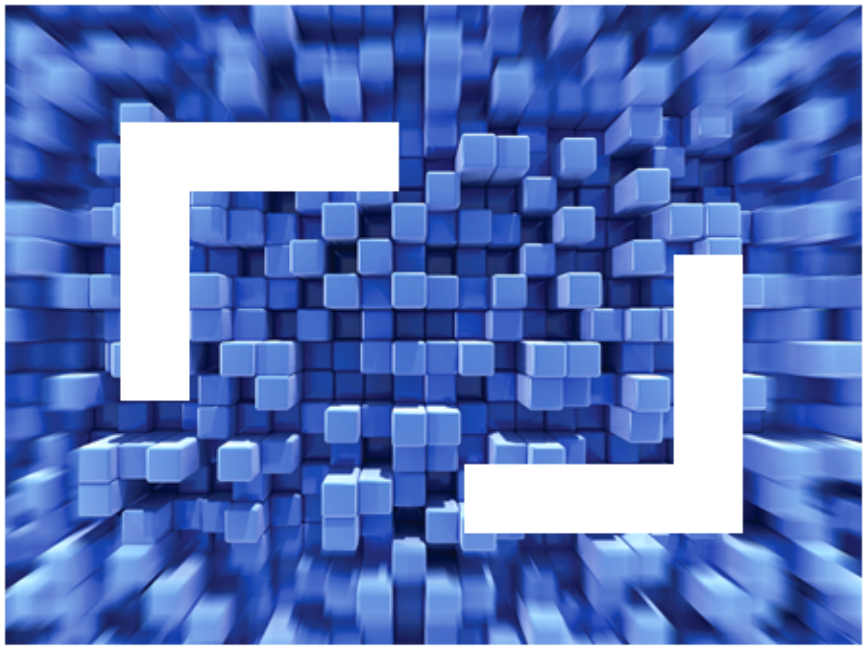


OnWeb 7.5.0

Training Guide for Host Integration

MICRO FOCUS
Leading the Evolution™

Micro Focus (IP) Ltd.
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK
http://www.microfocus.com

Table of Contents

Overview 1

OnWeb® rapidly transforms host applications into Web-based solutions, without the high cost of rewriting applications or changing back-end systems. It is a proven and easy-to-use application development framework for building enterprise-level extranet applications.

OnWeb comprises the following core components:

- OnWeb **Server** - a high performance, scalable, multi-user runtime environment that accesses and integrates multiple data sources in real-time, executes business logic, and delivers the application to the desktop. It runs either on Microsoft® Windows®, Solaris™, Linux®, or AIX® platforms, or IBM® iSeries™ systems and fully utilizes their performance capabilities.
- OnWeb **Designer** - a development environment for developing OnWeb applications.
- OnWeb **Administrator** - a tool for administering OnWeb Server.
- **OnWeb Source Server** - a database server used by Designer to store components of OnWeb applications.
- OnWeb **Application Manager** - a tool for deploying and managing OnWeb applications on a production server.
- **OnWeb Object Builder**- a tool for creating host transactions that will enable 3rd party development environments to access host information.

For more information about OnWeb, see Chapter 2, “Understanding OnWeb”.

Using this guide

This guide will help you develop an awareness of the fundamentals of OnWeb Host Integration development. You will learn how to create a basic Host Integration application, and gain a solid understanding of some of the internal workings. The aim of this guide is to provide the OnWeb developer with a solid foundation for proper Host Integration application development using OnWeb Designer.

To build the example scripts like the ones provided in this guide, you will need OnWeb Server 5 or greater and OnWeb Designer 2 or greater. In order to use the sample terminal-based objects, OnWeb must be installed on a machine with Internet access.

Each chapter explains a certain aspect of Host Integration development. Chapter 2 gives you an overview of OnWeb Host Integration development environment. Chapters 3 to 7 guide you through the process of creating OnWeb objects. Chapter 8 describes how to add error handling codes to your OnWeb Host Integration application. And Chapter 9 shows you how to create a conditional object.

The sample code included in this guide is simple and basic; error checking is not included in order to keep the code as clear and concise as possible. The sample code is intended only to display the particular aspect of OnWeb scripting described in the chapter.

OnWeb documentation

In order to use this guide effectively, you must have a working knowledge of HTML and FrontPage®, and should be familiar with the OnWeb Designer environment. To learn more about Designer, consult OnWeb Designer Help.

