message type to `TMessage`), you can use the new `UnwrapMessage` function that is declared in the `Messages` unit.

## Accessing Protected Members from Classes in Other Units

Another technique that involves what is now an invalid typecast is when you need to access the protected members of a class that is declared in another unit. In Delphi for Win32, you can declare a descendant of the class whose members you want to see:

```
type
  TPeekAtWinControl = class(TWinControl);
```

Then, by casting an arbitrary `TWinControl` descendant to `TPeekAtWinControl`, you could access the protected methods of `TWinControl`, because `TPeekAtWinControl` was defined in the same unit.

In general, this technique does not work in RAD Studio, because the arbitrary `TWinControl` descendant is not, in fact, an instance of `TPeekAtWinControl`. The cast leads to an invalid cast exception at runtime.

Because this is a widely used technique in Win32, the compiler will recognize this pattern and allow it. However, the compiler can't know what assembly a unit will be linked into when it compiles the source code. If the units are linked into assemblies, this technique will fail at runtime with a type exception.

When you need to cross assembly boundaries, one workaround is to introduce an interface that provides access to the protected members in question. Some of the classes in the VCL ( `TControl`, `TWinControl`, `TCustomForm`) now use this technique, and you can find the addition of interfaces to access protected members (`IControl`, `IWinControl`, `IMDIForm`).

### Creating and Destroying Objects

Specific language issues with programming in Delphi on the memory-managed .NET platform are explained in the topic [Memory Management Issues on the .NET Platform](#).

Because of differences in the way objects are instantiated and freed, it is not possible to have a `BeforeDestruction` or `AfterConstruction` method on a RAD Studio class. Any classes that override these methods must be rewritten.

The fact that these methods and the `OldCreateOrder` property do not exist in the VCL on the .NET platform impacts forms and data modules that relied on `OldCreateOrder` being **False**. The `OnCreate` and `OnDestroy` events now act as if the `OldcreateOrder` property is set to **True**, and will only be called from the constructor or destructor.

**Note:** Because `OnDestroy` is called from a destructor, it is not guaranteed to be called – if the application does not call `Free`, the object's destructor is not called, even though it is garbage collected.

### Working with the Unmanaged Win32 API

Most of the VCL is designed for working with the Windows API. This is handled in a way analogous to the way `Systems.Windows.Forms` works: The VCL is a managed API that calls into the Windows API, marshaling between the managed structures on the VCL side and the unmanaged types that the Windows API uses. Some units, particularly in the RTL, have been ported so that they sit on top of CLR rather than the Windows API. Such units are more flexible, because they can work with any .NET environment, even those that do not support the Windows operating system (for example, the Compact Framework, Mono, and so on). Units that require the Windows operating system are tagged with the **platform** directive. In units that are not tagged with the **platform** directive, any methods or classes that require Windows are tagged with the **platform** directive.

### Isolating Windows Dependencies

In order to maintain relative platform independence in RTL units, some methods functions that rely on Windows have been moved into the `WinUtils` unit. In addition, some classes have been changed to rely more on CLR than Windows.

`TObject`, `Exception`, `TPersistent`, and `TComponent`, all map directly to classes implemented in the .NET Framework. In this way they integrate more smoothly with other .NET applications. Because the corresponding CLR classes (`System.Object`, `System.Exception`, `System.Marshal`, and `System.Component`) do not include all the methods that the VCL requires, the missing methods are supplied by Delphi class helper declarations. In most cases, this mechanism is transparent. However, there are a few cases where it requires you to make minor tweaks to your code. For example, with `TComponent`, `FComponentState` is now a property of `TComponentHelper` rather than a true field of `TComponent`. This means that you can't use the `Include` and `Exclude` methods on `FComponentState`, because when passed a property, they operate on a copy of the property value, which does not alter `FComponentState`. Thus code such as

```
Exclude(FComponentState, csUpdating);
```

Must be rewritten as

```
FComponentState := FComponentState - [csUpdating];
```

TThread has also been changed to map to the CLR thread object. This means that the Thread handle is no longer an ordinal type, but is rather a reference to the underlying CLR thread object. It also means that TThread no longer supports a ThreadID, which is not supported by the CLR thread object. If your thread class requires a ThreadID, you should change it to derive from TWin32Thread instead.

## Calling the Windows API

Many Windows APIs have changed to use a more managed interface. Often, the types of parameters have changed, typically to eliminate pointers. One common change is the **PChar** types have been replaced by **string** or **StringBuilder**.

When your application calls a Windows API, it is making a call into an unmanaged DLL. Because of this, all parameter values must be marshaled into unmanaged memory, where Windows can work with it, and results are then unmarshalled back into managed memory. In most cases, this marshaling is handled automatically, based on the attributes that have been added to API declarations or type declarations. There are some cases, however, when your code must explicitly handle the marshaling – especially when dealing with a pointer on a structure. To do this marshaling, use the **System.Marshal** class. Another class that can be very useful when marshaling data to or from unmanaged memory is the **BitConverter** class. For example, the **Marshal** class does not include a method to read or write a double value, but it can read or write **Int64** values, which are the same size, and the **BitConverter** class can convert these to or from doubles:

```
// copy double into unmanaged memory:
Mem := Marshal.AllocHGlobal(SizeOf(Int64));
Marshal.WriteInt64(Mem, BitConverter.DoubleToInt64Bits(DoubleVariable));
...
// copy double from unmanaged memory
DoubleVariable := BitConverter.Int64BitsToDouble(Marshal.ReadInt64(Mem));
```

When using the **marshal** class, remember that you must always free any unmanaged memory you allocate – the garbage collector does not collect unmanaged memory.

## Working with Windows Messages

One of the changes in the way RAD Studio applications work with Windows is the way message handlers work. The basics of declaring and using messages handlers is the same, but the message-cracker types have changed from records to classes, and you can no longer simply typecast from one message-cracker type to another. Most of this has already been covered in the section on typecasts, but there are a few additional issues that bear mentioning:

- When porting code that sends a message, it is no longer sufficient to declare the message cracker on the stack, fill out its fields, and pass it to a call to **SendMessage**. You must now add a call to create the message cracker, because it is now a class.
- Inside a message handler, you can still call an inherited message handler using the inherited keyword. However, if you do this, you must now be sure that the message cracker type is the same as that in the inherited message handler. For example, if the inherited message handler has a parameter of type **TWMMouse**, and your message handler only needs **TMessage**, declaring your message handler to use **TMessage** and calling **inherited** will lead to an invalid cast exception at runtime. Thus, if you call the inherited message handler, you must now ensure that your message parameter matches that of the inherited handler.
- If a message has parameters that are pointers to records (or pointers to anything, for that matter), then the corresponding message cracker will have properties that represent those records. It is important to realize, however, that these are properties and not fields. Thus, you can read the fields of the record directly from the property, but if your handler needs to change any field values, you can no longer make assignments directly to the fields of the record. Instead, you must copy the record to a local variable, make your changes, and then assign the result back to the property.

Using Windows messages is somewhat more expensive in RAD Studio, because in addition to the overhead of working with the message queue, there is now the overhead of marshaling values to and from unmanaged memory. This is particularly expensive when a parameter represents a pointer (an object reference or a pointer to a structure). Such parameters are ultimately converted to a **WPARAM** or **LPARAM** using an **IntPtr**, which acts as a handle to a block of unmanaged memory that contains a copy of the structure. Object references are converted using a **GCHandle**. In most cases, the predefined message cracker types handle the marshaling of these parameters, to and from the **IntPtr**, but if you defining your own messages, you may need to perform your own marshaling. The



message cracker classes defined in the `Controls` unit illustrate how to handle these marshaling issues.

The VCL defines and uses a number of private message types. These are, for the most part, defined in the `Controls` unit, and have identifiers of the form `CM_XXX` or `CN_XXX`. Because of the extra overhead in marshaling messages, several of the `CM_XXX` message types have been changed or eliminated, replaced by other mechanisms that are less expensive in the .NET environment. The following table lists the message types that have changed, and how the same task is accomplished in RAD Studio:

Message type	Change
CM_FOCUSCHANGED	Replaced by a protected method ( <code>FocusChanged</code> ) on <code>TWinControl</code> . Replace message handlers by an override to the <code>FocusChanged</code> method. Instead of sending messages, call <code>FocusChanged</code> using the <code>IWinControl</code> interface.
CM_MOUSEENTER	Meaning of <code>LPARAM</code> has changed. It used to pass an object reference to the child control where the mouse entered – now it passes the index of that child in the <code>FWinControls</code> or <code>FControls</code> list.
CM_MOUSELEAVE	Meaning of <code>LPARAM</code> has changed. It used to pass an object reference to the child control where the mouse exited – now it passes the index of that child in the <code>FWinControls</code> or <code>FControls</code> list.
CM_BUTTONPRESSED	Replaced by a protected method ( <code>ButtonPressed</code> ) on <code>TSpeedButton</code> . This was only used by <code>TSpeedButton</code> . The <code>CMButtonPressed</code> message handler was replaced by <code>ButtonPressed</code> , which is called directly.
CM_WINDOWHOOK	Retired. <code>TApplication.HookMainWindow</code> and <code>TApplication.UnhookMainWindow</code> are both public methods that can be called directly.
CM_CONTROLLISTCHANGE	Replaced by a protected method ( <code>ControlListChange</code> ) on <code>TWinControl</code> . Replace message handlers by an override to the <code>ControlListChange</code> method.
CM_GETDATALINK	Replaced by a protected method ( <code>GetDataLink</code> ) on various data-aware controls. Call this using the new <code>IDataControl</code> interface. When creating your own data-aware control (that does not descend from an existing class in <code>DBCtrls</code> ), you must implement <code>IDataControl</code> if the control is to work in a <code>DBCGrid</code> .
CM_CONTROLCHANGE	Replaced by a protected method ( <code>GetDataLink</code> ) on various data-aware controls. Call this using the new <code>IDataControl</code> interface. When creating your own data-aware control (that does not descend from an existing class in <code>DBCtrls</code> ), you must implement <code>IDataControl</code> if the control is to work in a <code>DBCGrid</code> .
CM_CHANGED	Meaning of <code>LPARAM</code> has changed. It used to pass an object reference, now it passes a hash code for the object that changed.
CM_DOCKCLIENT	Replaced by a protected method ( <code>DockClient</code> ) on <code>TWinControl</code> . Replace message handlers by an override to the <code>DockClient</code> method.
CM_UNDOCKCLIENT	Replaced by a protected method ( <code>UndockClient</code> ) on <code>TWinControl</code> . Replace message handlers by an override to the <code>UndockClient</code> method.
CM_FLOAT	Replaced by a protected method ( <code>FloatControl</code> ) on <code>TControl</code> . Replace message handlers by an override to the <code>FloatControl</code> method.
CM_ACTIONUPDATE	Retired. <code>TApplication.DispatchAction</code> was promoted to public, and is called directly rather than using a message.
CM_ACTIONEXECUTE	Retired. <code>TApplication.DispatchAction</code> was promoted to public and is called directly rather than using a message.

## Changes to the Threading Model

Sometimes, Windows API calls require the use of the Single Threaded Apartment (STA) model to function properly on some operating systems. For example, on some versions of Windows 98, the `Open` and `Save` dialogs do not work unless your RAD Studio application uses the Single Threaded Apartment model. Any portion of the VCL that uses COM requires this model.

The threading model is established when the process first starts up. If you are creating an executable, this is easy: just add the `[STAThreadAttribute]` attribute to the line immediately preceding the **begin** statement in the `dpr` file. When creating a DLL, you can't force the threading model. However, you can call the `CheckThreadingModel` procedure in the `SysUtils` unit to raise an exception when the application calls a method that requires a particular threading model.

This restriction is fairly common in .NET. By default, Microsoft Visual Studio adds the `STAThreadAttribute` attribute to applications it creates.

## Migrating Variants

The Variant type is very different in RAD Studio. Whereas the Win32 compiler maps Variant onto the record type that COM uses for Variants, in RAD Studio, a Variant is more general. Any object (which in RAD Studio is any type) can act be manipulated as a Variant. Thus, in RAD Studio, you could assign a control to a Variant.

The Delphi Variant type is a Delphi language notion that is not CLS compliant. If you are writing code in RAD Studio that uses Variants, to the outside world will see, these will map to only as `System.Object`. Thus, to code written in other languages, the flexibility in type conversions that Delphi Variants support provide is not available.

## Changes to TVarRec

If your code uses Variants, chances are it should still work. However, because Variants are no longer based on the `TVarRec` type, any code that works with the internals of a Win32 Variant by getting into the underlying `TVarRec` record must be rewritten for .NET.

**Note:** Nearly all of the functions provided by the `Variants` unit are implemented in RAD Studio. If you need to get the `VarType` of a Variant, you can accomplish this and still maintain platform portable code.

## Changes to OLE Variants

The COM Interop layer automatically marshals Objects (and hence Variants). Thus, you can use RAD Studio Variants with COM. However, when using RAD Studio Variants with COM, you should restrict the types you assign to the Variant to COM-compatible types.

In Delphi for Win32, the compiler enforces COM restrictions on the kinds of data that can be assigned to an `OleVariant`. In RAD Studio, `OleVariant` is simply a synonym for `Variant`. It does nothing to ensure that the Variant value is a COM-compatible type.

## Changes to Custom Variants

Custom Variants are completely different in RAD Studio. Because Variants are just objects, you do not need to do anything at all to create a custom Variant – any class you define is already a Variant type. However, to work well as a custom Variant, it helps to implement some CLR interfaces: `IComparable`, `IConvertible`, and `ICloneable`. The Delphi compiler can use these to implement Variant operations. Even with these interfaces, however, other, arbitrary Variant types, can't be converted into your Variant (class) unless you implement a `FromObject` method:

```
class function FromObject(AObject: System.Object): TObject; static;
```

`FromObject` takes an arbitrary source object (the Variant to convert to your class type) and returns the corresponding instance of your class as a `TObject`.

## Working With Resources

RAD Studio can link Windows resources (`.res` files) into your assemblies. This means that when first porting an application, you do not need to change the way you declare and use resources, and it will still work. In some cases, this is what you want to do

anyway. For example, if you use custom cursors, it is simpler to use the Windows API `LoadCursor` function to add the cursor to `TScreen.Cursors` than to bring in the overhead of using `Cursor` and then obtaining a handle to the underlying cursor. However, for resources that are not Windows-specific (such as bitmaps, icons, and strings) you will probably want to update to a .NET resources file.

### Resource Strings

When you use the **resourcestring** keyword, RAD Studio automatically creates the string resources as .NET resources rather than Windows resources. This happens automatically and there is nothing special you need to do. The one thing to watch out for is that you no longer can use the `PResStringRec` type.

### Bitmaps

You can convert bitmaps into .NET resources using the `ResourceWriter` class. The resulting `resources` file can be linked into your RAD Studio application, or deployed as a satellite assembly. To use these converted bitmaps, `LoadFromResourceName` has new overloads for working with .NET resources (and the old version of `LoadFromResourceName` as well as the `LoadFromResourceID` method have been deprecated.) Thus, for example, if your bitmaps are in a `resources` file with a name such as `MyResources.en-US.resources`, you can load your bitmap as follows

```
MyBitmap.LoadFromResourceName('MyFirstBitmap', 'MyResources',  
System.Assembly.GetCallingAssembly);
```

Note that this example assumes the resources are compiled into the assembly that is making the method call that contains this line. If the resources are compiled into a different assembly, you can use `System.Assembly.GetAssembly` (using a type that is defined in the relevant assembly) or `System.Assembly.GetExecutingAssembly` (to obtain the currently executing assembly).

### Change to `TTreeView.OnCompare`

The signature for the `OnCompare` event in the `TTreeView` class has changed in the VCL for .NET. Existing code will cause a runtime exception when the event handler is called.

In Delphi 7, the signature was:

```
TTVCompareEvent = procedure(Sender: TObject; Node1, Node2: TTreeNode; Data: Integer; var  
Compare: Integer) of object;
```

In Delphi for .NET, the new signature is:

```
TTCmpareEvent = procedure(Sender: TObject; Node1, Node2: TTreeNode; Data: TTag; var Compare:  
Integer) of object;
```

### See Also

Porting Web Services to Delphi for .NET (see page 74)

Using Platform Invoke with Delphi for .NET (see page 44)

## 1.5.3 Porting VCL Applications

When porting VCL applications from Delphi 7 to RAD Studio, there are issues you need to consider. Along with basic language elements that need to be replaced or modified, there are strategies that you should follow to make sure that you port your applications fully and reliably.

This topic includes

- General Language Issues
- Renaming Packages
- New Language Features

- Porting Web Service Client Applications

### General Language Issues

Porting Delphi 7 applications to RAD Studio exposes several language issues in the .NET Framework. For instance, the .NET Framework considers pointers to be *unsafe* and so does not consider applications that use pointers to fall into the category of *managed* code. To be compliant with the .NET Framework, you need to modify your applications to avoid or circumvent the use of pointers, the pChar type, and other language-specific elements.

In addition, there are critical issues with the Win32 API, using crackers, migrating char types, and other topics.

### Renaming Packages

When porting a Delphi 7 package to RAD Studio, you will need to change the old package names in the "Requires" list to the corresponding new package names. The following table shows the old and new names.

Old Package Name	New Package Name
rtl	Borland.Delphi and Borland.VclRtl
vcl	Borland.Vcl
vclx	Borland.VclX
dbrtl	Borland.VclDbRtl
bdertl	Borland.VclBdeRtl
vcldb	Borland.VclDbCtrls
dbexpress	Borland.VclDbExpress
dbxcds	Borland.VclDbxCds
dsnapp	Borland.VclDSnap
dsnappcon	Borland.VclDSnapCon
vclactnband	Borland.VclActnBand

Borland.VclActnBand Packages are now installed by using [Component ► Installed .NET Components ► .NET VCL Components](#).

### New Language Features

Several new features have been added to the Delphi language to support programming concepts and features of the .NET platform and the CLS:

- Partitioning code into [namespaces](#)
- New [visibility specifiers](#) for class members
- Class static [methods](#), [properties](#), and [fields](#)
- [Class constructors](#)
- [Nested type](#) declarations within classes
- [Sealed classes](#)
- [Final](#) virtual methods
- [Operator overloads](#) in classes
- [.NET attributes](#)
- [Class helper](#) syntax

Programming in the garbage-collected environment of .NET brings a number of new issues related to allocating and disposing of objects. These issues are discussed in [Memory Management Issues on the .NET Platform](#).

### Porting Web Service Client Applications

The .NET Framework employs a major architectural shift in how it handles web services and web service clients. Your existing web service client applications need to be modified to operate on the .NET Framework. RAD Studio does not support the RIO components, and uses a more transparent .NET approach to managing web service client applications. You will need to eliminate RIO components and modify the way you access WSDL documents.

#### See Also

VCL.NET Overview (see page 71)

Language Issues in Porting VCL Applications to Delphi for .NET (see page 59)

Using Platform Invoke with Delphi for .NET (see page 44)

Porting Web Service Clients to Delphi for .NET (see page 74)

Porting a Delphi for Win32 Web Service Client Application to Delphi for .NET (see page 207)

---

## 1.5.4 VCL for .NET Overview

VCL for .NET is the programming framework for building RAD Studio applications using VCL components. RAD Studio and VCL for .NET are intended to help users leverage the power of Delphi when writing new applications, as well as for migrating existing Win32 applications to the .NET Framework.

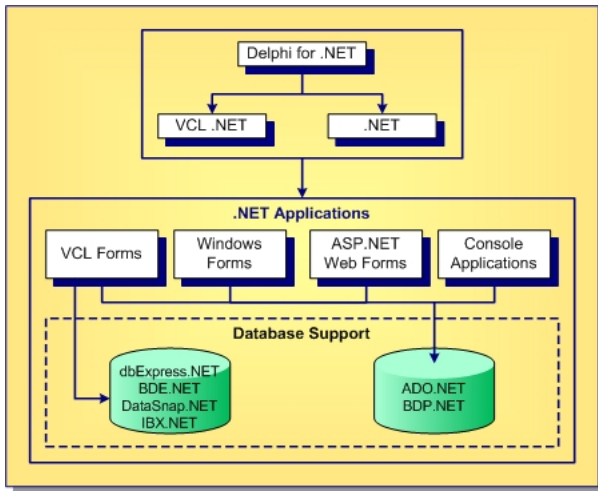
These technologies allow a Delphi developer to migrate to .NET, taking their Delphi skills and much of their current Delphi source code with them. RAD Studio supports Microsoft .NET Framework development with the Delphi language and VCL for .NET controls. RAD Studio ASP.NET also supports WebForms, and SOAP and XML Web Services application development.

VCL for .NET is a large subset of the most common classes in VCL for Win32. The .NET Framework was designed to accommodate any .NET-compliant language. In many cases Delphi source code that operates on Win32 VCL classes and functions recompiles with minimal changes on .NET. In some cases, the code recompiles with no changes at all. VCL for .NET is a large subset of VCL, therefore it supports many of the existing VCL classes. However, source code that calls directly to the Win32 API requires source code changes. Also, dependent third-party Win32 VCL controls need to be available in .NET versions for compatibility.

This section introduces:

- VCL for .NET Architecture
- VCL for .NET and the .NET Framework
- VCL for .NET Components
- Borland.VCL Namespace
- Porting Delphi Applications to RAD Studio
- Importing .NET Components for Use in VCL for .NET Applications

## VCL for .NET Architecture



VCL is a set of visual components for building Windows applications in the Delphi language. VCL for .NET is the same library of components updated for use in the .NET Framework. VCL for .NET and the .NET Framework coexist within RAD Studio. Both VCL for .NET and .NET provide components and functionality that allow you to build .NET applications:

- VCL for .NET provides the means to create VCL Forms applications, which are Delphi forms that are .NET-enabled, and use VCL for .NET components.
- VCL for .NET provides VCL non-visual components which have been .NET-enabled to access databases. You can also access databases through the ADO.NET and BDP.NET providers.
- .NET provides the means to build .NET Web Forms and Console applications, using .NET components, with Delphi code-behind.

You can build VCL Forms applications using VCL for .NET components. You can also build ASP.NET Web Forms applications using either VCL for .NET components or .NET components.

## VCL for .NET and the .NET Framework

The .NET Framework provides a library of components, classes, and low-level functionality that manages much of the common functionality, from the display of buttons to remoting functionality, without regard to the underlying implementation language. VCL for .NET and the .NET Framework are functionally equivalent. Like the .NET Framework, VCL for .NET provides libraries of components, controls, classes, and low-level functionality that help you build Web Forms and console applications that run on the current Windows .NET Framework platform.

VCL for .NET is not a replacement for the .NET Framework.

You will still need the .NET runtime to use VCL for .NET, but you can build complete applications using VCL for .NET components that will run on .NET platform.

You can build RAD Studio applications without using VCL for .NET, by creating Web Forms and Console applications using RAD Studio code.

You can use RAD Studio to create powerful .NET applications using .NET components, or VCL for .NET components that have been migrated from the Delphi VCL. If you have existing Delphi VCL applications that you want to run on Windows XP, you can easily port those applications by using RAD Studio.

## VCL for .NET Components

VCL for .NET consists of a set of visual and non-visual components. VCL for .NET builds on the concept of constructing applications visually, eliminating much manual coding.

## Visual Components

RAD Studio provides a set of visual components, or controls, that you can use to build your applications. In addition to the common controls, such as buttons, text boxes, radio buttons, and check boxes, you can also find grid controls, scroll bars, spinners, calendar objects, a full-featured menu designer, and more. These controls are represented differently in RAD Studio than they are in frameworks, such as the .NET Framework.

In an IDE for other languages, such as C# or Java, you will see code-centric representations of forms and other visual components. These representations include physical definitions, such as size, height, and other properties, as well as constructors and destructors for the components. In the **Code Editor** of RAD Studio you will not see a code representation of your VCL for .NET components.

RAD Studio is a resource-centric system, which means that the primary code-behind representations are of event handlers that you fill in with your program logic. Visual components are declared and defined in text files with the extensions `.dfm` (Delphi Forms) or `.nfm` (RAD Studio forms). The `nfm` files are created by RAD Studio as you design your VCL Forms on the Forms Designer, and are listed in the resource list in the Project Manager for the given project.

## Non-Visual Components

You can use non-visual components to implement functionality that is not necessarily exposed visually. For example, you can access data by using non-visual components like the BDP.NET components, which provide database connectivity and DataSet access. Even though these components do not have visual runtime behavior, they are represented by components in the **Tool Palette** at designtime. VCL for .NET provides a variety of non-visual components for data access, server functions, and more.

## Borland.VCL Namespace

VCL for .NET classes are found under the `Borland.Vcl` namespace. Database-related classes are in the `Borland.Vcl.DB` namespace. Runtime library classes are in the `Borland.Vcl.Rtl` namespace.

Unit files have been bundled in corresponding `Borland.Vcl` namespaces. In some cases, units have been moved. However, namespaces are identified in a way that will assist you in finding the functionality you want.

Source files for all of the RAD Studio objects are available in the `c:\Program Files\CodeGear\RAD Studio\5.0\Source` subdirectory.

## Porting Delphi Applications to RAD Studio

If you have existing applications written with an earlier version of Delphi, you might want to port them to .NET. In most cases, this will be easier than rewriting the applications. Because RAD Studio takes advantage of significant structural elements in the .NET Framework, you will need to perform some manual porting tasks to make your applications run. For example, the .NET Framework does not support pointers in safe code. So, any instance of a `pChar` or pointer variable will need to be changed to a .NET type. Many Delphi objects have been updated to accommodate these type restrictions, but your code may include references to pointers or unsupported types. For more information, refer to the Language Guide in this Help system.

## Importing .NET Components for Use in VCL for .NET Applications

RAD Studio provides the **.NET Import Wizard** to help you import .NET controls into VCL for .NET units and packages. For example, you can wrap all .NET components, like those from the `System.Windows.Forms` assembly, in ActiveX wrappers that can be deployed on VCL for .NET applications. Once you have imported the .NET components of your choice, you can add a completed package file containing the units for each component to the **Tool Palette**. You can also view and modify the individual unit files, which can be useful reference material when you are writing your own custom components.

## See Also

[Language Guide](#)

Porting VCL Applications (see page 69)

Data Providers for .NET (see page 27)

Importing .NET Controls to VCL for .NET (see page 165)

### Deploying Applications

Building a VCL for .NET Form Application (see page 149)

## 1.5.5 Porting Web Service Clients

RAD Studio web services use the .NET Framework as the services layer. As a consequence, any existing Delphi 7 or earlier web service client applications need to be modified to use the .NET Framework.

This topic includes:

- Changes and Additions to Your Applications
- Implementation Notes

### Changes and Additions to Your Applications

Make the following changes and additions to your applications:

- Remove RIO components from your applications. They will no longer work in the .NET Framework. Before deleting any components, save property information for the components, such as URLs.
- Remove Delphi 7 SOAP units from the **uses** clause and remove the **uses** reference to the Delphi 7 WSDL Importer-generated Interface proxy unit.
- Add a web reference by choosing the Add Web Reference command from the context menu of the Project Manager.

### Implementation Notes

The following notes address discrete issues that you may encounter when porting web service client applications.

#### Web Service Server Interoperability

Delphi 7 and the .NET SOAP implementations differ slightly. You need to set certain options on the server (TSOAPDomConv.Options property) to succeed in getting multiple clients to work. Set the following options to ensure that both client applications can process the SOAP packages and can transfer the correct encoding:

```
[soTryAllSchema, soRootRefNodesToBody, soUTF8InHeader, soUTF8EncodeXML]
```

### Handling Exceptions

The following table shows the corresponding exceptions between Delphi 7 and the .NET Framework.

Delphi 7 Exception	.NET Framework Exception
ERemotableException	System.Web.Services.Protocols. SoapException
ESOAPHTTPException	System.Net.WebException

### Faultcodes

In Delphi 7, the **ERemotableException.FaultCode** property returns a qualified name, such as *SOAP-ENV:Client.Login*. You must then extract the code using the **ExtractLocalName** function. The .NET Framework SoapException class provides the **SoapException.Code.Name** property directly.



## Monitoring SOAP Packets

In Delphi 7, the THTTPTRIO component provides **OnBeforeExecute** and **OnAfterExecute** events, which allow you to monitor requests and responses. You can implement similar functionality using the .NET Framework SoapExtension class. The following is part of an example that implements this functionality. You can find the full example on Borland's CodeCentral. The link is included in the link list at the end of this topic.

```
uses
  System.Xml, System.Web.Services, System.Web.Services.Protocols, System.IO;

type
  TSoapMessageEvent = procedure (Sender: TObject; const Xml: string) of object;

  TSoapMonitor = class(TObject)
  private
    FOnRequest: TSoapMessageEvent;
    FOnResponse: TSoapMessageEvent;
  protected
    procedure DoRequest(const Xml: string);
    procedure DoResponse(const Xml: string);
  public
    class function FormatXmlData(const Xml: string): string; static;
    property OnRequest: TSoapMessageEvent add FOnRequest remove FOnRequest;
    property OnResponse: TSoapMessageEvent add FOnResponse remove FOnResponse;
  end;
  .
  .
  .
{ TSoapMonitor }

procedure TSoapMonitor.DoRequest(const Xml: string);
begin
  if Assigned(FOnRequest) then
    FOnRequest(Self, Xml);
end;

procedure TSoapMonitor.DoResponse(const Xml: string);
begin
  if Assigned(FOnResponse) then
    FOnResponse(Self, Xml);
end;

class function TSoapMonitor.FormatXmlData(const Xml: string): string;
var
  Doc: XmlDocument;
  Sw: StringWriter;
  Xw: XmlTextWriter;
begin
  Doc := XmlDocument.Create;
  Doc.LoadXml(Xml);
  Sw := StringWriter.Create;
  Xw := XmlTextWriter.Create(sw);
  Xw.Formatting := Formatting.Indented;
  Xw.Indentation := 2;
  Xw.IndentChar := ' ';
  doc.Save(xw);
  Result := sw.ToString;
  Xw.Close;
  Sw.Close;
end;
```

### Working with SOAP Headers

In Delphi 7, you are required to send the header before every method call. The header object is freed after the call. In the .NET Framework, the header class persists after calling a method, until **new** is assigned, or until the header class is cleared by assigning **nil**.

### SOAP Attachments

The .NET Framework does not support MIME attachments. Delphi 7 SOAP does not support DIME attachments.

### See Also

ASP.NET Web Services Overview (🔗 see page 97)

Porting a Delphi for Win32 Web Service Client Application to Delphi for .NET (🔗 see page 207)

[CodeCentral Monitoring SOAP Packets Example](#)

# 1.6 Developing Web Applications with ASP.NET

ASP.NET is the programming model for building Web applications using the .NET Framework. This section provides the conceptual background for building ASP.NET applications using RAD Studio. In addition to supporting data access components within the .NET Framework, RAD Studio includes DB Web Controls. DB Web Controls work with .NET Framework providers and Borland Data Providers for .NET (BDP.NET) to accelerate Web application development.

## Topics

Name	Description
ASP.NET Overview (see page 79)	<p>ASP.NET is the .NET programming environment for building applications in HTML that run on the Web. This topic provides introductory information about the major components of the ASP.NET architecture and explains how ASP.NET integrates with other programming models in the .NET framework. This topic introduces:</p> <ul style="list-style-type: none"> <li>• ASP.NET Architecture</li> <li>• Web Forms</li> <li>• Data Access</li> <li>• Web Services</li> <li>• Designtime Features</li> <li>• Supported Web Servers</li> <li>• Sample Applications</li> </ul>
CodeGear DB Web Controls Overview (see page 81)	<p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p> <p>CodeGear DB Web Controls simplify database development tasks in combination with BDP.NET and .NET Framework data access components. DB Web Controls are data-aware controls that provide advanced functionality, including data-aware grid, navigator, calendar, combobox, and other popular components.</p> <p>This section introduces:</p> <ul style="list-style-type: none"> <li>• DB Web Controls Architecture</li> <li>• Data-aware Components Advantages</li> <li>• Supported Data Access Components</li> <li>• DB Web Controls Namespace</li> <li>• ASP.NET Application Deployment with DB Web Controls</li> </ul>
Using DB Web Controls in Master-Detail Applications (see page 83)	<p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p> <p>DB Web Controls allow you to build full-fledged master-detail applications, using the DBWebDataSource, DBWebGrid, and DBWebNavigator controls. To support master-detail applications, these controls must provide a way to specify cascading behavior.</p> <p>This topic includes information about:</p> <ul style="list-style-type: none"> <li>• Specifying Cascading Deletes</li> <li>• Specifying Cascading Updates</li> </ul>

DB Web Controls Navigation API Overview (see page 85)	<p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p> <p>Although you can use the standard DBWebNavigator control for most applications, you may need to exercise more control over the navigation in your application. The DB Web Controls now provide an API that allows you to fine-tune your navigation. For example, using the API, you can create a button that performs navigation directly, rather than using the standard DBWebNavigator control. Although you can hide buttons on the DBWebNavigator, you might want to place controls in different locations on the form. With DBWebNavigator,... more (see page 85)</p>
DB Web Control Wizard Overview (see page 86)	<p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p> <p>The CodeGear DB Web Controls are data-aware web components. These DB Web Controls allow you to encapsulate data-aware functionality into standard web controls. One benefit of this approach is that the data binding function is fulfilled by the control itself, eliminating the need to add a call to the DataBind method.</p> <p>The basic concepts involved in creating DB Web Controls are:</p> <ul style="list-style-type: none"> <li>• The ASP.NET Control Execution Lifecycle</li> <li>• Data Binding Concepts</li> <li>• Overriding ASP.NET Methods</li> <li>• Implementing DB Web Interfaces</li> <li>• Essential Code Modifications</li> </ul>
Using XML Files with DB Web Controls (see page 92)	<p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p> <p>The DBWebDataSource component provides a way for you to create and use XML and XSD files as the data source for an ASP.NET application. Typically, you only use these types of files with the DBWeb controls as a way of prototyping your application. By using XML files as the data source, you can eliminate potentially costly database resources during the design and development phase of your project.</p> <p>This topic covers the following issues.</p> <ul style="list-style-type: none"> <li>• XML files as data sources.</li> <li>• Suggested workflow strategy.</li> <li>• Authentication... more (see page 92)</li> </ul>
Working with DataViews (see page 94)	<p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p> <p>With DataViews you can set filters on a DataTable using the RowFilter property or place data in a specific order. You can find the DataView component under the <b>Data Components</b> area of the <b>Tool Palette</b>. This topic discusses:</p> <ul style="list-style-type: none"> <li>• Runtime Properties</li> <li>• Master-Detail Relationships</li> <li>• ClearSessionChanges Method</li> <li>• DataView Limitations</li> </ul>
Deploying ASP.NET Applications (see page 95)	<p>This topic provides information about:</p> <ul style="list-style-type: none"> <li>• Web Server Requirements</li> <li>• Pre-Deploy Recommendations</li> <li>• The RAD Studio ASP.NET Deployment Manager</li> </ul> <p>For additional deployment information, see the <a href="#">deploy.htm</a> file located, by default, at <code>C:\Program Files\CodeGear\RAD Studio\5.0.</code></p>

Working with WebDataLink Interfaces (see page 96)

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

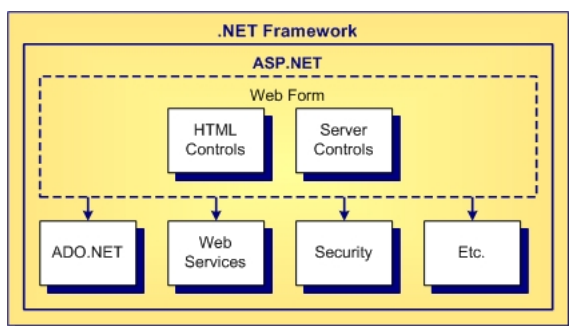
The characteristic that makes DB Web Controls different from traditional web controls is that the DB Web Controls automatically handle all data binding for you. Although you must still configure the links between data sources and controls at design time, all runtime binding is handled, without the need for you to add a data binding command in your code. When extending a DBWeb control using the **DBWeb Control Wizard**, you will implement several interfaces that provide the data binding capabilities. These... more (see page 96)

## 1.6.1 ASP.NET Overview

ASP.NET is the .NET programming environment for building applications in HTML that run on the Web. This topic provides introductory information about the major components of the ASP.NET architecture and explains how ASP.NET integrates with other programming models in the .NET framework. This topic introduces:

- ASP.NET Architecture
- Web Forms
- Data Access
- Web Services
- Designtime Features
- Supported Web Servers
- Sample Applications

### ASP.NET Architecture



The major components of the ASP.NET architecture are Web Forms, ASP.NET server controls, code-behind logic files, and compiled DLL files. Web Form pages contain HTML elements, text, and server controls. Code-behind files contain application logic for the Forms page. Compiled DLL files render dynamic HTML on the web server.

CodeGear provides tools to simplify ASP.NET development. If you are familiar with rapid application development (RAD) and object oriented programming (OOP) using properties, methods, and events, you will find the ASP.NET model for building Web applications familiar.

### Web Forms, Server Controls, and HTML Elements

Web Forms define the user interface for your Web application. Typically, a Web Form consists of a markup file (`.aspx`) that provides the visual presentation and a code-behind file (`.pas` or `.cs`) that provides the program logic. The code-behind file is compiled to a `.dll` and deployed to the server with the `.aspx` file. At runtime, the `.aspx` is compiled and linked against the code-behind `.dll`. This enables you to change the `.aspx` file without recompiling the code-behind file.

The Web Form `.aspx` file consists of ASP.NET server controls and static HTML elements. Server controls are declared in your code and can be accessed programmatically through properties, methods, and event handlers. They run on the web server and

render HTML to send back to the client.

HTML elements are static, client-side controls; they are not, by default, programmatically accessible. However, they are well suited for static text and images on a Web Form.

### Data Access

Web Forms can access data through ADO.NET. You can connect an ASP.NET application to an ADO.NET data source by using the data components included in the .NET Framework, the AdoDbx Client, Blackfish SQL, or the Borland Data Provider (BDP.NET) components.

### Web Services

Web Services provide application components to many distributed systems using XML-based messaging. A web service can be as simple as an XML message updating values in a remote application or can be an integral part of a sophisticated ASP.NET or ADO.NET application. Web Services and ASP.NET share a common .NET infrastructure that allows for seamless integration.

### Supported Web Servers

RAD Studio supports two servers for developing ASP.NET applications: Internet Information Services 6.0 (IIS) and Cassini. You can use both IIS and Cassini on the same computer, provided you configure them to use different ports.

- IIS is a comprehensive, scalable web server and is included with Windows Server 2003. You can deploy applications to a computer running IIS.
- Cassini is a web server this is used during the development process, but is not intended for application deployment. It is easier to use than IIS because there is no configuration. Cassini was developed by Microsoft and made available for free download with source. RAD Studio ships with a slightly customized version of Cassini that is integrated into our ASP.NET support.

When you create an ASP.NET application, RAD Studio prompts you to specify the web server and location for the application. You can set the default server and location for new applications, as well as the Cassini location and port, on the

[Tools ► Options ► ASP.NET](#) options page.

### Designtime Features

RAD Studio provides several designtime features to help you accelerate the development of Web Forms, HTML, and CSS files.

### Editing HTML and CSS Files

Many of the **Code Editor** features are also available when editing HTML and CSS files. Code Completion (**CTRL+SPACE**) and syntax highlighting are available for HTML and CSS files. Error Insight is available for HTML files and highlights invalid HTML with a wavy red underline. If you position the mouse over the highlighted HTML, a hint window is displayed indicating the probable cause of the error.

When displaying an HTML page, the internal HTML formatter automatically indents the HTML to improve readability. Alternatively, you can use HTML Tidy, the standard formatting tool from [www.w3c.org](http://www.w3c.org). You can use HTML Tidy as needed to format the file and check for errors by choosing the [Edit ► HTML Tidy](#) menu commands. Alternatively, you can set it as the default formatter, instead of the internal formatter. You can also define tags that HTML Tidy would otherwise detect as invalid, such as those prefixed with `asp:`. To access the HTML Tidy options, choose [Tools ► Options ► HTML Tidy Options](#).


The **Structure View** displays a hierarchical tree view of the HTML tags in the active HTML page and is useful for navigating large files. Double-clicking a node in the tree view positions the HTML file to the corresponding tag.