Other components of OnWeb documentation will also help increase your OnWeb knowledge base. You can access them from the Windows **Start** menu:

- OnWeb Scripting Help contains reference information related to the OnWeb scripting environment.
- OnWeb *Developer's Guide* discusses in detail the process of developing OnWeb applications.
- *OnWeb Samples Guide* describes NetTelecom, a sample Host Integration application distributed with OnWeb.
- OnWeb *Migration Guide* guides you through the process of converting your existing Salvo® Impact® application to an OnWeb application and describes issues related to upgrading you applications from one version of OnWeb to the next.

Understanding OnWeb 2

What is OnWeb Host Integration

Increasingly, businesses and organizations are looking to the Web as a means to extend host-based information to new users. To achieve results quickly and at reasonable cost, these organizations must fully leverage their existing applications and data sources.

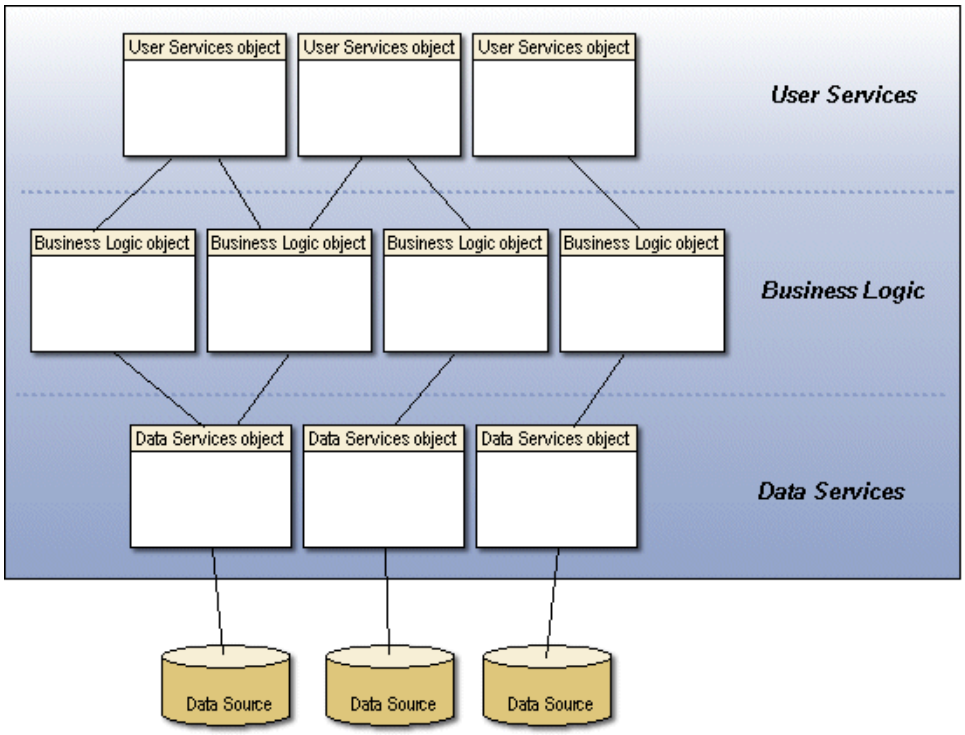
OnWeb Host Integration is the complete solution. OnWeb rapidly transforms host applications into Web-based solutions without the high cost of re-engineering applications or changing back-end systems. Simple point and click operation allows developers to integrate host applications and ODBC compliant database systems.

After deployment of a completed application, the following benefits are realized:

- Web access is provided for remote users to existing host resources via any standard configuration browser-equipped PC.
- Host information is made available to employees, business partners, and customers in easy-to-use graphical web pages.
- Cost for software development and administration is reduced through centrally managed, zero client footprint solutions.

OnWeb development environment

OnWeb's internal multi-tier architecture separates the User Services, Business Logic, and Data Services layers, making it invaluable to OnWeb application development.



OnWeb's top layer is the User Services tier, which allows you to easily create and modify the presentation of information, without changing any of the underlying business logic.

OnWeb executes Business Logic at its middle tier using a combination of data from multiple back-end sources. The Business Logic objects representing the business processes are developed and linked within Designer (the development environment), using "drag and drop" techniques. OnWeb's object-based development environment is designed to enable developers to rapidly create new applications with minimal coding. When coding is required, OnWeb offers the choice of using VBScript, JScript®, REXX, HTML, and standard SQL languages, which are ideal for rapid application development.

At the bottom level is OnWeb's Data Services tier, which insulates the application from changes to back-end systems. OnWeb separates data source access and definitions from the application logic, and uses Information Object® technology to ensure that system restructuring results in minimal changes to the OnWeb application.

Before examining the scripting syntax used to develop OnWeb applications, we will examine three of the most common internal OnWeb elements that you will be working with: the OnWeb objects, the Information Object (IObject), and the Table Object. For information about OnWeb objects, see Appendix A, "OnWeb Object Reference".

OnWeb objects

OnWeb objects are the OnWeb application logic, each containing their own properties and methods. Objects are executed to perform some specific pieces of application logic. Different types of objects can be created to accomplish the diverse logic requirements for your application.

Tier	Object	Function
Data Services	Data Source objects	Data connection
	Data Services objects	Data collection (ODBC/Terminal)
Business Logic	Business Logic objects	Data manipulation
User Services	User Services objects	Data presentation (HTML)

Think of each object as being a self-contained object that can be executed independently by OnWeb. It should only perform a single independent task to conform to the object-oriented paradigm. By connecting multiple objects together, they can be used to execute a complete application. This framework allows you to easily alter an existing application by simply connecting a new object.

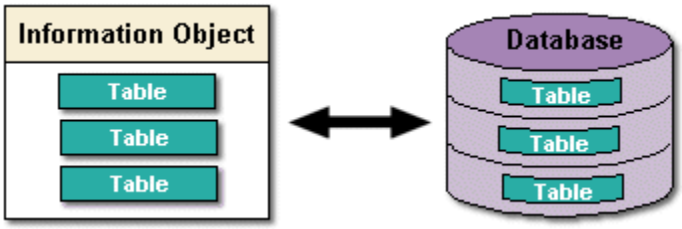
Table object

An OnWeb Table object can be equated to a table of data similar to a relational database table. An OnWeb Table object, however, is kept in memory, which allows it to support a more flexible and powerful access method than a typical relational database. You can use relational database access methods, which move the current row position one row at a time until you are at the row you want to access, or you can directly access a specific cell by using the Table object's cell property.

Information Object

The OnWeb Information Object can be seen as the representation of the data returned from an OnWeb object (explained in subsequent chapters). The Information Object is the nucleus of OnWeb's development communication channel, by which data is dispatched between different OnWeb objects.

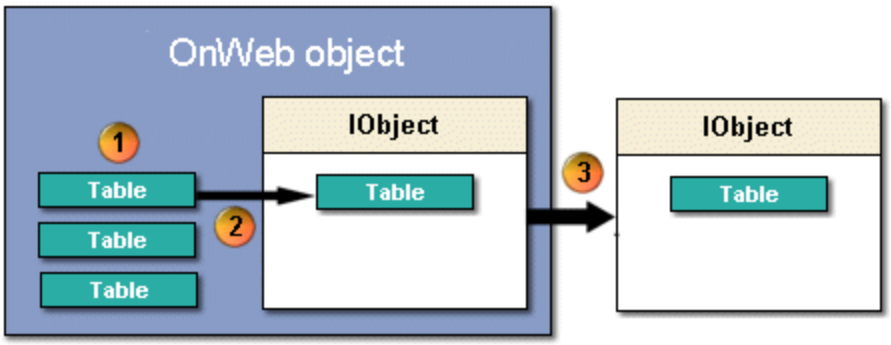
For simplicity's sake, consider OnWeb's Information Object in the same manner as you would a simple relational database. The Information Object can contain various objects (strings, tables, dictionaries and even other Information Objects). If you think of it as a database, that can contain a variable number of tables, it will help you understand its fundamental role as you begin creating your first sample object.



It is important to recognize that every OnWeb object contains its own internal Information Object, referred to as the IObject. It is also essential to understand that after an object has finished executing, only its own internal IObject is returned. Therefore, any collected or processed data that you want returned must be inserted into the object's own IObject. Data that has not been included within the IObject is removed from memory when the object has finished executing.

Remember, the only thing returned from an object is its own Information Object (IObject). In order to help you comprehend the IObject when you are learning OnWeb application development, think of it as a normal relational database. Like a database, it can include multiple tables, each containing data that you want to return from the object.

The following diagram shows how OnWeb returns data processed from one of its objects:



1 - A Table Object is created and filled with data.