### Designer Flow Layout and Grid Layout

When designing a Web Form, you can use either *grid layout* or *flow layout* for the Designer. In grid layout, controls are arranged by absolute position and you can reposition them by dragging them on the form. An optional, visible grid is also available to help you align controls. If you drag a control from the **Tool Palette** onto the Web Form, or if you click the control on the **Tool Palette** and then click Web Form, the control is added using absolute positioning.

In flow layout, controls are arranged top to bottom on the Web Form, and you can reposition them by using the arrow keys. If you

double-click a control on the **Tool Palette**, it will be added to the Web Form in flow layout.

The layout for an individual control can be changed by using the **Absolute Layout** button  on the **HTML Design** toolbar at the top of the Designer.

To permanently change the layout for new files created with RAD Studio, you can edit the [page.aspx](#) template file located at, by default, [CodeGear\RAD Studio\5.0\ObjRepos\DelphiDotNet](#).

### Sample Applications

RAD Studio includes several ASP.NET sample applications in the [Demos](#) directory. Many of the sample applications include a [readme](#) file that explains the application and lists any prerequisites. Before you attempt to open a sample application in the IDE:

- Check for a [readme](#) file in the application's directory and follow any set up instructions.
- Create a virtual directory for the sample application to avoid [resource cannot be found](#) errors in the browser at runtime (see the procedure listed at the end of this topic).

### See Also

[ADO.NET Overview](#) ( see page 14)

[Web Services Overview](#) ( see page 97)

[Building an ASP.NET Application](#) ( see page 170)

[DB Web Controls for ASP.NET](#) ( see page 81)

[Deploying ASP.NET Applications](#) ( see page 95)

[Creating a Virtual Directory for Demo Applications](#) ( see page 188)

[System.Web Namespace](#)

[.NET Framework Developer's Guide ASP.NET Web Applications \(MSDN\)](#)

---

## 1.6.2 CodeGear DB Web Controls Overview

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

CodeGear DB Web Controls simplify database development tasks in combination with BDP.NET and .NET Framework data access components. DB Web Controls are data-aware controls that provide advanced functionality, including data-aware grid, navigator, calendar, combobox, and other popular components.

This section introduces:

- DB Web Controls Architecture
- Data-aware Components Advantages
- Supported Data Access Components
- DB Web Controls Namespace
- ASP.NET Application Deployment with DB Web Controls

### DB Web Controls Architecture

DB Web Controls are a set of visual and non-visual components that speed up the creation of ASP.NET applications by providing drag-and-drop capabilities along with a powerful data source discovery mechanism. For the most part, DB Web Controls are common GUI web controls for ASP.NET applications. The connector control, the `DBWebDataSource` control, acts as a data-aware connector between the visual controls and the underlying data source. In other words, the `DBWebDataSource`

control acts as a conduit for the data that is stored in a data source and the controls that display that data on your ASP.NET form. The DBWebDataSource control can reference both .NET Framework ADO.NET and BDP.NET components. For example, the in-memory DataSet that is generated by an ADO.NET adapter (such as the SQLDataAdapter) or by one of the BDP.NET adapters (such as the BDPDataAdapter). Additionally, you can use the DBWebDataSource to link to other types of data source providers, such as text files, arrays, or collections.

### Data-Aware Components Advantages

Typically, when you create an ASP.NET application that features controls that expose data from an underlying data source, such as a database, you need to manually configure the binding between the data source and the controls. This means figuring out the syntax and parameters for each control that must be bound to the data source.

The major advantage of using DB Web Controls is that once you have connected one DBWebDataSource control to your data source, all of the DB Web Controls on your ASP.NET page that reference the DBWebDataSource automatically bind to the underlying data source. You do not need to add any code to accomplish the data binding.

DB Web Controls provide the following advantages over standard web controls:

- Eliminates a need to call the **WebControl.DataBind** method. Normally, each ASP.NET control on the web form requires that you add this call in the Page\_Load routine or the control will not display data at runtime.
- Provides a design-time view of the data.
- Posts changes back to the DataSet automatically. Typically, ASP.NET controls require code to post back changes.
- Maintains current row position.
- Manages change and row state automatically. This means that clients from different machines can operate independently, without regard to the server-side state.

In addition to these general advantages, DB Web Controls provide the following specific advantages:

- The DBWebDataSource maintains an ordered list of changes so that the user can undo changes in the order in which they were made.
- The DBWebNavigator control provides navigation capabilities for grids, multiple text controls, and can be extended to standard web controls.
- The DBWebDataGrid provides built-in capabilities for paging with numbers and icons, for adding Edit and Delete columns, and other advanced capabilities. In other words, you no longer need to code these features into your grid control.

### Supported Data Access Components

DB Web Controls are compatible with .NET Framework ADO.NET and CodeGear BDP.NET data access components. Any data source that can be accessed by one of these providers can serve as the underlying data source for the DB Web Controls. In addition, many of the DB Web Controls, like many .NET web controls in general, can access other objects as data sources, such as arrays, collections, and files.

### DB Web Controls Namespace

The namespace for DB Web Controls is **Borland.Data.Web**. By using reflection, you can learn much about the structure of the namespace and the controls. You can add the namespace to your project, then open it in the **Code Editor**. This opens the **Reflection Editor** and gives you a hierarchical view of all of the controls and their members.

Control	Description
DBWebDataSource	Acts as a bridge between the data source and the DBWeb controls.
DBWebAggregateControl	Text box control that displays aggregate values from a specified column.
DBWebCalendar	A calendar control.
DBWebCheckBox	A check box control.
DBWebDropDownList	A combo box control.
DBWebGrid	A data grid.



DBWebImage	An image control.
DBWebLabel	A label.
DBWebLabeledTextBox	A text box with an attached label.
DBWebListBox	A list box control.
DBWebMemo	A memo field control.
DBWebNavigationExtender	A non-visual component that allows you to define standard web control buttons as navigation controls.
DBWebNavigator	A navigation bar.
DBWebRadioButtonList	A radio button group.
DBWebSound	A sound control, which uses the default media player on your system.
DBWebTextBox	A text box.
DBWebVideo	A video control, which uses the default media player on your system.

### ASP.NET Application Deployment with DB Web Controls

After creating an ASP.NET project with DB Web Controls, deploy your ASP.NET application as usual. No special considerations are required.

#### See Also

Data Providers for .NET (see page 27)

Building an Application with DB Web Controls (see page 183)

Building an ASP.NET Application (see page 170)

Using XML Files with DB Web Controls (see page 92)

Working with DataViews (see page 94)

WebDataLink Interfaces (see page 96)

Deploying ASP.NET Applications (see page 95)

## 1.6.3 Using DB Web Controls in Master-Detail Applications

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

DB Web Controls allow you to build full-fledged master-detail applications, using the DBWebDataSource, DBWebGrid, and DBWebNavigator controls. To support master-detail applications, these controls must provide a way to specify cascading behavior.

This topic includes information about:

- Specifying Cascading Deletes
- Specifying Cascading Updates

### Cascading Deletes

In a master-detail application, the application typically uses an OnApplyChanges event to send the DataSet changes to the server. It is necessary for the master data adapter's update method (in BDP.NET, the AutoUpdate event) to be called prior to the detail data adapter's update method. Otherwise, insertion of detail rows fails if the master row has not yet been inserted. If the

master row is deleted prior to the detail row, the server might return an error.

The property `CascadingDeletes` has been added to the `DBWebDataSource` control. The `CascadingDeletes` property specifies how the server deletes rows in master-detail applications. The `CascadingDeletes` property provides the following three options:

- `NoMasterDelete` (Default)
- `ServerCascadeDelete`
- `ServerNoForeignKey`

**Note:** When DB Web Controls are connected to a `DataTable` that is a detail table in a relation, the control's rows are automatically limited to the rows controlled by the current parent row in the master table.

### NoMasterDelete

This option does not allow deletion of a master row containing detail rows. This option should be used when the server enforces a foreign constraint between master and detail, but it does handle cascading deletes. You must:

1. Delete detail rows.
2. Apply the changes with an apply event (for example, the `BdpDataAdapter.AutoUpdate` event).
3. Delete the master row.
4. Call the apply event (for example, the `BdpDataAdapter.AutoUpdate` event).

This option is the default value for the `CascadingDeletes` property.

### ServerCascadeDelete

This option allows deletion of the master row. This option should be specified whenever the server is set up to automatically handle cascading deletes. When a master row is deleted, the detail rows will automatically disappear from view. Any time prior to applying the change, you can undo the parent row deletion and all the detail rows come back into view. If the server is not set up to handle cascading deletes, an error may occur when attempting to send changes to the server.

### ServerNoForeignKey

This option automatically deletes all detail rows whenever a master row is deleted. This option should be specified whenever there are no foreign key constraints between the master-detail tables on the server. Like the **ServerCascadeDelete** option, when a master row is deleted, the detail rows will automatically disappear from view. Any time prior to applying the change, it is possible to undo the master row deletion to redisplay the detail rows. If you specify this option and foreign key constraints exist between master and detail tables, an error will be thrown by the server when attempting to delete the master table.

### Cascading Updates

In a master-detail application, the application typically uses an `OnApplyChanges` event to send the `DataSet` changes to the server. It is necessary for the update method of the master data adapter (in `BDP.NET`, the `AutoUpdate` event) to be called prior to the update method of the detail data adapter. Otherwise, insertion of detail rows fails if the master row has not yet been inserted. If the master row is deleted prior to the detail row, the server might return an error.

The property `CascadingUpdates` has been added to the `DBWebDataSource` control. This property specifies how the server updates foreign-key values in master-detail applications. The `CascadingUpdates` property provides the following three options:

- `NoMasterUpdate` (default)
- `ServerCascadeUpdate`
- `ServerNoForeignKey`

**Note:** When DB Web Controls are connected to a `DataTable` that is a detail table in a relation, the rows of the control are automatically limited to the rows controlled by the current parent row in the master table.

### NoMasterUpdate

This option does not allow changes to the foreign key value of a master row if it has any associated detail rows. This option is the default value for the CascadingUpdates property.

### ServerCascadeUpdate

This option allows you to change the foreign key value of the master row. You should use this option whenever the server automatically handles cascading updates. When the foreign key value of a master row is changed, the key value is changed automatically in the detail rows. Anytime prior to applying the change, you can undo the change to the master row and all the detail key changes will be undone also. If the server is not set up to handle cascading updates, an error might occur when attempting to update the changes to the server.

### ServerNoForeignKey

This option also allows changing the foreign key value of the parent row, but should be used whenever there is no foreign key between the master and detail tables on the server.

### See Also

Working with DataViews (see page 94)

Building Applications with DBWeb Controls (see page 183)

---

## 1.6.4 DB Web Controls Navigation API Overview

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

Although you can use the standard DBWebNavigator control for most applications, you may need to exercise more control over the navigation in your application. The DB Web Controls now provide an API that allows you to fine-tune your navigation. For example, using the API, you can create a button that performs navigation directly, rather than using the standard DBWebNavigator control. Although you can hide buttons on the DBWebNavigator, you might want to place controls in different locations on the form. With DBWebNavigator, for instance, if you hide all buttons but Previous and Next, they still appear side by side. To place the buttons on opposite sides of the form, use the navigation API methods or the DBWebNavigationExtender control. Both allow you to turn standard web control buttons into navigation controls.

To provide this capability, the DBWebDataSource implements new IDBDataSource methods, each of which perform a specific navigation task. You include these methods in the **Form\_Load** event. You are not required to include click events.

The following methods are provided:

- RegisterNextControl
- RegisterPreviousControl
- RegisterFirstControl
- RegisterLastControl
- RegisterInsertControl
- RegisterDeleteControl
- RegisterUpdateControl
- RegisterCancelControl
- RegisterUndoControl

- RegisterUndoAllControl
- RegisterApplyControl
- RegisterRefreshControl
- RegisterGoToControl

**See Also**

CodeGear DBWeb Controls (see page 81)

Building an Application with DBWeb Controls (see page 183)

## 1.6.5 DB Web Control Wizard Overview

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

The CodeGear DB Web Controls are data-aware web components. These DB Web Controls allow you to encapsulate data-aware functionality into standard web controls. One benefit of this approach is that the data binding function is fulfilled by the control itself, eliminating the need to add a call to the `DataBind` method.

The basic concepts involved in creating DB Web Controls are:

- The ASP.NET Control Execution Lifecycle
- Data Binding Concepts
- Overriding ASP.NET Methods
- Implementing DB Web Interfaces
- Essential Code Modifications

**The ASP.NET Control Execution Lifecycle (CEL)**

Anytime an ASP.NET web forms page is displayed, ASP.NET performs what Microsoft calls the CEL. This consists of a number of steps, which are represented by methods:

1. Initialize
2. Load view state
3. Process postback data
4. Load
5. Send postback change notifications
6. Handle postback events
7. Prerender
8. Save state
9. Render
10. Dispose
11. Unload

You can add logic to any or all of these events by adding code to given methods, such as the **Page\_Load** method or the **OnInit** method. Most often, however, you will need to override the **Render** method.

**Data Binding**

In ASP.NET you can bind to a variety of data sources including databases, text files, XML files, arrays, and collections. In RAD

Studio, controls provide a simple property-based interface to data sources. In the **Object Inspector**, you can bind a selected control to a data source that is identified to your project by way of the BDP.NET controls, SQL client controls, or other data or file controls. Each type of data control has different sets of binding requirements. For instance, any collection control, such as the listbox control, data grid, or listview control, must bind to a data source that implements the ICollection interface. Other controls, like buttons and text boxes, do not have this requirement.

When you are programming with web controls, you must add the code to perform the data binding. For example, if you created an instance of a data grid, the command that you would add would look like:

```
dataGrid1.DataBind();
```

When using DB Web Controls, you no longer need to add this code. DB Web Controls handle the data binding operation for you. The DBWebDataSource component serves as a bridge between your data source component and the specific DB Web control you want to use. The DBWebDataSource creates and manages the data binding between the data source and the control. Although you can physically add the code to instantiate a DB Web control and to perform the data binding, it is unnecessary to do so. You can drop your components onto a web form and select the linkages from property drop down list boxes in the **Object Inspector**.

**Note:** When creating a new DB Web control or extending an existing control, you may need to add code to perform binding of some properties.

### Overriding ASP.NET Methods

The main method you will need to override is the Render method (or the **RenderContents** method). The Render method is responsible for displaying your controls visibly on the web page. When you define the Render method and pass it an instance of the HtmlTextWriter class, you are indicating that whatever you code in the method is to be written to the ASP.NET page in HTML. The Write method of the HtmlTextWriter class writes a sequential string of HTML characters onto a Web Forms page.

The following example shows how the control is declared in the file that is built by the **DB Web Control Wizard**. This is only a small segment of the code that is provided for you.

```
/// TWebControl1 inherits from the WebControl class of System.Web.UI.WebControls.
```

```
TWebControl1 = class(System.Web.UI.WebControls.WebControl)
```

When creating your own controls or extending existing controls, you must override the Render method to display your control. The Render method is responsible for sending output to an instance of an HtmlTextWriter class. HtmlTextWriter sends a sequence of HTML characters to the web forms page. The HTML characters are the representation in HTML of your control. For example, a web grid control is represented on a web forms page as an HTML table. Each control has its own HTML representation, but when you extend a control, you need to modify how the HTML is emitted to accurately represent your new control.

```
/// The following lines declare the Render method.
/// Output represents an instance of the HtmlTextWriter class.
/// HtmlTextWriter is the class that writes HTML characters to
/// the ASP.NET Web Forms page.

    strict protected
        procedure Render(Output: HtmlTextWriter); override;

implementation

{$REGION 'Control.Render override'}

/// The following procedure is the overridden Render method
/// You can include additional logic in the procedure to affect
/// the behavior of the control. This method, as written, does
/// nothing but write out a sequence of HTML characters that
/// define TWebControl1.
```

```

procedure TWebControl1.Render(Output: HtmlTextWriter);
begin
    Output.Write(Text);
end;

```

You would need to implement the preceding code even if you were trying to extend the capabilities of a standard web control. To extend one of the DB Web Controls you need to make more adjustments to this code.

## Implementing DB Web Interfaces

When you run the **DB Web Control Wizard**, the wizard creates a code file for you, containing the basic code you need to extend a DB Web control. This file is similar to the file you would create if you were trying to extend a standard web control. The major difference is that the **DB Web Control Wizard** adds implementations of specific DB Web interfaces, which provide automatic access to a data source, tables, columns and their respective properties. Because the DB Web Controls handle so much of the postback and data binding automatically, you need to implement several specific interfaces to add this functionality to your control.

## Essential Code Modifications

When you create a new **DB Web Control Library**, the **DB Web Control Wizard** creates a file template for you. This file contains the major elements you need to include in your project to create or extend a control. You will need to add or modify the following elements:

- Change the `ToolboxBitmap` attribute to specify your own icon for the Tool Palette, if necessary.
- Change the control declaration to specify the control you intend to inherit.
- Declare the correct `Render` method.
- Implement the `IDBWebDataLink` interface.
- Implement the `IDBWebColumnLink` and `IDBWebLookupColumnLink` interfaces, if necessary.
- Modify or extend the `Render` method.
- Modify hidden field registration, if necessary.
- Set data binding on specific properties, if necessary.

## Change the ToolboxBitmap Attribute

If you have a bitmap icon available for use in the Tool Palette, specify its path in the `ToolboxBitmap` attribute in the **DB Web Control Library** file. The code might look something like this:

```

[ToolboxBitmap(typeof(WebControl1)]
['WebControl1.bmp'])

```

Make sure that you include the bitmap file in your project.

## Change the Control Declaration

You can specify the ancestor more specifically. For example, if your control is an extended version of a `DBWebGrid` control, the code would look like this:

```
MyDataGrid = class(Borland.Data.Web.DBWebGrid, IPostBackDataHandler, IDBWebDataLink)
```

## Declare the Correct Render Method

Your control can inherit from either the `Control` namespace or the `WebControls` namespace. `WebControls` actually derives from the `Control` namespace.

The major difference for you is that `WebControls` defines all of the standard web controls, so if you plan on extending the

capabilities of a web control like a textbox or a data grid, your control needs to inherit from [WebControls](#).

By inheriting from [WebControls](#), you are able to use all of the appearance properties of your base control. Typically, if you want to create a control that has a UI, inherit from [System.Web.UI.WebControls](#). In the [DB Web Control Library](#) file, you will override the `RenderContents` method.

If your control inherits from [Control](#), you need to supply the UI definition when you override the `Render` method. Typically, if you want to create a control that has no UI, you inherit from [System.Web.UI.Control](#). In the [DB Web Control Library](#) file, you will override the `Render` method.

### Implement the IDBWebDataLink Interface

This interface provides the access to a data source. You need to implement this interface for any DB Web control you intend to extend. The implementation is handled for you in the [DB Web Control Library](#) file.

### Modify or Extend the Render Method

In the `Render` or `RenderContents` method, depending on which namespace you inherit from, you can override the properties of the base class. In the [DB Web Control Library](#) file the following code is automatically included for you:

```
procedure TWebControl1.Render(Output: HtmlTextWriter);
begin
    Output.Write(Text);
end;
```

This method passes the definition of your control to an instance of `HtmlTextWriter`, called `Output` in this case. The `Text` property will contain the HTML text that is to be rendered. If you wanted to code directly within the method, You could add code, as follows:

```
procedure TWebControl1.Render(Output: HtmlTextWriter);
begin
    Output.WriteFullBeginTag("html");
    Output.WriteLine();

    Output.WriteFullBeginTag("body");
    Output.WriteLine();

    Output.WriteEndTag("body");
    Output.WriteLine();

    Output.WriteEndTag("html");
    Output.WriteLine();
end;
```

This results in an ASP.NET web page with the following HTML code:

```
<html>
  <body>
</body>
</html>
```

The use of the `Text` property, however, makes the code easier to work with. Once you have defined your control and its properties, along with various HTML tags, you can pass the entire structure to the `Text` property. From that point forward, you need only refer to the `Text` property to act upon the control. You define the properties of your control and pass them to the `HtmlTextWriter` by creating a `Text` property that contains the control definition. It is instructive to look at the source code for some of the existing DB Web Controls. For example, the following code shows the definition of the `Text` property for the `DBWebNavigator` control.

```
protected string Text{
    get
    {
        // Create a new instance of StringWriter.
```

```

        StringWriter sw = new StringWriter();

// Create a new instance of HtmlTextWriter.
        HtmlTextWriter tw = new HtmlTextWriter(sw);

// Call the DataBind procedure.
        DataBind();

// Call the AddButtons procedure.
        AddButtons();

// Call the SetButtonsWidth procedure.
        SetButtonsWidth();

// Add a style to a panel.
        ClassUtils.AddStyleToWebControl(FPanel, this.Style);

// Render the HTML start tag for a panel control.
        FPanel.RenderBeginTag(tw);

// Call the HtmlTextWriter.Write method and pass the table
// and tablerow tags to the web forms page.
        tw.Write("<table><tr>");

// If the ButtonType is set to ButtonIcons, iteratively create and render buttons
// to the web forms page.
        if( ButtonType == NavigatorButtonType.ButtonIcons )
        {
            for( int i = 0; i < IconNavButtons.Count; i++ )
            {
// Write the first table cell tag.
                tw.Write("<td>");

// Instantiate an image button.
                ImageButton b = (IconNavButtons[i] as ImageButton);

// Render the button on the web page.
                b.RenderControl(tw);

// Write the closing table cell tag.
                tw.Write("</td>");
            }
        }
        else
        {
            // If the ButtonType is something other than ButtonIcons, iteratively create and
            // Render default navigation buttons to the web forms page.
            {
                for( int i = 0; i < NavButtons.Count; i++ )
                {
// Write the first table cell tag.
                    tw.Write("<td>");

// Instantiate a button.
                    Button b = (NavButtons[i] as Button);

// Render the button on the web page.
                    b.RenderControl(tw);

// Write the closing table cell tag.

```



```

        tw.Write("</td>");
    }
}

// Write the closing row and table tags.
tw.Write("</tr></table>");

// Render the Panel end tag.
FPanel.RenderEndTag(tw);
return sw.ToString();
}
}

```

### Modify Hidden Field Registration

The **DB Web Control Library** file includes a call to register a hidden field, which identifies the key for a read-write control. If you are creating a read-only control, you can remove or comment out this call. The call is as shown in the following sample:

```

Page.RegisterHiddenField(DBWebDataSource.IdentPrefix +
DBWebConst.Splitter + IDataLink.TableName, self.ID);

```

### Set Data Binding on Specific Properties

If you need other properties data bound, other than the Text property, you can add that data binding code in the same location where you find that the Text property is being bound. Typically, there is a call to `DataBind` in the `PreRender` method. The `DataBind` procedure itself is similar to the following sample, taken from the `DBWebLabeledTextBox` control source code. You can see in the following code that a number of properties are set after checking to see if the `FColumnLink` (from the `IDBWebDataColumnLink` interface) is bound to some data source.

```

public override void DataBind()
{
    try
    {
        FTextBox.ReadOnly = FReadOnly;
        FTextBox.ID = this.ID;
        base.DataBind();
        ClassUtils.SetBehaviorProperties(FPanel, this);
        ClassUtils.SetOuterAppearanceProperties(FPanel, this);
        ClassUtils.SetSizeProperties(FPanel, this);
        if( !ClassUtils.IsEmpty(FLabel.Text) )
        {
            ClassUtils.SetInnerAppearanceProperties(FLabel, this);
            SetProportionalSize();
            SetLabelFont();
            FTextBox.Text = null;
        }

        // If there is a data source.
        if( IColumnLink.DBDataSource != null )
        {
            // And if there is bound data.
            if( FColumnLink.IsDataBound )
            {
                // Then set behavior properties.
                ClassUtils.SetBehaviorProperties(FTextBox, this);

                // Set appearance properties.
                ClassUtils.SetAppearanceProperties(FTextBox, this);
            }
        }
    }
}

```

```

// Set size properties.
        ClassUtils.SetSizeProperties(FTextBox, this);
        object o = IColumnLink.DBDataSource.GetColumnValue(Page,
IColumnLink.TableName, IColumnLink.ColumnName);

// If the page and the table and column names are not null,
// it means there is already bound data.
// Put the string representation of the page, table, and
// column names into the textbox.
        if( o != null )

            FTextBox.Text = Convert.ToString(o);

        else

// Otherwise, clear the textbox and bind it and
// its properties to the specified
column.
            FTextBox.Text = "";
            FTextBox.DataBind();
    }
}

```

**See Also**

CodeGear DB Web Controls Overview (🔗 see page 81)

Working with WebDataLink Interfaces (🔗 see page 96)

Using the DB Web Control Wizard (🔗 see page 195)

## 1.6.6 Using XML Files with DB Web Controls

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

The DBWebDataSource component provides a way for you to create and use XML and XSD files as the data source for an ASP.NET application. Typically, you only use these types of files with the DBWeb controls as a way of prototyping your application. By using XML files as the data source, you can eliminate potentially costly database resources during the design and development phase of your project.

This topic covers the following issues.

- XML files as data sources.
- Suggested workflow strategy.
- Authentication and caching issues.

**XML Files as Data Sources**

XML has become another standard data source for many applications, but for ASP.NET applications in particular. When working with data that does not require strong security and therefore can be sent over HTTP as text, XML files provide a simple solution. Because the files are text, they are easy to read. Because the XML tags describe the data, you can understand and process the data structures with little difficulty.

Despite their obvious advantages over more complex data structures, XML files do have some drawbacks. For one thing, they are not secure, therefore, it is not a good idea to pass sensitive data, such as credit card numbers or personal identification (PIN) numbers, over the Internet by way of XML files. Another drawback is the lack of concurrency control over XML records, unlike database records.

Nonetheless, the self-describing nature and the lightweight data format of XML files makes them a natural choice as data sources for ASP.NET applications. The DBWebDataSource control, in particular, has been built to handle XML files as well as other types of data sources. There are no special requirements for using XML files, no unique drivers or communication layers beyond those that come with RAD Studio, so you will find it easy to work with XML files as data sources.

### Suggested Workflow Strategy

You use the DBWebDataSource control to create the XML file for your application and to connect the XML file with a DataSet object. The basic workflow strategy is this:

- Build an ASP.NET application, with a connection to your target database. Use DBWeb controls, including a DBWebDataSource and specify a non-existent XML file. When you run the application, your DataSet receives the result set from the target database and the DBWebDataSource then fills the XML file with tagged data representing the DataSet.
- From this point forward, you can eliminate the data adapter and data connection, keeping only a DataSet, the DBWebDataSource, and the reference to the XML file. Your DBWeb controls will pull data from the XML file and DataSet rather than from the database. For more information, follow the links to specific procedures on building and using XML files with DBWeb controls.

### Authentication and Caching Issues

The DB Web Controls support automatic reading of an XML file by the DBWebDataSource component at both designtime and runtime. To support XML files, the DBWebDataSource component includes caching properties. If you use XML caching, the XML file data is automatically read into the DataSet whenever a data source is loaded.

If you do not implement user authentication in your application, you will likely only use this feature for prototyping. Otherwise, without user authentication, users may experience permissions errors when trying to access a single XML file concurrently. When multiple clients are using the application, the XML file is constantly being overwritten by different users. One way to avoid this is to write logic in your server application to check row updates and notify various clients when there is a conflict. This is similar to what a database system does when it enforces table-level or row-level locking. When using a text file, like an XML file, this level of control is more difficult to implement.

However, if you implement user authentication, you can create a real-world application by setting the UseUniqueFileName property. This property specifies that the DBWebDataSource control will create uniquely named XML files for each client that uses accesses the XML file specified in the XMLFileName property of the DBWebDataSource. This helps avoid data collisions within a multi-user application. The drawback to this approach is that each XML file will contain different data and your server application will need built-in logic to merge the unique data from each client XML file.

Read-write applications using XMLFileName require that all web clients have write access to the XML files to which they are writing. If the web client does not have write access, the client will get a permissions error on any attempt to update the XML file. You must grant write access to the clients who will use the application.

### See Also

CodeGear DB Web Controls Overview (see page 81)

Creating a DB Web XML File (see page 185)

Building a Briefcase Application with DB Web Controls (see page 182)

---

## 1.6.7 Working with DataViews

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

With DataViews you can set filters on a DataTable using the RowFilter property or place data in a specific order. You can find the DataView component under the **Data Components** area of the **Tool Palette**. This topic discusses:

- Runtime Properties
- Master-Detail Relationships
- ClearSessionChanges Method
- DataView Limitations

### Runtime Properties

At designtime, when a DBWeb control points to a DataView, the control is automatically updated whenever there is a change to any DataView property that controls the rows to be displayed. To change the DataView properties at runtime, you must make sure that the change is in place prior to the rendering of any of the DB Web Controls.

For example, if you use a listbox to set the filter, you would also:

- Set the listbox AutoPostBack property to **True**.
- Add code in the **Page\_Load** event to handle setting the RowFilter.
- Add code in the **Page\_Load** event to call the ClearSessionChanges method after the RowFilter has been changed.

Assume you have two tables on a form. You bind an ASP.NET listbox to one table that contains lookup values. These values serve as a filter for the second table, whose values display in a DBWebGrid. Set the AutoPostBack property in the listbox to **True**, handle the RowFilter setting in **Page\_Load**, and call ClearSessionChanges after changing the RowFilter.

**Tip:** If you set the AutoRefresh property to **False**, which is the default, you might end up using cached data. Review the WorldTravel demo in \Demos\DBWeb to see an example of how this is handled.

### Master-Detail Relationships

You can make a DataView the master table in a master-detail relationship by adding a row filter. Set up a master-detail relationship with two or more DataTables within a single DataSet, then connect the DataView to the master DataTable. When the DBWebDataSource connects to the DataView, the DB Web Controls will let you select either the parent table, which is the DataView, or the detail table.

### ClearSessionChanges Method

The ClearSessionChanges method notifies the DBWebDataSource that the DataSet has changed and that existing row, column, and changed data information is no longer valid. All pending changes are removed. If you try to call this method from a DBWebNavigator button click event, the DBWebNavigator button will not work.

### DataView Limitations

There are some limitations with the DataView:

- Inserted rows in a DataView behave differently than inserted rows in a DataTable.
- A DataView does not allow multiple inserts of null records. This means that you must add data to an inserted row before adding a new inserted row.
- If an inserted row is deleted, that row is removed from the DataView and you cannot use Undo to recall it.
- If an inserted row contains a single non-null value, and that value is set to null, the row can be deleted in some cases and cannot be recalled.
- DBWeb controls do not provide full support for the DataViewSort property. If a sort field is encountered, the values for the fields contained in the Sort property cannot be changed, and the insert key will be disabled on the DBWebNavigator.

### See Also

Data Providers for .NET (see page 27)

Building an ASP.NET Application (see page 170)

## 1.6.8 Deploying ASP.NET Applications

This topic provides information about:

- Web Server Requirements
- Pre-Deploy Recommendations
- The RAD Studio ASP.NET Deployment Manager

For additional deployment information, see the [deploy.htm](#) file located, by default, at `C:\Program Files\CodeGear\RAD Studio\5.0`.

### Web Server Requirements

Before deploying your application to a web server, consider the following web server requirements:

- Internet Information Services (IIS) 6.0 must be installed and operational on the web server.
- The .NET Framework must be installed on the web server.
- ASP.NET must be enabled on the web server.
- The ASPNET account on the web server must be configured with the correct permissions.

For information on installing IIS, see the documentation that accompanies your Windows operating system. For information on performing the other tasks listed above, see the link to ASP.NET platform requirements at the end of this topic.

### Pre-Deploy Recommendations

Before you deploy your application, you should disable debugging and rebuild the application to make it smaller and more efficient:

- For a Delphi ASP.NET or C# application, update the application [web.config](#) file to disable debugging. For details, see the link to using the Deployment Manager at the end of this topic.
- For a C# application, choose **Project ► Options** and change the **Debug/Release** option set to the **Release** option set and recompile the application.

### The RAD Studio ASP.NET Deployment Manager

While you can use the XCOPY command-line tool to copy your entire project directory to a web server, only a subset of those files are actually required for deployment. For example, the [.aspx](#), [.config](#), and [.dll](#) files are required, but the Delphi-specific files such as the [.bdsproj](#), [.dcuil](#), and [.pas](#) files are not required.

RAD Studio includes the ASP.NET Deployment Manager to assist you in deploying ASP.NET applications. You can use it to deploy to a remote computer by using a share or an FTP connection, or to your local computer.

When you add a Deployment Manager to your project, an XML file ([.bdsdeploy](#)) is added to the project directory and a **Deploy** tab is added to the IDE. You provide destination and connection information on the **Deploy** tab and optionally modify the suggested list of files to copy, then the Deployment Manager copies the files to the deployment destination.

### See Also

[Deploying .NET Framework Applications \(MSDN\)](#)

[ASP.NET Platform Requirements \(MSDN\)](#)

[Deploying Applications Overview](#)

Using the ASP.NET Deployment Manager (see page 196)

## 1.6.9 Working with WebDataLink Interfaces

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

The characteristic that makes DB Web Controls different from traditional web controls is that the DB Web Controls automatically handle all data binding for you. Although you must still configure the links between data sources and controls at design time, all runtime binding is handled, without the need for you to add a data binding command in your code. When extending a DBWeb control using the **DBWeb Control Wizard**, you will implement several interfaces that provide the data binding capabilities. These interfaces are discussed in this topic.

- IDBWebDataLink
- IDBWebColumnLink: IDBWebDataLink
- IDBWebLookupColumnLink: IDBWebColumnLink

### IDBWebDataLink

All DB Web Controls implement this interface. The interface defines a data source and a data table, allowing you to connect to and access data from a variety of data sources, including databases, text files, arrays, and collections. If your control only needs to access data at the table level, you implement this interface.

### IDBWebColumnLink:IDBWebDataLink

This interface is implemented by DBWeb column controls, such as DBWebImage, DBWebTextBox, and DBWebCalendar, among others. The interface defines a column name to which a column control is linked. In combination with the IDBWebDataLink interface, this interface provides access to standard table and column data.

### IDBWebLookupColumnLink:IDBWebColumnLink

This interface is implemented by DBWeb lookup controls, such as DBWebListBox, DBWebRadioGroup, and DBWebDropDownList. The interface defines a TableName within a DataSet, a ColumnName representing a table that contains the data to be displayed in the lookup, and the column containing the values which, when a value is selected, are to be placed into the ColumnName field linked to the control. By default, the ColumnName field is the same as DataTextField. Lookup controls contain not only a text property, usually the item that is displayed in the control, such as a listbox, but also a value property. The value property might be identical to the text property, or it might contain a completely different piece of data, such as an identification number. For example, you might choose to display product names in a listbox or a drop down listbox, but set the values for each displayed item to their respective product IDs. When a user selects a product name, the product ID is passed to the application, rather than the name of the product itself. One benefit of this approach is to eliminate processing confusion between products with similar names.

### See Also

DBWeb Controls (see page 81)

Building an Application with DB Web Controls (see page 183)

Building an ASP.NET Application (see page 170)

## 1.7 Developing Web Services with ASP.NET

Web Services is a programmable entity that provides a particular element of functionality, such as application logic. Web Services is accessible to any number of potentially disparate systems through the use of Internet standards, such as XML and HTTP. Applications built with ASP.NET Web Services can be either stand-alone applications or subcomponents of a larger web application and can provide application components to any number of distributed systems using XML-based messaging. RAD Studio provides a number of methods that can help you build, deploy, and use applications with ASP.NET Web Services. For more general information about Web Services, refer to the Microsoft .NET SDK Documentation.

### Topics

Name	Description
ASP.NET Web Services Overview (see page 97)	Web Services is an Internet-based integration methodology that enables applications, independent of any platform or language, to connect and exchange information. Web Services is tightly integrated with the ASP.NET model used for the .NET Framework. Unlike traditional native Windows applications, ASP.NET Web Services applications contain objects and methods that are exposed over the Web using simple messaging protocol stacks. Any client can invoke a Web Services application over HTTP using a WebMethod. Like any method that can be accessed by way of a simple Windows Form application, a WebMethod provides some defined functionality. Unlike other types of methods, however, the... more (see page 97)
Web Services Protocol Stack (see page 100)	Understanding the Web Services infrastructure requires that you have some exposure to Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). Because the infrastructure already exists, as a developer of XML web services, you can leverage the existing technology by using standard Web protocols such as XML and HTTP. CodeGear provides an easy way to create, deploy, and use web services without concern for back-end processing so you can focus more on designing your services. This topic provides the conceptual background to understand how the protocol stack contributes... more (see page 100)
ASP.NET Web Services Support (see page 102)	ASP.NET Web Services support VCL.NET Forms and ASP.NET Web Forms. These forms can be used to create client applications that access Web Services applications. Use the <b>Add Web Reference</b> feature to add the desired ASP.NET Web Services application to the client application. Using the <b>UDDI Browser</b> you can locate Web Services applications you might want to use. RAD Studio provides simple tools to develop and deploy your ASP.NET Web Services applications. Additionally, RAD Studio helps you import WSDL documents that describe particular Web Services applications and expose their functionality to the client application. You can use the sample WebMethod provided... more (see page 102)

### 1.7.1 ASP.NET Web Services Overview

Web Services is an Internet-based integration methodology that enables applications, independent of any platform or language, to connect and exchange information. Web Services is tightly integrated with the ASP.NET model used for the .NET Framework. Unlike traditional native Windows applications, ASP.NET Web Services applications contain objects and methods that are exposed over the Web using simple messaging protocol stacks. Any client can invoke a Web Services application over HTTP using a WebMethod. Like any method that can be accessed by way of a simple Windows Form application, a WebMethod provides some defined functionality. Unlike other types of methods, however, the WebMethod is accessed by way of a web browser. For more general information about Web Services, refer to the Microsoft .NET Framework SDK Documentation.

CodeGear provides tools to develop and access ASP.NET Web Services using a variety of techniques. As modular objects, web services can be reused without additional coding.

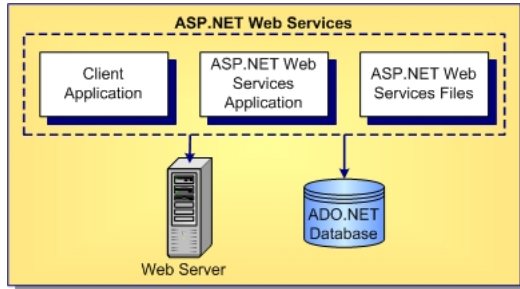


The following topics provide a brief introduction to the architecture of ASP.NET Web Services, the basic fundamentals of Web Services communication, and to the files created when you develop ASP.NET Web Services.

This topic introduces:

- ASP.NET Web Services Architecture
- Web Services Prerequisites
- Web Services Scenarios
- ASP.NET Web Services Files

### ASP.NET Web Services Architecture



The major components of the ASP.NET Web Services architecture include a client application, an ASP.NET Web Services application, several files such as code files in the development language, `.asmx` files, and compiled `.dll` files. You need a web server to house both ASP.NET Web Services application and the client. Optionally, you might include a database server for storage and access of ASP.NET Web Services data.

### Web Service Prerequisites

Before you begin developing a Web Services application, become familiar with the following concepts:

- **XML (Extensible Markup Language).** XML is a user-defined, human-readable structural description of data. Any data, dataset, or document that you intend to send to, or receive from, a web service is formatted in XML.
- **SOAP (Simple Object Access Protocol).** SOAP is the standard messaging protocol that is used for communication between web services and their clients. SOAP uses XML to format its messages, and contains the parameters or return values needed by servers and clients.
- **WSDL (Web Services Description Language).** WSDL is the language that describes a web service. A web service can be defined in any number of implementation languages. As a single-purpose utility, each web service must publish a description of its interface, which allows clients to interact with it. The WSDL document, at a minimum, describes the required parameters a client must provide and the result a client can expect to receive. The result description typically consists of the return data type.
- **UDDI (Universal Description, Discovery, and Integration).** UDDI is an industry initiative that provides a standard repository where businesses can publish web services for use by other companies. The UDDI repository contains links to, and descriptions of, a variety of web services. You can use the UDDI browser in the IDE to locate web services, download WSDL documents, and access additional information about web services and the companies that provide them.