2 - The Table to be returned is stored in the object's own Information Object (IObject).

3 - Once the object has finished executing, it returns its own IObject, containing all the tables of information.

Until you have a firm grasp of the returning IObject, just think of the executing object as returning a database record and think of the IObject as a database which contains a single table.

Creating Data Services Objects 3

Our first example is designed to demonstrate the most basic and widely used aspect of OnWeb scripting. This example demonstrates the steps involved when creating a basic OnWeb Data Services object (a table object), populating it, and placing it into the returning OnWeb object's IObject.

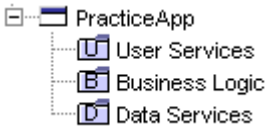
Creating an OnWeb application

Before you can create the Data Services object, you must create a project and an application.

▶ To create a new OnWeb application

1. Start OnWeb Designer.
2. In the OnWeb Server Logon dialog box, enter your OnWeb user name and password to establish a connection with OnWeb Server.
3. In the OnWeb Designer dialog box, select **New Project** and click **OK**.
4. In the New Project dialog box, enter *PracticeProject* in the **Name** box, and click **OK**.
5. In the OnWeb Application Wizard dialog box, select **New Application** and click **OK**.
6. In the Create Application dialog box, enter a name in the **Name** box, select **Host Integration**, and click **OK**. For this example, type *PracticeApp* as the application name.

Note that the application is now represented in the tree-list area. A newly created application is assigned a standard 3-tier configuration (Data Services, Business Logic, and User Services).



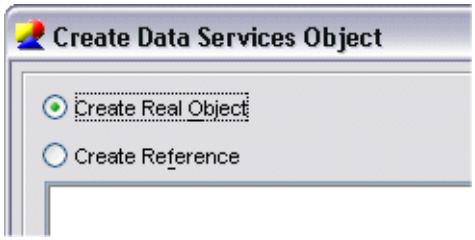
Creating a basic Data Services object

You will create a simple table object containing a single column within its schema collection. You will then insert the literal string “Hello World!” into the first row/column (cell) of the table, and place the table object into the OnWeb object’s IObject.

▶ To create a Data Services object

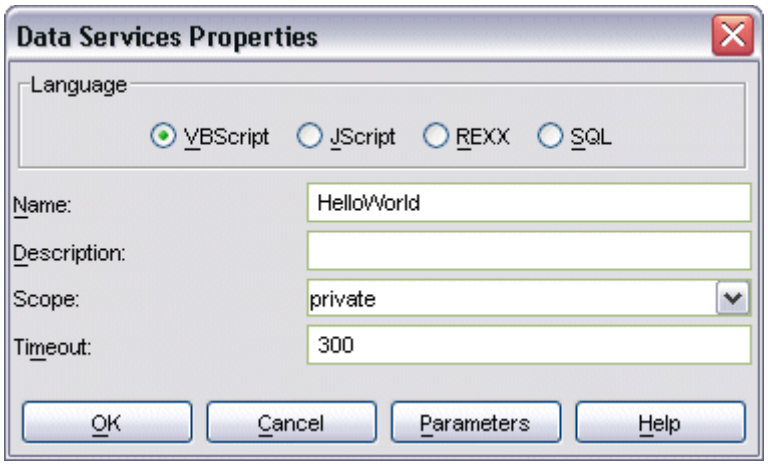
1. Double-click Data Services under your new application name to open the Data Services white-board.
2. From the **New** menu, choose **Data Services** and then click anywhere within the Data Services white-board to place your new Data Services object.

   In the Create Data Services Object dialog box that appears, you can specify whether this will be a new Data Services object or a reference to an existing one. For this example, create a new Data Services object.



3. Select **Create Real Object**.
4. Click **OK**.

The Data Services Properties dialog box appears, allowing you to specify various property values for this new Data Services object.



5. In the **Language** area, select **VBScript**.

Note: While you are free to use any of the available OnWeb scripting languages, this guide will focus on using VBScript for all of its examples. OnWeb has an object-oriented development environment and this fits well with the object-oriented nature of VBScript.

6. Give the new Data Services object a meaningful name. Since the example is to return the literal string “Hello World!”, name the new Data Services object “HelloWorld”.

7. You are also going to allow this Data Services object to be executed as a stand-alone component, so make its scope public by selecting “public” in the **Scope** list.

   To keep things simple, leave the rest of the values at their respective defaults. They will be discussed in future chapters.