### Web Service Scenarios

Current web services provide simple information sources that you can easily incorporate into applications, such as stock quotes, weather forecasts, and sports scores. As the demand for access to business logic over the web increases, companies are finding ways of providing their customers with a class of applications to analyze and aggregate information. For example, a financial institution might provide a web service to consolidate and continuously update customer financial information, such as stock portfolio, 401(k), bank account, and loan information for display in a spreadsheet, web site, or a personal digital assistant (PDA). This saves customer from having to manually collect and combine the information on their own. Although much of this information is available through the web today, a web service will simplify accessing and consolidating information and will ensure greater reliability.



You can use web services for solutions in the following areas:

- **Enterprise Application Integration (EAI).** A web service could allow multiple business partners to exchange inventory, order, or financial data, for example, without specifically knowing the precise data layout in which data is stored for each partner. For instance, many customer relationship management (CRM) or other front-end applications store customer data in a format that is not entirely compatible with the way a back-end enterprise resource planning (ERP) system stores its financial or inventory information. However, a sales organization may wish to use its CRM solution to process real-time orders with up-to-date inventory information from the ERP system. A web service could be a solution to managing the transformation of CRM requests to ERP storage and from ERP responses to CRM confirmations.
- **Business-to-business (B2B) integration.** Similar to the EAI solution, a B2B solution could take advantage of a Web Services capability to provide cached data for large orders. B2B transactions, unlike business-to-consumer (B2C) transactions, often consist of high-volume transactions that would be prohibitive to execute at the level of a B2C transaction. For instance, a consumer might order one box of pencils from an online stationery store, but a business might order a thousand boxes monthly, with multiple shipping addresses. The scale and complexity of a B2B transaction requires the intervention of a web service to help simplify and process the transaction quickly and with consistency.
- **Business-to-consumer integration.** B2C web services typically manage web-based transactions. For example, a web service that allows you to look up postal codes eliminates the need for businesses to create a new program every time the service is included on a web site. Some commerce sites might use web services to help manage currency conversion when taking international sales orders.
- **Mobile (Smart client applications).** Because the small footprint of a mobile client requires that memory usage be reserved for only the most important system functions, and because mobile clients are, by definition, linked to the Internet by way of their wireless communication protocols, Web services play a vital role in providing lightweight but powerful applications to mobile devices. Web services allow mobile device users to perform a variety of tasks which require little more than data input at the device and data display of the results. All processing can occur on a remote web service, thus decreasing bandwidth requirements on the mobile device itself.
- **Distributed and Peer-to-Peer.** For certain types of distributed and peer-to-peer applications, web services play an important role. If you use distributed computing over an uncontrolled network (such as the Internet) rather than over a LAN or corporate network, you might use web services. Web services do not require state maintenance, thus offering potentially improved performance, particularly where a request-response behavior is not absolutely required. For applications that require strict request-response behavior and high security, you should consider using an older, more controlled model, such as COM or .NET remoting.

### ASP.NET Web Services Files

Certain files are automatically generated when you create applications with ASP.NET Web Services. These files enable the ASP.NET Web Services to render their services through a web server. The following table lists the files and their descriptions.

File	Description
<a href="#">.asmx</a>	When you create an ASP.NET Web Services application, a text file is automatically generated with the <a href="#">.asmx</a> extension. The required Web Services directive is placed at the top of this file to correlate between the URL address of the web service and its implementation. Within the <a href="#">.asmx</a> file, you add Web Services logic to the methods visible by the client application. The <a href="#">.asmx</a> file acts as the base URL for clients calling the XML web service. This file is compiled into an assembly, along with other files, for deployment.
code-behind	When you create an ASP.NET Web Service application, a code-behind file is generated with a language-specific extension. You add your Web Services logic to the public method to process Web Services requests and responses.
compiled DLL files	Web Services DLL files provide dynamic services on the web server.
<a href="#">.wsdl</a>	This file is generated when you click the <b>Add Web Reference</b> feature to add the web service to your client application. It describes the Web Services interface available to the client.
<a href="#">.map</a>	This file enables the discovery of a web service that is exposed on a given server. It also contains links to other resources that describe the web service.

### See Also

Web Services Protocol Stack (see page 100)

ASP.NET Web Services Support (📖 see page 102)

Building an ASP.NET "Hello World" Web Services Application (📖 see page 206)

Accessing an ASP.NET "Hello World" Web Services Application (📖 see page 202)

[Microsoft Overview of Web Services](#)

## 1.7.2 Web Services Protocol Stack

Understanding the Web Services infrastructure requires that you have some exposure to Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). Because the infrastructure already exists, as a developer of XML web services, you can leverage the existing technology by using standard Web protocols such as XML and HTTP.

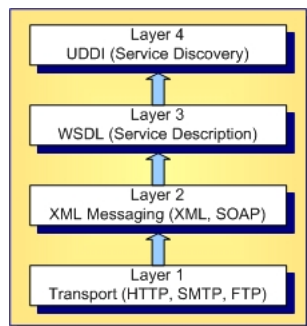
CodeGear provides an easy way to create, deploy, and use web services without concern for back-end processing so you can focus more on designing your services.

This topic provides the conceptual background to understand how the protocol stack contributes to Web Services functionality:

- How web services access and expose their services via the Web
- How XML passes information through standard SOAP and HTTP
- How a client can identify a web service offering
- How web services are discovered and accessed

### Layers of the Web Services Protocol Stack

Web services consist of sets of internet protocols and standards for exchanging data between applications. The Web Services Protocol Stack describes the layering of the set of internet protocols or rules used to design, discover, and implement web services.



The major components or layers of a Web Service Protocol Stack include:

- **Transport Layer**—transports messages between applications
- **XML Messaging Layer**—encodes messages in XML that can be understood by both client and server
- **WSDL Layer**—describes the service provided
- **UDDI Layer**—centralizes services with a common registry

### Transport Layer

The Transport layer is the first component in the stack and is responsible for moving XML messages between applications. The Transport protocol most commonly used is the standard HTTP protocol. Other commonly used Web protocols are SMTP and FTP.

## XML Messaging

The messaging layer in the protocol stack is based on an XML model. XML is widely used in Web Services applications and is the foundation for all web services. XML is just one of the standards enabling web services to map between technology domains. You will find many resources on the Web that describe XML messaging. For more information, refer to the World Wide Web Consortium (W3C) site on Messaging listed in the link list below.

The XML Messaging specification is a broadly-defined umbrella under which a number of more specific protocols are defined. SOAP is one of the more popular standards, and is one of the most significant standards in communicating web services over the network. XML provides a means for communicating over the Web using an XML document that both requests and responds to information between two disparate systems. SOAP allows the sender and the receiver of XML documents to support a common data transfer protocol for effective networked communication. You will find many resources on the Web that describe SOAP. For more information, refer to the W3C site for SOAP listed in the link list below.

## WSDL Layer

This layer represents a way of specifying a public interface for a web service. It contains information on available functions, on data types for XML messaging, binding information about the transport protocol, and the location of the specific web service.

Any client application that wants to know about a service, what data it expects to receive, whether or not it delivers any results, and the supported transport, uses WSDL to find that information. When you create a Web Service, it must be described and advertised to its potential customers before it can be used. WSDL provides a common format for describing and publishing that web service information. Typically, WSDL is used with SOAP, and the WSDL specification includes a SOAP binding.

Use CodeGear's **Add Web Reference** feature to obtain a WSDL document for your web service. The WSDL document, or proxy file, is copied to the client and is used to call the server. This proxy file is named `References.*`, where the file name extension reflects the language type. For more information about WSDL, refer to the W3C WSDL site listed in the link list below.

## UDDI Layer

This layer represents a way to publish and find web services over the Web. You can think of this layer as the White and Yellow Pages of your phonebook. The White pages of web services provides general information about a specific company, for instance, their business name, description, and address. The Yellow Pages includes the classification of data for the services offered, for instance, industry type and products.

The protocol you use to publish your web services is known as UDDI. The UDDI Business Registry allows anyone to search existing UDDI data and enables you to register your company and its services. With RAD Studio, your data automatically gets published to the registry, or a distributed directory for business and web services.

## See Also

ASP.NET Web Services Overview (🔗 see page 97)

ASP.NET Web Services Support (🔗 see page 102)

Building an ASP.NET "Hello World" Web Services Application (🔗 see page 206)

Accessing an ASP.NET "Hello World" Web Services Application (🔗 see page 202)

[SOAP](#)

[XML Messaging](#)

[WSDL](#)

## 1.7.3 ASP.NET Web Services Support

ASP.NET Web Services support VCL.NET Forms and ASP.NET Web Forms. These forms can be used to create client applications that access Web Services applications. Use the **Add Web Reference** feature to add the desired ASP.NET Web Services application to the client application. Using the **UDDI Browser** you can locate Web Services applications you might want to use.

RAD Studio provides simple tools to develop and deploy your ASP.NET Web Services applications. Additionally, RAD Studio helps you import WSDL documents that describe particular Web Services applications and expose their functionality to the client application. You can use the sample WebMethod provided by RAD Studio, which lets you create and access an ASP.NET Web Services application.

This topic includes:

- ASP.NET Web Services Client Support
- ASP.NET Web Services Server Support
- ASP.NET Web Services Namespaces

### ASP.NET Web Services Client Support

You can create a Web Services application that is simply a provider, or a server application. This application resides on a web server and can be accessed by any client that understands the application architecture. If you want to consume a Web Services application yourself, you need to create a client application. RAD Studio provides different tools you can use to build client applications:

- Web Forms
- Web References

### Virtues of ASP.NET Web Forms

If you need to provide a thin-client application that performs simple data manipulation or satisfies a single-purpose requirement, consider using ASP.NET Web Forms. Web Forms are platform-independent interfaces that display in a web browser and invoke Web Services applications over a simple protocol like HTTP.

You can also create an ASP.NET Web Services application as a console application which can be accessed through either a console window, or by another Web Services application, even one without a client.

### Add Web Reference

You can add a Web Reference to your client application to access web services. A Web Reference refers to either a WSDL document or an XML schema, which is imported into your client application. The WSDL document or XML schema describes a web service. When you import one of these documents, RAD Studio generates the interfaces and class definitions needed for calling that web service. Right-click the **WebService** node in the **Project Manager** and select **Add Web Reference**. A **UDDI Browser** appears. To add the web service to your client application, you must navigate within the browser and locate the WSDL document for the web service.

### ASP.NET Web Services Server Support

The ASP.NET Web Services application you build in RAD Studio, provides programmatic access to the application logic of one or more web services. You define the services you want to expose, how the services are to be used, and the infrastructure that receives and processes requests and responses.

When you create a new ASP.NET Web Service application, the **New ASP.NET Application** dialog box lets you specify the

name and location of the ASP.NET Web Services application, and automatically creates the files required for deployment. When you specify the application settings, RAD Studio generates the `.asmx` file that acts as a base URL for clients calling the ASP.NET Web Services application.

**ASP.NET Web Services Namespaces**

For more information on System.Web.Services namespaces, refer to the Microsoft .NET Framework SDK.

**See Also**

ASP.NET Web Services Overview (📖 see page 97)

Web Service Protocol Stack (📖 see page 100)

Building an ASP.NET "Hello World" Web Services Application (📖 see page 206)

Accessing an ASP.NET "Hello World" Web Service Application (📖 see page 202)



# 2 Procedures

This section provides how-to information for various areas of RAD Studio development.

Topics

Name	Description
Database Procedures (📄 see page 106)	This section provides how-to information on developing database applications.
Interoperable Applications Procedures (📄 see page 139)	This section provides how-to information on building interoperable applications.
Modeling Procedures (📄 see page 141)	This section provides how-to information for modeling applications.
VCL for .NET Procedures (📄 see page 145)	This section provides how-to information on developing VCL for .NET applications.
ASP.NET Procedures (📄 see page 167)	This section provides how-to information on developing ASP.NET Web Forms applications.
Web Services Procedures (📄 see page 202)	This section provides how-to information on developing and using web services.

## 2.1 Database Procedures

This section provides how-to information on developing database applications.

### Topics

Name	Description
Adding a New Connection to the Data Explorer ( <a href="#">see page 107</a> )	You can add new connections to the <b>Data Explorer</b> , which persist as long as the connection object exists.
Adding a BDP Reconcile Error dialog to your BDP Application ( <a href="#">see page 108</a> )	You can modify your BDP applications to call the BDP Reconcile Error dialog to handle an update exception (as occurs sometimes when two people are trying to simultaneously update the same row of a database table).
Browsing a Database in the Data Explorer ( <a href="#">see page 109</a> )	Once you have a live connection, you can use the <b>Data Explorer</b> to browse database objects.
Connecting to the AdoDbx Client ( <a href="#">see page 110</a> )	You can establish a database connection using AdoDbx Client in several ways.
Creating Database Projects from the Data Explorer ( <a href="#">see page 111</a> )	You can drag and drop data from the <b>Data Explorer</b> to any forms such as Windows Forms or Web Forms, and Global.asax files, to populate datasets and quickly build a database project. This allows you to automatically hook up database components to your project and eliminates the need to provide a connection string, which can be prone to errors if entered manually.
Creating Table Mappings ( <a href="#">see page 112</a> )	Using the TableMappings property, you can map columns between a data source and an in-memory dataset. This allows you to use different, often more descriptive names for your dataset columns. You can also map a column in a database table to a column in the dataset different from that which is selected by default. The TableMappings property also allows you to create a dataset that contains fewer or more columns than those retrieved from the database schema.
Executing SQL in the Data Explorer ( <a href="#">see page 113</a> )	You can write, edit, and execute SQL in an <b>SQL Window</b> , which is available from within the <b>Data Explorer</b> .
Handling Errors in Table Mapping ( <a href="#">see page 114</a> )	Whenever you perform any type of comparison function between a data source and an in-memory data representation, there is potential for error. Errors can occur when a data source and its corresponding dataset do not share uniform numbers of columns, or when column types in a data source do not correspond to the column types in the dataset. In addition, other, internal errors can occur for which there is no design-time workaround. You can use both the MissingMappingAction property and the MissingSchemaAction property to respond to errors in your table mapping operations. Use the MissingMappingAction when you want to specify... more ( <a href="#">see page 114</a> )
Migrating Data Between Databases ( <a href="#">see page 115</a> )	The DataExplorer makes it easy to migrate data from one database to another, and even between providers. The DataExplorer lets you quickly copy a table from one database and paste it into another database. Both the structure and the data for the table or tables is migrated.  Data migration is supported by the BdpCopyTable class, which is available as a design-time component from the <b>Tool Palette</b> . You can use this component to programmatically migrate data.  <b>Note:</b> The BdpCopyTable class does not copy foreign keys or dependent objects.
Modifying Connections in the Data Explorer ( <a href="#">see page 116</a> )	You can modify connections in a variety of ways from the <b>Data Explorer</b> .
Modifying Database Connections ( <a href="#">see page 117</a> )	The basic elements of a connection string tend to be the same from one database type to another. However, each database type supports slightly different connection string syntax. This topic addresses those differences.
Building a Database Application that Resolves to Multiple Tables ( <a href="#">see page 122</a> )	RAD Studio supports multi-table resolution with BDP.NET. Specifically, the DataSync and DataHub components are designed to provide and resolve a .NET DataSet from multiple heterogeneous data sources. In addition, these components support the display of live data at design-time, and provide and resolve master-detail data by generating optimal SQL for resolving to BDP data sources.  The DataHub acts as a conduit between a DataSet and a DataSync. The DataPort property for a DataHub can be set to any IDataProvider implementation. DataSync implements IDataProvider and has a Providers collection that can contain any .NET data provider that implements IDbDataAdapter. The GetData... more ( <a href="#">see page 122</a> )



Passing Parameters in a Database Application (see page 124)	The following procedures describe a simple application that allows you to pass a parameter value at runtime to a DataSet. Parameters allow you to create applications at design time without knowing specifically what data the user will enter at runtime. This example process assumes that you already have your sample Interbase Employee database set up and connected. For purposes of illustration, this example uses the default connector IBConn1, which is set to a standard location. Your database location may differ.
Using the Data Adapter Preview (see page 126)	CodeGear RAD Studio provides a tool that enables communication between a data source and a dataset. You can use the <b>Data Adapter Preview</b> to specify what data to move into and out of the dataset either in the form of SQL statements or stored procedures that are invoked to read or write a database.
Using the Command Text Editor (see page 127)	In order to create a DataSet, your BdpDataAdapter needs to have at least a SQL Select statement defined for the CommandText property. This statement, once built, appears as the CommandText of the BdpCommand object for the BdpDataAdapter. You can enter this Select statement manually, or you can use the <b>Command Text Editor</b> to construct the statement, along with Update, Insert, and Delete statements, using a simple point-and-click mechanism. Using this method, once you have a connection to a live data source, you will be able to see the names of tables and columns in the <b>Command Text Editor</b> . You... more (see page 127)
Using the Data Adapter Designer (see page 128)	The Data Adapter contains, at a minimum, a SQL Select statement of the SELECT command property. You can enter this statement yourself, or using the <b>Data Adapter</b> designer you can construct the Select, along with the Update, Insert, and Delete statements. The BdpCommandBuilder constructs the Update, Insert, and Delete statements based on the tables and columns you have selected. The <b>Data Adapter</b> designer uses a live connection to retrieve metadata from which you can build the appropriate SQL statements for manipulating the data you want to move from a DataSet back into your database.
Using the Connection Editor Designer (see page 128)	Each connection object can support multiple named connections. These connections can represent connections to multiple databases and database types.
Using Standard DataSets (see page 129)	The standard DataSet provides an in-memory representation of one or more tables or views retrieved from a connected data source. Because of the level of indirection used in coding the underlying data structure, you are only able to see the column names from your data source at runtime. When you generate a DataSet, it retrieves everything you specified in your Select statement in the Data Adapter Configuration dialog. You can limit your columns by changing the Select statement and creating a new DataSet.
Using Typed DataSets (see page 132)	Typed DataSets provide certain advantages over standard DataSets. For one thing, they are derived from an XML hierarchy of the target database table. The XML file containing the DataSet description allows the system to provide extensive code-completion capabilities not available when using standard DataSets. Strong typing of DataSet methods, properties, and events allows compile-time type checking, and can provide a performance improvement in some applications.
Connecting to a Database using the dbExpress Driver Framework (see page 134)	This procedure tells how to use the dbExpress driver framework to connect to a database and read its records. In the sample code, the dbExpress ini files contain all the information about the particular database connection, such as driver, user name, password, and so on.
Building a Distributed Database Application (see page 136)	Data remoting is fundamental to developing distributed database applications. The .NET remoting technology provides a flexible and extensible framework for interprocess communication. With .NET remoting you can interact with objects in different application domains, in different processes running on the same machine, or in different machines on a network.  Using the RemoteServer and RemoteConnection components, you can easily migrate a client/server application that uses DataHub and DataSync components to a multi-tier DataSet remoting application. RemoteServer implements IDataService and publishes itself as a singleton server activated object (SAO). On the client side, the RemoteConnection properties form the URL for connecting to... more (see page 136)

## 2.1.1 Adding a New Connection to the Data Explorer

You can add new connections to the **Data Explorer**, which persist as long as the connection object exists.

**To add a new connection**

1. Choose **View ► Data Explorer**. This displays the **Data Explorer**.
2. Select a provider from the tree list.
3. Right-click to display a pop-up menu.
4. Choose **Add New Connection**. This displays the **Add New Connection** dialog.
5. Enter the name of the new connection.
6. Click **OK**.

**Tip:** If you need to modify your new connection settings, right-click on your new connection and scroll down to modify a connection

. A **Connection Editor** dialog appears. Enter your connection settings and click **OK**.

**See Also**

ADO.NET Component Designers (see page 21)

Browsing a Database (see page 109)

Executing SQL in the Data Explorer (see page 113)

Modifying Connections (see page 116)

## 2.1.2 Adding a BDP Reconcile Error dialog to your BDP Application

You can modify your BDP applications to call the BDP Reconcile Error dialog to handle an update exception (as occurs sometimes when two people are trying to simultaneously update the same row of a database table).

**To add a BDP Reconcile Error dialog:**

1. Add a BDPDataAdapter component to your existing WinForm.
2. Choose the Events tab on the Object Inspector window
3. Double-click in the content section of the blank pull-down list next to the OnUpdateError event. This will populate the first level of the pull-down list. It will also create the code for the BdpDataAdapter method definition and implementation.
4. Add the lines that are in bold below to the method implementation to handle the event (the following example is using the C# language):

```
private void bdpDataAdapter1_OnUpdateError(object sender,
Borland.Data.Common.BdpUpdateEventArgs e)
{
    Borland.Data.Common.ReconcileErrorForm f = new
Borland.Data.Provider.ReconcileErrorForm( e );
    f.ShowDialog();
}
```

5. Save the changes to your WinForm.

The BDP Reconcile Error dialog will now appear whenever one user is trying to modify data in the same row of a database that another user is working on. The dialog works as follows. As each row in a table is updated

Your new Error Reconcile Form will display four columns in the upper portion of the window, and six radio buttons in the bottom portion of the window. The following table describes each of the columns.

Column Label	Meaning
Column Name	The names of the columns of the table in which an error has occurred.
Current Row	The contents of the row that is currently in contention.
Original Row	What the row contained before the contentious data was entered.
Server Row	The last update that was saved to the Server. (This represents what the row contains on the server.)

The three radio buttons on the lower left portion of the window allow you to indicate how to continue processing after handling the error. You can only choose one option from the following three choices.

Radio Button Label	Meaning
Retry update using primary key	The error will be cleared, and then the update will be attempted again with the primary key. If the data row from the server cannot be found, this option will be disabled.
Skip current row and continue	Choose this option when you have decided not to attempt to update changes for the current row, but you want to try to update the rest of the rows.
Abort updates	The latest updates will not be applied, and error will be cleared, but no more updates will be attempted.

The three radio buttons in the lower right portion of the window allow you to indicate which data to write to the database. You can only choose one option from the following three choices.

Radio Button Label	Meaning
Use original values	Place the data from the Original Row column (described previously) into the row where the contention occurred.
Use server values	Place the data from the Server Row column, (described previously) into the row where the contention occurred.
Use current values	Place the data from the Current Row column, (described previously) into the row where the contention occurred.

#### See Also

ADO.NET Component Designers (🔗 see page 21)

Browsing a Database (🔗 see page 109)

Executing SQL in the Data Explorer (🔗 see page 113)

Modifying Connections (🔗 see page 116)

## 2.1.3 Browsing a Database in the Data Explorer

Once you have a live connection, you can use the **Data Explorer** to browse database objects.

#### To browse database objects

1. Choose **View ► Data Explorer**.
2. Expand a provider node to expose the list of available connections.
3. Expand a connection node to view the list of database objects (tables, views, and procedures).

**Note:** If you receive an error because your connection is not live, you should refresh your provider, and/or modify your connection.

**To retrieve data from the database**

1. Expand a connection in the **Data Explorer**.
2. Double-click a table name or view name to retrieve data. This operation returns a result set into a tabbed **Data Explorer** page in the **Code Editor**.

**Tip:** You can also select a table in the Data Explorer and right-click to display a pop-up menu with a **Retrieve Data From Table** command.

**To run a stored procedure**

1. Choose **View ► Data Explorer**.
2. Expand a connection in the **Data Explorer** and locate a stored procedure.
3. Double-click the stored procedure to view its parameters. The parameters open in a separate page on the design surface.
4. Edit input parameters as necessary.
5. Click the Execute button in the top left corner of the page to execute the procedure. The result set appears in a datagrid.

**Tip:** You can also select a procedure in the Data Explorer and right-click to display a pop-up menu with an **Execute** command.

**See Also**

ADO.NET Component Designers (🔗 see page 21)

Adding a New Connection (🔗 see page 107)

Executing SQL in the Data Explorer (🔗 see page 113)

Modifying Connections (🔗 see page 116)

## 2.1.4 Connecting to the AdoDbx Client

You can establish a database connection using AdoDbx Client in several ways.

**To make a connection using dbxconnections.ini file**

1. The `ConnectionName` property referenced in the code sample is the name of a connection in the `dbxconnections.ini` file.
2. Use the following Delphi code to connect:

```
uses System.Data.Common
...
var
    Factory: System.Data.Common.DbProviderFactory;
    Connection: System.Data.Common.DbConnection;
begin
    Factory := System.Data.Common.DbProviderFactories.GetFactory('Borland.Data.AdoDbxClient');
    Connection := Factory.CreateConnection();
    Connection.ConnectionString := 'ConnectionName=IBConnection';
    Connection.Open;
end
```

**To make a connection using a System.Configuration file**

1. For this to work, the property settings in `dbxconnections.ini` and `dbxdriver.ini` for the database you are connecting to must be migrated to the `machine.config` file. Here is an example of connection string text to add to the `<connectionStrings>` section of

machine.config:

```
<add name="IBConnection"
  connectionString="ConnectionName=IBCONNECTION;
  drivervname=Interbase;
  database=workerbee:C:\Borland\Interbase\examples\database\employee.gdb;
  rolename=RoleName;
  user_name=user;
password=password;
  sqldialect=3;localecode=0000;blobsize=-1;
  commitretain=False;waitonlocks=True;interbase trans isolation=ReadCommitted;
  trim char=False" providerName="Borland.Data.AdoDbxClient"/>
```

2. Use the following Delphi code to connect:

```
var
  Factory: System.Data.Common.DbProviderFactory;
  Connection: System.Data.Common.DbConnection;
  Config: System.Configuration.Configuration;
  ConnectSection: System.Configuration.ConnectionStringsSection;
  CurrentSettings: System.Configuration.ConnectionStringSettings;
begin
  Factory:= System.Data.Common.DbProviderFactories.GetFactory('Borland.Data.AdoDbxClient');
  Connection:= Factory.CreateConnection();
  Config:=
System.Configuration.ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
  ConnectSection:= Config.ConnectionStrings;
  CurrentSettings:= ConnectSection.ConnectionStrings['IBConnection'];
  Connection.ConnectionString:= CurrentSettings.ConnectionString;
  Connection.Open;
end;
```

#### See Also

AdoDbx Client Overview (see page 5)

[Deploying the AdoDbx Client](#)

## 2.1.5 Creating Database Projects from the Data Explorer

You can drag and drop data from the **Data Explorer** to any forms such as Windows Forms or Web Forms, and Global.asax files, to populate datasets and quickly build a database project. This allows you to automatically hook up database components to your project and eliminates the need to provide a connection string, which can be prone to errors if entered manually.

#### To create a database project from the Data Explorer

1. Make sure you have a live connection to a database.
2. From the View menu, select **Data Explorer**.
3. Choose **File ► New ► Other** and select a Delphi for .NET project. Typically, this will be either a Windows Form, a VCL Form, or an ASP.NET application.
4. Expand the **Data Explorer Tree** by drilling down to the **Table** or **View** level. If the connection to your database is live, the small red x will disappear when you expand the connection node for the database. If it's not live, you may need to modify the connection string.
5. Using the cursor, grab one of the tables named in the list.
6. Drag and drop the table object onto your form. An AdoDbxConnection and an AdoDbxDataAdapter appear in the component tray.
7. Specify the appropriate database properties for each database component. For instance, set the Active property to *True* if you want to be able to view data in your component at design time.

**Note:** A DataGrid will not appear automatically so make sure you drop a DataGrid component onto your form to appropriately display data, when necessary.

#### See Also

ADO.NET Component Designers (🔗 see page 21)

Browsing a Database (🔗 see page 109)

Executing SQL in the Data Explorer (🔗 see page 113)

Modifying Connections (🔗 see page 116)

---

## 2.1.6 Creating Table Mappings

Using the TableMappings property, you can map columns between a data source and an in-memory dataset. This allows you to use different, often more descriptive names for your dataset columns. You can also map a column in a database table to a column in the dataset different from that which is selected by default. The TableMappings property also allows you to create a dataset that contains fewer or more columns than those retrieved from the database schema.

#### To create a table mapping

1. Create an application.
2. Add and configure database components.
3. Set the table mappings in the TableMappings dialog.

**Note:** This procedure assumes you are using BDP.NET database components.

#### To create an application

1. Choose **File ► New ► Windows Forms Application** for either Delphi for .NET or C#.
2. Click the Data Explorer tab to display your data sources.
3. Expand the list and locate a live data source.
4. Drag-and-drop a table name onto your Windows Form to add a data source to your application. You should see two objects in the **Component Tray**: a BdpDataAdapter and a BdpConnection.

For more information about how to create database applications, refer to the additional ADO.NET and database topics in this Help system.

#### To configure the database components

1. Select the BdpDataAdapter icon in the **Component Tray**.
2. Click the **Configure Data Adapter** designer verb to open the Data Adapter Configuration dialog.
3. Select the **DataSet** tab.
4. Click the **New DataSet** radio button.
5. Click **OK**. This creates a new dataset and displays an icon for it in the **Component Tray**.

#### To set table mappings

1. Select the BdpDataAdapter icon in the **Component Tray**.
2. Double-click the **Collections** field for the **TableMappings** property in the **Object Inspector**. This displays the **TableMappings** dialog.

3. If you want to use an existing dataset as a model for the columns, check the **Use a dataset to suggest table and column names** check box. This provides you with a list of column names from an existing dataset based on the schema of that dataset. The column names are not linked to anything when you use this process.
4. If you checked the **Use a dataset to suggest table and column names** check box, you can choose the dataset from the DataSet drop down list.
5. Select the source table from the **Source table** drop down list. If there is more than one table in the data source, their names appear in the drop down list.
6. If you chose to use a dataset to suggest table and column names, and that dataset contains more than one table, you can select the table you want to use from the **Dataset table** drop down list. The column names from the source table and from the dataset should appear in the **Column mappings** grid. As they are displayed by default, they represent the mapping from source to dataset; in other words, the data adapter reads data from each column named on the left side of the grid and stores the data in the dataset column named in the corresponding field on the right side of the grid. You can change the names on either side by typing new names or by selecting different tables. This allows you to store queried data into different dataset columns than the ones created in the dataset by default.
7. If you want to modify a mapping, type a new name in the Dataset table column next to the target Source table column. This results in the data from the Source table column being stored in the new dataset column.

**Note:** If you want to reset the column names so that the dataset columns match the data source columns, you can click the Reset button.

#### To delete a mapping

1. Select the grid row that you want to delete.
2. Click **Delete**. This will cause the query to ignore that column in the source table and to not fill the dataset column with any data.

#### See Also

ADO.NET Overview (see page 14)

ADO.NET Component Designers (see page 21)

Handling Errors in Table Mappings (see page 114)

---

## 2.1.7 Executing SQL in the Data Explorer

You can write, edit, and execute SQL in an **SQL Window**, which is available from within the **Data Explorer**.

#### To open a SQL Window

1. Choose **View ► Data Explorer**.
2. Select a connection.
3. Right-click the connection and choose **SQL Window**. This opens a tabbed **SQL Window** in the **Code Editor**.

#### To execute SQL

1. Enter a valid SQL statement or stored procedure name in the multi-line text box at the top of the **SQL Window**.
2. Click **Execute SQL**. If the SQL statement or stored procedure is valid, the result set appears in the bottom pane of the **SQL Window**.

**Note:** The SQL statement or stored procedure must operate against the current connection and its target database. You cannot execute SQL against a database to which you are not connected.

3. Click **Clear All SQL** to clear the SQL statement or stored procedure from the multi-line text box.

#### See Also

ADO.NET Component Designers (🔗 see page 21)

Browsing a Database (🔗 see page 109)

Adding a New Connection (🔗 see page 107)

Modifying Connections (🔗 see page 116)

---

## 2.1.8 Handling Errors in Table Mapping

Whenever you perform any type of comparison function between a data source and an in-memory data representation, there is potential for error. Errors can occur when a data source and its corresponding dataset do not share uniform numbers of columns, or when column types in a data source do not correspond to the column types in the dataset. In addition, other, internal errors can occur for which there is no designtime workaround. You can use both the `MissingMappingAction` property and the `MissingSchemaAction` property to respond to errors in your table mapping operations. Use the `MissingMappingAction` when you want to specify how the adapter should respond when the mapping is missing. Use the `MissingSchemaAction` when you want to specify how the adapter should respond when it tries to write data to a column that isn't defined in the dataset.

#### To set the `MissingMappingAction` property

1. Once you have created an `AdoDbxDataAdapter` and have set up your table mappings, click the drop down list next to the `MissingMappingAction` property in the Object Inspector.
2. Select *Passthrough* if you want the adapter to load the data source column data into a dataset column of the same name, or, if there is no corresponding dataset column, if you want the adapter to perform the action specified in the `MissingSchemaAction` property.
3. Select *Ignore* if you want to keep data from being loaded when data source columns are not properly mapped to dataset columns. This could occur if mapped columns are of incompatible data types, lengths, or have other errors.
4. Select *Error* if you want the adapter to raise an error that you can trap.

#### To set the `MissingSchemaAction` property

1. Select *Add* if you want the data source table or column added to the dataset and its schema. Setting the `MissingMappingAction` property to *Passthrough* and the `MissingSchemaAction` to *Add* results in a duplication of data source table and column names in the dataset.
2. Select *AddWithKey* if you want the data source table or column added to the dataset and its schema along with the table's or column's primary key information.
3. Select *Ignore* if you don't want a table or column added to the dataset, when that table or column aren't already represented in the dataset schema. Specify *Ignore* when you want the dataset loaded only with data explicitly specified in the table mappings. This may be necessary if your adapter calls a stored procedure or a user-defined SQL statement that returns more columns than are defined in the dataset.
4. Select *Error* if you want the adapter to raise an error that you can trap.

#### See Also

ADO.NET Overview (🔗 see page 14)

ADO.NET Component Designers (🔗 see page 21)

Creating Table Mappings (🔗 see page 112)



## 2.1.9 Migrating Data Between Databases

The DataExplorer makes it easy to migrate data from one database to another, and even between providers. The DataExplorer lets you quickly copy a table from one database and paste it into another database. Both the structure and the data for the table or tables is migrated.

Data migration is supported by the BdpCopyTable class, which is available as a designtime component from the **Tool Palette**. You can use this component to programmatically migrate data.

**Note:** The BdpCopyTable class does not copy foreign keys or dependent objects.

### To migrate multiple tables

1. Choose **View ► Data Explorer**.
2. Right-click a provider type, such as Interbase, and choose **Migrate Data**. The **Data Explorer** page for data migration opens in the **Code Editor**. This data migration page lets you select one or more tables from a source provider connection and a destination connection to which the tables will be migrated.
3. Choose a connection from the Source Connection drop-down list box. The tables associated with this connection appear in the list box beneath the connection.
4. Choose a connection from the Destination Connection drop-down list box. The tables associated with this connection appear in the list box beneath the connection.
5. Select one or more tables to migrate from the list of tables associated with the source connection. To select consecutive tables, click the first table, press and hold down the **SHIFT** key, and then click the last table. To select nonconsecutive tables, press and hold down **CTRL**, and then click each table.
6. Click the Include (>) button to include these tables for migration to the destination connection. The selected tables appear in the list of tables for the destination connection. If one of the selected tables has the same name as a table in the destination connection, it cannot be migrated.
7. Click **Migrate** to copy the tables to the destination connection. The Data Migration page shows the progress as SQL types are mapped, tables are created, data is retrieved from the source connection, and data is populated in the new table in the destination connection. The result of each operation is reported for each table.
8. Right-click the Tables node in the destination provider and choose **Refresh**. Nodes for any new tables appear.
9. Double-click a new table node to confirm its structure and contents. The table opens in a page on the design surface.

### To migrate a single table

1. Choose **View ► Data Explorer**.
2. Expand the Tables node in the source provider, and select the database table containing the data and structure you want to migrate. You must have a valid connection to expand the provider nodes.
3. Right-click the table you want to migrate and choose **Copy Table**.
4. Expand the Tables node of the provider into which you want to migrate the data.
5. Right-click any table and choose **Paste Table**. The New Table Name dialog box appears.
6. Enter a name for the new table and click **OK**.
7. Right-click the Tables node in the destination provider and choose **Refresh**. A node for the new table appears.
8. Double-click the new table node to confirm its structure and contents. The table opens in a page on the design surface.

### See Also

Data Providers for Microsoft .NET (🔗 see page 27)

Modifying Connections in the Data Explorer (🔗 see page 116)

Browsing a Database in the Data Explorer (see page 109)

---

## 2.1.10 Modifying Connections in the Data Explorer

You can modify connections in a variety of ways from the **Data Explorer**.

### To modify connections

1. Choose **View ► Data Explorer**.
2. Select a provider.
3. Right-click to display a pop-up menu to view your options.

### To refresh a connection

1. Choose **View ► Data Explorer**.
2. Select a provider.
3. Right-click to display a pop-up menu.
4. Choose **Refresh**. This operation reinitializes all connections defined for the selected provider.

### To delete a connection

1. Choose **View ► Data Explorer**.
2. Select a connection.
3. Right-click to display a pop-up menu.
4. Choose **Delete Connection**. This displays a confirmation message that asks if you want to delete the connection.
5. Click **OK**.

### To modify a connection

1. Choose **View ► Data Explorer**.
2. Select a connection.
3. Right-click to display a pop-up menu.
4. Choose **Modify Connection**. This displays the **Connections Editor** dialog.
5. Make changes to the appropriate values in the editor.
6. Click **OK**.

### To close a connection

1. Choose **View ► Data Explorer**.
2. Select a connection.
3. Right-click to display a pop-up menu.
4. Choose **Close Connection**. If the connection is open, this operation closes it.

**Note:** If the Close Connection command is disabled in the menu, the connection is not open.

### To rename a connection

1. Choose **View ► Data Explorer**.

2. Select a connection.
3. Right-click to display a pop-up menu.
4. Choose **Rename Connection**. This displays **Rename Connection** dialog.
5. Enter a new name.
6. Click **OK**. The **Data Explorer** displays the connection with its new name.

**See Also**

ADO.NET Component Designers (see page 21)

Browsing a Database (see page 109)

Executing SQL in the Data Explorer (see page 113)

Adding a New Connection (see page 107)

## 2.1.11 Modifying Database Connections

The basic elements of a connection string tend to be the same from one database type to another. However, each database type supports slightly different connection string syntax. This topic addresses those differences.

**To modify different types of database connections**

1. Click on the **Data Explorer** tab in the IDE.
2. Select the database type of your choice.
3. Right-click to display the popup menu.
4. Choose **Modify Connection** to display the **Connections Editor**. The properties in the Connections Editor are organized into three categories: Connections, Options, and Provider Settings. The Connections options designate the database and authentication parameters. The Options area includes various database-specific database options, including transaction isolation types. The Provider Settings area specifies assemblies and the client libraries required to accomplish the connection to the given database.

**Note:** All of the procedures in this topic assume that you already have installed a database client, server, or both, and that the database instance is running.

**To modify an InterBase connection**

1. Either enter the database name or navigate to the database on your local disk or a network drive, by clicking the ellipsis button to browse. The standard supplied databases are typically installed into C:\Program Files\Common Files\CodeGear Shared\Data.
2. Enter the password and username. By default, these are `masterkey` and `sysdba`, respectively.
3. Set the following options, if necessary. The default values are shown in the following table.

Option	Description	Default
CommitRetain	Commits the active transaction and retains the transaction context after a commit.	False
LoginPrompt	Determines if you want the user to be prompted for a login every time the application tries to connect to the database.	False
QuoteObjects	Specifies that table names, column names, and other objects should be quoted or otherwise delimited when included in a SQL statement. This is required for databases that allow spaces in names, such as MS Access.	False

RoleName	If there is a role for you in the database, you can enter the rolename here. The role is generally an authentication alias, that combines your identify with your access rights.	myRole
ServerCharSet	Specifies the character set on the server.	—
SQLDialect	Sets or returns the SQL dialect used by the client.	3
TransactionIsolation	Shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction, resulting in non-repeatable reads or phantom data. This specifies the value for the TADOdbxTransaction. IsolationLevel property.	ReadCommitted
WaitOnLocks	Specifies that a transaction wait for access if it encounters a lock conflict with another transaction.	False

4. You should be able to accept the defaults for the following Provider Settings:

Option	Default
Assembly	Borland.Data.Interbase,Version= <i>Current Product Version</i> ,Culture=neutral,PublicKeyToken= <i>Token #</i>
Provider	Interbase
VendorClient	gds32.dll

5. Click **Test** to see if the connection works.

6. Click **OK** to save the connection string.

**Note:** If you are writing ASP.NET applications, and are running the ASP.NET Web Forms locally for testing purposes, you might need to modify the path statement that points to your database, to include the `localhost:` designation. For example, you would modify the path shown earlier in this topic as such: `localhost:C:\Program Files\Common Files\CodeGear Shared\Data\employee.gdb`.

**Note:** Your connection string should resemble something like

```
database=C:\Program Files\Common Files\CodeGear Shared\Data\EMPLOYEE.GDB;
assembly=Borland.Data.Interbase,Version=2.0.0.0,Culture=neutral,PublicKeyToken=91d62ebb5b0d1b1b;
vendorclient=gds32.dll;provider=Interbase;username=sysdba;password=masterkey
```

### To modify an MS SQL Server connection

1. Enter the database name in the **Database** field of the **Connections Editor**. For example, use one of the sample MS SQL Server databases, such as Pubs or Northwind. There is no need to add the file extension to the name.
2. Enter the hostname. If you are using a local database server, enter `(local)` in this field.
3. If you are deferring to your OS authentication, set **OSAuthentication** to **True**.
4. If you are using database authentication, enter the password and username into the appropriate fields. By default, the SQL Server database username is `sa`.
5. Change the database options if necessary. The default values are shown in the following table.

Option	Description	Default
BlobSize	Specifies the upper limit of the size of any BLOB field.	1024
LoginPrompt	Determines if you want the user to be prompted for a login every time the application tries to connect to the database.	False

QuoteObjects	Specifies that table names, column names, and other objects should be quoted or otherwise delimited when included in a SQL statement. This is required for databases that allow spaces in names, such as MS Access.	False
TransactionIsolation	Shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction, resulting in non-repeatable reads or phantom data. This specifies the value for the TAdoDbxTransaction. IsolationLevel property.	ReadCommitted

6. You should be able to accept the defaults for the following Provider Settings:

Option	Default
Assembly	Borland.Data.Mssql,Version= <i>Current Product Version</i> ,Culture=neutral,PublicKeyToken= <i>Token #</i>
Provider	MSSQL
VendorClient	sqloledb.dll

7. Click **Test** to see if the connection works.

8. Click **OK** to save the connection string.

**Note:** If you are writing ASP.NET applications, and are running the ASP.NET Web Forms locally for testing purposes, you might need to modify the path statement that points to your database, to include the `localhost:` designation, prepended to the path.

**Note:** Your connection string should resemble something like

```
assembly=Borland.Data.Mssql,Version=2.0.0.0,Culture=neutral,PublicKeyToken=91d62ebb5b0d1b1b;
vendorclient=sqloledb.dll;osauthentication=True;database=Pubs;username=;hostname=(local);password=;
provider=MSSQL
```

### To modify a DB2 connection

1. Enter the path to the database.
2. Enter the password and username into the appropriate fields.
3. Set the following database options, if necessary. The default values are shown in the following table.

Option	Description	Default
LoginPrompt	Determines if you want the user to be prompted for a login every time the application tries to connect to the database.	False
QuoteObjects	Specifies that table names, column names, and other objects should be quoted or otherwise delimited when included in a SQL statement. This is required for databases that allow spaces in names.	False
TransactionIsolation	Shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction, resulting in non-repeatable reads or phantom data. This specifies the value for the TAdoDbxTransaction. IsolationLevel property.	ReadCommitted

4. You should be able to accept the defaults for the following Provider Settings:

Option	Default
Assembly	Borland.Data.Db2,Version= <i>Current Product Version</i> ,Culture=neutral,PublicKeyToken= <i>Token #</i>
Provider	DB2

VendorClient	db2cli.dll
--------------	------------

- Click **Test** to see if the connection works.
- Click **OK** to save the connection string.

### To modify an Oracle connection

- Enter the path to the database.
- If you are deferring to your OS authentication, set **OSAuthentication** to **True**. This means that the system defers to your local system username and password to login to the database.
- If you are using database authentication, enter the password and username into the appropriate fields. For example, the typical Oracle username and password for the sample database is **SCOTT** and **TIGER**, respectively.
- Set the following database options, if necessary. The default values are shown in the following table.

Option	Description	Default
LoginPrompt	Determines if you want the user to be prompted for a login every time the application tries to connect to the database.	False
QuoteObjects	Specifies that table names, column names, and other objects should be quoted or otherwise delimited when included in a SQL statement. This is required for databases that allow spaces in names.	False
TransactionIsolation	Shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction, resulting in non-repeatable reads or phantom data. This specifies the value for the TAdoDbxTransaction. IsolationLevel property.	ReadCommitted

- You should be able to accept the defaults for the following Provider Settings:

Option	Default
Assembly	Borland.Data.Oracle,Version= <i>Current Product Version</i> ,Culture=neutral,PublicKeyToken= <i>Token #</i>
Provider	Oracle
VendorClient	oci.dll

- Click **Test** to see if the connection works.
- Click **OK** to save the connection string.

### To modify an MS Access connection

- Either enter the database name or navigate to the database on your local disk or a network drive, by clicking the ellipsis button to browse. If you have the Office Component Toolkit installed, you might find Northwind in `C:\Program Files\Office Component Toolpack\Data\Northwind.mdb`.
- Enter the username and password. By default, you can generally try `admin` for the username and leave the password field empty.
- Set the following database options, if necessary. The default values are shown in the following table.

Option	Description	Default
BlobSize	Specifies the upper limit of the size of any BLOB field.	1024
LoginPrompt	Determines if you want the user to be prompted for a login every time the application tries to connect to the database.	False

QuoteObjects	Specifies that table names, column names, and other objects should be quoted or otherwise delimited when included in a SQL statement. This is required for databases that allow spaces in names, such as MS Access.	False
TransactionIsolation	Shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction, resulting in non-repeatable reads or phantom data. This specifies the value for the TAdoDbxTransaction. IsolationLevel property.	ReadCommitted

4. You should be able to accept the defaults for the following Provider Settings:

Option	Default
Assembly	Borland.Data.Msacc,Version= <i>Current Product Version</i> ,Culture=neutral,PublicKeyToken= <i>Token #</i>
Provider	MSAccess
VendorClient	msjet40.dll

5. Click **Test** to see if the connection works.

6. Click **OK** to save the connection string.

**Note:** Your connection string should resemble something like

```
database=C:\Program Files\Office Component Toolpack\Data\Northwind.mdb;
assembly=Borland.Data.Msacc,Version=2.0.0.0,Culture=neutral,PublicKeyToken=91d62ebb5b0d1b1b;
vendorclient=msjet40.dll;provider=MSAccess;username=admin;password=
```

### To modify a Sybase connection

1. Enter the path to the database.
2. Enter the password and username into the appropriate fields.
3. Set the following database options, if necessary. The default values are shown in the following table.

Option	Description	Default
BlobSize	Specifies the upper limit of the size of any BLOB field.	1024
ClientAppName	Client application name set by the middle-tier application.	—
ClientHostName	Client host name set by the middle-tier application.	—
LoginPrompt	Determines if you want the user to be prompted for a login every time the application tries to connect to the database.	False
PacketSize	Specifies the number of bytes per network packet transferred from the database server to the client.	512
QuoteObjects	Specifies that table names, column names, and other objects should be quoted or otherwise delimited when included in a SQL statement. This is required for databases that allow spaces in names, such as MS Access.	False
TransactionIsolation	Shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction, resulting in non-repeatable reads or phantom data. This specifies the value for the TAdoDbxTransaction. IsolationLevel property.	ReadCommitted

4. You should be able to accept the defaults for the following Provider Settings:

Option	Default
Assembly	Borland.Data.Sybase,Version= <i>Current Product Version</i> ,Culture=neutral,PublicKeyToken= <i>Token #</i>
Provider	Sybase
VendorClient	libct.dll

5. Click **Test** to see if the connection works.

6. Click **OK** to save the connection string.

**Note:** Your connection string should resemble something like

```
assembly=Borland.Data.Sybase,Version=2.0.0.0,Culture=neutral,  
PublicKeyToken=91d62ebb5b0d1b1b;vendorclient=libct.dll;database=Pubs;  
username=admin;hostname=host1;password=;provider=Sybase
```

#### See Also

ADO.NET Overview (see page 14)

Database Providers for .NET (see page 27)

ADO.NET Component Designers (see page 21)

Building an ASP.NET Database Application (see page 171)

Creating Database Projects from the Data Explorer (see page 111)

Passing Parameters in a Database Application (see page 124)

## 2.1.12 Building a Database Application that Resolves to Multiple Tables

RAD Studio supports multi-table resolution with BDP.NET. Specifically, the DataSync and DataHub components are designed to provide and resolve a .NET DataSet from multiple heterogeneous data sources. In addition, these components support the display of live data at designtime, and provide and resolve master-detail data by generating optimal SQL for resolving to BDP data sources.

The DataHub acts as a conduit between a DataSet and a DataSync. The DataPort property for a DataHub can be set to any IDataProvider implementation. DataSync implements IDataProvider and has a Providers collection that can contain any .NET data provider that implements IDbDataAdapter. The GetData method for DataSync iterates through all the DataProviders in the collection and returns a DataSet. SaveData resolves DataSet changes back to the database through the DataProvider collection. While resolving changes through a BdpDataAdapter the resolver generates optimal SQL. For non-BDP data providers, their respective CommandBuilder is used.

Building a database application that resolves multiple tables consists of the following steps:

1. Create a simple database project from the **Data Explorer** with multiple BdpDataAdapter objects to connect to multiple providers
2. Add and configure a DataSync component to connect the providers
3. Add and configure a DataHub component to connect the DataSync to a DataSet



### To create a database project from the Data Explorer

1. Choose **File ► New ► Windows Forms Application** for either Delphi for .NET. The Windows Forms designer appears.
2. Choose **View ► Data Explorer** to access the **Data Explorer**.
3. Expand the **Data Explorer Tree** to expose the providers and database tables you want to use. You must have a live connection to exprovider nodes. If you do not have a live connection, you may need to modify the connection string.
4. Drag and drop tables from one or more providers onto your form. For each table you drag onto your form, a BdpConnection and a BdpDataAdapter appear in the component tray. If you add multiple tables from the same provider, you can delete all but one BdpConnection for that provider.
5. Configure each BdpDataAdapter component. There is no need to set the Active or DataSet properties, as the DataSet will be populated by the DataHub component.
6. Add a DataSet component to your form from the **Data Components** category of the **Tool Palette**.
7. Add and configure a DataGrid component to your form from the **Data Controls** category of the **Tool Palette**. Set the DataSource property for the DataGrid to the name of the added DataSet component (for example, dataSet1).

### To add and configure a DataSync component

1. Drag a DataSync component onto your form from the **Borland Data Provider** category of the **Tool Palette**.
2. In the **Component Tray**, select the DataSync component.
3. In the **Object Inspector**, select the Providers property, and click the ellipsis button to open the **DataProvider Collection Editor**.
4. In the **DataProvider Collection Editor**, add a DataProvider for each table you want to provide and resolve. You should have a DataProvider for each BdpDataAdapter in your project.
5. For each DataProvider, select the DataProvider in the Members pane, and set the DataAdapter property to the appropriate BdpDataAdapter.
6. When you have finished configuring your DataProviders, click OK to close the **DataProvider Collection Editor**.
7. In the **Object Inspector**, set the CommitBehavior property to specify how failures are handled during resolving. There are three options for resolving logic:
  - Atomic—transactions are attempted for each provider. If a transaction fails, no further transactions are attempted, and all preceding transactions are rolled back. If there are no failed transactions, all transactions are committed.
  - Individual—a transaction is attempted for a provider, and if it succeeds, it is committed. The next transaction is attempted, and if it succeeds, it is committed, and so on. If a transaction fails for a provider, that transaction is rolled back, and no further transactions are attempted.
  - ForceIndividual—a transaction is attempted for a provider, and if it succeeds, it is committed. The next transaction is attempted, and if it succeeds, it is committed, and so on. If a transaction fails for a provider, that transaction is rolled back, and the next transaction is attempted.

### To add and configure a DataHub component

1. Drag a DataHub component onto your form from the **Borland Data Provider** category of the **Tool Palette**.
2. In the **Component Tray**, select the DataHub component.
3. In the **Object Inspector**, set the DataPort property to the added DataSync component (for example, DataSync1).
4. Set the DataSet property to the added DataSet (for example, dataSet1)
5. Choose **Run ► Run**. The application compiles and displays a Windows Form with a DataGrid.

### See Also

ADO.NET Overview (🔗 see page 14)

Data Providers for .NET (🔗 see page 27)

ADO.NET Component Designers (see page 21)

Building a Distributed Database Application (see page 136)

## 2.1.13 Passing Parameters in a Database Application

The following procedures describe a simple application that allows you to pass a parameter value at runtime to a DataSet. Parameters allow you to create applications at design time without knowing specifically what data the user will enter at runtime. This example process assumes that you already have your sample Interbase Employee database set up and connected. For purposes of illustration, this example uses the default connector IBConn1, which is set to a standard location. Your database location may differ.

### To pass a parameter

1. Create a data adapter and connection to the Interbase `employee.gdb` database.
2. Add a text box control, a button control, and a data grid control to your form.
3. Configure the data adapter.
4. To add a parameter to the data adapter.
5. Configure the data grid.
6. Add code to the button Click event..
7. Compile and run the application.

### To create a data adapter and connection

1. Choose **File ► New ► Windows Forms Application** for either Delphi for .NET or C#. The Windows Forms designer appears.
2. Click on the **Data Explorer** tab and drill down to find the IBConn1 connection under the Interbase node.
3. Drag and drop the EMPLOYEE table onto the Windows Form. This creates a BdpDataAdapter and BdpConnection and displays their icons in the **Component Tray**.
4. Select the data adapter icon, then click the **Configure Data Adapter** designer verb in the **Designer Verb** area at the bottom of the Object Inspector. This displays the **Data Adapter Configuration** dialog.
5. Rewrite the SQL statement that is displayed in the Select tab of the dialog to:

```
SELECT EMP_NO, FIRST_NAME, LAST_NAME, SALARY FROM EMPLOYEE WHERE FIRST_NAME = ?;
```

As you can see, this statement is limiting the number of fields. It also contains a ? character as part of the Where clause. The ? character is a wildcard that represents the parameter value that your application passes in at runtime. There are at least two reasons for using a parameter in this way. The first reason is to make the application capable of retrieving numerous instances of the data in the selected columns, while using a different value to satisfy the condition. The second reason is that you may not know the actual values at design time. You can imagine how limited the application might be if we retrieved only data where `FIRST_NAME = 'Bob'`.

6. Click the **DataSet** tab.
7. Click **New DataSet**.
8. Click **OK**. This creates the DataSet that represents your query.

### To add a parameter to the data adapter

1. Select the data adapter icon, then expand the properties under SelectCommand in the **Fill** area of the **Object Inspector**. You should be able to see your Select statement in the SelectCommand property drop down list box.
2. Change the ParameterCount property to 1.

3. Click the (Collection) entry next to the Parameters property. This displays the **BdpParameter Collection Editor**.
4. Click **Add** to add a new parameter.
5. Rename the parameter to *emp*.
6. Set BdpType to *String*, DbType to *Object*, Direction to *Input*, Source Column to *FIRST\_NAME*, and ParameterName to *emp*.
7. Click **OK**.
8. In the **Object Inspector**, set the Active property under Live Data to **True**.

#### To add controls to the form

1. Drag and drop a TextBox control onto the form.
2. Drag and drop a Button onto the form.
3. Change the Text property of the button to *Get Info*.
4. Drag and drop a DataGrid data control onto the form.
5. Arrange the controls how you want them to appear, making sure that the DataGrid is long enough to display four fields of data.

#### To configure the data grid

1. Select the data grid.
2. Set the DataSource property to the name of the DataSet (dataSet1 by default).
3. Set the DataMember property to *Table1*. This should display the column names of the columns specified in the SQL statement that you entered into the data adapter.

#### To add code to the button Click event

1. Double-click the button to open the Code Editor.
2. In the button1\_Click event code block, add the following code:

```
bdpSelectCommand1.Close();
/* This closes the command to make sure that we will pass the parameter to */
/* the most current
bdpSelectCommand.

*/

        bdpDataAdapter1.Active = false;
/* This clears the data adapter so that we don't maintain old
data                                     */

        bdpSelectCommand1.Parameters["emp"].Value = textBox1.Text;
/* This sets the parameter value to whatever value is in the text field.    */

        bdpDataAdapter1.Active = true;
/* This re-activates the data adapter so the refreshed data appears in the data grid. */
Self.bdpSelectCommand1.Close();
/* This closes the command to make sure that we will pass the parameter to */
/* the most current
bdpSelectCommand.

*/

        Self.BdpDataAdapter1.Active := false;
/* This clears the data adapter so that we don't maintain old
data                                     */

        Self.bdpSelectCommand1.Parameters['emp'].Value := textBox1.Text;
/* This sets the parameter value to whatever value is in the text field.    */
```

```
Self.BdpDataAdapter1.Active := true;  
/* This re-activates the data adapter so the refreshed data appears in the data grid. */
```

If you have changed the names of any of these items, you need to update these commands to reflect the new names.

3. Save your application.

### To compile and run the application

1. Press **Shift + F9** to compile the application.
2. Press **F9** to run the application.
3. Type one of the names John, Robert, Roger, Kim, Terri, Katherine, or Ann into the text box.
4. Click the button. This displays the employee number, first name, last name, and salary of the employee with that name in the data grid. If there is more than one person with the same first name, the grid displays all occurrences of employees with that name.

### See Also

ADO.NET Overview (see page 14)

Data Providers for Microsoft .NET (see page 27)

Building a Windows Forms Database Application

---

## 2.1.14 Using the Data Adapter Preview

CodeGear RAD Studio provides a tool that enables communication between a data source and a dataset. You can use the **Data Adapter Preview** to specify what data to move into and out of the dataset either in the form of SQL statements or stored procedures that are invoked to read or write a database.

### To use the Data Adapter Preview

1. After you have dropped a **BdpDataAdapter** component onto the designer, click the **Configure Data Adapter** designer verb that appears at the bottom of the **Object Inspector**.
2. Click the **Preview** tab to display the **Data Adapter Preview**.
3. To limit the number of rows fetched, click the **Limit rows** check box.
4. Enter the number of rows you want the result set to contain, in the **Rows to fetch** text box.
5. Click **Refresh** to re-execute the query and to refill the list box with the new number of rows.

### See Also

ADO.NET Overview (see page 14)

ADO.NET Component Designers (see page 21)

Building a Windows Forms Database Application

Using the Command Text Editor (see page 127)

Using the Connection Editor Designer (see page 128)

Using the Generate Dataset Designer (see page 132)

Using the Data Adapter Designer (see page 128)

## 2.1.15 Using the Command Text Editor

In order to create a DataSet, your BdpDataAdapter needs to have at least a SQL Select statement defined for the CommandText property. This statement, once built, appears as the CommandText of the BdpCommand object for the BdpDataAdapter. You can enter this Select statement manually, or you can use the **Command Text Editor** to construct the statement, along with Update, Insert, and Delete statements, using a simple point-and-click mechanism. Using this method, once you have a connection to a live data source, you will be able to see the names of tables and columns in the **Command Text Editor**. You can pick from listboxes to build the statement. Also, if you create your BdpDataAdapter using the **Data Explorer** and a live connection to a data source, a boilerplate Select statement is created for you in the form `select * from tablename`. You can use this statement to return all rows from the named data source, or you can modify the statement prior to generating the DataSet.

### To generate the commands

1. Select a connection from the **Connection** drop-down list box. This must be a BdpConnection you have already defined. Your associated BdpDataAdapter object must also be defined and must have the DataSet Active property set to **True**. This populates the Tables and Columns list boxes with data from the database.
2. Select a table from the Tables list box.
3. Select each column that you want to appear in your SQL statements. As you select the column names, they appear in the SQL text box.
4. Select the check box next to each statement type you want to generate.
5. Click the **Generate SQL** button.

### See Also

ADO.NET Overview (see page 14)

ADO.NET Component Designers (see page 21)

Building a Windows Forms Database Application

Using the Connection Editor Designer (see page 128)

Using the Data Adapter Designer (see page 128)

Using the Generate DataSet Designer (see page 132)

Using the Data Adapter Preview (see page 126)

## 2.1.16 Using the Data Adapter Designer

The Data Adapter contains, at a minimum, a SQL Select statement of the SELECT command property. You can enter this statement yourself, or using the **Data Adapter** designer you can construct the Select, along with the Update, Insert, and Delete statements. The BdpCommandBuilder constructs the Update, Insert, and Delete statements based on the tables and columns you have selected. The **Data Adapter** designer uses a live connection to retrieve metadata from which you can build the appropriate SQL statements for manipulating the data you want to move from a DataSet back into your database.

### To invoke the commands

1. Select a connection from the **Connection** drop-down list box. This must be a BdpConnection you have already defined. This populates the Tables and Columns list boxes with data from the database.

2. Select a table from the Tables list box.
3. Select each column that you want to appear in your SQL statements.
4. Select the check box next to each statement type you want to generate.
5. Click the **Generate SQL** button.
6. Edit the generated text if desired, or reselect different columns and click **Generate SQL** again.
7. Click **OK**.

**Note:** Command components are automatically created as needed based on the selections in the dialog.

### See Also

ADO.NET Overview (see page 14)

ADO.NET Component Designers (see page 21)

Building a Windows Forms Database Application

Using the Connection Editor Designer (see page 128)

Using the Command Text Editor (see page 127)

Using the Generate DataSet Designer (see page 132)

Using the Data Adapter Preview (see page 126)

---

## 2.1.17 Using the Connection Editor Designer

Each connection object can support multiple named connections. These connections can represent connections to multiple databases and database types.

### To add a new connection

1. Select an existing BdpConnection component in the designer, or drop a BdpConnection component onto the designer to create a new object.
2. Click the component designer tab at the bottom of the **Object Inspector** to display the **Connection Editor** dialog.
3. Click **Add** to display the **Add New Connection** dialog.
4. Select a provider from the **Provider Name** drop-down list box.
5. Enter a new name for the connection in the **Connection Name** text box.
6. Click **OK**.
7. Enter the appropriate values for your particular data source.
8. Click **OK**.

### To remove a connection

1. Select the connection type until it is highlighted.
2. Click **Remove**. A **Confirm Delete** dialog box appears.
3. Click **Yes**.

### To rename a connection

1. Right-click on the connection and choose **Rename**.

2. Type the new name of the connection.
3. Click **OK**.

**See Also**

ADO.NET Overview (🔗 see page 14)

ADO.NET Component Designers (🔗 see page 21)

Connection Pooling Options (🔗 see page 8)

Building a Windows Forms Database Application

Using the Command Text Designer (🔗 see page 127)

Using the Data Adapter Designer (🔗 see page 128)

Using the Generate Dataset Designer (🔗 see page 132)

Using the Data Adapter Preview (🔗 see page 126)

---

## 2.1.18 Using Standard DataSets

The standard DataSet provides an in-memory representation of one or more tables or views retrieved from a connected data source. Because of the level of indirection used in coding the underlying data structure, you are only able to see the column names from your data source at runtime. When you generate a DataSet, it retrieves everything you specified in your Select statement in the Data Adapter Configuration dialog. You can limit your columns by changing the Select statement and creating a new DataSet.

**To use DataSets**

1. Generate a DataSet.
2. Add multiple tables to a DataSet.
3. Define primary keys for DataTables in the DataSet.
4. Define column properties for your DataSet columns.
5. Define constraints for your columns.
6. Define relationships between tables in your DataSet.

**To generate a DataSet**

1. From the **Data Explorer**, select a data source.
2. Drill down in the tree, then drag and drop the name of a table onto your Windows Form or Web Form. This creates the BdpDataAdapter and BdpConnection for that data source and displays icons for those objects in the **Component Tray**.  
**Note:** You can also drag a data source only onto the form, rather than a table, but in that case, RAD Studio creates only a connection object for you. You must still create and configure the BdpDataAdapter object explicitly.
3. Click the BdpDataAdapter icon (named bdpDataAdapter1, by default) to select it.
4. Click the **Configure Data Adapter** designer verb in the **Designer Verb area** at the bottom of the **Object Inspector**. This displays the **Data Adapter Configuration** dialog.
5. If the SQL statement that is pre-filled on the dialog is acceptable, click the **DataSet** tab, otherwise, modify the SQL statement, then click the **DataSet** tab.
6. Select the **New DataSet** radio button.

**Tip:** You can accept the default name or change the name of the DataSet.

7. Click **OK** to generate the DataSet. A DataSet icon appears in the **Component Tray** indicating that your DataSet has been created.

**Note:** By reviewing the code for the DataSet in the Code Editor, you can see that the columns are defined as generic dataColumns, whose columnName properties are assigned the value of the column name from the database table. This differs from how a typed DataSet is constructed, wherein the object name is constructed from the actual database column name, rather than assigned as a property value.

### To add multiple tables to one DataSet

1. From the **Data Explorer**, select a data source.
2. Drill down in the tree, then drag and drop the names of multiple tables, one at a time, onto your Windows Form or Web Form. This creates the BdpDataAdapter for each table and one BdpConnection for that data source and displays icons for those objects in the **Component Tray**.
3. Click the BdpDataAdapter icon (named bdpDataAdapter1, by default) to select it.
4. Click the **Configure Data Adapter** designer verb in the **Designer Verb area** at the bottom of the **Object Inspector**. This displays the **Data Adapter Configuration** dialog.
5. If the SQL statement that is pre-filled on the dialog is acceptable, click the **DataSet** tab, otherwise, modify the SQL statement, then click the **DataSet** tab.
6. Select the **New DataSet** radio button.

**Tip:** You can accept the default name or change the name of the DataSet.

7. Click **OK** to generate the DataSet. A DataSet icon appears in the **Component Tray** indicating that your DataSet has been created.
8. Repeat the Data Adapter configuration for each of the other data adapters, but select **Existing Data Set** on the **DataSet** tab when generating the DataSets for all data adapters except the first one you configure. This generates a DataTable for each data adapter and stores them all in one DataSet.

**Note:** It is also possible to generate multiple DataSets, either one for each data adapter, or combinations of DataTables.

### To define primary keys for each DataTable in the DataSet

1. Select each data adapter in turn and set the Active property under **Live Data** in the **Object Inspector** to **True**.
2. Select the DataSet in the **Component Tray**.
3. In the **Object Inspector**, in the Tables property, click the ellipsis button. This displays the **Tables Collection Editor**. If you have set all of the data adapters' Active properties to **True**, the **Tables Collection Editor** will contain one member for each DataTable stored in the corresponding DataSet.
4. Select a table from the members list.
5. In the Primary Key field in the Table Properties, click on the DataColumn[] entry to display a pop-up list of column names.
6. Click the gray check box next to the column name of the column or columns that comprise the Primary Key. The number 1 appears in the gray check box when selected.
7. Define Column properties and Constraints for your Primary Key columns.

### To define column properties for your DataSet columns

1. In the Tables Collection Editor, click the (Collections) entry next to Columns in the Table Properties pane. This displays the Columns Collection Editor for the selected column.



2. Set the property values for the individual columns.
3. Repeat the process for each column.

### To define constraints for your columns

1. In the **Tables Collection Editor**, click the (Collections) entry next to Constraints in the **Table Properties** pane. This displays the Constraints Collection Editor for the selected column.
2. Click **Add** to add either a Unique Constraint or a Primary Key Constraint.
3. If you selected Unique Constraint, the Unique Constraint dialog appears. Select one or more of the displayed column names. You can also select the Primary Key check box if you want to set the column as a primary key. By setting the Unique Constraint on a column, you are enforcing the rule that all values in the column must be unique. This is useful for columns that contain identification numbers, such as employee numbers, social security numbers, part numbers, and so on.

**Note:** If you have already defined a primary-foreign key relationship between two tables, you may not be able to set a column as a primary key, based on the fact that it may already be set as the primary key, or based on a conflict with another relationship.

4. If you selected Foreign Key Constraint, the Foreign Key Constraint dialog appears. Select the tables you want to relate by choosing them from the Parent table and Child table drop down lists.
5. Click Key Columns to select the primary key column from the list.
6. Click Foreign Key Columns to select the foreign key column from the list.

**Warning:** The primary key and foreign key columns must have the same data type and must contain unique values. Columns that can contain duplicates are not good choices for primary or foreign keys. It is common to choose the same column name from each table for your primary-foreign key relationship.

### To define relationships between tables in the DataSet

1. Once you have defined primary keys for each DataTable, select the DataSet in the **Component Tray** if it is not already selected.
2. Click the ellipsis button next to the Relations property in the **Object Inspector**. This displays the blank **Relations Collection Editor** dialog.
3. Click **Add**. This displays the **Relation editor** dialog
4. From the Parent table and Child table dropdown lists, choose the tables you want to relate.
5. Click the Key Columns field to choose a Primary Key column from the list of column names from the parent table.
6. Click the Foreign Key Columns field to choose a Foreign Key column from the list of column names from the child table.

**Note:** If you have already performed this procedure while setting constraints for your DataTables, you may find that all of the appropriate values are already established.

**Warning:** The primary key and foreign key columns must have the same data type and must contain unique values. Columns that can contain duplicates are not good choices for primary or foreign keys. It is common to choose the same column name from each table for your primary-foreign key relationship.

7. Click **OK**.
8. Repeat the process to define additional relations between the same DataTables.

### See Also

ADO.NET Overview (🔗 see page 14)

ADO.NET Component Designers (🔗 see page 21)

Building a Windows Forms Database Application

Using the Command Text Designer (🔗 see page 127)

Using the Connection Editor Designer (🔗 see page 128)

Using the Data Adapter Designer (🔗 see page 128)

Using the Data Adapter Preview (🔗 see page 126)

Using Typed DataSets (🔗 see page 132)

---

## 2.1.19 Using Typed DataSets

Typed DataSets provide certain advantages over standard DataSets. For one thing, they are derived from an XML hierarchy of the target database table. The XML file containing the DataSet description allows the system to provide extensive code-completion capabilities not available when using standard DataSets. Strong typing of DataSet methods, properties, and events allows compile-time type checking, and can provide a performance improvement in some applications.

### To create a strongly typed DataSet

1. From the Database Explorer, select the data source you want to use.
2. Drag and drop the name of the database table you want to use onto your form. This displays a BdpConnection icon and a BdpDataAdapter icon in the **Component Tray**.
3. Select the BdpDataAdapter.
4. Click the **Configure Data Adapter** designer verb in the **Designer Verb** area beneath the **Object Inspector**. This displays the **Data Adapter Configuration** dialog.
5. Modify the pre-filled SQL statement if you like.
6. Click **OK**.  
**Note:** Do not create a DataSet by selecting the DataSet tab in the **Configure Data Adapter** dialog. That tab applies only to standard DataSets.
7. Click the **Generate Typed Dataset** designer verb in the **Designer Verb** area beneath the **Object Inspector**. This displays the **Generate Dataset** dialog.
8. Select the database table you want to use.
9. Click **OK**. This creates an instance of the typed DataSet and displays an icon *<DataSet Name>1* in the **Component Tray**. For example, if your DataSet is *DataSet1*, the new instance will be named *dataSet11*. You will also see that an XML .xsd file and a new program file appear in the **Project Manager** under your project.

### To modify how columns appear

1. After you have created a new typed DataSet, drop a **DataGrid** component onto your form.
2. Set the DataSource property to point to the typed DataSet and the DataMember property to point to the target table.
3. Click the *(Collection)* entry next to the TableStyles property. This displays the **DataGridTableStyle Collection Editor**.
4. Click **Add** to add a new member to the members list.
5. Click the drop down list next to the MappingName property.
6. Click the *(Collection)* entry next to the GridColumnStyles property. This displays the **DataGridColumnStyle Collection Editor**.
7. Click **Add** to add a new item to the members list.

**Note:** By default the item is created as a Text Box Column. You can also expand the Add

button and select the BoolColumn if you want a boolean.

- Click the MappingName property, select the column you want to display in your grid, then change any additional properties you want, including the header name that will appear as the column header in the runtime grid.
- Click **OK** twice.

**Note:** When you build and run the application, only the columns that you explicitly defined by following the steps in this procedure appear.

### To modify the structure of the dataset

- In the **Project Manager**, double-click the .xsd file that contains the XML definition of your dataset.
- Edit the XML file to reflect how you want the dataset to be structured. You can change data types, names, and anything else about the structure.
- If you have the program code file (<dataset>.cs or <dataset>.pas) open in the **Code Editor**, close it now.
- Choose **Project ► Compile** to recompile the .xsd file. If you re-open the program code file, you will see that the file contains the changes you made to the XML in the .xsd file.

### To set the Namespace property for a dataset

- In the **Project Manager**, double-click the .xsd file that contains the XML definition of your dataset.
- Find the targetNamespace property.
- Change the following text to a relevant namespace:  
`http://www.changeme.now/DataSet1.xsd`
- If you have the program code file (<dataset>.cs or <dataset>.pas) open in the **Code Editor**, close it now.
- Choose **Project ► Compile** to recompile the .xsd file. If you re-open the program code file, you will see that the InitClass() class now contains the new namespace.

### See Also

ADO.NET Overview (🔗 see page 14)

ADO.NET Component Designers (🔗 see page 21)

Building a Windows Forms Database Application

Using the Command Text Editor (🔗 see page 127)

Using the Connection Editor Designer (🔗 see page 128)

Using the Data Adapter Designer (🔗 see page 128)

Using the Data Adapter Preview (🔗 see page 126)

Using Standard DataSets (🔗 see page 129)

[Using Annotations with a Typed DataSet](#)

## 2.1.20 Connecting to a Database using the dbExpress Driver Framework

This procedure tells how to use the dbExpress driver framework to connect to a database and read its records. In the sample code, the dbExpress ini files contain all the information about the particular database connection, such as driver, user name,

password, and so on.

### To connect to a database and read its records

1. Configure the connections ini file with the information about the database you are connecting to. This includes setting the driver name, user name, password, and so on.
2. Obtain a TDBXConnectionFactory, which is returned by TDBXConnectionFactory.GetConnectionFactory.
3. Get a TDBXConnectionobject returned by TDBXConnectionFactory.GetConnection.
4. Open the database connection by calling TDBXConnection.Open on the TDBXConnection instance.
5. Get a TDBXCommandobject by calling TDBXConnection.CreateCommand on the TDBXConnection instance.
6. Set the TDBXCommand's Textproperty to the desired SQL command. Call TDBXCommand.Prepare on the TDBXCommand instance.
7. Execute the SQL query by calling TDBXCommand.ExecuteQuery, which returns a TDBXReader instance.
8. Read the first database record by calling TDBXReader.Next. Call this method to retrieve successive database records.
9. Get whatever information you want from the database. For instance, TDBXReader.GetColumnCount returns the number of database columns. The TDBXReader properties Value and ValueIndex contain the data type and value for a given column number in the current record.

```
// This sample connects to a database using the ini files.
// These files must be configured for the database.
// Once connected, the sample reads values and displays the
// ANSI values for the first 100 records in a listbox.

// Get a TDBXConnection using a TDBXConnectionFactory.
// ConnectionName = section in the connections ini file.
class function TForm1.BuildConnectionFromConnectionName(
  ConnectionName: WideString): TDBXConnection;
var
  ConnectionFactory: TDBXConnectionFactory;
  ConnectionProps: TDBXProperties;
begin
  ConnectionFactory := TDBXConnectionFactory.GetConnectionFactory;
  ConnectionProps := ConnectionFactory.GetConnectionProperties(ConnectionName);
  Result := ConnectionFactory.GetConnection(ConnectionProps,
    ConnectionProps.Values[TDBXPropertyNames.UserName],
    ConnectionProps.Values[TDBXPropertyNames.Password] );
end;

procedure Connect;
var
  connection: TDBXConnection;
  command: TDBXCommand;
  reader: TDBXReader;
  value: TDBXValue;
  valueType: TDBXValueType;
  colCountStr: string;
  i, j: Integer;
  numCols: integer;
  ListBox1: TListBox;

const
  sqlCommand = 'select * from employee';

begin
  // Open connection to DB.
  connection := BuildConnectionFromConnectionName('ConnectionName');
  connection.Open;
```

```

// Get command
command := connection.CreateCommand();
command.Text := sqlCommand;

// Execute query
command.Prepare;
reader := command.ExecuteReader;

// Get values from DB
if reader.Next then
begin
numCols := reader.GetColumnCount;
Str(numCols, colCountStr);
ListBox1.Items.Add('Number of columns = ' + colCountStr);
j := 1;
repeat
for i := 0 to reader.GetColumnCount - 1 do
begin
valueType := reader.ValueType[i];
if valueType.DataType = TDBXDataTypes.AnsiStringType then
begin
value := reader.Value[i];
ListBox1.Items.Add(valueType.Name + ' = ' +
value.GetString);
end
else
ListBox1.Items.Add(valueType.Name);
end;
Inc(j);
until (j > 100) or not reader.Next;

reader.Next;
end;

// Free resources
command.Free;
end;

```

## 2.1.21 Building a Distributed Database Application

Data remoting is fundamental to developing distributed database applications. The .NET remoting technology provides a flexible and extensible framework for interprocess communication. With .NET remoting you can interact with objects in different application domains, in different processes running on the same machine, or in different machines on a network.

Using the RemoteServer and RemoteConnection components, you can easily migrate a client/server application that uses DataHub and DataSync components to a multi-tier DataSet remoting application. RemoteServer implements IDataService and publishes itself as a singleton server activated object (SAO). On the client side, the RemoteConnection properties form the URL for connecting to the RemoteServer. Channel specifies the protocol to use (TCP/IP or HTTP), Port specifies the port on which the RemoteServer is listening for requests, and URI refers to the unique resource identifier for the RemoteServer.

Building a distributed application with data remoting components consists of the following steps:

- Build a server-side Windows Forms application with one or more connections to a BDP.NET data provider, a DataSync component to collect the connections and set the commit behavior, and a RemoteServer component to set the communication protocol and URI for communicating with clients
- Build a client-side Windows Forms application with RemoteConnection component with properties to specify the connection to the server-side application, a DataHub component for passing data to and from a DataSet, and a DataGrid to display the data

**Note:** The RemoteServer component is hosted in Windows Forms applications without adding any additional code manually.

### To create the server-side application

1. Choose **File ► New ► Windows Forms Application** for either Delphi for .NET or C#. The Windows Forms designer appears.
2. Choose **View ► Data Explorer** to access the **Data Explorer**, and expand the **Data Explorer Tree** to expose the providers and database tables you want to use. You must have a live connection to expand provider nodes. If you do not have a live connection, you may need to modify the connection string.
3. Drag and drop tables from one or more providers onto your form. For each table you drag onto your form, a **BdpConnection** and a **BdpDataAdapter** appear in the component tray. If you add multiple tables from the same provider, you can delete all but one **BdpConnection** for that provider.
4. Configure each **BdpDataAdapter** component. There is no need to set the **Active** or **DataSet** properties, as the **DataSet** will be populated by the **DataHub** component on the client-side.
5. Drag a **DataSync** component onto your form from the **Borland Data Provider** category of the **Tool Palette**, and configure the following **DataSync** properties in the **Object Inspector**:

Property	Description
Providers	Specifies a collection of <b>DataProviders</b> to use as data sources. Click the ellipsis button to open the <b>DataProvider Collection Editor</b> , and add a <b>DataProvider</b> for each table you want to provide and resolve.
CommitBehavior	Specifies the logic (Atomic, Individual, or ForceIndividual) for handling failures during resolving.

6. Drag a **RemoteServer** component onto your form from the **Borland Data Provider** category of the **Tool Palette**, and configure the following **RemoteServer** properties in the **Object Inspector**:

Property	Description
DataSync	Specifies the <b>DataSync</b> that needs remoting. Select the <b>DataSync</b> from the drop-down list in the <b>Object Inspector</b> .
AutoStart	Specifies whether or not to start the remote server automatically when the application runs. Set this property to <b>True</b> .
ChannelType	Specifies the channel type: <b>Http (HTTP)</b> or <b>Tcp (TCP/IP)</b> . Select the channel type from the drop-down list in the <b>Object Inspector</b> .
Port	Specifies the port the remote server will be listening on. Enter a new value, or accept the default port value, <b>8000</b> .
URI	Specifies the universal resource identifier for the remote server. By default, the <b>URI</b> property is the same as the <b>Name</b> property.

7. Choose **Run ► Run** to start the server-side application.

### To create the client-side application

1. Choose **File ► New ► Windows Forms Application** for either Delphi for .NET or C#. The Windows Forms designer appears.
2. Drag a **DataSet** component onto your form from the **Data Components** category of the **Tool Palette**.
3. Drag a **DataGrid** component to your form from the **Data Controls** category of the **Tool Palette**, and set the **DataSource** property for the **DataGrid** to the name of the added **DataSet** component (for example, **dataSet1**).
4. Drag a **RemoteConnection** component onto your form from the **Borland Data Provider** category of the **Tool Palette**, and configure the following **RemoteConnection** properties in the **Object Inspector**:

Property	Description
ProviderType	Specifies the type of provider published by the remote server. In this case, the property should be set to <b>Borland.Data.Provider.DataSync</b> . If the remote server is running, you can select this value from the drop-down list. Otherwise, you must enter the value.

ChannelType	Specifies the channel type: Http (HTTP) or Tcp (TCP/IP). Select the channel type from the drop-down list in the <b>Object Inspector</b> . This should match the setting for the remote server.
Host	The name or IP address of the remote server.
Port	Specifies the port the remote server will be listening on. Enter a new value, or accept the default port value, <b>8000</b> . This should match the setting for the remote server.
URI	Specifies the universal resource identifier for the remote server. This should match the URI property for the RemoteServer component in the remote server application.

5. Drag a DataHub component onto your form from the **Borland Data Provider** category of the **Tool Palette**, and configure the following DataHub properties in the **Object Inspector**:

Property	Description
DataPort	Specifies the data source. Set the DataPort property to the added RemoteConnection component (for example, RemoteConnection1).
DataSet	Specifies the DataSet to hold the data retrieved from the specified data source. Set this property to the added DataSet (for example, dataSet1).

6. Choose **Run ▶ Run**. The application compiles and displays a Windows Form with DataGrid.

#### See Also

ADO.NET Overview (🔗 see page 14)

Data Providers for .NET (🔗 see page 27)

ADO.NET Component Designers (🔗 see page 21)

Building a Database Application that Resolves to Multiple Tables (🔗 see page 122)

## 2.2 Interoperable Applications Procedures

This section provides how-to information on building interoperable applications.

### Topics

Name	Description
Adding a J2EE Reference (see page 139)	RAD Studio provides a way to generate a .NET assembly from a J2EE archive. After creating the assembly, a reference is automatically added to your project.
Adding a Reference to a COM Server (see page 139)	

### 2.2.1 Adding a J2EE Reference

RAD Studio provides a way to generate a .NET assembly from a J2EE archive. After creating the assembly, a reference is automatically added to your project.

#### To add a J2EE reference

1. In the **Project Manager**, right-click on the top-level project node and choose **Add J2EE Reference**. The **Select a J2EE Archive** dialog displays.
2. In the **Select a J2EE Archive** dialog, navigate to the `.jar` or `.ear` file from which to generate the .NET assembly.
3. In the **Select J2EE Archive** dialog, click **Open**. The **Select EJBs from List** dialog displays.
4. In the **Select EJBs from List** dialog, you may generate an assembly for all EJBs, or for individual EJBs in the archive.

### 2.2.2 Adding a Reference to a COM Server

#### To Add a Reference to a COM Server

1. In the **Project Manager**, right-click the **References** tree node of your project, and select **Add Reference**.
2. In the **Add Reference** dialog box, click the **COM Imports** tab. The IDE will scan the system registry for all registered type libraries and COM servers.
3. Select the item or items you want to reference in your project.  
**Tip:** You can individually select multiple items from the list by holding down the **CTRL** key as you click each item. To select a range of items, select the first item, then hold down the **SHIFT** key as you select the second item.
4. Click the **Add Reference** button. All of the items you selected will appear in the **New References** list in the bottom portion of the dialog.  
**Tip:** You can remove items from the **New References** list. Select the item or items and click the **Remove** button.
5. If the COM component you want to reference does not appear in the list, click the **Browse** button to add an explicit reference to it.
6. In the **Select a reference** dialog box, navigate to the folder where the component is located.



7. Select it, and click **Open**.
8. When you have selected all of the COM servers you wish to add, click **OK**.

After you click the **OK** button in the **Add Reference** dialog, the IDE will generate interop assemblies for each item you selected (unless a Primary Interop Assembly has already been created). These assemblies will be named `Interop.LibraryName.dll`, where `LibraryName` is the name of the component's type library (note this name might differ from the control's DLL file name). The generated assemblies will be stored in a folder called `COMImports`, under your project directory. Each generated interop assembly will be set to Copy Local, meaning, when the project is built, the assembly will be copied to the build target folder automatically.

The `COMImports` folder might not exist, for example, if you move the project to a new machine, or if you delete it on the machine where the project resides. If the `COMImports` folder does not exist when the project is reopened, the IDE will recreate it and regenerate the interop assemblies. In order for this to work, the COM servers must first be registered on the machine where the project resides.

If a Primary Interop Assembly for the COM server exists, the IDE will not generate a new interop assembly. Instead a reference to the Primary Interop Assembly will be added, and the Copy Local setting will be turned off, since Primary Interop Assemblies are deployed in the Global Assembly Cache.

**Note:** To see the Copy Local setting on any referenced assembly, right click the mouse on the assembly in the Project Manager

. The Copy Local setting is an item on the context menu. The project will still retain references to the interop assemblies, even if the `COMImports` folder could not be regenerated. In this case, the **Project Manager** will highlight the referenced assembly to indicate that it currently does not exist on the machine.

#### See Also

COM Interop Overview (🔗 see page 38)

Adding an ActiveX Control to the Tool Palette

## 2.3 Modeling Procedures

This section provides how-to information for modeling applications.

### Topics

Name	Description
Exporting a Code Visualization Diagram to an Image (see page 141)	You can export a code visualization diagram to an image and then open the image in any graphic viewer that supports the Windows bitmap (.bmp) file format.
Importing and Exporting a Model Using XML Metadata Interchange (XMI) (see page 142)	RAD Studio supports XML version 1.1. Please see the link to the OMG website at the end of this document for more information on XML or to download the complete specification.
Using the Model View Window and Code Visualization Diagram (see page 143)	Code Visualization allows you view and navigate the logical structure of your application, as opposed to the file-centric view of the <b>Project Manager</b> .
Using the Overview Window (see page 144)	Large, real-world models will not fit within the diagram window. To help you view the diagram, you can use the <b>Overview</b> window.

### 2.3.1 Exporting a Code Visualization Diagram to an Image

You can export a code visualization diagram to an image and then open the image in any graphic viewer that supports the Windows bitmap (.bmp) file format.

#### To export a diagram to an image

1. Open a project.
2. Click the **Model View** tab.
3. Right-click the diagram node in the tree and choose Export to Image
4. Adjust zoom settings, if necessary.
5. Click **Save**.
6. Name the image and click **Save**.

#### See Also

UML Features in Delphi for .NET

Integrated Modeling Tools Overview

Code Visualization Overview (see page 54)

Importing and Exporting a Model Using XMI (see page 142)

Using the Model View Window and Code Visualization Diagram (see page 143)

Using the Overview Window (see page 144)

Adding Columns to a Component

Using the OCL Expression Editor

## 2.3.2 Importing and Exporting a Model Using XML Metadata Interchange (XMI)

RAD Studio supports XMI version 1.1. Please see the link to the OMG website at the end of this document for more information on XMI or to download the complete specification.

### To import a model in XMI format

1. Export the model from the modeling tool, using the XMI format. When exporting from Rational Rose, choose XMI version 1.1, using the Unisys extension.
2. In RAD Studio, choose **File ► New ► Other**.
3. Select **ECO Windows Forms Application** from the **New Items** dialog box.

**Note:** You can use either Delphi for .NET, or C# when starting the new ECO application.

4. Open the **Model View Window**, right-click the top-level project node in the tree, and choose **Import Project from XMI**.
5. In the **XMI Import** dialog box, click the **Browse** button to navigate to the XMI file you exported in step one.
6. Click the **Import** button in the **XMI Import** dialog box.

RAD Studio will generate ECO-enabled, Delphi or C# source code for the model elements on the class diagrams in the XMI file.

### To export a model in XMI format

1. Open the **Model View Window**, right-click the top-level project node in the tree, and choose **Export Project to XMI**.
2. In the **XMI Export** dialog box, select the XMI version and XMI encoding appropriate for the tool you will ultimately use to open the model file.
3. Click the **Browse** button to navigate to the destination folder.
4. Enter a target file name for the exported file.
5. Click the **Export** button in the **XMI Export** dialog box.

### See Also

[Object Management Group \(OMG\) Website](#)

UML Features in Delphi for .NET

Integrated Modeling Tools Overview

Code Visualization Overview (see page 54)

Using the Model View Window and Code Visualization Diagram (see page 143)

Using the Overview Window (see page 144)

Adding Columns to a Component

Using the OCL Expression Editor

## 2.3.3 Using the Model View Window and Code Visualization Diagram

Code Visualization allows you view and navigate the logical structure of your application, as opposed to the file-centric view of the **Project Manager**.

### To Display the Model View Window

1. Start a new project or load an existing one.
2. Select **View ► Model View**. The **Model View window** will open, showing the elements of your project in a tree view.

### Using the Model View window with Code Visualization diagrams

To...	...do this
View or hide nested elements within a UML package, class, or interface	Click the plus sign (+) next to the element's icon to view nested items, or click the minus sign (-) to hide nested items.
View a <b>Code Visualization diagram</b> for a .NET namespace, or Delphi unit	Expand the namespace or unit icon (📁) and double-click the diagram icon (📊) in the <b>Model View tree</b> .
View a <b>Code Visualization diagram</b> for the entire project	Expand the project icon (📁) in the <b>Model View tree</b> , and double-click the diagram icon.
Open the <b>source code editor</b> on a specific item in the <b>Model View tree</b>	Right-click the item and choose Open Source. Note that a .NET namespace can span multiple source files. You cannot open a source file for a namespace directly from the <b>Model View tree</b> .
Open the <b>Code Visualization diagram</b> on a specific item in the <b>Model View tree</b>	Right-click the item and choose Show Element on Diagram.

The Code Visualization diagram has a set of functions that can help you view large models, show or hide attributes, properties, etc., and to move from the graphical depiction on the diagram, directly to the source code for that item.

### Using the Code Visualization diagram

To...	...do this
Rearrange items on the diagram	Click the item and drag it to a new location.
View or hide the attributes, operations, properties, and nested types for an item on the diagram	Click the plus sign (+) next to the category (attributes, operations, etc.) you want to view. Click the minus sign (-) to hide items of a particular category.
Perform an automatic layout of the items on the diagram	Right-click anywhere in the <b>Code Visualization diagram</b> window and choose <b>Layout ► Do Full Layout</b> , or <b>Layout ► Optimize Sizes</b> .
Print the diagram	Right-click anywhere in the <b>Code Visualization diagram</b> window and choose Print.
Open the <b>source code editor</b> on a specific item on the <b>Code Visualization diagram</b>	Right-click the item, and choose Go to Definition.
Save the diagram as an image	Right-click anywhere in the <b>Code Visualization diagram</b> and choose Export to Image.

**Note:** With the exception of the Code Visualization Overview, and Using the Overview Window, the links below are only available in the RAD Studio Architect edition.

**See Also**

Code Visualization Overview (see page 54)

Using the Overview Window (see page 144)

UML Features in Delphi for .NET

Integrated Modeling Tools Overview

Importing and Exporting a Model Using XMI (see page 142)

Building an ECO Enabled User Interface

Adding Columns to a Component

Using the OCL Expression Editor

---

## 2.3.4 Using the Overview Window

Large, real-world models will not fit within the diagram window. To help you view the diagram, you can use the **Overview** window.

**To Scroll the Model with the Overview Window**

1. Click the **Overview** button in the lower right corner of the diagram. A miniature view of the entire diagram is displayed in its own sizable window. A smaller rectangle within the **Overview** window contains that portion of the model that is currently displayed on the diagram.
2. Click and drag the floating rectangle within the **Overview** window to dynamically scroll to the portion of the diagram you want to view.
3. Click anywhere outside of the **Overview** window to close it.

**Tip:** The Overview

window has a sizing grip in the upper left corner. Click and drag the sizing grip to resize the **Overview** window. As you resize the window, its contents are scaled to fit the current size.

Resizing is useful when the model is large enough that you cannot read the text on the miniature representation in the **Overview** window.

**Note:** With the exception of the Code Visualization Overview, and Using the Model View Window and Code Visualization Diagram, the links below are only available in the RAD Studio Architect edition.

**See Also**

Code Visualization Overview (see page 54)

Using the Model View Window and Code Visualization Diagram (see page 143)

UML Features in Delphi for .NET

Integrated Modeling Tools Overview

Importing and Exporting a Model Using XMI (see page 142)

Adding Columns to a Component

Using the OCL Expression Editor

## 2.4 VCL for .NET Procedures

This section provides how-to information on developing VCL for .NET applications.

### Topics

Name	Description
Building VCL Forms Applications With Graphics (📖 see page 147)	<p>Each of the procedures listed below builds a VCL Form application that uses graphics. Build one or more of the examples.</p> <ol style="list-style-type: none"> <li>1. Draw straight lines.</li> <li>2. Draw rectangles and ellipses.</li> <li>3. Draw a polygon.</li> <li>4. Display a bitmap image.</li> <li>5. Place a bitmap in a combo box.</li> </ol>
Building a VCL.NET Forms ADO.NET Database Application (📖 see page 148)	<p>The following procedure describes how to build an ADO.NET database application.</p> <p>Building a VCL.NET ADO.NET application consists of the following major steps:</p> <ol style="list-style-type: none"> <li>1. Set up the database connection.</li> <li>2. Set up the dataset.</li> <li>3. Set up the data provider, client dataset, and data source.</li> <li>4. Connect a DataGrid to the connection components.</li> <li>5. Run the application.</li> </ol>
Building a VCL Forms Application (📖 see page 149)	<p>The following procedure illustrates the essential steps to building a VCL Forms application using RAD Studio.</p>
Creating Actions in a VCL Forms Application (📖 see page 150)	<p>Using RAD Studio, the following procedures illustrate how to create actions using the ActionList tool. You will set up a simple application and create an edit menu with cut and paste actions that can be used to cut and paste to a memo.</p> <p>Creating the VCL application consists of the following major steps:</p> <ol style="list-style-type: none"> <li>1. Add main menu, actionlist, and memo tools to a form.</li> <li>2. Create the cut and paste actions.</li> <li>3. Add the actions to the main menu and associate with the edit action category.</li> <li>4. Build and run the application.</li> </ol>
Building a VCL Forms Hello World Application (📖 see page 151)	<p>The Windows Forms <i>Hello World</i> application demonstrates the essential steps for creating a VCL Forms application. The application uses a VCL Form, a control, an event, and displays a dialog in response to a user action.</p> <p>Creating the <i>Hello World</i> application consists of the following steps:</p> <ol style="list-style-type: none"> <li>1. Create a VCL.NET Form with a button control.</li> <li>2. Write the code to display "Hello World" when the button is clicked.</li> <li>3. Run the application.</li> </ol>

Using ActionManager to Create Actions in a VCL Forms Application (see page 152)	<p>Using RAD Studio, the following procedure illustrates how to create actions using ActionManager. It sets up a simple user interface with a text area, as would be appropriate for a text editing application, and describes how to create a file menu item with a file open action.</p> <p>Building the VCL application with ActionManager actions consists of the following major steps:</p> <ol style="list-style-type: none"> <li>1. Add a file open action to the ActionManager on a form.</li> <li>2. Create the main menu.</li> <li>3. Add the action to the menu.</li> <li>4. Build and run the application.</li> </ol>
Building a VCL Forms dbExpress.NET Database Application (see page 153)	<p>The following procedures describe how to build a dbExpress database application.</p> <p>Building a VCL Forms dbExpress.NET application consists of the following major steps:</p> <ol style="list-style-type: none"> <li>1. Set up the database connection.</li> <li>2. Set up the unidirectional dataset.</li> <li>3. Set up the data provider, client dataset, and data source.</li> <li>4. Connect a DataGrid to the connection components.</li> <li>5. Run the application.</li> </ol>
Building an Application with XML Components (see page 155)	This example creates a VCL Forms application that uses an XMLDocument component to display contents in an XML file.
Making Changes Required Due to 64-bit .NET 2.0 Support (see page 157)	<p>Changes have been made to support 64-bit .NET 2.0. These changes might require minor code changes so that existing applications work correctly. This document describes these changes in detail. There are two general areas:</p> <ol style="list-style-type: none"> <li>1. Changed code patterns</li> <li>2. Windows API declarations for some callbacks and records</li> </ol> <p><b>Warning:</b> These changes are required due to changes made to support 64-bit systems. They must be made for applications intended for both 32-bit and 64-bit systems.</p>
Creating a New VCL.NET Component (see page 159)	You can use the New VCL Component wizard to create a new VCL.NET component and add it to the <b>Tool Palette</b> .
Displaying a Bitmap Image in a VCL Forms Application (see page 160)	<p>These procedures load a bitmap image from a file and displays it to a VCL form.</p> <ol style="list-style-type: none"> <li>1. Create a VCL form with a button control.</li> <li>2. Provide a bitmap image.</li> <li>3. Code the button's onClick event handler to load and display a bitmap image.</li> <li>4. Build and run the application.</li> </ol>
Drawing Rectangles and Ellipses in a VCL Forms Application (see page 161)	<p>These procedures draw a rectangle and ellipse in a VCL form.</p> <ol style="list-style-type: none"> <li>1. Create a VCL form.</li> <li>2. Code the form's OnPaint event handler to draw a rectangle and ellipse.</li> <li>3. Build and run the application.</li> </ol>
Drawing a Rounded Rectangle in a VCL Forms Application (see page 162)	<p>These procedures draw a rounded rectangle in a VCL form.</p> <ol style="list-style-type: none"> <li>1. Create a VCL form.</li> <li>2. Code the form's OnPaint event handler to draw a polygon.</li> <li>3. Build and run the application.</li> </ol>

Drawing Straight Lines In a VCL Forms Application (🔗 see page 163)	<p>These procedures draw two diagonal straight lines on an image in a VCL form.</p> <ol style="list-style-type: none"> <li>1. Create a VCL form.</li> <li>2. Code the form's OnPaint event handler to draw the straight lines.</li> <li>3. Build and run the application.</li> </ol>
Placing a Bitmap Image in a Control in a VCL Forms Application (🔗 see page 164)	<p>These procedures add a bitmap image to a combo box in a VCL forms application.</p> <ol style="list-style-type: none"> <li>1. Create a VCL form.</li> <li>2. Place components on the form.</li> <li>3. Set component properties in the Object Inspector.</li> <li>4. Write event handlers for the component's drawing action.</li> <li>5. Build and run the application.</li> </ol>
Importing .NET Controls to VCL.NET (🔗 see page 165)	<p>You might want to use .NET components on your VCL.NET forms. There is no direct way to use .NET components. You can, however, wrap the components in an ActiveX wrapper, which then can be added to your VCL.NET application. RAD Studio provides the <b>.NET Import Wizard</b> to accomplish this task.</p>

## 2.4.1 Building VCL Forms Applications With Graphics

Each of the procedures listed below builds a VCL Form application that uses graphics. Build one or more of the examples.

1. Draw straight lines.
2. Draw rectangles and ellipses.
3. Draw a polygon.
4. Display a bitmap image.
5. Place a bitmap in a combo box.

### See Also

VCL.NET Overview (🔗 see page 71)

Drawing Straight Lines In a VCL Application (🔗 see page 163)

Drawing Rectangles and Ellipses in a VCL Application (🔗 see page 161)

Drawing a Rounded Rectangle in a VCL Application (🔗 see page 162)

Displaying a Bitmap Image in a VCL Application (🔗 see page 160)

Placing A Bitmap Image In a Combo Box of a VCL Application (🔗 see page 164)

## 2.4.2 Building a VCL.NET Forms ADO.NET Database Application

The following procedure describes how to build an ADO.NET database application.

Building a VCL.NET ADO.NET application consists of the following major steps:

1. Set up the database connection.



2. Set up the dataset.
3. Set up the data provider, client dataset, and data source.
4. Connect a DataGrid to the connection components.
5. Run the application.

#### To add an ADO connection component

1. Choose **File ► New ► Other ► Delphi for .NET Projects ► VCL Forms Application**. The **VCL Forms Designer** displays.
2. From the **dbGO** category of the **Tool Palette**, place a TADOConnection component on the form.
3. Double-click the TADOConnection component to display the **ConnectionString** dialog.
4. If necessary, select **Use Connection String**; then click the **Build** button to display the **Link Properties** dialog box.
5. On the **Provider** page of the dialog, select **Microsoft Jet 4.0 OLE DB Provider**; then click the **Next** button to display the **Connections** page.
6. On the **Connections** page, click the ellipsis button to browse for the `dbdemos.mdb` database. The default path to this database is `C:\Program Files\Common Files\Borland Shared\Data`.
7. If it is not already filled in, enter `Admin` in the **User name** field and select the **Blank password** check box.
8. Click **Test Connection** to confirm the connection. A dialog appears, indicating the status of the connection.
9. Click **OK** twice to close the **Data Link Properties** dialog box and the **ConnectionString** dialog box.

#### To set up the dataset

1. From the **dbGO** category, double-click a TADODataset component to place it on the form.
2. In the **Object Inspector**, set the Connection property drop-down list from the **Linkages** category to `ADOConnection1`.
3. Set the CommandText to an SQL command, for example, `Select * from orders`. You can either type the Select statement in the **Object Inspector** or click the ellipsis button to the right of CommandText to display the **Command Text Editor** where you can build your own query statement.

**Tip:** If you need additional help while using the CommandText Editor, click the **Help** button or press `F1`.

4. Set the Active property to **True** to open the dataset. You are prompted to log in.
5. Enter `Admin` for the username.
6. Leave the password field blank.

#### To add the provider

1. From the **Data Access** category of the **Tool Palette**, double-click a TDataSetProvider component to place it at the top of the form.
2. In the **Object Inspector**, set the DataSet property to `ADODataset1`.

#### To add client dataset

1. From the **Data Access** category of the **Tool Palette**, double-click a TClientDataSet component to place it to the right of the DataSetProvider component on the form.
2. In the **Object Inspector**, set the ProviderName property to `DataSetProvider1`.
3. Set the Active property to **True** to allow data to be passed to your application. A data source connects the client dataset with data-aware controls. Each data-aware control must be associated with a data source component to have data to display and manipulate. Similarly, all datasets must be associated with a data source component for their data to be displayed and manipulated in data-aware controls on the form.

**To add the data source**

1. From the **Data Access** category of the **Tool Palette**, double-click a TDataSource component to place it to the right of the ClientDataSet on the form.
2. In the **Object Inspector**, set the DataSet property to `ClientDataSet1`.

**To connect a DataGrid to the DataSet**

1. From the **Data Controls** area of the **Tool Palette**, double-click a TDBGrid component to place it on the form.
2. In the **Object Inspector**, set the DataSource property to `DataSource1`.
3. Select **Run ▶ Run**. You are prompted to log in.
4. Enter `Admin` for the username.
5. Leave the password field blank.
6. Click **OK**. The application compiles and displays a VCL form with a DBGrid.

**See Also**

VCL.NET Overview (see page 71)

Building a VCL Forms dbExpress.NET Database Application (see page 153)

---

## 2.4.3 Building a VCL Forms Application

The following procedure illustrates the essential steps to building a VCL Forms application using RAD Studio.

**To create a VCL Form**

1. Choose **File ▶ New ▶ Other**. The **New Items** dialog appears.
2. Select **Delphi for .NET Projects**.
3. Double-click **VCL Forms Application**. The **VCL Forms Designer** displays.
4. From the **Tool Palette**, place components onto the form to create the user interface.
5. Write the code for the controls.

**To associate code with a control**

1. Double-click a component on the form. The **Code Editor** displays, cursor in place within the event handler block.
2. Code your application logic.
3. Save and compile the application.

**See Also**

VCL.NET Overview (see page 71)

---

## 2.4.4 Creating Actions in a VCL Forms Application

Using RAD Studio, the following procedures illustrate how to create actions using the ActionList tool. You will set up a simple application and create an edit menu with cut and paste actions that can be used to cut and paste to a memo.

Creating the VCL application consists of the following major steps:

1. Add main menu, actionlist, and memo tools to a form.
2. Create the cut and paste actions.
3. Add the actions to the main menu and associate with the edit action category.
4. Build and run the application.

#### To add the main menu, actionlist, and memo to a form

1. Choose **File ► New ► Other ► Delphi for .NET Projects ► VCL Forms Application** to create a new form.
2. Click the **Design** tab to switch to the **VCL Form Designer**.
3. From the **Standard** category of the **Tool Palette**, place a **TMainMenu**, **TActionList**, and **TMemo** component on the form.

#### To create the actions

1. Double-click **ActionList1** on the form. The **ActionList Editor** displays.
2. Select **New Standard Action** from the drop-down list to display the **Standard Action Classes** dialog box.
3. Scroll to the **TEditCut** action, select it, and click **OK**. **EditCut1** displays in the **Actions** list in the editor.
4. Select **New Standard Action** from the drop-down list to display the **Standard Action Classes** dialog box.
5. Scroll to the **TEditPaste** action, select it, and click **OK**. **EditPaste1** displays in the **Actions** list in the editor.
6. Close the **ActionList Editor** window.

#### To add the cut and paste actions to the edit category in the main menu

1. Double-click **MainMenu1** on the form. The **MainMenu1 Editor** displays with the first blank command category selected.
2. In the **Object Inspector**, enter **Edit** for the **Caption** property and press **ENTER**. **Edit** displays as the first command category.
3. Click **Edit** to display a blank action just below it.
4. Click the blank action to select it.
5. In the **Object Inspector**, select **EditCut1** from the drop-down list of actions in the **Action** property, located in the **Linkage** category.
6. If not already filled in, expand the list of Action properties, enter **Cut** for the **Caption** property, enter **Edit** for the category, and press **ENTER**. **Cut** displays as the first action.
7. In the **MainMenu1 Editor**, click the second blank action beneath **Cut** to select it.
8. In the **Object Inspector**, select **EditPaste1** from the drop-down list of actions in the **Action** property, located in the **Linkage** category.
9. Expand the list of Action properties, and if necessary, enter **Paste** for the **Caption** property, enter **Edit** for the category, and press **ENTER**. **Paste** displays as the second action.

#### To build and run the application

1. Save all files in the project.
2. Choose **Run ► Run**. The application executes, displaying a form with the main menu bar and the **Edit** menu.
3. In the application, select text in the memo.
4. Choose **Edit ► Cut**. The text is cut from the memo.
5. Choose **Edit ► Paste**. The text is pasted back into the memo.

#### See Also

VCL.NET Overview (see page 71)

Building a VCL Forms Application (see page 149)

## 2.4.5 Building a VCL Forms Hello World Application

The Windows Forms *Hello World* application demonstrates the essential steps for creating a VCL Forms application. The application uses a VCL Form, a control, an event, and displays a dialog in response to a user action.

Creating the *Hello World* application consists of the following steps:

1. Create a VCL.NET Form with a button control.
2. Write the code to display "Hello World" when the button is clicked.
3. Run the application.

### To create a VCL Form

1. Choose **File ► New ► Other ► Delphi for .NET Projects ► VCL Forms Application**. The **VCL Forms Designer** displays.
2. Click the **Design** tab to display the form view.
3. From the **Standard** category of the **Tool Palette**, place a TButton component on the form.

### To display the "Hello World" string

1. Select **Button1** on the form.
2. In the **Object Inspector**, double-click the OnClick event handler on the **Events** tab. The **Code Editor** appears, with the cursor in the TForm1.Button1Click event handler block.
3. Place the cursor before the `begin` reserved word and then press **Return**. This creates a new line above the code block.
4. Insert the cursor on the new line created, and type the following variable declaration:

```
var s: string;
```

5. Insert the cursor within the code block, and type the following code:

```
s:= 'Hello World!';  
ShowMessage(s);
```

### To run the "Hello World" application

1. Save your project files.
2. Choose **Run ► Run** to build and run the application. The form displays with a button called **Button1**.
3. Click **Button1**. A dialog box displays the message "Hello World!".
4. Click **OK** to close the message dialog.
5. Close the VCL form to return to the IDE.

### See Also

VCL.NET Overview (🔗 see page 71)

Building a VCL Forms Application (🔗 see page 149)

## 2.4.6 Using ActionManager to Create Actions in a VCL Forms Application

Using RAD Studio, the following procedure illustrates how to create actions using ActionManager. It sets up a simple user

interface with a text area, as would be appropriate for a text editing application, and describes how to create a file menu item with a file open action.

Building the VCL application with ActionManager actions consists of the following major steps:

1. Add a file open action to the ActionManager on a form.
2. Create the main menu.
3. Add the action to the menu.
4. Build and run the application.

#### To add a file open action to ActionManager

1. Choose **File ► New ► Other ► Delphi for .NET Projects ► VCL Forms Application** to create a new form.
2. From the **Additional** page of the **Tool Palette**, add a TActionManager component to the form.
3. Double-click the TActionManager component to display the **Action Manager** editor.

**Tip:** To display captions for nonvisual components such as TActionManager, choose Tools->Environment Options . On the **Designer** tab, check **Show component captions**, and click **OK**.

4. If necessary, click the **Actions** tab.
5. Select **New Standard Action** from the drop-down list to display the **Standard Action Classes** dialog.
6. Scroll to the **File** category, and click the TFileOpen action.
7. Click **OK** to close the dialog.
8. In the **Action Manager** editor, select the **File** category. **Open...** displays in the **Actions:** list box.
9. Click **Close** to close the editor.

#### To create the main menu and add the File action to it

1. From the **Additional** page of the **Tool Palette**, place a TActionMainMenuBar component on the form.
2. Open the **Action Manager** editor, and select the **File** category from the **Categories** list box.
3. Drag **File** to the blank menu bar. File displays on the menu bar.
4. Click **Close** to close the editor.

#### To build and run the application

1. Select **Run ► Run**. The application executes, displaying a form with the main menu bar and the **File** menu.
2. Select **File ► Open**. The **Open** file dialog displays.

#### See Also

VCL.NET Overview (🔗 see page 71)

Building a VCL Forms Application (🔗 see page 149)

---

## 2.4.7 Building a VCL Forms dbExpress.NET Database Application

The following procedures describe how to build a dbExpress database application.

Building a VCL Forms dbExpress.NET application consists of the following major steps:

1. Set up the database connection.
2. Set up the unidirectional dataset.
3. Set up the data provider, client dataset, and data source.
4. Connect a DataGrid to the connection components.
5. Run the application.

#### To add a dbExpress connection component

1. Choose **File ► New ► VCL Forms Application**. The **VCL Forms Designer** displays.
2. From the **dbExpress** category of the **Tool Palette**, place a TSQLConnection component on the form.
3. Double-click the TSQLConnection component to display the **Connection Editor**.
4. In the **Connection Editor**, set the **Connection Name** list to **IBConnection**.
5. In the **Connections Setting** box, specify the path to the InterBase database file called **employee.gdb** in the **Database** field. By default, the file is located in **C:\Program Files\Common Files\CodeGear Shared\Data**.
6. Accept the value in the **User\_Name** field (**sysdba**) and **Password** field (**masterkey**).
7. To test the connection, click the button with the checkmark on it (just above the **Connection Name** list).

**Note:** By default, you are prompted to log in. Use the masterkey password. If the connection works a confirmation message appears. If you cannot connect to the database, make sure you have installed Interbase and that the server is started.

8. Click **OK** to close the **Connection Editor** and save your changes.

#### To set up the unidirectional dataset

1. From the **dbExpress** category of the **Tool Palette**, place a TSQLDataSet component at the top of the form.
2. In the **Object Inspector**, set the SQLConnection property drop-down list to **SQLConnection1**.
3. Set the CommandText to a SQL command, for example, **Select \* from sales**. For the SQL command, you can either type a **Select** statement in the **Object Inspector** or click the ellipsis to the right of CommandText to display the **Command Text Editor** where you can build your own query statement.

**Tip:** If you need additional help while using the Command Text Editor

, click the **Help** button or press **F1**.

4. In the **Object Inspector**, set the Active property to **True** to open the dataset.

#### To add the provider

1. From the **Data Access** category of the **Tool Palette**, place a TDataSetProvider component at the top of the form.
2. In the **Object Inspector**, set the DataSet property drop-down list to **SQLDataSet1**.

#### To add client dataset

1. From the **Data Access** category of the **Tool Palette**, place a TClientDataSet component to the right of the DataSetProvider component on the form.
2. In the **Object Inspector**, set the ProviderName drop-down to **DataSetProvider1**.
3. Set the Active property to **True** to allow data to be passed to your application.

A data source connects the client dataset with data-aware controls. Each data-aware control must be associated with a data source component to have data to display and manipulate. Similarly, all datasets must be associated with a data source component for their data to be displayed and manipulated in data-aware controls on the form.

#### To add the data source

1. From the

**Data Access** category of the **Tool Palette**, place a TDataSource component to the right of the ClientDataSet on the form.

2. In the **Object Inspector**, set the DataSet property drop-down to `ClientDataSet1`.

#### To connect a DataGrid to the DataSet

1. From the **Data Controls** category of the **Tool Palette**, place a TDBGrid component on the form.
2. In the **Object Inspector**, set the DataSource property drop-down to `DataSource1`.
3. Save all files in the project.
4. Select **Run ▶ Run**. You are prompted to enter a password.
5. Enter `masterkey` as the password. The application compiles and displays a VCL.NET form with a DBGrid.

#### See Also

VCL.NET Overview (see page 71)

## 2.4.8 Building an Application with XML Components

This example creates a VCL Forms application that uses an XMLDocument component to display contents in an XML file.

#### The basic steps are:

1. Create an XML document.
2. Create a VCL form.
3. Place an XMLDocument component on the form, and associate it with the XML file.
4. Create VCL components to enable the display of XML file contents.
5. Write event handlers to display XML child node contents.
6. Compile and run the application.

#### To create the XML document

1. Copy the text below into a file in a text editor.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE StockHoldings [
  <!ELEMENT StockHoldings (Stock+)>
  <!ELEMENT Stock (name)>
  <!ELEMENT Stock (price)>
  <!ELEMENT Stock (symbol)>
  <!ELEMENT Stock (shares)>
]>

<StockHoldings>
  <Stock exchange="NASDAQ">
    <name>Borland</name>
    <price>10.375</price>
    <symbol>BORL</symbol>
    <shares>100</shares>
  </Stock>

  <Stock exchange="NYSE">
    <name>MyCompany</name>
    <price>8.75</price>
    <symbol>MYCO</symbol>
    <shares type="preferred">25</shares>
  </Stock>
```

```
</StockHoldings>
```

2. Save the file to your local drive as an XML document. Give it a name such as `stock.xml`.
3. Open the document in your browser. The contents should display without error.

**Note:** In the browser, you can choose View->Source  
to view the source file in the text editor.

### To create a form with an XMLDocument component

1. Start a new project.
2. Choose **File ► New ► Other**.
3. In the **New Items** dialog box, select **Delphi for .NET Projects**.
4. Double-click **VCL Forms Application**. The **VCL Forms Designer** displays.
5. From the **Internet** category on the **Tool Palette**, place an **TXMLDocument** component on the form.
6. In the **Object Inspector**, click the ellipsis button next to the **FileName** property, browse to the location of the XML file you created, and open it. The XML file is associated with the **TXMLDocument** component.
7. In the **Object Inspector**, set the **Active** property to **True**.

### To set up the VCL components

1. From the **Standard** page on the **Tool Palette**, place a **TMemo** component on the form.
2. From the **Standard** page on the **Tool Palette**, place two **TButton** components on the form just above **Memo1**.
3. In the **Object Inspector** with **Button1** selected, enter **Borland** for the **Caption** property.
4. In the **Object Inspector** with **Button2** selected, enter **MyCompany** for the **Caption** property.

### To display child node contents in the XML file

1. Select **Button1**.
2. In the **Object Inspector** double-click the **OnClick** event on the **Events** tab. The code displays with the cursor in the `TForm1.Button1Click` event handler block.
3. Enter the following code to display the stock price for the first child node when the **Borland** button is clicked:

```
BorlandStock:=XMLDocument1.DocumentElement.ChildNodes[0];
Price:= BorlandStock.ChildNodes['price'].Text;
Memo1.Text := Price;
```

4. Add a **var** section just above the code block, above the `begin` statement in the event handler, and enter the following local variable declarations:

```
var
    BorlandStock: IXMLNode;
    Price: string;
```

5. Select **Button2**.
6. In the **Object Inspector** double-click the **OnClick** event on the **Events** tab. The code displays with the cursor in the `TForm1.Button2Click` event handler block.

7. Enter the following code to display the stock price for the second child node when the **MyCompany** button is clicked:

```
MyCompany:=XMLDocument1.DocumentElement.ChildNodes[1];
Price:= MyCompany.ChildNodes['price'].Text;
Memo1.Text := Price;
```

8. Add a **var** section just above the code block, above the `begin` statement in the event handler, and enter the following local variable declarations:

```
var
    MyCompany: IXMLNode;
```



```
Price: string;
```

### To compile and run the application

1. Select **Run ▶ Run** to compile and execute the application. The application displays two buttons and a memo.
2. Click the **Borland** button. The stock price displays.
3. Click the **MyCompany** button. The stock price displays.

### See Also

VCL.NET Overview (see page 71)

Building a VCL Forms Application (see page 149)

## 2.4.9 Making Changes Required Due to 64-bit .NET 2.0 Support

Changes have been made to support 64-bit .NET 2.0. These changes might require minor code changes so that existing applications work correctly. This document describes these changes in detail. There are two general areas:

1. Changed code patterns
2. Windows API declarations for some callbacks and records

**Warning:** These changes are required due to changes made to support 64-bit systems. They must be made for applications intended for both 32-bit and 64-bit systems.

### To make required changes for various code patterns

1. **Case statements** Using a handle in a case statement no longer compiles. Replace statements like this:

```
case Msg.WParam of
```

with one of the following:

```
case Msg.WParam.ToInt64 of // VCL.NET specific
case Int64(Msg.WParam) of  // Compatible with VCL/Win32
```

2. **Casting enumerations** A handle can no longer be directly type cast to an enumerated type. Replace code like this:

```
LEnum := TEnum(GetWindowLong(...));
```

with this:

```
LEnum := TEnum(GetWindowLong(...).ToInt64);
```

3. **Using handles with sets** Overloading set operators isn't supported, so code such as:

```
if Msg.WParam in [1..5] then
```

must be replaced with one of the following:

```
if Msg.WParam.ToInt64 in [1..5] then // VCL.NET specific
if Int64(Msg.WParam) in [1..5] then // Compatible with VCL/Win32
```

4. **Indexing into arrays** Using a handle to index into an array no longer compiles. Code such as this:

```
S := StrArray[Msg.WParam];
```

must be changed to one of the following:

```
S := StrArray[Msg.WParam.ToInt64]; // VCL.NET specific
S := StrArray[Int64(Msg.WParam)]; // Compatible with VCL/Win32
```

5. **Assuming a specific handle size** Code that is expected to run on 64-bit platforms should not assume the value of a handle

or an `IntPtr` is in the `Int32` range. In the case of an `IntPtr`, the `ToInt32` method throws an overflow exception if it doesn't. Code such as this:

```
LIntPtr := IntPtr.Create(Buffer.ToInt32 + 10);
```

needs to be changed to this form:

```
LIntPtr := IntPtr.Create(Buffer.ToInt64 + 10);
```

Similarly, the following code:

```
var
    DC: HDC;
begin
    ...
    SendMessage(Window, Message, Integer(DC), 0);
end;
```

must be replaced by:

```
var
    DC: HDC;
begin
    ...
    SendMessage(Window, Message, WPARAM(DC), 0);
end;
```

### To make required changes in various Windows callbacks

1. Examine the first column of the table below to determine if you have used any of these units.
2. If you have used any unit, determine if you have used any of the callbacks for that unit listed in the second column of the table.
3. For each occurrence of such an callback, change the parameters noted in column 3 appropriately.

Unit	Callback	Details
DDEml	TFNCallback	Data1 and Data2 param types changed
MMSysm	TFNDriverProc	dwDriverId param type changed
MMSysm	TFNDrvCallBacK	dwUser, dw1 and dw2 param types changed
MMSysm	TFNTimeCallBacK	dwUser, dw1 and dw2 param types changed
RichEdit	TEditStreamCallBacK	dwCookie param type changed
Windows	TFNWndProc	WParam, LParam and Result types changed
Windows	TFNWndEnumProc	LParam param type changed
Windows	TFNTimerProc	P3 param type changed
Windows	TFNAPCProc	dwParam param type changed
Windows	TFNDlgProc	Result type changed
Windows	TFNSendAsyncProc	P3 param type changed
Windows	TFNPropEnumProcEx	P4 param type changed
WinInet	PFN_AUTH_NOTIFY	dwContext param type changed

### To make required changes in various Variant records

1. Examine the first column of the table below to determine if you have used any of these units.
2. If you have used any unit, determine if you have used any of the records for that unit listed in the second column of the table.
3. For each such record, find all places where a field was passed as a `var` parameter that has been changed to a property and

modify the code appropriately. The compiler warns you whether you need to update or not.

Unit	Record
ActiveX	TPictDesc
ActiveX	TPropSpec
ActiveX	TStgMedium
ActiveX	TTypeDesc
ActiveX	TVarDesc
ActiveX	TVariantArg
CommCtrl	TPropSheetHeader
CommCtrl	TPropSheetPage
CommCtrl	TTVInsertStruct
CommCtrl	TTVInsertStructA
CommCtrl	TTVInsertStructW
MMSysThemed	TMixerControlDetails
ShlObj	TStrRet
Windows	TInput
Windows	TProcessHeapEntry
Windows	TSystemInfo
WinSpool	TPrinterNotifyInfoData

#### See Also

Changes Required Due to 64-bit .NET 2.0 Support (see page 58)

## 2.4.10 Creating a New VCL.NET Component

You can use the New VCL Component wizard to create a new VCL.NET component and add it to the **Tool Palette**.

#### General procedure for creating a new VCL.NET component

1. Create a package ( [.dll](#) ).
2. Create your new .NET VCL component:
  1. Specify the ancestor component.
  2. Specify the class name.
3. Register your component and add it to the VCL.

#### To create a package

1. Choose File->New->Other->Delphi for .NET Projects->Package.
2. RAD Studio creates the package and displays it in the **Project Manager**.

### To create your new .NET VCL component

1. Choose **File ► New ► Other ► Delphi for .NET Projects ► New Files ► VCL Component**. This displays the first page of the **New VCL Component** dialog box and loads the list of ancestor components.
2. On the Ancestor Component page of the **New VCL Component** dialog box, select an ancestor component from the list and click **Next**.
3. On the **Component** page of the **New VCL Component** dialog box, review the default values for **Class Name**, **Palette Page**, **Unit Name**, and **Search Path**. You can accept the default values or enter values of your choice.
4. Click **Next**. At this point, you should see that the **Project Manager** now lists your new .NET VCL component inside the package you have created.

### To register your component in the VCL

1. Choose Component->Installed .NET Components.
2. Select the **.NET VCL Components** tab.
3. Click **Add**.
4. In the **Browse to select a VCL .NET Package/Assembly** dialog box, click the package that contains the .NET VCL component you have just created.
5. On the **Tool Palette**, find the unit in which your .NET VCL component was installed. In that unit, you should see your .NET VCL component listed.

### See Also

VCL for .NET Overview (see page 71)

## 2.4.11 Displaying a Bitmap Image in a VCL Forms Application

These procedures load a bitmap image from a file and displays it to a VCL form.

1. Create a VCL form with a button control.
2. Provide a bitmap image.
3. Code the button's onClick event handler to load and display a bitmap image.
4. Build and run the application.

### To create a VCL form and button

1. Choose **File ► New ► Other ► Delphi for .NET Projects ► VCL Forms Application**. The **VCL Forms Designer** displays.
2. From the **Standard** category in the **Tool Palette**, place a TButton component on the form.

### To provide a bitmap image

1. Create a directory in which to store your project files.
2. Locate a bitmap image and copy it to your project directory.
3. Save all files in your project to your project directory.

### To write the OnClick event handler

1. In the **Input** category of the **Events** tab, double-click the **Button1OnClick** event. The **Code Editor** displays with the cursor in the `TForm1.Button1Click` event handler block.

2. Enter the following event handling code, replacing `MyFile.bmp` with the name of the bitmap image in your project directory:

```
Rect := TRect.Create(0,0,100,100);
Bitmap := TBitmap.Create;
try
    Bitmap.LoadFromFile('MyFile.bmp');
    Form1.Canvas.Brush.Bitmap := Bitmap;
    Form1.Canvas.FillRect(Rect);
finally
    Form1.Canvas.Brush.Bitmap := nil;
    Bitmap.Free;
end;
```

**Tip:** You can change the size of the rectangle to be displayed by adjusting the `Rect` parameter values.

3. In the var section of the code, add these variable declarations:

```
Bitmap : TBitmap;
Rect : TRect;
```

#### To run the program

1. Save all files in your project.
2. Choose **Run ▶ Run**.
3. Click the button to display the image bitmap in a 100 x 100-pixel rectangle in the upper left corner of the form.

#### See Also

VCL.NET Overview (🔗 see page 71)

Drawing Straight Lines In a VCL Application (🔗 see page 163)

Drawing Rectangles and Ellipses in a VCL Application (🔗 see page 161)

Drawing a Rounded Rectangle in a VCL Application (🔗 see page 162)

Placing A Bitmap Image in a Control in a VCL Forms Application (🔗 see page 164)

## 2.4.12 Drawing Rectangles and Ellipses in a VCL Forms Application

These procedures draw a rectangle and ellipse in a VCL form.

1. Create a VCL form.
2. Code the form's `OnPaint` event handler to draw a rectangle and ellipse.
3. Build and run the application.

#### To create a VCL form

1. Choose **File ▶ New ▶ Other ▶ Delphi for .NET Projects ▶ VCL Forms Application**.
2. In the **Object Inspector**, click the **Design** tab, if necessary, to display `Form1`.

#### To write the `OnPaint` event handler

1. In the **Object Inspector**, click the **Events** tab.
2. Double-click the `OnPaint` event in the **Miscellaneous** category on the **Events** tab. The **Code Editor** displays with the cursor in the `TForm1.FormPaint` event handler block.
3. Enter the following event handling code:

```
Canvas.Rectangle (0, 0, ClientWidth div 2, ClientHeight div 2);  
Canvas.Ellipse (0, 0, ClientWidth div 2, ClientHeight div 2);
```

**To run the program**

1. Save all files in your project.
2. Choose **Run ▶ Run**.
3. The application executes, displaying a rectangle in the upper left quadrant, with an ellipse in the middle of the rectangle.

**See Also**

VCL.NET Overview (🔗 see page 71)

Drawing a Rounded Rectangle in a VCL Forms Application (🔗 see page 162)

---

## 2.4.13 Drawing a Rounded Rectangle in a VCL Forms Application

These procedures draw a rounded rectangle in a VCL form.

1. Create a VCL form.
2. Code the form's OnPaint event handler to draw a polygon.
3. Build and run the application.

**To create a VCL form**

1. Choose **File ▶ New ▶ Other ▶ Delphi for .NET Projects ▶ VCL Forms Application**.
2. In the **Designer**, click the form, if necessary, to display **Form1** properties in the **Object Inspector**.

**To write the OnPaint event handler**

1. In the **Object Inspector**, click the **Events** tab.
2. Double-click the OnPaint event handler in the **Visual** category. The **Code Editor** displays with the cursor in the `TForm1.FormPaint` event handler block.
3. Enter the following event handling code:

```
Canvas.RoundRect(0, 0, ClientWidth div 2,  
ClientHeight div 2, 10, 10);
```

**To run the program**

1. Save all files in your project.
2. Select **Run ▶ Run**.
3. The application executes, displaying a rounded rectangle in the upper left quadrant of the form.

**See Also**

VCL.NET Overview (🔗 see page 71)

Drawing Rectangles and Ellipses in a VCL Forms Application (🔗 see page 161)

## 2.4.14 Drawing Straight Lines In a VCL Forms Application

These procedures draw two diagonal straight lines on an image in a VCL form.

1. Create a VCL form.
2. Code the form's `OnPaint` event handler to draw the straight lines.
3. Build and run the application.

### To create a VCL form and place an image on it

1. Choose **File ► New ► Other ► Delphi for .NET Projects ► VCL Forms Application**.
2. Click the **Design** tab, if necessary, to display `Form1` properties in the **Object Inspector**.

### To write the `OnPaint` event handler

1. In the **Object Inspector**, click the **Events** tab.
2. In the **Visual** category, double-click the `OnPaint` event. The **Code Editor** displays with the cursor in the `TForm1.FormPaint` event handler block.
3. Enter the following event handling code:

```
with Canvas do
begin
    MoveTo(0,0);
    LineTo(ClientWidth, ClientHeight);
    MoveTo(0, ClientHeight);
    LineTo(ClientWidth, 0);
end;
```

### To run the program

1. Save all files in your project.
2. Choose **Run ► Run**. The application executes, displaying a form with two diagonal crossing lines.

**Tip:** To change the color of the pen to green, insert this statement following the first `MoveTo( )` statement in the event handler code: `Pen.Color := clRed;` Experiment using other canvas and pen object properties.

### See Also

VCL.NET Overview (🔗 see page 71)

Building VCL Forms Applications With Graphics (🔗 see page 147)

Drawing Rectangles and Ellipses in a VCL Forms Application (🔗 see page 161)

Drawing a Rounded Rectangle in a VCL Forms Application (🔗 see page 162)

Displaying a Bitmap Image in a VCL Forms Application (🔗 see page 160)

## 2.4.15 Placing a Bitmap Image in a Control in a VCL Forms Application

These procedures add a bitmap image to a combo box in a VCL forms application.

1. Create a VCL form.
2. Place components on the form.
3. Set component properties in the Object Inspector.
4. Write event handlers for the component's drawing action.
5. Build and run the application.

#### To create a VCL form with a ComboBox component

1. Choose **File ► New ► Other ► Delphi for .NET Projects ► VCL Forms Application**. The **VCL Forms Designer** displays.
2. Click the **Design** tab to display the form.
3. From the **Win32** category of the **Tool Palette**, place an **TImageList** component on the form.
4. From the **Standard** category of the **Tool Palette**, place a **TComboBox** component on the form.

#### To set the component properties

1. Select **ComboBox1** on the form.
2. In the **Object Inspector**, set the **Style** property drop-down to **csOwnerDrawFixed**.
3. In the **Object Inspector**, click the [...] next to the **Items** property. The **String List Editor** displays.
4. Enter a string you would like to associate with the bitmap image, for example, **MyImage** and then click **OK**.
5. Double-click **ImageList1** in the form. The **ImageList Editor** displays.
6. Click the **Add** button to display the **Add Images** dialog.
7. Browse your local drive to locate a bitmap image to display in the combobox.
8. Select a very small image such as an icon. Copy it to your project directory, and click **Open**. The image displays in the **ImageList Editor**.
9. Click **OK** to close the editor.

#### To add the event handler code

1. In the **Designer**, select **ComboBox1**.
2. In the **Object Inspector**, click the **Events** tab.
3. Double-click the **OnDrawItem** event. The **Code Editor** displays with cursor in the code block of the **DrawItem** event handler.
4. Enter the following code for the event handler:

```
ComboBox1.Canvas.FillRect(rect);  
ImageList1.Draw(ComboBox1.Canvas, Rect.Left, Rect.Top, Index);  
ComboBox1.Canvas.TextOut(Rect.Left+ImageList1.Width+2,  
    Rect.Top, ComboBox1.Items[Index]);
```

#### To run the program

1. Save all files in your project.
2. Choose **Run ► Run**. The application executes, displaying a form with a combo box.
3. Click the combobox drop-down. The bitmap image and the text string display as a list item.

#### See Also

VCL.NET Overview (🔗 see page 71)

Building VCL Forms Applications With Graphics (🔗 see page 147)



## 2.4.16 Importing .NET Controls to VCL.NET

You might want to use .NET components on your VCL.NET forms. There is no direct way to use .NET components. You can, however, wrap the components in an ActiveX wrapper, which then can be added to your VCL.NET application. RAD Studio provides the **.NET Import Wizard** to accomplish this task.

### To use .NET components in a VCL.NET Form

1. Run the WinForm Control Import Wizard.
2. Build the package to create an assembly file.
3. Add the assembly to the **Tool Palette**.

### To run the WinForm Control Import Wizard

1. Choose **File ► New ► Other ► Delphi for .NET Projects ► WinForm Controls Package**. This starts the **WinForm Control Import Wizard**.
2. Specify the following file:  
`c:\Windows\Microsoft.NET\Framework\v1.1.4322\System.Windows.Forms.dll`
3. Click **Next**. This displays the second page of the **WinForm Control Import Wizard** and lists all of the available components.
4. Check the check boxes next to the components you want to import.  
**Note:** If you want to import all components, click the Check All button.
5. Click **Next**. This displays the third page of the Wizard, which provides generation options for the units.
6. Accept the defaults, and click **Next**. This displays the fourth page of the Wizard, which allows you to set a location and a name for the package file.
7. Click **Next**. This displays the fifth page of the Wizard, which allows you to overwrite any existing files of the same name.
8. Click **Next**. This initiates the generation process and displays status messages for each file as it is created, including the package (.dpk) file.
9. If you want to import additional controls, click **New**. Otherwise, click **Finish**. The package containing the units appears in the **Project Manager**.

### To build and add the package

1. Select the package node in the **Project Manager**.
2. Choose **Project ► Build <Project Name>** from the main menu where <Project Name> is the name of your project. This creates the assembly file containing the package and the units.
3. Choose **Components ► Installed .NET Components**.
4. Click the **.NET VCL Components** tab.
5. Click **Add**.
6. Locate the package assembly, select it, and click **Open**. The location depends on your project options directory locations. The file might also end up in your default documents directory.
7. Click **OK**. The individual controls appear in the **Tool Palette** under the **WinForm Controls** category. You can now add the individual controls to your VCL.NET form applications.

### See Also

VCL.NET Overview (see page 71)

## 2.5 ASP.NET Procedures

This section provides how-to information on developing ASP.NET Web Forms applications.

### Topics

Name	Description
Building an ASP.NET Application (see page 170)	The following procedures describe the general steps required to build a simple ASP.NET project. For more advanced topics, refer to the related information following the procedure.
Building an ASP.NET Database Application (see page 171)	<p>The following procedure describes the minimum number of steps required to build a simple ASP.NET database application such as BDP.NET. After generating the required connection objects, the project displays data in a DataGrid.</p> <p>For a more detailed database sample using other database technologies for ASP.NET 2.0, see Building an ASP.NET Application with Database Controls 1 (see page 174).</p> <p>BDP.NET includes component designers to facilitate the creation of database applications. Instead of dropping individual components on a designer, configuring each in turn, use BDP.NET designers to rapidly create and configure database components. The following procedure demonstrates the major components of ASP.NET, ADO.NET, and... more (see page 171)</p>
Developing an ASP.NET Application with Database Controls, Part 1 (see page 174)	<p>This extended example shows how to create a Web page that contains a <b>GridView</b> control and a database control. This example illustrates:</p> <ul style="list-style-type: none"> <li>• Adding an ASP.NET component to a form.</li> <li>• Using the <b>Smart Tasks</b> window.</li> <li>• Configuring and connecting to a data source.</li> <li>• Using SQL to read data from a database.</li> <li>• Running the application in a browser.</li> </ul> <p><b>Note:</b> You can only create ASP.NET projects in CodeGear RAD Studio, not Win32 personalities.</p>
Building an ASP.NET Application with Database Controls, Part 2 (see page 176)	<p>This example shows further development of a Web page containing a <b>GridView</b> control and database control. The steps add more capability to the <b>GridView</b> control, including editing. This example illustrates:</p> <ul style="list-style-type: none"> <li>• Using the <b>Smart Tasks</b> window.</li> <li>• Changing the configuration of a data source.</li> <li>• Running the application in a browser.</li> </ul>
Building an ASP.NET Application with Database Controls, Part 3 (see page 177)	<p>This example shows further development of a Web page with a <b>GridView</b> control and database control. It adds a <b>FormView</b> control to insert rows. This example illustrates:</p> <ul style="list-style-type: none"> <li>• Adding an ASP.NET component to a form.</li> <li>• Using the <b>Smart Tasks</b> window.</li> <li>• Editing markup to change a component's appearance.</li> <li>• Running the application in a browser.</li> </ul>
Building an ASP.NET "Hello World" Application (see page 178)	Though simple, the ASP.NET "Hello World" application demonstrates the essential steps for creating an ASP.NET application. The application uses a Web Form, controls, and an event that will display a result in response to a user action.

Building an ASP.NET SiteMap (see page 179)	<p>This example shows how to create a Web site with a SiteMap, <b>Menu</b>, <b>SiteMapPath</b> and master page. A web.sitemap file is an XML file that describes the structure of a web site. A master page serves as a template for pages in the Web site. A <b>Menu</b> and <b>SiteMapPath</b> allow you to navigate a Web site. This example illustrates:</p> <ul style="list-style-type: none"> <li>• Creating a Web sitemap.</li> <li>• Creating a master page.</li> <li>• Adding a <b>Menu</b> and <b>SiteMapPath</b> to a master page.</li> <li>• Using the <b>Smart Tasks</b> window.</li> <li>• Adding content pages referencing the master page to the site.</li> <li>• Running the application in a browser.</li> </ul> <p>The example... more (see page 179)</p>
Creating a Briefcase Application with DB Web Controls (see page 182)	<p>You can use DB Web Controls, XML caching, and the BDP.NET data adapters to create server-side briefcase applications. You can only create this type of application when using user authentication, to guarantee that each user has a unique copy of the XML file.</p> <p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p>
Building an Application with DB Web Controls (see page 183)	<p>The following procedures describe the minimum number of steps required to build a simple ASP.NET database application using DB Web Controls and BDP.NET. After generating the required connection objects, the project displays data in a DBWebGrid with a DBWebNavigator. Additional information is provided for other common DB Web Controls.</p> <p>Users should already be familiar with creating an ASP.NET project using BDP.NET.</p> <p>Building the simple ASP.NET application with DB Web Controls and BDP.NET consists of three major steps:</p> <ol style="list-style-type: none"> <li>1. Prepare an ASP.NET project with BDP.NET or other connection components.</li> <li>2. Drag and drop a DBWebDataSource onto the Designer and set its DataSource property... more (see page 183)</li> </ol>
Converting HTML Elements to Server Controls (see page 184)	<p>Unlike Web controls, HTML elements can not, by default, be controlled programmatically. However, you can convert an HTML element to a server control and then write code to access or modify the element. Most of the HTML elements that appear in the <b>Tool Palette</b> can be converted by using the Run As Server Control command. HTML elements that do not appear on the <b>Tool Palette</b>, such as <code>body</code>, can be converted manually.</p> <p>The following procedures explain how to convert an HTML <code>table</code> element by using the Run As Server Control command, and how to convert a <code>body</code> element... more (see page 184)</p>
Creating an XML File for DB Web Controls (see page 185)	<p>You can use XML files as your data source, particularly if you want to prototype applications without reading from and writing to a database. First you must create the XML file. The DBWebDataSource control provides a powerful way to create the XML file based on real database data. This procedure assumes that you can create a connection to a live database containing the data you want to use.</p> <p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p>
Creating Metadata for a DataSet (see page 187)	<p>When you choose to use an XML file for a data source in an ASP.NET application using DB Web Controls, you may need to create the metadata to structure the XML data in your DataSet. If you chose to create an XML file without an XML schema file (.xsd), you need to manually create the metadata. This procedure assumes that you have already created an XML file containing data.</p> <p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p>

Creating a Virtual Directory (see page 188)	<p>When you create an ASP.NET application, the IDE automatically creates a virtual directory for you based on the settings in the <b>New ASP.NET Web Application</b> dialog box.</p> <p>However, the IDE can also create a virtual directory for an application that you did not create within the IDE, such as the demo applications found in the <b>DBWeb</b> folder.</p> <p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p>
Adding Aggregate Values with DBWebAggregateControl (see page 188)	<p>You can use DBWebAggregateControl to apply one of several standard aggregation functions to a data column. The control displays the aggregate value in a text box, which also support a linked caption.</p> <p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p>
Debugging and Updating ASP.NET Applications (see page 189)	<p>During the installation of RAD Studio, the install program requested permission to update the <b>machine.config</b> file on your computer. This information is necessary for debugging RAD Studio applications under IIS. If you replied <b>Yes</b> to that prompt, CodeGear debugger information was written to <b>machine.config</b> and will be available to the applications that you created with Delphi 8. <b>You need not perform this procedure.</b></p> <p>If you replied <b>No</b> to that prompt, the debugger information is written to the application <b>web.config</b> file when you create an ASP.NET application with RAD Studio. However, you will need to add this information manually to <b>web.config</b>... more (see page 189)</p>
Deploying an ASP.NET Application using Blackfish SQL to a system without RAD Studio (see page 190)	You can deploy an ASP.NET application using Blackfish SQL to a system without RAD Studio.
Generating HTTP Messages in ASP.NET (see page 191)	<p>When attempting to debug your ASP.NET applications, you may find that the error messages are cryptic or even meaningless. This may be the result of having a specific option set in your Internet Explorer browser. To assist your debugging efforts, you should change this option.</p>
Binding Columns in the DBWebGrid (see page 191)	<p>There may be times when you want to modify the order in which columns appear in a DBWebGrid control. You can accomplish this task by binding columns manually, from within the <b>Property Builder</b>.</p> <p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p>
Setting Permissions for XML File Use (see page 192)	<p>You need to grant rights to clients who will be using your ASP.NET applications, if you want to avoid a permissions error when using an XML file as a data source. There are two ways to do this, as described in the following procedures.</p> <p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p>
Troubleshooting ASP.NET Applications (see page 193)	<p>Unlike traditional window-based applications, web applications are dependent on servers and resources that are not directly within the control of the application or the user. Web applications are often hybrid combinations of client, server, and network resources.</p> <p>The areas you need to check include ASP.NET installation, IIS installation and configuration, and security. All three of these areas are extensive and complex. The following procedures provide solutions to some of the most common problems.</p> <p><b>Note:</b> The following suggestions apply only to IIS 5.1.</p>
Using the DB Web Control Wizard (see page 195)	<p>The <b>DB Web Control Wizard</b> helps you create a data-aware web control based on a standard web control.</p> <p><b>Note:</b> DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.</p>
Using the ASP.NET Deployment Manager (see page 196)	<p>You can add an ASP.NET Deployment Manager to an ASP.NET application project to assist you with deploying the application. The Deployment Manager determines which files are required for deployment, requests the destination directory name and connection information, and then copies the files to the destination directory. The Deployment Manager generates a list of files to copy based on the names of the files in your project directory, but you can include or exclude files as needed.</p> <p>You can use the right mouse button, when the Deployment Manager window is displayed, to see options for displaying, copying, deleting, modifying, and filtering... more (see page 196)</p>

Using the HTML Tag Editor (📖 see page 199)	When you are creating or editing an HTML file, you can use the <b>Tag Editor</b> window, beneath the Form Designer, to edit the HTML tags. If you are using an ASP.NET Web Form, the Tag Editor is not supported. If you are using an HTML Form, you can display the Tag Editor in the Designer by selecting <b>View ► Tag Editor</b> .  The <b>Tag Editor</b> lets you review and modify HTML tags while viewing the corresponding controls in the Designer window, above it. The <b>Tag Editor</b> allows you to use the <b>Code Completion</b> , <b>Error Insight</b> , and <b>Live Template Completion... more</b> (📖 see page 199)
Working with ASP.NET User Controls (📖 see page 200)	User controls provide a way to reuse common user interface functionality across ASP.NET web applications. For example, you might create user control that encapsulates a login screen. You could then add the user control to any Web Form that requires the login screen functionality. For more information about user controls, click the link at the end of this topic.

## 2.5.1 Building an ASP.NET Application

The following procedures describe the general steps required to build a simple ASP.NET project. For more advanced topics, refer to the related information following the procedure.

### To create an ASP.NET project

1. Create a new ASP.NET project by clicking **File ► New ► ASP.NET Web Application**. The **New ASP.NET Web Application** dialog box appears.
2. In the **Name** field, enter the name of your project.
3. In the **Location** field, accept the default path or enter another project path.

### To change Web server settings (optional)

1. In the **New ASP.NET Web Application** dialog box, click **View Server Options**. The dialog expands to show additional server options.
2. Set the various read and write attributes of the project as needed or accept the defaults.

**Tip:** In most cases, the default settings will suffice.

3. Click **OK**. The Web Forms Designer appears.

### To create an ASP.NET page

1. Make sure the Designer is displayed.
2. From the **Tool Palette**, drag components onto the Designer to define the user interface.
3. Add code-behind logic to components.

### To add code-behind logic to a component

1. In the Designer, double-click the component to which you wish to apply logic. The code-behind Designer appears, cursor in place between event handler brackets.
2. Add your logic.

### To view and edit markup

1. Click the **ApplicationName.aspx** tab on the **Code Editor** to view the markup, which shows the HTML and ASP.NET component information.
2. Edit the markup.
3. Save the file.

### To run the application

1. Run the application by clicking **Run ▶ Run**.
2. Alternatively, right-click the **Code Editor** and select **View In Browser** on the context menu.
3. The application saves and compiles. Once you compile the application, the generated `.aspx` file displays HTML in the default web browser.

### See Also

ASP.NET Overview (see page 79)

Building an ASP.NET "Hello World" Application (see page 178)

Building an ASP.NET Application with Database Controls 1 (see page 174)

Building an ASP.NET Application with Database Controls 2 (see page 176)

Building an ASP.NET Application with Database Controls 3 (see page 177)

Building an ASP.NET SiteMap (see page 179)

---

## 2.5.2 Building an ASP.NET Database Application

The following procedure describes the minimum number of steps required to build a simple ASP.NET database application such as BDP.NET. After generating the required connection objects, the project displays data in a DataGrid.

For a more detailed database sample using other database technologies for ASP.NET 2.0, see Building an ASP.NET Application with Database Controls 1 (see page 174).

BDP.NET includes component designers to facilitate the creation of database applications. Instead of dropping individual components on a designer, configuring each in turn, use BDP.NET designers to rapidly create and configure database components. The following procedure demonstrates the major components of ASP.NET, ADO.NET, and BDP.NET at work.

Building an ASP.NET application with database components consists of four major steps:

1. Create an ASP.NET project.
2. Configure database connection components and a data source.
3. Add a DataBind call.
4. Connect a DataGrid to the connection components.

**Tip:** For testing purposes, use the employee.gdb database included with Interbase, if included with your version of the product.

### To create an ASP.NET project

1. Choose **File ▶ New ▶ ASP.NET Web Application**. The **New ASP.NET Web Application** dialog appears.
2. In the **Name** field, enter the name of your project.
3. In the **Location** field, use the default or enter the project path.

### To change Web server settings (optional)

1. In the **New ASP.NET Web Application** dialog, click **View Server Options**. The dialog expands to show additional server options.
2. Set the various read and write attributes of the project as needed or accept the defaults.

**Tip:** In most cases, the default settings will suffice.

3. Click **OK**. The Web Forms Designer appears.

#### To configure data components

1. Drag and drop a BdpDataAdapter component onto the Designer. If necessary, select BdpDataAdapter.
2. In **Object Inspector**, select **Configure Data Adapter**. The **Data Adapter Configuration** dialog appears.
3. If necessary, select the **Command** tab. From the **Connection** drop-down, select **New Connection**.
4. The **Borland Data Provider: Connections Editor** dialog appears.

**Tip:** Alternatively, use Data Explorer to drag and drop a table on to the Designer surface. Data Explorer sets the connection string automatically.

#### To set up a connection

1. In **Borland Data Provider: Connections Editor**, select the appropriate item from the **Connections** list.
2. In **Connection Settings**, enter the **Database** path.

**Note:** If referring to a database on the local disk, prepend the path with localhost:

. If using Interbase, for example, you would enter the path to your Interbase database: `localhost:C:\Program Files\Borland\Interbase\Examples\Database\employee.gdb` (or whatever the actual path might be for your system).

3. Complete the **UserName** and **Password** fields for the database as needed.
4. Click **Test** to confirm the connection. A dialog appears confirming the status of the connection.
5. Click **OK** to return to the **Borland Data Provider: Connections Editor** dialog.
6. Click **OK** to return to the **Data Adapter Configuration** dialog. In the **Command** tab, the areas for **Tables** and **Columns** are updated with information from your connection.

#### To set a command

1. In the **Select** area, enter an SQL command.

**Tip:** For Interbase's employee.gdb database, you might enter `select * from SALES`, as an example.

2. Click the **Preview Data** tab.
3. Click **Refresh**. Column and row data appear.
4. Click the **DataSet** tab.
5. Confirm that **New DataSet** is selected.
6. Click **OK**. New components for DataSet and BdpConnection appear on the Designer.
7. Select BdpDataAdapter component.
8. In **Object Inspector**, select the Active property drop-down and set the value to **True**.

#### To connect a DataGrid to a DataSet

1. Drag and drop a DataGrid web control onto the Designer. If necessary, select DataGrid.
2. In **Object Inspector**, select the DataSource property drop-down. Select the DataSet component that you generated previously (the default is DataSet1).
3. In **Object Inspector**, select the DataMember property drop-down. Select the appropriate table. The DataGrid displays data from the DataSet.

**To add a DataBind call**

1. Use the **Object Inspector** drop-down to select the Web Form (**WebForm1** is the default).
2. In **Object Inspector**, select the **Events** tab.
3. Set the Load event to **Page\_Load**.
4. In **Object Inspector**, double-click **Page\_Load**. The code-behind Designer appears, cursor in place between event handler brackets.
5. Code the DataBind call:

```
this.dataGrid1.DataBind();  
Self.dataGrid1.DataBind();
```

**Note:** If you are using data aware controls, for instance from a third-party provider, you may not need to code the DataBind call.

6. Choose **Run ▶ Run**. The application compiles and the HTTP server displays a Web Form with the datagrid.

While presenting a minimum number of steps required to build a database project, the preceding procedure demonstrates the major components of the ASP.NET, ADO.NET, and BDP.NET architectures at work, including: providers, datasets, and adapters. The adapter connects to the physical data source via a provider, sending a command that will read data from the data source and populate a dataset. Once populated, a datagrid displays data from the dataset.

Once created, use other BDP.NET designers to modify and maintain the components of your project.

**See Also**

ASP.NET Overview (🔗 see page 79)

ADO.NET Overview (🔗 see page 14)

Data Providers for .NET (🔗 see page 27)

ADO.NET Component Designers (🔗 see page 21)

Creating Database Projects from the Data Explorer (🔗 see page 111)

Building an ASP.NET Application with Database Controls 1 (🔗 see page 174)

Building an ASP.NET Application with Database Controls 2 (🔗 see page 176)

Building an ASP.NET Application with Database Controls 3 (🔗 see page 177)

---

## 2.5.3 Developing an ASP.NET Application with Database Controls, Part 1

This extended example shows how to create a Web page that contains a **GridView** control and a database control. This example illustrates:

- Adding an ASP.NET component to a form.
- Using the **Smart Tasks** window.
- Configuring and connecting to a data source.
- Using SQL to read data from a database.
- Running the application in a browser.

**Note:** You can only create ASP.NET projects in CodeGear RAD Studio, not Win32 personalities.



**To create a page containing a GridView control and a database control**

1. Create a new ASP.NET project by clicking **File ► New ► ASP.NET Web Application**.
2. In the **New ASP.NET Web Application** dialog, name your project:
  - In the **Location** field, accept the default path or enter another project path.
  - Choose either Cassini or IIS as your server in the **Server** drop down menu. Click **OK**.
3. (Optional) In the **New ASP.NET Web Application** dialog box, click **View Server Options**. The dialog expands to show additional server options. Set the various read and write attributes of the project as needed or accept the defaults.  
**Tip:** In most cases, the default settings suffice.
4. Make sure that the **Design** tab is selected in the **Code Editor**. In the **Tool Palette** under **Data Web 2.0**, double-click the **GridView** component to add the component to your application.
5. Display the **GridView** component **Smart Tasks** window either by clicking the arrow on the upper right corner of the **GridView** or by clicking **Smart Tasks** on the **GridView** context menu.
6. To connect the **GridView** to a data source, click **<New data source>** in the **Choose Data Source** drop down list. This displays the **Choose a Data Source Type** dialog, part of the **Data Source Configuration Wizard**.
7. In the **Choose a Data Source Type** dialog, select **Database**. Enter an ID for the data source and click **OK** to display the **Choose Your Data Connection** dialog.
8. On the **Choose Your Data Connection** dialog, click **New Connection**.
9. The **Choose Data Source** dialog lists several data sources, determined by your configuration. The list might include:
  - AdoDbx: AdoDbx Client
  - Microsoft SQL Server: Microsoft SQL Server
  - Blackfish SQL: Blackfish SQL Server Select the **Data source** and the **Data provider** for that data source from the lists. Then click **Continue**.
10. On the **Add Connection** dialog, enter the appropriate connection information.
  - Click **Advanced** to enter more detailed connection information for your data source, if necessary.
  - Click the **Test Connection** button to test the connection. Click **OK** to close the dialog.
11. On the **Choose Your Data Connection** dialog, click **Next** to continue with data source configuration.
12. On the **Save the Connection String to the Application Configuration File** dialog, enter a name and check **Yes, save this connection as**. This makes the data source's **ConnectionString** property reference the name you entered rather than the actual connection string. The web.config file in this project contains the connection string based on the information you have entered. Click **Next**.
13. On the **Configure the Select Statement** dialog, set the radio button to either specify a custom SQL Select statement or generate a SQL Select statement for the database you are connected to. If generating SQL, select the table in the **Name** drop down list, check the desired **Columns**, and optionally click the **WHERE**, **ORDER BY** or **Advanced** buttons to qualify the SQL statement. The **SELECT statement** field shows the generated SQL statement. If directly specifying the SQL, click **Next** to display the **Define Custom Statements or Stored Procedures** dialog and complete the fields. Optionally, click **Query Builder** to display a dialog that allows you to graphically create a SELECT statement. You can create SQL statements for SELECT, UPDATE, INSERT and DELETE statements by clicking the corresponding tabs. Click **Next** in either dialog to continue with data source configuration.  
**Note:** If you directly specify SQL, you cannot use the feature to automatically generate INSERT, UPDATE or DELETE statements.  
**Note:** Query Builder  
can only build SELECT statements.
14. (Optional) On the **Test Query** dialog, click **Test Query** to test the SQL query. If the query succeeds, the appropriate data is

displayed in the dialog. Click **Finish** to complete configuring the data source.

15. (Optional) If you select the **SQLDataSource** component on the designer tab, note that the **ConnectionString** property contains the connection string information. Connection string data is also stored in the web.config file in the **connectionStrings** element.
16. (Optional) Click the **WebForm.aspx** tab on the **Code Editor** to view the markup, which shows the HTML and ASP.NET component information. You can see that the **GridView Columns** property contains fields bound to the table column names of your database.
17. To view the page in a Web browser, right-click the **Code Editor** and select **View In Browser** on the context menu. The project builds, displaying a build status window, which is closed upon completing the build. The browser displays a table for the database you're working with.

This completes Part 1 of the example.

#### See Also

ASP.NET Overview (🔗 see page 79)

Building an ASP.NET Application with Database Controls (🔗 see page 176)

Building an ASP.NET Application with Database Controls (🔗 see page 177)

Building an ASP.NET SiteMap (🔗 see page 179)

---

## 2.5.4 Building an ASP.NET Application with Database Controls, Part 2

This example shows further development of a Web page containing a **GridView** control and database control. The steps add more capability to the **GridView** control, including editing. This example illustrates:

- Using the **Smart Tasks** window.
- Changing the configuration of a data source.
- Running the application in a browser.

#### To add editing to the GridView control

1. Open or resume development with the project described in Building an ASP.NET Database Application 1 (🔗 see page 174). At the end of that phase, the **Design** pane showed a **GridView** control and a **SqlDataSource** control in the **Design** pane of the **Code Editor**.
2. To enable **GridView** display features, open the **GridView** component's **Smart Tasks** window by clicking on the arrow near the upper right corner of the **GridView** control. Check **Enable Sorting**, **Enable Selection** and **Enable Paging**. The design time representation of the **GridView** gets updated.
3. To view the page in a Web browser, right-click the **Code Editor** and click **View In Browser** on the context menu. Paging, sorting and selecting are now supported by this grid. You can sort each column by clicking the column name at the top of the column. Select doesn't appear to work, because the format of the **GridView** doesn't highlight selected rows. Paging is not enabled if the data is less than one page.
4. Return to the **Designer** pane of the **Code Editor**. Open the **GridView Smart Tasks** window and click **Auto Format** to display the **Auto Format** dialog. Choose **Classic**. The webform1.aspx file's description of the **asp:GridView** component now has style properties that were modified by selecting **Classic** on the Auto Format dialog.
5. At this point, row selection is enabled. To enable row editing, you may need to reconfigure the connection. On the **GridView Smart Tasks** window, click **Configure Data Source**. Click **Next** in the **Choose Your Data Connection** dialog. Click the **Advanced** button in the **Configure the Select Statement** dialog. On the **Advanced SQL Generation Options** dialog, check **Generate INSERT, UPDATE, and DELETE statements**. Click **OK** to close the **Advanced SQL Generation Options** dialog.

**Note:** You cannot use the Advanced SQL Generation Options

dialog if you chose to generate your own SQL statements in the **Configure the Select Statement** dialog.

6. Click **Next** then **Finish** in the **Configure the Select Statement** dialog to exit the wizard.
7. The **GridView Smart Tasks** window now contains **Enable Editing** and **Enable Deleting** check boxes. Check both of these. The **GridView** control on the **Design** pane now shows **Edit** and **Delete** links.
8. To view the page in a Web browser, right-click the **Code Editor** and click **View In Browser** on the context menu. The **Edit**, **Delete** and **Select** links should work for each row.

This completes Part 2 of the example.

#### See Also

ASP.NET Overview (🔗 see page 79)

Building an ASP.NET Application with Database Controls (🔗 see page 174)

Building an ASP.NET Application with Database Controls (🔗 see page 177)

Building an ASP.NET SiteMap (🔗 see page 179)

---

## 2.5.5 Building an ASP.NET Application with Database Controls, Part 3

This example shows further development of a Web page with a **GridView** control and database control. It adds a **FormView** control to insert rows. This example illustrates:

- Adding an ASP.NET component to a form.
- Using the **Smart Tasks** window.
- Editing markup to change a component's appearance.
- Running the application in a browser.

#### To add a FormView control

1. Open or resume development with the project described in Building an ASP.NET Database Application 2 (🔗 see page 176). At the end of that phase, the **Design** pane showed a **GridView** control and a **SqlDataSource** control in the **Designer** pane of the **Code Editor**.
2. Make sure that the **Design** tab is selected in the **Code Editor**. Add a new line to the end of the Web design by pressing the **Ctrl-End** keys then the **Enter** key. In the **Tool Palette** under **Web Data**, double-click a **FormView** component, which is added to the form.
3. Display the **Smart Tasks** window for the **FormView** component by clicking the tab on the upper right of the component. In the drop down menu for **Choose Data Source**, select the data source that you previously added and configured. On the **Design** tab, the **FormView** component shows the data fields.
4. Click **Auto Format** on the **Smart Tasks** window for the **FormView**. On the **Auto Format** dialog, select **Slate** and click **OK**.
5. In the **Design** tab, select the **FormView** component. In the **Object Inspector**, change the value of the **DefaultMode** property to **Insert**. This change allows you to use the **FormView** control to add rows to the database.
6. The **FormView** control's initial appearance has labels and fields on the same line. The **FormView** control's appearance is defined by the markup in the **InsertItemTemplate** entry in the markup file Default.aspx. Display the text in Default.aspx by clicking the **Default.aspx** tab at the bottom of the **Default.aspx** pane. To put the labels and fields on separate lines, edit Default.aspx, and add **<br/>** tags immediately after the column labels. The **FormView** control now has labels and fields on separate lines.

7. View the page in a Web browser by right-clicking the **Code Editor** and selecting **View In Browser** from the context menu. The form is now displayed with the **FormView** control for inserting rows.

This completes the example.

#### See Also

- ASP.NET Overview (🔗 see page 79)
- Building an ASP.NET Application with Database Controls (🔗 see page 174)
- Building an ASP.NET Application with Database Controls (🔗 see page 176)
- Building an ASP.NET SiteMap (🔗 see page 179)

---

## 2.5.6 Building an ASP.NET "Hello World" Application

Though simple, the ASP.NET "Hello World" application demonstrates the essential steps for creating an ASP.NET application. The application uses a Web Form, controls, and an event that will display a result in response to a user action.

#### To create an ASP.NET project

1. Create a new ASP.NET project by clicking **File ► New ► ASP.NET Web Application**. The **New ASP.NET Web Application** dialog box appears.
2. In the **Name** field, enter `HelloWorld` for the application name.
3. In the **Location** field, accept the default or enter another path.

#### To change Web server settings (optional)

1. In the **New ASP.NET Web Application** dialog box, click **View Server Options**. The dialog expands to show additional server options.
2. Set the various read and write attributes of the project as needed or accept the defaults.

**Tip:** For most ASP.NET projects, the default settings will suffice.

3. Click **OK**. The Web Forms Designer appears.

#### To create the ASP.NET page

1. If necessary, click **Design** view.
2. From the **Web Controls** category of the **Tool Palette**, drag a Button component onto the Designer surface. The Button control appears on the Designer. Make sure the control is selected.
3. In **Object Inspector**, set the **Text** property to `Hello, world!`.

#### To associate code with the button control

1. In the Designer, double-click the Button control. The code-behind Designer appears, cursor in place between event handler brackets.
2. Code the application logic:

```
button1.Text = "Hello, developer!";  
button1.Text := 'Hello, developer!';
```
3. Choose **File ► Save** to save the application.

**To run the "Hello World" application**

1. Choose **Run ▶ Run**. The application compiles and the HTTP server displays a Web Form in your default browser with the "Hello, world!" button.
2. Click the "Hello, world!" button. The server updates the page with the response, "Hello, developer!".
3. Close the Web browser to return to the IDE.

**See Also**

ASP.NET Overview (🔗 see page 79)

Building an ASP.NET application (🔗 see page 170)

Building an ASP.NET database application (🔗 see page 171)

## 2.5.7 Building an ASP.NET SiteMap

This example shows how to create a Web site with a SiteMap, **Menu**, **SiteMapPath** and master page. A web.sitemap file is an XML file that describes the structure of a web site. A master page serves as a template for pages in the Web site. A **Menu** and **SiteMapPath** allow you to navigate a Web site. This example illustrates:

- Creating a Web sitemap.
- Creating a master page.
- Adding a **Menu** and **SiteMapPath** to a master page.
- Using the **Smart Tasks** window.
- Adding content pages referencing the master page to the site.
- Running the application in a browser.

The example creates the Web site in several stages, each building on the previous ones.

**Note:** You can only create ASP.NET projects in CodeGear RAD Studio, not Win32 personalities.

**To add a SiteMap**

1. Create a new ASP.NET project by clicking **File ▶ New ▶ ASP.NET Web Application**.
2. In the **New ASP.NET Web Application** dialog, name your project or use the default name. In the **Location** field, accept the default path or enter another project path. Choose either Cassini or IIS as your server in the **Server** drop down menu.
3. (Optional) In the **New ASP.NET Web Application** dialog box, click **View Server Options**. The dialog expands to show additional server options. Set the various read and write attributes of the project as needed or accept the defaults.

**Tip:** In most cases, the default settings suffice.

4. Add a new web.sitemap file to the project by clicking **File ▶ New ▶ Other**. In the **New Items** dialog under **New ASP.NET Files**, select **Site Map** and click **OK**. The IDE displays a new tab **Web.sitemap** with sitemap template text.

5. Replace the template text on the **Web.sitemap** tab with this text:

```
<?xml version="1.0" encoding="utf-8" ?>
<sitemap>
  <siteMapNode title="Home" url="~/home.aspx" >
    <siteMapNode title="Book 1" url="~/book1/book1.aspx" >
      <siteMapNode title="Chapter 1" url="~/book1/chapter1/chapter1.aspx" >
        </siteMapNode>
      </siteMapNode>
    </siteMapNode>
  </siteMapNode>
</sitemap>
```

```
</siteMapNode>
</siteMap>
```

This XML defines a hierarchy of pages in the site. Save this change by clicking **File ► Save** or entering **Ctrl-S**.

### To add a master page

1. Add a new master page to the project by clicking **File ► New ► Other**. In the **New Items** dialog under **New ASP.NET Files**, select **ASP.NET Master Page** and click **OK**. On the **New Master Page** dialog, the default name of the master page is **MasterPage1.master**; you can enter a different name, but you should use the **.master** extension. Click **OK**. The IDE displays a new tab **MasterPage1.master** containing a new master page with one content placeholder. The default content of a master page is a single **ContentPlaceHolder** control, which is used to indicate where content can be inserted into content pages that reference this master page. This example uses only one **ContentPlaceHolder**, but you can have more than one **ContentPlaceHolder** on a master page.
2. Click the **MasterPage1.master** tab at the bottom of the master page to view its markup. Before adding the **Menu** and **SiteMapPath** controls to the master page, add some **<div>** elements to control the positioning of these controls. To do this, replace the text in the master page's **body** element with the following text:

```
<body>
  <form runat="server">
    <div id="titlediv" style="WIDTH: 100%">
      <h1>Sample</h1>
    </div>
    <div id="breadcrumbdiv" style="WIDTH: 100%">
    </div>
    <div id="menudiv" style="FLOAT: left; WIDTH: 33%">
    </div>
    <div id="contentdiv" style="FLOAT: left; WIDTH: 66%">
      <asp:contentplaceholder id="ContentPlaceholder1" runat="server">
      </asp:contentplaceholder>
    </div>
  </form>
</body>
```

Save this change by clicking **File ► Save** or entering **Ctrl-S**. The master page now has several divisions that appear as rectangles in the **Design** pane. The content placeholder has also shifted to the right side of the page. Master pages are used to build content pages. Everything that's in a master page appears on a content page based on that master. If the master is changed, that change propagates to the content pages derived from it.

### To add a menu and SiteMapPath to the master page

1. To add a menu, view the **Design** pane for the master page. Drag a **Menu** control from the tool palette under **Navigation** into the **<div>** immediately to the left of the **ContentPlaceHolder**. Position the **Menu** so its top aligns with the top of the **ContentPlaceHolder**.
2. Display the **Menu Smart Tasks** window by either clicking the arrow on the upper right of the **Menu** or clicking **Smart Tasks** on the **Menu** context menu.
3. To connect the **Menu** to a data source, click **<New data source>** in the **Choose Data Source** drop down list in the **Smart Tasks** window. This displays the **Choose a Data Source Type** dialog.
4. On the **Choose a Data Source Type** dialog, select the **Site Map** data source type. Enter an ID for the data source or use the default ID and click **OK** to add the data source.
5. To customize the **Menu** control, click on the **Menu**. In the **Object Inspector**, change the **StaticDisplayLevels** property to 3. This is the number of levels displayed in the **Menu**. Note that the menu displays data from the SiteMap XML file in the **Design** pane.
6. To add a **SiteMapPath**, view the **Design** pane for the master page. Drag a **SiteMapPath** control from the tool palette under **Navigation** into the second **<div>** from the top of the master page, just above the **<div>** containing the **Menu**. A **SiteMapPath** is also referred to as **bread crumbs**, since it shows the path you used to get to a page and allows you to retrace your steps.

The menu and SiteMapPath are used for navigating the site.

### To add a content page

1. Create a new content page by clicking **File ► New ► Other**. In the **New Items** dialog under **New ASP.NET Files**, select **ASP.NET Content Page** and click **OK**. In the **New Content Page** dialog, name the page `Home.aspx`. Select `~/MasterPage1.master` in the **Master page file** drop down menu. In the resulting content page, the gray elements are the ones the content page inherits from the master page: everything except the **ContentPlaceHolder** control. These elements are grayed to indicate that they are read only. Only the **ContentPlaceHolder** control can be modified.
2. On the **Home.aspx** pane, click the **Home.aspx** tab at the bottom. On the first line, change the Title attribute to `Title="Home"`. Save this change by clicking **File ► Save** or entering **Ctrl-S**.
3. The content page (`home.aspx`) has a single Content control that corresponds to the **ContentPlaceHolder** control in the master page (`masterpage1.master`). To customize the content page, on the **Home.aspx** pane, click the **Design** tab at the bottom. Click inside the **ContentPlaceHolder** element and type `Home`.

This content page will serve as the site's home page.

### To create site content

1. To create site content, you need to create content pages and folders to contain them. To create a new folder, right-click the project in the **Project Manager** and click **Add New ► Folder**. Enter `Book1` for the folder's name.
2. Right click the **Book1** folder in the project manager and click **Add New ► Other**. In the **New Items** dialog under **New ASP.NET Files**, select **ASP.NET Content Page** and click **OK**. In the **New Content Page** dialog, name the page `Book1.aspx`. Select `~/MasterPage1.master` in the **Master page file**.
3. On the **Book1.aspx** pane, click the **Book1.aspx** tab at the bottom. On the first line, change the Title attribute to `Title="Book1"`. Save this change by clicking **File ► Save** or entering **Ctrl-S**.
4. On the **Book1.aspx** pane, click the **Design** tab at the bottom. Click inside the **ContentPlaceHolder** element and enter `Book1`.
5. Add a folder inside the **Book1** folder. Right-click the **Book1** folder in the **Project Manager** and click **Add New ► Folder**. Enter `Chapter1` for the folder's name.
6. Right click the **Chapter1** folder in the project manager and click **Add New ► Other**. In the **New Items** dialog under **New ASP.NET Files**, select **ASP.NET Content Page** and click **OK**. In the **New Content Page** dialog, name the page `Chapter1.aspx`. Select `~/MasterPage1.master` in the **Master page file**.
7. On the **Chapter1.aspx** pane, click the **Chapter1.aspx** tab at the bottom. On the first line, change the Title attribute to `Title="Chapter1"`. Save this change by clicking **File ► Save** or entering **Ctrl-S**.
8. On the **Chapter1.aspx** pane, click the **Design** tab at the bottom. Click inside the **ContentPlaceHolder** element and enter `Chapter1`.

The pages and folders have been added to correspond to the appropriate items in the sitemap XML file. This completes adding site content.

### To finish the site

1. The file `Default.aspx` is no longer needed. Right click on **Default.aspx** in the project manager and click **Delete**. Click **Yes** in the **Confirm** dialog.
2. You must define a start page, which is the first page displayed when the application runs. In the **Project Manager**, right-click **Home.aspx** and select **Set as Start Page** in the context menu.

The application is now ready to run.

### To view the page in a Web browser

1. To run the application and view in a browser, click **Run ► Run**. The project builds, displaying a build status window, which is closed upon completing the build.
2. The Home page should be displayed in a Web browser.
3. Click **Book1** in the menu to display the Book1 page. Click **Chapter1** to display the Chapter1 page.



4. Note that the **SiteMapPath** gets updated to show the currently displayed page's location in the site. You can click the nodes there to go to previous pages in the path.

This completes the project.

#### See Also

ASP.NET Overview (📖 see page 79)

Building an ASP.NET 2.0 Database Application 1 (📖 see page 174)

Building an ASP.NET 2.0 Database Application 2 (📖 see page 176)

Building an ASP.NET 2.0 Database Application 3 (📖 see page 177)

---

## 2.5.8 Creating a Briefcase Application with DB Web Controls

You can use DB Web Controls, XML caching, and the BDP.NET data adapters to create server-side briefcase applications. You can only create this type of application when using user authentication, to guarantee that each user has a unique copy of the XML file.

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

#### To create a briefcase application

1. Create a BDP.NET application.
2. Add a DBWebDataSource control and link to the BDP DataSet.
3. Configure the DBWebDataSource control to generate XML and XSD files.
4. Configure the AutoUpdateCache and UseUniqueFileName properties.
5. Configure an OnApplyChangesRequest to call the BdpDataAdapterAutoUpdate method.
6. Run the application.

#### To configure the AutoUpdateCache and UseUniqueFileName properties

1. Build a standard ASP.NET database application using the BDP.NET components and the DBWebDataSource component.
2. Specify XML and XSD filenames for non-existent files in the DBWebDataSource component.

**Note:** It is best to create these files in the project directory or in a subdirectory of the project directory, typically on your web server.

3. Set AutoUpdateCache to **True**.
4. Set UseUniqueFileName to **True**.
5. Select the **Events** tab for the DBWebDataSource component.
6. Double-click the OnApplyChangesRequest field to display the event handler in the **Code Editor**.
7. Add the following code:

```
BdpDataAdapter1.AutoUpdate;
```

8. Choose **Run ▶ Run**. The first time the application runs, it creates the XSD file using the server metadata.

The first time a user runs the application, the application retrieves data from the server. When the user changes data, thereafter, the application saves those changes to the server in a unique filename based on the username. If the user shuts down the application and runs it again at a later time, the application restores the user's specific data. At this point, the user can undo or



modify the data. Anytime the `OnApplyChangesRequest` is called successfully, the application deletes the unique user files and creates new ones.

**Warning:** If the tables or columns accessed by the application are altered after the application has run, you must delete the XSD file to avoid a mismatch between the XSD file and the server metadata. Otherwise, you can experience runtime errors and unpredictable behavior.

#### See Also

Borland DB Web Controls Overview (see page 81)

Using XML Files with DB Web Controls (see page 92)

Building an Application with DB Web Controls (see page 183)

## 2.5.9 Building an Application with DB Web Controls

The following procedures describe the minimum number of steps required to build a simple ASP.NET database application using DB Web Controls and BDP.NET. After generating the required connection objects, the project displays data in a `DBWebGrid` with a `DBWebNavigator`. Additional information is provided for other common DB Web Controls.

Users should already be familiar with creating an ASP.NET project using BDP.NET.

Building the simple ASP.NET application with DB Web Controls and BDP.NET consists of three major steps:

1. Prepare an ASP.NET project with BDP.NET or other connection components.
2. Drag and drop a `DBWebDataSource` onto the Designer and set its `DataSource` property to a `DataSet`, `DataRowView` or `DataTable`.
3. Drag and drop a `DBWebGrid` and other control onto the Designer.

**Note:** DB Web Controls (`Borland.Data.Web` namespace) are being deprecated in 2007. You should use standard Web controls instead.

#### To prepare an ASP.NET project for DB Web Controls

1. Create an ASP.NET project.
2. Set up BDP.NET or other data access components, setting the `DataSource` property to an existing `DataSet`, `DataRowView`, or `DataTable`.

**Tip:** For more information about setting up BDP.NET data access components, see the related procedure for building an ASP.NET database application. Instead of using a `DataGrid` and adding a `DataBind` call, in the following procedure you use DB Web Controls without a `DataBind` call.

#### To configure a DBWebDataSource

1. Place a `DBWebDataSource` component on the **Designer**.
2. In the **Object Inspector**, select the `DataSource` property.
3. Select an existing data source (by default, this is called `dataSet1`).

#### To configure DB Web Controls

1. Place a `DBWebNavigator` component on the **Designer**.
2. In the **Object Inspector**, select a data source in the `DBDataSource` property drop-down.
3. In the **Object Inspector**, select a `DataTable` from the `TableName` property drop-down.

**Tip:** If no `TableName` is available, verify that the `BdpDataAdapterActive` property is set to **True**.

4. Place a DBWebGrid on the **Designer**.
5. In the **Object Inspector**, select the data source from the DBDataSource property drop-down.
6. In the **Object Inspector**, select a DataTable from the TableName property drop-down. The grid displays data.
7. Place other DB Web Controls as needed.
8. Set the values for DBDataSource, TableName, and other properties as appropriate.

**Note:** For data-aware Column Controls (such as DBWebTextBox, DBWebImage, DBWebMemo, and DBWebCalendar) additionally set the ColumnName property. For data-aware lookup controls (such as DBWebDropDownList, DBWebListBox, and DBWebRadioButtonList), also set the LookupTableName, the DataTextField, and the DataValueField properties.

9. Choose **Run ▶ Run**. The application compiles and the HTTP server displays a Web Form with a DBWebGrid displaying data.

**Tip:** Dragging web components from the Tool Palette

places them in an absolute position on an ASP.NET web form. Double-clicking components in the **Tool Palette** leaves them in ASP.NET flow layout. Flow layout is much easier to manage. For instance, controls in an absolute position on a web form can overwrite other controls if they change sizes at runtime. Overwriting might occur when you add rows to and remove rows from a grid control, making the grid control change size.

#### See Also

Building an ASP.NET Database Application

CodeGear DB Web Controls (🔗 see page 81)

Using XML Files with DB Web Controls (🔗 see page 92)

DB Web Control Wizard Overview (🔗 see page 86)

ASP.NET Overview (🔗 see page 79)

ADO.NET Overview (🔗 see page 14)

Data Providers for .NET (🔗 see page 27)


ADO.NET Component Designers (🔗 see page 21)

## 2.5.10 Converting HTML Elements to Server Controls

Unlike Web controls, HTML elements can not, by default, be controlled programmatically. However, you can convert an HTML element to a server control and then write code to access or modify the element. Most of the HTML elements that appear in the **Tool Palette** can be converted by using the Run As Server Control command. HTML elements that do not appear on the **Tool Palette**, such as `body`, can be converted manually.

The following procedures explain how to convert an HTML `table` element by using the Run As Server Control command, and how to convert a `body` element manually.

#### To convert an HTML table element to a server control

1. With an ASP.NET application open, display the Designer.
2. From the **Tool Palette**, add the **HTML Table** element from the **HTML Elements** category to the Designer.
3. Right-click the **Table** element on the Designer and choose Run As Server Control. The server control icon  is added to the **Table** element. In the `.aspx` file, the `id="TABLE1"` and `runat="server"` attributes are added to the `table` tag. In the code-behind file, `TABLE1` is declared using `System.Web.UI.HtmlControls.HtmlTable`.

4. You can now reference `TABLE1` in your code. To demonstrate this, add a **Button** from the **Web Controls** category of the **Tool Palette** to the Designer.
5. Double-click the button. The **Code Editor** opens and is positioned at the click event for the button.
6. Add the following code to the event handler to change the background color of the table to blue. Note that `TABLE1` is the id that was added automatically to the `table` tag in Step 3.

```
TABLE1.BgColor := 'blue';  
TABLE1.BgColor = "blue";
```

7. Choose **Run ▶ Run** to run the application.
8. Click the button to change the table color.

#### To convert an HTML body element to a server control manually

1. With an ASP.NET application open, display the `.aspx` file.
2. Add the `runat="server"` and `id="identifier"` attributes to the `body` tag, where `identifier` is a descriptive identifier, such as `bodytag`.
3. Add the following declaration to the strict protected section of the code-behind file:

```
bodytag: System.Web.UI.HtmlControls.HtmlGenericControl;  
protected System.Web.UI.HtmlControls.HtmlGenericControl bodytag;
```

4. You can now reference `bodytag` in your code. To demonstrate this, add a **Button** from the **Web Controls** category of the **Tool Palette** to the Designer.
5. Double-click the button. The **Code Editor** opens and is positioned at the click event for the button.
6. Add the following code to change the background color of the Web Form to yellow.

```
bodytag.Attributes['bgcolor'] := 'yellow';  
bodytag.Attributes["bgcolor"] = "yellow";
```

7. Choose **Run ▶ Run** to run the application.
8. Click the button to change the background color of the form.

#### See Also

[Choosing Between HTML Elements and Web Controls](#)

## 2.5.11 Creating an XML File for DB Web Controls

You can use XML files as your data source, particularly if you want to prototype applications without reading from and writing to a database. First you must create the XML file. The `DBWebDataSource` control provides a powerful way to create the XML file based on real database data. This procedure assumes that you can create a connection to a live database containing the data you want to use.

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

#### To create and use an XML file

1. Create an ASP.NET application using DB Web Controls.
2. Specify the XML file as a data source for a new ASP.NET application.

#### To create an ASP.NET application using DBWeb Controls

1. Choose **File ▶ New ▶ ASP.NET Web Application** for either Delphi for .NET.

2. Create a database connection and data adapter using the BDP.NET controls or other data adapter controls.
3. Drag and drop a DBWebDataSource control onto the **Designer** from the **DB Web** area of the **Tool Palette**.
4. In the XMLFileName property or in the XMLSchemaFile property, specify a new file name of a file that does not yet exist.
5. Generate a DataSet from the data adapter.
6. Set the DataSource property of the DBWebDataSource to `dataSet1`.
7. Set the Active property of the data adapter to **True**.
8. Choose **Run ▶ Run**. This runs the application but also creates the XML file or XSD file and fills it with data from the DataSet.

#### To specify the XML file as a data source for a new ASP.NET application

1. Choose **File ▶ New ▶ ASP.NET Web Application** for either Delphi for .NET.
2. Drag and drop a **DataSet** component onto the **Designer** from the **Data Components** area of the **Tool Palette**.
3. Drag and drop a DBWebDataSource control onto the **Designer** from the **DB Web** area of the **Tool Palette**.
4. Specify the existing XML file name in the XMLFileName property of the DBWebDataSource control.  
**Note:** If you created an XSD file instead of an XML file, you specify the XSD file name in this step.
5. Specify the DataSet component in the DataSource property of the DBWebDataSource control.
6. Drag and drop a DBWebGrid control onto the **Designer** from the **DB Web** area of the **Tool Palette**.
7. Set the DBDataSource property of the DBWebGrid to the name of the DBWebDataSource
8. Choose **Run ▶ Run** to display the application. The application pulls data from the DataSet and XML file to fill the DBWebGrid.

**Warning:** It is possible for you to specify an existing XML file in the XMLFileName property of your DBWebDataSource along with an active BdpDataAdapter and its DataSet. You can run the application and the DBWeb controls will display the data from the XML file. However, this is not the intended use or behavior of the XML capabilities of the DBWebDataSource. Although your XML file data may display properly, the results of an update or any other operations on the data will be unpredictable.

#### See Also

- Using XML Files with DBWeb Controls (📖 see page 92)
- Borland DBWeb Controls Overview (📖 see page 81)
- Building an Application with DBWeb Controls (📖 see page 183)
- Setting Permissions for XML File Use (📖 see page 192)
- Building an ASP.NET Database Application (📖 see page 171)

## 2.5.12 Creating Metadata for a DataSet

When you choose to use an XML file for a data source in an ASP.NET application using DB Web Controls, you may need to create the metadata to structure the XML data in your DataSet. If you chose to create an XML file without an XML schema file (.xsd), you need to manually create the metadata. This procedure assumes that you have already created an XML file containing data.

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

**To set up the application**

1. Choose **File ► New ► ASP.NET Web Application** for either Delphi for .NET.
2. Drag and drop a DBWebDataSource control onto the form.
3. Drag and drop a DataSet component onto the form.
4. Click the ellipsis button (...) next to the XMLFileName property of the DBWebDataSource and locate your XML file.
5. Select the DataSet component in the **Component Tray**.
6. Click the **Tables (Collection)** property to display the **Tables Collection Editor**.

**To create the metadata**

1. Click **Add** to add a new table to the collection. For the sake of illustration, we'll use the following XML records.

```
<?xml version="1.0" standalone="yes"> /// XML Declaration
<NewSongs>

  /// <song> becomes the table name in your DataSet.
  <song>

    /// <songid> becomes Column1 in your DataSet.
    <songid>1001</songid>

    /// <title> becomes Column2 in your DataSet.
    <title>Mary Had a Little Lamb</title>
  </song>
</song>
  <songid>1003</songid>
  <title>Twinkle, Twinkle Little Star</title>
</song>
</NewSongs>
```

2. Change the **TableName** property to `song`.
3. Click the **Columns (Collection)** property to display the **Columns Collection Editor**.
4. Click **Add** to add a new column.
5. Change the **ColumnName** property to `songid`.
6. Click **Add** to add another new column.
7. Change the **ColumnName** property to `title`.
8. Click **Close** to close the **Columns Collection Editor**.
9. Click **Close** to close the **Tables Collection Editor**. You have now created the metadata to match the XML file data.

**See Also**

Using XML Files with DBWeb Controls (see page 92)

## 2.5.13 Creating a Virtual Directory

When you create an ASP.NET application, the IDE automatically creates a virtual directory for you based on the settings in the **New ASP.NET Web Application** dialog box.

However, the IDE can also create a virtual directory for an application that you did not create within the IDE, such as the demo applications found in the **DBWeb** folder.

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls

instead.

#### To create a virtual directory for an existing application

1. Open the ASP.NET application project file in the IDE.
2. Choose **Project ► Options ► Debugger ► ASP.NET**. The default application settings are displayed. Accept the default settings or change them as needed.
3. If you are creating a virtual directory for use with Internet Information Server (IIS), click the **Server Options** button to display the **Configure Virtual Directory** dialog. If you change the name of the virtual directory or its alias, you can also change the permissions associated with the virtual directory.
4. Click **OK** to return to the project options.
5. Click **OK** to exit the project options.

The virtual directory is created for you, enabling you to run the application.

---

## 2.5.14 Adding Aggregate Values with DBWebAggregateControl

You can use DBWebAggregateControl to apply one of several standard aggregation functions to a data column. The control displays the aggregate value in a text box, which also support a linked caption.

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

#### To create and configure a DBWebAggregateControl

1. Create a new ASP.NET web application and add your database connection, data adapter, dataset, and DBWebDataSource component to the application..
2. Set the Active property of BdpDataAdapter to **True**.
3. Place a DBWebAggregateControl component on the **Web Form Designer**.
4. Set the DBDataSource property of the DBWebAggregateControl to your DBWebDataSource1, which is the default name of the DBWebDataSource component.
5. Set the TableName property.
6. Choose the AggregateType property value from the drop down list.
7. Choose the ColumnName property from the drop down list. The text box is filled with the value based on the type of aggregate you selected and the values in the column you selected.

**Note:** If you think there may be NULL values in your selected column, set the IgnoreNullValues property to **True**, otherwise you may get an error.

#### To set the caption for DBWebAggregateControl

1. In the **Object Inspector** enter the caption in the Caption property field.
2. Choose a position from the CaptionPosition property drop down list.

#### See Also

CodeGear DB Web Controls (🔗 see page 81)

Building an Application with DB Web Controls (🔗 see page 183)

## 2.5.15 Debugging and Updating ASP.NET Applications

During the installation of RAD Studio, the install program requested permission to update the `machine.config` file on your computer. This information is necessary for debugging RAD Studio applications under IIS. If you replied **Yes** to that prompt, CodeGear debugger information was written to `machine.config` and will be available to the applications that you created with Delphi 8. **You need not perform this procedure.**

If you replied **No** to that prompt, the debugger information is written to the application `web.config` file when you create an ASP.NET application with RAD Studio. However, you will need to add this information manually to `web.config` for applications that were created with Delphi 8. Otherwise, attempting to debug your Delphi 8 application with RAD Studio may result in the following error:

```
Unable to start debugging on the web server. Unable to attach to ASP.NET worker process
(typically aspnet_wp.exe or w3wp.exe).
```

### To update the web.config file for a Delphi 8 ASP.NET application

1. Open the `web.config` file in the IDE or a text editor.
2. Replace the following lines:

```
<compilation
  debug="true"
  defaultLanguage="c#">
</compilation>
```

with this:

```
<compilation defaultLanguage="c#" debug="true">
<assemblies>
  <add assembly="Borland.dbkasp, Version=9.0.0.1,
Culture=neutral, PublicKeyToken=b0524c541232aae7"/>
</assemblies>
</compilation>

<httpModules>
  <add name="DbgConnect" type = "Borland.DbkAsp.DbkConnModule,
Borland.dbkasp,Version=9.0.0.1, Culture=neutral,
PublicKeyToken=b0524c541232aae7"/>
</httpModules>
```

3. Save the `web.config` file.
4. Open the application project in the IDE and run it.

**Note:** Before deploying an ASP.NET application, you should disable debugging and remove debugger references from the `web.config`

file, as described in the topic listed below.

### See Also

Using the ASP.NET Deployment Manager (see page 196)

## 2.5.16 Deploying an ASP.NET Application using Blackfish SQL to a system without RAD Studio

You can deploy an ASP.NET application using Blackfish SQL to a system without RAD Studio.

**To deploy an ASP.NET application using Blackfish SQL to a system that does not have CodeGear RAD Studio**

1. Create an ASP.NET application. In the **web.config** file in the project, set the **connectionString** entry to use the Blackfish SQL Local Client. Here is a sample entry from a **web.config** file:

```
<connectionStrings>
  <!-- Local Connection String : Use the string below for connecting to the provider using
the local client -->
  <add name="BlackfishSqlAspNet"
    connectionString="database=|DataDirectory|bsql_aspnetdb.jds;
    user=user;password=password;create=true"
    providerName="Borland.Data.BlackfishSQL.LocalClient" />
</connectionStrings>
```

2. Build the application.
3. Deploy the application as described in Using the ASP.NET Deployment Manager (see page 95).
4. Deploy the following files to the **bin** directory:
  - Borland.Delphi.dll
  - Borland.VclRtl.dll
  - Borland.Web.Provider.dll
  - Borland.Data.BlackfishSQL.LocalClient.dll
  - BlackfishSQL.slip
5. Copy a Blackfish SQL database to the **App\_Data** directory that is created in the project directory when the application runs.
6. Run the application.

### See Also

Deploying ASP.NET Applications (see page 95)

Using the ASP.NET Deployment Manager (see page 95)

## 2.5.17 Generating HTTP Messages in ASP.NET

When attempting to debug your ASP.NET applications, you may find that the error messages are cryptic or even meaningless. This may be the result of having a specific option set in your Internet Explorer browser. To assist your debugging efforts, you should change this option.

**To generate more meaningful error messages**

1. In Internet Explorer (assuming you are using IE) choose **Tools ► Internet Options**.
2. Click the **Advanced** tab.
3. Deselect the **Show friendly HTTP error messages** check box.
4. Click **OK**. This turns off friendly messages and provides meaningful ASP.NET messages.



**See Also**

ASP.NET Overview (🔗 see page 79)

Troubleshooting ASP.NET Applications (🔗 see page 193)

---

## 2.5.18 Binding Columns in the DBWebGrid

There may be times when you want to modify the order in which columns appear in a DBWebGrid control. You can accomplish this task by binding columns manually, from within the **Property Builder**.

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

**To open the Property Builder**

1. Start a new ASP.NET application.
2. Add a data provider.
3. Add a DBWebDataSource object and connect it to a generated dataset.
4. Add a DBWebGrid control to your Web form.
5. Click the **Property BuilderDesigner verb**, located at the bottom of the **Object Inspector**. This displays the **Property Builder**.

**To change column order**

1. On the **Property Builder**, click the **General** tab.
2. Set the **DataSource** to the DBWebDataSource, or to the dataset the DBWebDataSource points to.
3. Click the **Columns** tab.
4. Select the columns you want to appear in the **Available Columns** list.
5. Click the right-arrow button to add the columns to the **Selected Columns** list.
6. Rearrange the column order, if you like, in the **Selected Columns** list.
7. You can change the column heading name as it appears in the grid by changing the **Header** text.
8. Click **Apply**.
9. Click **OK**.

**Warning:** If you choose to bind columns in this way, you must set the AutoGenerateColumns property to **False**. Setting this property to **True** raises a runtime error, and does not allow the visible restriction of columns at designtime. If the same column is bound to a grid more than once, you may get a runtime error.

**See Also**

CodeGear DBWeb Controls (🔗 see page 81)

Building an Application with DBWeb Controls (🔗 see page 183)

## 2.5.19 Setting Permissions for XML File Use

You need to grant rights to clients who will be using your ASP.NET applications, if you want to avoid a permissions error when using an XML file as a data source. There are two ways to do this, as described in the following procedures.

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

### To give users rights when the `UseUniqueFileName` property is false

1. Right-click the Windows Start menu and choose **Explore**.
2. Choose **Tools ► Folder Options**.
3. Choose the **View** tab.
4. Uncheck the **Use Simple File Sharings** option.
5. Click **Apply to All Folders**.
6. Click **OK**.
7. Locate the XML file being used in the project, then right-click and select **Properties**.
8. If available, select the **Security** tab.
9. Add user **Everyone** and set **Full Rights** to the file.

### To give users rights when `UseUniqueFileName` is true and user authentication is in use

1. On the Windows Control Panel **User Accounts** dialog, create a new user.
2. In the IIS virtual directory where your web application is built, create a new folder named **CacheFiles**. Typically, your IIS virtual directories are in the `C:\Inetpub\wwwroot` directory.
3. Using the Windows Explorer, located the folder **CacheFiles**.
4. Right-click and choose **Properties**.
5. Choose the **Security** tab and add the user you created in Step 1.
6. Add **Full Rights** to the folder.
7. Move the XML file to this folder.
8. Set the `XMLFileName` property of the `DBWebDataSource` in your application to this file.

**Note:** You must make sure that the **Use Simple File Sharings** option in your Windows **Folder Options** is unchecked.

### See Also

Using XML Files with DB Web Controls (▢ see page 92)

Creating an XML File for DB Web Controls (▢ see page 185)

## 2.5.20 Troubleshooting ASP.NET Applications

Unlike traditional window-based applications, web applications are dependent on servers and resources that are not directly within the control of the application or the user. Web applications are often hybrid combinations of client, server, and network

resources.

The areas you need to check include ASP.NET installation, IIS installation and configuration, and security. All three of these areas are extensive and complex. The following procedures provide solutions to some of the most common problems.

**Note:** The following suggestions apply only to IIS 5.1.

#### To troubleshoot your ASP.NET application

1. Install or reinstall ASP.NET.
2. Create or check your ASP.NET user account.
3. Install or reinstall IIS.
4. Start or restart IIS.
5. Configure IIS to recognize your application.
6. Add document types to IIS.
7. Set anonymous authentication.
8. Check your database connection, if applicable.

#### To install or reinstall ASP.NET

1. Choose **Start ► Run** to display the **Run** dialog box.
2. Type `cmd /e` in the **Open** drop down list box.
3. Click **OK**.
4. Change directories to `c:\Windows\Microsoft.NET\Framework\v1.1.4322`.
5. Enter the command `aspnet_regiis.exe -i`.
6. Press **Enter**.

**Note:** If you want to know the various command flags for the `aspnet_regiis.exe` utility, follow the basic command with a `?` character instead of the `-i` flag.

#### To create or check your ASP.NET user account

1. Choose **Start ► Control Panel ► User Accounts** to display the list of user accounts on your system.
2. If you do not have an ASPNET user account, create one.
3. Restart your machine.

**Warning:** Do not give your ASPNET user administrator privileges. This opens up a security hole in your system and makes deployed ASP.NET applications vulnerable to hacking. Instead, create an impersonated user.

#### To install or reinstall IIS

1. Choose **Start ► Control Panel ► Add or Remove Programs**. This displays the **Add or Remove Programs** dialog box.
2. Click **Add/Remove Windows Components**. This displays the **Windows Components Wizard**.
3. Check the **Internet Information Services (IIS)** check box.
4. Click **Next**.
5. Click **Finish**.
6. Start IIS.

### To restart IIS

1. Choose **Start ► Control Panel ► Administrative Tools ► Internet Information Services**.
2. Select the local computer node.
3. Right-click and select **Restart IIS....** This displays the **Stop/Start/Reboot** dialog.
4. Choose the task you want to accomplish from the drop down list box.
5. Click **OK**.

### To configure IIS to recognize your application

1. In the IIS console, locate the folder or virtual directory containing your web application. If there is not a folder or virtual directory, you will need to create a virtual directory.
2. Select the folder.
3. Right-click and select **Properties**.
4. Click the **Virtual Directory** tab.
5. Under the **Application Settings** area, click the **Create** button. If the **Remove** button is displayed instead, you can remove, then create the virtual directory again, if necessary.

### To add document types to IIS

1. Choose **Start ► Control Panel ► Administrative Tools ► Internet Information Services**.
2. Select **Default Web Site**.
3. Right-click and select **Properties**.
4. Click the **Documents** tab.
5. Click **Add**. This displays the **Add Default Document** dialog box.
6. Add **WebForm1.aspx** in the **Default Document Name** textbox.
7. Click **OK** twice.

### To set anonymous authentication

1. In the IIS console, locate the folder or virtual directory containing your web application. If there is not a folder or virtual directory, you will need to create a virtual directory.
2. Select the folder.
3. Right-click and select **Properties**.
4. Click the **Directory Security** tab.
5. Click **Edit**.
6. Select the **Anonymous Access** check box.
7. In the **User name:** field, enter the name of the ASPNET user you created.
8. Check the **Integrated Windows authentication** check box or add your own password.
9. Click **OK** twice.

### To check your database connection

1. Click the **Data Explorer** tab to display your database connections.
2. Expand the provider list to display a valid database connection.
3. Right-click and choose **Modify Connection**. This displays the **Connections Editor**.
4. If the Database connection string does not contain the **localhost** specifier, prepend it to the connection string, as in the

following example:

```
localhost:C:\Program Files\Common Files\Borland Shared\Data\EMPLOYEE.GDB
```

5. Make sure all of your other connection options are set property.
6. Click **Test** to make sure the connection is alive.

#### See Also

ASP.NET Overview (see page 79)

[Microsoft Discussion Groups](#)

Creating a Virtual Directory (see page 188)

Generating HTTP Messages in ASP.NET (see page 191)

---

## 2.5.21 Using the DB Web Control Wizard

The **DB Web Control Wizard** helps you create a data-aware web control based on a standard web control.

**Note:** DB Web Controls (Borland.Data.Web namespace) are being deprecated in 2007. You should use standard Web controls instead.

#### To start the DB Web Control Wizard

1. Choose **File ► New ► Other ► Delphi for .NET Projects ► DB Web Control Library**. This displays the **New DB Web Control Wizard**.
2. Enter a name for the control in the **Control Name** textbox.
3. Select **Bind to DataTable**. This informs the wizard to add to the control file code that implements `IDBWebDataLink`. This interface defines the means to access data source and table information.
4. Select **Bind to DataColumn** if you want to bind to a column, for instance, if your control supports a single type of data. This informs the wizard to add to the control file code that implements `IDBWebColumnLink`. This interface defines the means to access a column in the table accessed by way of `IDBWebDataLink`.
5. If you select **Bind to DataColumn** and your control is one of the lookup controls, such as a listbox, radio button group, or check box control, and you want the new control to be a lookup control also, check the **Supports Lookup** check box. This informs the wizard to add to the control file code that implements `IDBWebLookupColumnLink`. This interface defines the means to access the lookup table, the text field and value field of the column accessed by way of `IDBWebColumnLink`.

The **DB Web Control Wizard** creates a template file and displays it in the **Code Editor**. You then modify this file to inherit from a specific DB Web control.

#### See Also

DB Web Control Wizard Overview (see page 86)

Borland DB Web Controls Overview (see page 81)

---

## 2.5.22 Using the ASP.NET Deployment Manager

You can add an ASP.NET Deployment Manager to an ASP.NET application project to assist you with deploying the application. The Deployment Manager determines which files are required for deployment, requests the destination directory name and connection information, and then copies the files to the destination directory. The Deployment Manager generates a list of files to copy based on the names of the files in your project directory, but you can include or exclude files as needed.

You can use the right mouse button, when the Deployment Manager window is displayed, to see options for displaying, copying, deleting, modifying, and filtering destination files.

When the Show Assembly References option is enabled, the Deployment Manager window displays all of the assemblies referenced by the project. The system assemblies are shown, but disabled (grayed). These disabled assemblies can't be deployed.

The External Files.... option allows you to pick the external files that you want to deploy. A dialog box with a check list box is pre-populated with the BDP database libraries, since one of these often needs to be deployed. You can also add files to the list using a **File ► Open** dialog. The list box has a column that indicates the destination subdirectory for the external file. You can edit the destination path. The files that are checked when you click OK will be shown in the Deployment Manager.

See the links at the end of this topic for more information about the right-click options for the Deployment Manager.

### Considerations

- To enable IIS debugging of RAD Studio applications, during the installation of RAD Studio, the install program requested permission to update the `machine.config` file on your computer. If you replied **Yes** to that prompt, CodeGear debugger information was written to `machine.config`. If you replied **No** to that prompt, that debugger information is written to the application `web.config` file when you create an ASP.NET application with RAD Studio. Before deploying the application, you should disable debugging to optimize the application, as described in the following procedure. Additionally, if you chose not to update `machine.config`, you should remove references to the CodeGear debugger modules in `web.config`, because those modules might not be available on the deploy target computer.
- Consider maintaining a separate `web.config` file for deployment purposes. For example, you might maintain a file named `web.config.deploy` and rename it to `web.config` during deployment. Use the Deployment Manager Change Destination Filename command to rename the file.
- You can create the destination directory while using the Deployment Manager, however, you will then need to use IIS to create the virtual directory before using the application. Alternatively, you can deploy to an existing virtual directory.
- When deploying to an FTP site, the Deployment Manager will retain your FTP connection information. You may save your FTP connection password, however, it will be saved as unencrypted, plain text.
- You can add multiple Deployment Managers to an ASP.NET project and configure them to deploy to different destination directories.
- Some of the commands that are available in the Deployment Manager are also available in the **Project Manager** context menu.

### To remove debugger references in the web.config file

1. In the IDE or a text editor, open the `web.config` file that you will use for the deployed ASP.NET application.
2. In the `<compilation>` section, change `debug="true"` to `debug="false"`.
3. Skip this step if you chose to update `machine.config` during the installation of RAD Studio (see the **Considerations** above for details). Remove or comment out the following references to the CodeGear debugger assembly and modules:

```
<assemblies>
  <add assembly="Borland.dbkasp, Version=9.0.0.1,
    Culture=neutral, PublicKeyToken=b0524c541232aae7"/>
</assemblies>

<httpModules>
  <add name="DbgConnect" type =
    "Borland.DbKAsp.DbKConnModule,Borland.dbkasp,Version=9.0.0.1,
    Culture=neutral,
    PublicKeyToken=b0524c541232aae7"/>
</httpModules>
```


4. Save the file and recompile the application.

### To deploy an ASP.NET application

1. In the IDE, open the ASP.NET application project to be deployed.



- Choose **File ► New ► Other ► Deployment ► ASP.NET Deployment** and click **OK**. (The **Deployment** node is not displayed in the **New Items** dialog box unless an ASP.NET project is open.) The **Deploy** tab is displayed and a **.bdsdeploy** file is added to the project directory and displayed in the **Project Manager**. The files required for deployment are listed on the left side of the **Deploy** tab under **Source Files**.

**Tip:** Only files that have been saved are displayed in the list; save any new files and refresh the Deployment Manager to display the files.

- In the **Destination** drop-down list, select either **Folder Location** or **FTP Location**. If you select **Folder Location**, the **Browse For Folder** dialog box is displayed. You can select an existing directory or click **Make New Folder** to create a new one. If you select **FTP Location**, the **FTP Site** dialog box is displayed. Enter the connection information. Click **Help** for an explanation of each field. Click **OK** to return to the Deployment Manager.
- If you selected an FTP location, check the **Connected** check box to connect and display the files, if any, in the destination directory.
- Review the files in the **Source Files** list. Click a file to display detailed file information in the text box below the file list.
- To copy all of the files to the destination directory, click the **Copy All New or Modified Files to Destination** button  on the toolbar at the top of the Deployment Manager. The files are copied immediately to the destination directory and displayed in the **Destination Files** list. To modify the file list, right-click anywhere in the file list and use the context menu commands, or use the file list status buttons, as described below.




**Tip:** To select a file in the list, click the file name. To select multiple files, press **CTRL**

and click the files. To select a range of files, press **CTRL+SHIFT**, click the first file in the range and then the last file in the range.

Context Menu Command	Description
Refresh	Redisplays the Deployment Manager to reflect changes in the file lists.
Copy Selected File(s) to Destination	Copies the selected files to the destination directory.
Delete Selected Destination File(s)	Deletes the selected files from the destination directory after displaying a confirmation prompt for each file.
Change Destination Filename	Displays a dialog for renaming the selected file in the destination directory.
Copy All New and Modified Files to Destination	Copies all of the files marked with  to the destination directory. This command is also available on the Deployment Manager toolbar and by right-clicking the <b>.bdsdeploy</b> node in the <b>Project Manager</b> .
Delete All Destination Files Not in Project	Deletes any of the files marked with  from the destination directory after displaying a confirmation prompt for each file.
Show Ignored Groups and Files	Displays all of the files in the project directory, even those that are not required to deploy the application.
Ignore Group(s)	Causes the selected file to be ignored by the Deployment Manager.
Ignore File(s)	Causes all of the files in a node of the source files list to be ignored by the Deployment Manager.
Enable Logging	Logs the operations performed by the Deployment Manager in a file named <b>DeployLog.txt</b> in the project directory.
View Log	Displays the log file in the default text editor.

- When you are satisfied with the deployment criteria, save your changes to the **.bdsdeploy** file. When you reopen the project, you can open the Deployment Manager from the **Project Manager** and deploy the application as is, or modify the deployment criteria as described above.

The following buttons indicate the status of the files in the file list and can be used to copy or delete the file, as described below.

File List Status Button	Description
	The file is eligible to copy (it does not exist in the destination directory, or the source file has changed since it was last copied to the destination). Click the button to copy the file to the destination directory.
	The file exists in the destination directory, but not in the project directory. You can probably safely delete it from the destination directory. Click the button to delete the file from the destination directory.
	The status of the file in the is unknown. It might have a later time stamp than the file in the project directory. Click the button to replace the file in the destination directory.

#### To create an IIS virtual directory for a new destination directory

1. Open IIS on the computer where you deployed the application. On Windows XP, for example, choose **Start ► Control Panel ► Administrative Tools ► Internet Information Services**.
2. In the **Internet Information Services** dialog box, expand the tree view to display the local computer node.
3. Right-click the **Default Web Site** node and choose **New ► Virtual Directory**. The **Virtual Directory Creation Wizard** is displayed.
4. Follow the prompts on each page of the wizard to create the virtual directory.

For more information about virtual directories, refer to the IIS online Help system.

#### See Also

Deploying ASP.NET Applications (📖 see page 95)

## 2.5.23 Using the HTML Tag Editor

When you are creating or editing an HTML file, you can use the **Tag Editor** window, beneath the Form Designer, to edit the HTML tags. If you are using an ASP.NET Web Form, the Tag Editor is not supported. If you are using an HTML Form, you can display the Tag Editor in the Designer by selecting **View ► Tag Editor**.

The **Tag Editor** lets you review and modify HTML tags while viewing the corresponding controls in the Designer window, above it. The **Tag Editor** allows you to use the **Code Completion**, **Error Insight**, and **Live Template Completion** features that are also available in the **Code Editor**. Refer to the links at the end of this topic for more information about using each of these features.

The **Tag Editor** works with one tag at a time, unless you have the Document object selected or you have zoomed out from a tag. (When the document object is selected, you'll see the item "DOCUMENT" on the Object Inspector.)

Use the zoom buttons to zoom out to a tag's parent and zoom back in to the selected child tag. Zooming isn't specific to the tag, it's more generic to the markup in the document itself. For example, if the cursor is on a tag in your HTML markup, and you use the Zoom command, it will take you to the outer tag, or one level above the attribute where the cursor is positioned.

Validation against standard HTML style rules occurs automatically. If validation fails, the incorrect element is highlighted in red in the **Designer**, and **Error Insight** appears in the **Tag Editor** to help you correct the problem.

**Note:** The Tag Editor

is not available in an ASP.NET application.



**To view HTML code for an individual control**

1. With the Designer displayed, drag an HTML element from the **Tool Palette** to the Designer surface. The **Tag Editor** displays the HTML code.
2. To view the individual control's code, click anywhere on the Designer surface to deselect the control. The HTML code appears in the tag editor window, with syntax highlighting. The gray header of the tag editor now displays the higher level tag, usually the FORM tag that defines this particular Web Form.

**Note:** If a control is defined using several lines of HTML code, when you select the control, the first line of the code is displayed in the gray header of the tag editor. The additional code appears below in the tag editor window.

**To view the HTML code for all controls**

1. With the Designer displayed, drag several HTML elements from the **Tool Palette** to the Designer surface. The editor displays the HTML code for each element as you drop them on the Designer surface.
2. Click anywhere on the Designer surface to deselect all controls. This displays the code for all the controls in the tag editor, with syntax highlighting.

**To modify a control**

1. Click anywhere on the Designer surface to deselect all controls.
2. Locate the tag that corresponds to the control you want to modify.
3. Modify the code, and the change is immediately reflected in the control on the Designer surface.
4. Save your project to make the modifications permanent.

**To change editor properties**

1. Choose **Tools ► Options ► HTML/ASP.NET Options**.
2. Change any code editor properties.
3. Click **OK**. Your changes take effect immediately.

**To zoom between contents of the form and the form container**

1. To zoom out so that you can view the HTML form definition, click the left-hand blue arrow in the gray header of the tag editor.  
**Note:** You can only use this feature when the cursor is somewhere in the tag editor, rather than on the Designer surface.
2. To zoom in so that you can view only the content within the FORM tags, click the right-hand blue arrow in the gray header of the tag editor.

**Note:** You can only use this feature when the cursor is somewhere in the tag editor, rather than on the Designer surface.

**To close the Tag Editor**

1. Choose **Tools ► Options ► HTML/ASP.NET Options**.
2. Uncheck the **Display Tag Editor** option.
3. Click **OK**.

**See Also**

[Using the Code Editor](#)

[Customizing the Code Editor](#)

[Using Live Templates](#)

[Using Code Insight](#)

---

## 2.5.24 Working with ASP.NET User Controls

User controls provide a way to reuse common user interface functionality across ASP.NET web applications. For example, you might create user control that encapsulates a login screen. You could then add the user control to any Web Form that requires the login screen functionality. For more information about user controls, click the link at the end of this topic.

### To create an ASP.NET user control

1. Open an ASP.NET application.
2. Choose **File ► New ► Other ► Delphi for .NET Projects ► New ASP.NET Files** and double-click on **ASP.NET User Control**. A new `.ascx` file is added to the **Project Manager** and the empty page is displayed in the Designer. Optionally, rename the `.ascx` file by right-clicking it in the **Project Manager** and choosing **Rename**. Any associated files, such as the `.pas` or `.resx` files, are also renamed.
3. Design the page by adding controls, setting properties, and adding code to the code-behind `.pas` file as needed.
4. Save and compile the project.

### To add an ASP.NET user control to a Web Form

1. Open the Web Form to which you want to add the user control. Make sure the Designer is displayed.
2. Choose **Insert ► Insert User Control** to display the **Insert User Control** dialog box.
3. Select a user control from the drop-down list or use the **Browse** button to navigate to a user control file (`.ascx`).
4. Click **OK** to add the user control to the Web Form.
5. Optionally, in the **Object Inspector**, provide a descriptive name for the user control button with the **Id** property.
6. Save and compile the project.

The Web Form is displayed in the browser and the user control button is replaced with its encapsulated controls.

**Tip:** The runtime appearance of the user control depends on the appearance of the encapsulated page and controls, not the position of the user control button. If you are adding multiple user controls to a page, run the application to ensure that the controls do not overlap each other.

### See Also

[Web Forms User Controls](#)

## 2.6 Web Services Procedures

This section provides how-to information on developing and using web services.

### Topics

Name	Description
Accessing an ASP.NET "Hello World" Web Services Application (see page 202)	If you want to consume the Web Services application you created, you must create a client application to access your ASP.NET Web Services application. This process requires different development steps to achieve the desired output.
Adding Web References in ASP.NET Projects (see page 204)	If you want to consume a web service, you must create a client application, and add a Web Reference. These procedures describe how to create an ASP.NET client application that consumes a third-party web service. The client application consumes the DeadOrAliveWS web service available from the XMethods Web site. This web service lets you query a simple database of celebrities and their respective birthdates and expiration dates.
Building an ASP.NET "Hello World" Web Services Application (see page 206)	Building an application with ASP.NET Web Services lets you expose functionality to your client application over a Web connection. These steps walk you through building a simple "Hello World" application with ASP.NET Web Services. Once built, the application exposes all of its objects and methods through a WebMethod that you create and access through a web browser.
Porting a Delphi for Win32 Web Service Client Application to Delphi for .NET (see page 207)	The following steps are required to port your Delphi for Win32 Web Services client application to Delphi for .NET.

### 2.6.1 Accessing an ASP.NET "Hello World" Web Services Application

If you want to consume the Web Services application you created, you must create a client application to access your ASP.NET Web Services application. This process requires different development steps to achieve the desired output.

#### To access a simple "Hello World" ASP.NET Web Services application

1. Create a client application.
2. Add a Web Reference for an XML web service.
3. Create the code-behind logic.
4. Run the client application.

#### To create a client application

1. Choose **File ► New ► Other**. A **New Items** dialog box appears.
2. Select any type of application to create your client, such as a Windows Forms application or an ASP.NET Web application. For this example, we will create a Windows Forms application (either Delphi for .NET or C#).
3. Click **OK**. A **New Project** dialog box appears.

#### To add a Web Reference for an ASP.NET Web Services application

1. Choose **Project ► Add Web Reference**.
2. From the **CodeGear UDDI Browser** web dialog box, enter the following URL in the address text box at the top:

`http://localhost/WebService1/WebService1.aspx`

**Note:** The name of your application may not be WebService1. In that case, use your own application name in place of

WebService1 in the example preceding example.

3. Press **Enter**.

**Note:** If you need to determine the proper path and you are using IIS, you can open the IIS Administrator from the Windows XP Control Panel Administrative Tools. Find the WebService you have saved and compiled in the list of IIS web sites, then review the name of the site and the name of the .asmx

file. If you have entered the proper path, this should display information about the WebMethods.

4. Click the **Service Description** link to view the WSDL document.

5. Click **Add Reference** to add the WSDL document to the client application. A **Web References** folder is added to the Project directory in the **Project Manager** which contains the `WebService1.wsdl` file and the dialog box disappears.

### To create the code-behind logic

1. Add a Button to the Windows Form.

2. Double-click the Button to view the code-behind file.

3. For a Delphi for .NET client, implement the Click event in the **Code Editor** with the following code:

```
procedure TForm1.Button1_Click(sender: System.Object; e: System.EventArgs);
var
    ws: TWebService1;
begin
    ws := TWebService1.Create;
    button1.Text := ws.HelloWorld();
end;
```

When you added the Web Reference to your application, RAD Studio used the WSDL to generate a proxy class representing the "Hello World" web service. The Click event uses methods from the proxy class to access the web service. For a Delphi for .NET client, you may need to add the unit name of the proxy class (for example, `localhost.WebService1`) to the **uses** clause of your Windows Form unit to prevent errors in your Click event.

4. For a C# client, implement the Click event in the **Code Editor** with the following code:

```
private void button1_Click(object sender, System.EventArgs e)
{
    TWebService1 ws = new TWebService1();
    button1.Text = ws.HelloWorld();
}
```

### To run the client application

1. Save the application.

2. Compile and run the project.

3. Click the Button on your client application. The "Hello World" caption appears on the button.

### See Also

ASP.NET Web Services Overview (see page 97)

ASP.NET Web Services Support (see page 102)

Building an ASP.NET "Hello World" Web Services Application (see page 206)

## 2.6.2 Adding Web References in ASP.NET Projects

If you want to consume a web service, you must create a client application, and add a Web Reference. These procedures describe how to create an ASP.NET client application that consumes a third-party web service. The client application consumes

the DeadOrAliveWS web service available from the XMethods Web site. This web service lets you query a simple database of celebrities and their respective birthdates and expiration dates.

### To create an ASP.NET project

1. Choose **File ► New ► Other**. The **New Items** dialog box appears.
2. Double-click the **ASP.NET Web Application** icon in the **Delphi for .NET Projects** item categories. The **New ASP.NET Application** dialog box appears.
3. In the **Name** field, enter a name for your project.
4. In the **Location** field, enter a path for your project.

**Tip:** Most ASP.NET projects reside in the IIS directory `Inetpub\wwwroot`

5. If necessary, click the **View Server Options** button to change your Web server settings.

**Tip:** The default Server Options will usually be sufficient, so this step is optional.

6. Click **OK**. The Web Forms Designer appears.

### To design the ASP.NET web page

1. If necessary, click **Design** view.
2. From the **Web Controls** category of the **Tool Palette**, place a Button component onto the Designer surface. The Button control appears on the Designer. Make sure the control is selected.
3. In **Object Inspector**, set the **Text** property to `Dead or Alive?`.
4. From the **Web Controls** category of the **Tool Palette**, place a TextBox component onto the Designer above the Button. This is where you type your query to the Web Service.
5. Place a Label component below the Button. This is where the results of the web service query are displayed.

Use the UDDI browser to locate the DeadOrAlive Web Service on the internet. This allows you to use the methods and objects published by the Web Service Definition Language (WSDL).

### To add the Web Reference for DeadOrAliveWS

1. Choose **Project ► Add Web Reference**.
2. In the **CodeGear UDDI Browser** web dialog box, click the **XMethods Full** link in the list of available UDDI directories. A list of various web services published on the XMethods Web site appears.
3. Find and click the **DeadOrAliveWS** link.

**Tip:** You can use Ctrl+F

to search within the **CodeGear UDDI Browser**.

4. Click the link to the WSDL file:

`http://www.abundanttech.com/webservices/deadoralive/deadoralive.wsdl`

A WSDL document appears. This XML document describes the interface to the DeadOrAliveWS web service.

5. Click **Add Reference** to add the WSDL document to the client application. A **Web References** folder containing a **com.abundanttech.www** node is added to the Project directory in the **Project Manager**.

### To write the application logic

1. If necessary, click **Design** view.
2. Double-click the **Dead or Alive?** button to view the code-behind file.
3. For a Delphi for .NET Web Services application, implement the Click event in the **Code Editor** with the following code :

```

procedure TWebForm1.Button1_Click(sender: System.Object; e: System.EventArgs);
var
    result: DataSet;
    ws: DeadOrAlive;
    currentTable: DataTable;
    currentRow: DataRow;
    currentCol: DataColumn;
begin
    //This initializes the web service
    ws := DeadOrAlive.Create;

    //Send input to the web service
    result := ws.getDeadOrAlive(TextBox1.Text);

    //parse results and display them
    Label1.Text := '';
    for currentTable in result.Tables do
        begin
            Label1.Text := Label1.Text + '<p>' + #13#10;
            for currentRow in currentTable.Rows do
                begin
                    for currentCol in currentTable.Columns do
                        begin
                            Label1.Text := Label1.Text + currentCol.ColumnName + ': ';
                            Label1.Text := Label1.Text + (currentRow[currentCol]).ToString;
                            Label1.Text := Label1.Text + '<br>' + #13#10;
                        end;
                    end;
                end;
            Label1.Text := Label1.Text + '</p>';
        end;
    end;

```

When you added the Web Reference to your application, RAD Studio used the WSDL to generate a proxy class representing the "Hello World" web service. The Click event uses methods from the proxy class to access the web service. For Delphi for .NET Web Services, you may need to add the unit name of the proxy class, `abundanttech.deadoralive`, to the **uses** clause of your Web Form unit to prevent errors in your Click event.

### To run the application

1. Choose **Project ► Build All Projects**. Now your project is built and resides on your ASP.NET server.
2. Open a Web browser.
3. Type the URL of your Web Application's `.aspx` file and press **Enter**.

**Tip:** If you are using Microsoft IIS, the URL is the path of the `.aspx`

file after `Inetpub\wwwroot`. For example, if the path of your Web Application is

`c:\Inetpub\wwwroot\WebApplication1` and your `.aspx` file is named "`WebForm1.aspx`", the URL would be `http://localhost/WebApplication1/WebForm1.aspx`.

4. If necessary, enter your user name and password for your ASP.NET server. The web page for your web application appears.
5. Enter the name of a celebrity (for example, Isaac Asimov) in the text box and click the **Dead or Alive?** button. Your web application requests the information from the DeadOrAliveWS web service and displays the result in the label.

**Note:** If no information is displayed, that name may not be in the database. Check your spelling or try a different name.

### See Also

Creating ASP.NET Web Applications

ASP.NET Web Services Overview (see page 97)

Building an ASP.NET "Hello World" Web Services Application (see page 206)

Accessing an ASP.NET "Hello World" Web Services Application (see page 202)

TDataset

---

## 2.6.3 Building an ASP.NET "Hello World" Web Services Application

Building an application with ASP.NET Web Services lets you expose functionality to your client application over a Web connection. These steps walk you through building a simple "Hello World" application with ASP.NET Web Services. Once built, the application exposes all of its objects and methods through a WebMethod that you create and access through a web browser.

### To create a simple "Hello World" application with ASP.NET Web Services

1. Create an ASP.NET Web Services application.
2. Create a WebMethod.
3. Test and run the ASP.NET Web Services application.

**Note:** Currently, using RAD Studio you can only create web services using the code-behind method. You cannot use the code inline method, in which you code your web service in the <ServiceName>.asmx

file. Currently, RAD Studio does not support the code inline method of creating web services.

### To create an ASP.NET Web Services application

1. Choose **File ► New ► Other**. A **New Items** dialog box appears.
2. Select the **ASP Projects** folder for the language you are using.
3. Select **ASP.NET Web Service Application**. An **Application Name** dialog box appears.
4. Enter a name and location of the application in the fields and retain all other default settings.

**Note:** If you are using the Cassini Web Server, you need to change the Location and Server entries. You also need to make sure you configure the Cassini Web Server before trying to run this application. Choose **Tools->Options**

and select **ASP.NET Options** to set the **Path** and the **Port** for Cassini.

5. Click **OK**. A `WebService1.asmx` file and a `WebService1.asmx.<filetype>`, are automatically created for you.

### To create a WebMethod

1. Select the **WebService.pas** or **WebService.asmx.cs** tab at the bottom of the **Code Editor**. If you named your web service application something other than the default, that will be the name that appears on the tab. The code for the "Hello World" application is already included in the WebMethod that is created for you when you created the Web Services application.
2. Uncomment the sample WebMethods in the code-behind file. Delphi for .NET applications have two "Hello World" WebMethods to uncomment; one in the Interface module and the other in the Implementation module.
3. Choose **Project ► Build <project name>** to build your project.
4. Run your project. This invokes the browser which hosts the Web Service. The pages you see will include sample SOAP and HTTP code that you can use to access the WebMethods. You can run the samples and see how the results are passed to an output XML file.

### To test and run the XML web service manually

1. From a web browser, enter the location of the `WebService1.asmx` file on your localhost:

`http://localhost/WebService1/WebService1.asmx`

The pages you see will include sample SOAP and HTTP code that you can use to access the WebMethods. You can run the

samples and see how the results are passed to an output XML file.

**Note:** You may need to use a slightly different syntax than that shown in this step. For instance, on some Windows XP machines, the *localhost* identifier should be your machine name. For instance, if your machine name is *MyMachine*, the syntax would be: `http://MyMachine/WebService1/Webservice1.asmx`.

2. Test the two methods from a web browser.

#### See Also

ASP.NET Web Services Overview (see page 97)

ASP.NET Web Services Support (see page 102)

Accessing an ASP.NET "Hello World" Web Services Application (see page 202)

---

## 2.6.4 Porting a Delphi for Win32 Web Service Client Application to Delphi for .NET

The following steps are required to port your Delphi for Win32 Web Services client application to Delphi for .NET.

#### To port your web service

1. Change the existing RIO form components.
2. Change the **uses** clause.
3. Add a web reference.
4. Change the web service invocation code.

#### To change your existing form components

1. Copy and save the web reference URL from your existing RIO component.
2. Delete the HTTPRIO component from the form if it was not dynamically created.

#### To change the uses clause

1. Remove any Delphi for Win32 SOAP units from the clause. These include, but are not restricted to `InvokeRegistry`, `RIO`, and `SOAPHTTPClient`.

**Warning:** The preceding list of units is not inclusive. Make sure you identify all SOAP units, regardless of naming convention. Not all of the units include the word SOAP in the name.

2. Remove the reference to the Delphi for Win32 WSDL Importer-generated Interface proxy unit.
3. Remove the proxy unit from the project.

#### To add a web reference

1. Open a Delphi for Win32 project in RAD Studio and choose **Project ► Add Web Reference**. Once you have saved the project, the UDDI Browser appears.
2. Enter the URL you want to use, either a service you are already familiar with, or the one saved from your RIO component into the list box at the top of the Browser.

**Note:** If you want to locate a WSDL file on your local disk, you can click the ellipsis button next to the list box and search for the document. You can also navigate to one of the web service sites listed in the UDDI Browser if you want to use a published service.



3. Click the **Add Reference** button to add the WSDL document to your project. RAD Studio creates the necessary web reference and the corresponding proxy unit based on the WSDL document. A new Web References node appears in the **Project Manager**. Expand it to see the associated WSDL and proxy code files.
4. Choose **File ► Use Unit**.

#### To change the web service invocation code

1. In the code file for your application, locate the code that invokes the web service. Assume it looks something like this:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  HelloService: Service3Soap;
begin
  // The next line will be slightly different if you have
  // used a component or generated the method dynamically.

  // This is how it will look if you used a component.
  HelloService := (HTTPRIO1 as Service3Soap);

  // This is how it will look if created dynamically.
  // GetService3Soap is the global method in the proxy unit.
  HelloService := GetService3Soap;

  Caption := HelloService.HelloWorld;
end;
```

2. Change the **var** section from this:

```
var
  // This is the type of the old proxy interface.
  HelloService: Service3Soap;

to

var
  // This is the type of the new proxy class.
  HelloService: Service3;
```

This assumes the name of your service is Service3. Change the name accordingly.

**Note:** You will see that what was formerly created as an interface is now created as a class. The .NET Framework provides automatic garbage collection, and so certain restrictions placed on the use of classes in previous versions of Delphi may no longer apply when using RAD Studio.

3. Change the first line in the procedure block from this:

```
HelloService := (HTTPRIO1 as Service3Soap);

to:
```

```
HelloService := Service3.Create;
```

The updated code should look like this:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  HelloService: Service3;
begin
  HelloService := Service3.Create;
  Caption := HelloService.HelloWorld;
end;
```

Your code is most likely more complex than this example. However, these instructions cover the basic steps for porting any Delphi for Win32 application that uses web services to RAD Studio.

**See Also**

Porting Web Services to Delphi for .NET ([🔗](#) see page 74)

Web Services Overview ([🔗](#) see page 97)

# Index

- 
- .NET controls
  - VCL and .NET controls 165
- .NET data types
  - ADO.NET 11
- A**
  - Accessing an ASP.NET "Hello World" Web Services Application 202
  - Adding a BDP Reconcile Error dialog to your BDP Application 108
  - Adding a J2EE Reference 139
  - Adding a New Connection to the Data Explorer 107
  - Adding a Reference to a COM Server 139
  - Adding Aggregate Values with DBWebAggregateControl 188
  - Adding Web References in ASP.NET Projects 204
  - ADO.NET
    - Adapter Preview Editor 126
    - architecture 14
    - ASP.NET 79
    - Command Text Editor 127
    - CommandText Editor 128
    - connection components 148
    - database applications 148
    - namespace 14
    - overview 14, 58, 71
  - ADO.NET application 171
  - ADO.NET Component Designers 21
  - ADO.NET Overview 14
  - AdoDbx
    - BDP migration 16
  - AdoDbx Client Connecting
    - building 110
  - AdoDbx Client Overview 5
  - AdoDbx.NET
    - data providers 27
    - data types 11
  - AdoDbx.NET components
    - ADO 27
  - AdoDbx.NET Data Types 11
  - AdoDbxCient
    - overview 5
  - ancestor
    - base 159
  - ASP.NET
    - architecture 79
    - DB Web Controls 183
    - overview 79
  - ASP.NET application 170, 174, 176, 177, 178, 179
  - ASP.NET errors
    - HTTP messages 191
  - ASP.NET lifecycle
    - ASP.NET processing 86
  - ASP.NET Overview 79
  - ASP.NET page 170, 178
  - ASP.NET Procedures 167
  - ASP.NET project 170, 178
  - ASP.NET user account
    - security holes 193
  - ASP.NET Web Services Overview 97
  - ASP.NET Web Services Support 102
  - AutoUpdateCache
    - UseUniqueFileName 182
- B**
  - BDE.NET
    - database 6
  - BDP Connection Pooling Overview 8
  - BDP migration 16
  - BDP Migration Overview 16
  - binding
    - components 195
  - Binding Columns in the DBWebGrid 191
  - bitmap images 164
  - bitmaps
    - combobox 164

- Blackfish SQL
    - overview 20
  - Blackfish SQL Overview 20
  - briefcase application
    - AutoUpdateCache property 182
  - browsing a database 109
  - Browsing a Database in the Data Explorer 109
  - building
    - VCL forms applications with XML components 155
    - VCL Forms hello world 151
    - VCL Forms menus 152
  - Building a Database Application that Resolves to Multiple Tables 122
  - Building a Distributed Database Application 136
  - Building a VCL Forms Application 149
  - Building a VCL Forms dbExpress.NET Database Application 153
  - Building a VCL Forms Hello World Application 151
  - Building a VCL.NET Forms ADO.NET Database Application 148
  - Building an Application with DB Web Controls 183
  - Building an Application with XML Components 155
  - Building an ASP.NET "Hello World" Application 178
  - Building an ASP.NET "Hello World" Web Services Application 206
  - Building an ASP.NET Application 170
  - Building an ASP.NET Application with Database Controls, Part 2 176
  - Building an ASP.NET Application with Database Controls, Part 3 177
  - Building an ASP.NET Database Application 171
  - Building an ASP.NET SiteMap 179
  - Building VCL Forms Applications With Graphics 147
- C**
- callback functions
    - functions 44
  - cascading deletes
    - database 83
  - cascading updates
    - database 83
  - Changes Required Due to 64-bit .NET 2.0 Support 58
  - clearing text 152
  - ClearSessionChanges method 94
  - client dataset
    - TClientDataSet 153
  - Code Visualization 54, 141, 143, 144
    - exporting diagrams 141
  - Code Visualization Overview 54
  - code-behind logic 170, 178
  - CodeGear DB Web Controls Overview 81
  - COM interfaces
    - interfaces 44
  - COM Interop 38, 139
    - Interop assemblies in the IDE 38
    - SDK Tools 38
    - Terminology 38
  - combobox events
    - OnDrawItem 164
  - component designers
    - Command Text Editor 21
    - configure data adapter 21
    - Connection Editor 21
    - Dataset 21
    - relationship 21
    - Stored Procedure Dialog 21
  - components
    - connection 81
    - data-aware 81
    - importing 71
  - Concepts 1
  - configuring
    - connection components 171
    - data components 171
    - IIS 148, 170
    - Web server 171, 178
  - connecting
    - data source 171
    - DataBind 171
    - DataGrid 171
  - connecting DataGrids 148

Connecting to a Database using the dbExpress Driver Framework 134

Connecting to the AdoDbx Client 110

connections

database connections 117

MS Access connection 117

MS SQL Server connection 117

Sybase connection 117

constructors

destructors 59

Converting HTML Elements to Server Controls 184

Creating a Briefcase Application with DB Web Controls 182

Creating a New VCL.NET Component 159

Creating a Virtual Directory 188

Creating Actions in a VCL Forms Application 150

Creating an XML File for DB Web Controls 185

Creating Database Projects from the Data Explorer 111

Creating Metadata for a DataSet 187

Creating Table Mappings 112

## D

data

migration 115

remoting 136

data binding

DB Web Control binding 86

data explorer

definition 21

Data Explorer

executing SQL 113

ISQLDataSource 21

modifying connections 116

data providers

architecture 27

Data Providers for .NET (AdoDbx)

ADO.NET 27

Data Providers for Microsoft .NET 27

data types

ADO.NET 27

database

adding a new connection 128

Command Text Editor 127

connections 117

creating project 111, 122, 136

DataAdapter Designer 128

DataSet Designer 129

dbExpress 30, 31

dbExpress for .NET 9

dbGo for .NET 10

Database

adding a new connection 107

adding error reconcile 108

browsing objects 109

executing SQL 113

modifying connections 116

using data adapter preview tab 126

using generate dataset designer 132

database components

configuring 112

Database Procedures 106

DataHub component

configuring 122

DataSet

table mappings 112

XML files 187

DataSnap .NET Client

database 6

DataSync component

configuring 122

DataView limitations

inserting records 94

DataViews

runtime properties 94

DB Web Control Wizard

custom controls 86

DB Web Control Wizard Overview 86

DB Web Controls

architecture 81

- ASP.NET 81
  - configuring 183
  - library 195
  - namespace 81
  - preparing projects 183
- DB Web Controls Navigation API Overview 85
- DB Web interfaces 86
- DB2
  - AdoDbx.NET 11
- dbExpress Components overview 9
- dbExpress Framework 30
- dbExpress Framework Compatibility 31
- dbExpress.NET
  - database 6
- dbGo
  - database 6
- dbGo Components Overview 10
- DBWeb Controls
  - WebDataLink interfaces 96
- DBWebAggregateControl
  - aggregate values 188
- DBWebDataSource
  - configuring 183
- Debugging and Updating ASP.NET Applications 189
- Delphi for .NET
  - Web Forms 79
- deploying
  - ADO.NET applications 24
  - ASP.NET Deployment Manager 190, 196
  - BDE applications for .NET 24
  - BDP.NET applications 24
  - COM Interop applications 43
  - database applications 24
  - dbExpress applications for .NET 24
  - dbGo applications for .NET 24
- Deploying an ASP.NET Application using Blackfish SQL to a system without RAD Studio 190
- Deploying ASP.NET Applications 95
- Deploying COM Interop Applications 43
- Deploying Database Applications for the .NET Framework 24

- deployment
  - DB Web Controls 81
- Developing an ASP.NET Application with Database Controls, Part 1 174
- Developing Applications with Unmanaged Code 38
- Developing Applications with VCL.NET Components 57
- Developing Database Applications with ADO.NET 3
- Developing Reports for .NET Applications 56
- Developing Web Applications with ASP.NET 77
- Developing Web Services with ASP.NET 97
- Displaying a Bitmap Image in a VCL Forms Application 160
- displaying bitmap images 160
- displaying child nodes 155
- drawing
  - polygons 162
  - rectangles and ellipses 161
  - straight lines 163
- Drawing a Rounded Rectangle in a VCL Forms Application 162
- Drawing Rectangles and Ellipses in a VCL Forms Application 161
- Drawing Straight Lines In a VCL Forms Application 163

## E

- ECO framework 142
- event handlers
  - OnPaint 161, 163
- Executing SQL in the Data Explorer 113
- Exporting a Code Visualization Diagram to an Image 141

## G

- Generating HTTP Messages in ASP.NET 191
- Getting Started with InterBase Express 32

## H

- Handling Errors in Table Mapping 114
- HTML tag editor
  - editing HTML tags 199
- HTTP
  - messages 193

## HTTP messages

errors 191

**I**

## IBX.NET

InterBase 6

## IDBWebColumnLink

IDBWebDataLink 96

IDBWebLookupColumnLink 96

## IDBWebDataLink 96

## IIS

troubleshooting 193

## Importing .NET Controls to VCL.NET 165

## Importing and Exporting a Model Using XML Metadata Interchange (XMI) 142

## install ASP.NET

reinstalling 193

## Interbase

connections 117

## InterBase

ADO.NET 11

components 32

## Interfaces

AdoDbx.NET 27

## Interoperable Applications Procedures 139

**L**

## Language Issues in Porting VCL Applications to RAD Studio 59

## logical data types

AdoDbx.NET 11

**M**

## Making Changes Required Due to 64-bit .NET 2.0 Support 157

## master-detail

DataViews 94

## menus

actions 150

example application 150

standard actions 150

VCL forms 150

## metadata

DataSet mappings 187

XML schema files 187

## migrating

tables 115

## Migrating Data Between Databases 115

## MissingMappingAction property

errors 114

## MissingSchemaAction property

errors 114

## Modeling Concepts 54

## Modeling Procedures 141

## Modifying Connections in the Data Explorer 116

## Modifying Database Connections 117

## modifying DB Web controls

extending controls 86

## MS SQL

ADO.NET 11

**N**

## New Language Features 69

## null values in aggregates

caption property 188

**O**

## object references

passing references 44

## Oracle

ADO.NET 11

## override Render

Render 86

**P**

## parameters

database 124

## Pascal

language changes 69

## Passing Parameters in a Database Application 124

PIInvoke

platform invoke 44

Placing a Bitmap Image in a Control in a VCL Forms Application 164

pointer types

porting 59

porting

VCL.NET porting 71

Porting a Delphi for Win32 Web Service Client Application to Delphi for .NET 207

Porting VCL Applications 69

Porting Web Service Clients 74

Procedures 105

property builder

DB Web Controls 191

## R

Rave Reports 56

creating new reports 56

resolving

multiple tables 122

resources

migrating 59

## S

sample

ASP.NET hello world 178

sending

commands 171

ServerSupport

ASP.NET 79

Setting Permissions for XML File Use 192

Stored Procedure Overview 29

stored procedures

overview 29

string

char 59

structures

pointers 44

Sybase

ADO.NET 11

## T

table mappings

configuring 112

deleting 112

errors 114

tables

mappings 112

Tag Editor 199

TDataSetProvider

dbExpress provider 153

TDataSource

dbExpress data source 153

TDBGrid

dbExpress connection 153

Troubleshooting ASP.NET Applications 193

typecasts

crackers 59

## U

unidirectional dataset

TSQLDataSet 153

unidirectional datasets

dbExpress components 153

unmanaged functions

Win32 API 44

user authentication

UseUniqueFileName 192

user control

ASP.NET 200

ASP.NET user control 200

user interfaces

ADO.NET 14

Using ActionManager to Create Actions in a VCL Forms Application 152

Using COM Interop in Managed Applications 38

Using DB Web Controls in Master-Detail Applications 83

Using DrInterop 43



- Using Platform Invoke with Delphi for .NET 44
- Using Rave Reports in RAD Studio 56
- Using Standard DataSets 129
- Using the ASP.NET Deployment Manager 196
- Using the Command Text Editor 127
- Using the Connection Editor Designer 128
- Using the Data Adapter Designer 128
- Using the Data Adapter Preview 126
- Using the DB Web Control Wizard 195
- Using the HTML Tag Editor 199
- Using the Model View Window and Code Visualization Diagram 143
- Using the Overview Window 144
- Using Typed DataSets 132
- Using XML Files with DB Web Controls 92

## V

- variants

- TVarRec 59

- VCL

- porting 69

- VCL and .NET controls

- .NET control package 165

- VCL applications

- Client DataSets 148

- data sources 148

- DataSet providers 148

- dbExpress database applications 153

- events 149

- forms 149

- graphics 147

- Windows Forms project 149

- VCL for .NET

- graphics 160

- VCL for .NET Database Technologies 6

- VCL for .NET Overview 71

- VCL for .NET Procedures 145

- VCL form

- bitmap image 164

- VCL Form

- adding button and event code 151

- creating new 151

- running example application 151

- VCL forms

- menus 150

- VCL.NET

- architecture 71

- namespace 71

- porting 59

- VCL.NET components

- VCL.NET 71

- virtual directory, creating in IDE 188

- Virtual Library Interfaces 52

## W

- web references 204

- web service clients

- porting 69

- web services

- accessing 204

- accessing a web services application 202

- architecture 97

- ASP.NET 97

- building an application 206

- client support 102

- creating a client application 202

- discovery 100

- files 97

- namespaces 102

- porting 74, 207

- porting applications 74

- porting web services forms 74

- porting Win32 to .NET 207

- prerequisites 97

- protocol 100

- scenarios 97

- server support 102

- service description 100

service transport 100

UDDI browser 204

web references 204

xml messaging 100

#### Web Services

ASP.NET 79

client 202

Web Services Procedures 202

Web Services Protocol Stack 100

#### Win32 API

unmanaged code 59

WinForm Control Import Wizard

.NET controls in VCL 165

Working with ASP.NET User Controls 200

Working with DataViews 94

Working with WebDataLink Interfaces 96

## X

#### XML

DB Web Controls 185

XML and authentication

XML caching 92

XML file permissions

permissions 192

XML files

DBWeb data sources 92

DBWeb XML files 185

XML advantages 92