8. Click **OK** to create the new Data Services object.

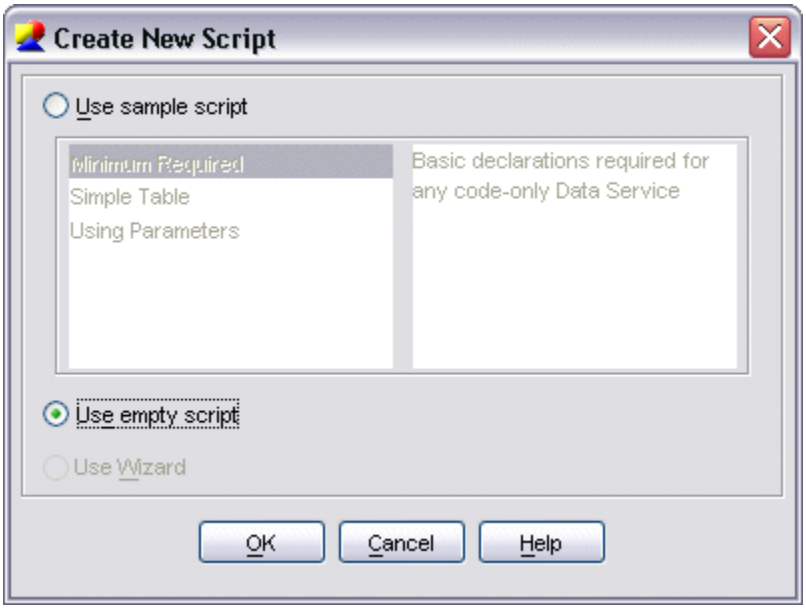
Adding the code

Now let's get to the heart of our Data Services object and write some scripting logic to make it do something.

▶ To add code to the Data Services object

1. Right-click the new HelloWorld Data Services object and choose **Edit Script** from the menu.

   The first time you edit the script, the Create New Script dialog box appears, giving you the option to use a previously created sample script.



2. Click **Use empty script** and click **OK**.

   The Designer default script editor appears. If you had specified a sample script, it would have been inserted within the text area.

3. Type the following script segment in the script editor.

```
Sub Collect()
   Set tTable = Onweb.CreateObject _
      ("Onweb.IComponent.Table","HelloTable")
   Set cColumnOne = _
      Onweb.CreateObject("Onweb.ColumnDef")
   cColumnOne.Name = "Col1"
   cColumnOne.Type = "String"

   tTable.Schema.Add cColumnOne
   tTable.InsertRow
   tTable("Col1") = "Hello World!"
   IObject.Contents.Add tTable

End Sub
```

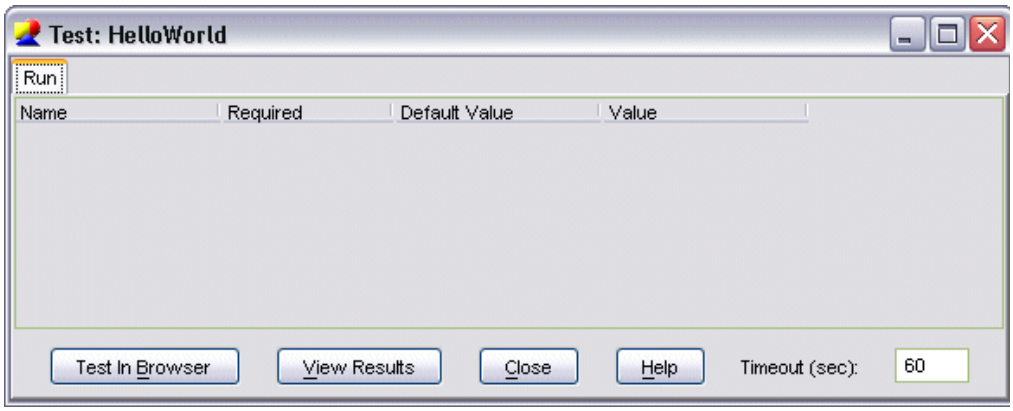
4. Save the script.

   You have now finished creating your first object. The next step is to test your new object to make sure everything executes as planned.

   You do not need to close the Designer script editor in order to test your scripts. Designer allows you to have multiple windows open (script editor, test dialog box, parameters list, etc.).

5. Right-click the HelloWorld Data Services object and choose **Test** from the menu.

   This opens the Test dialog box, which allows you to execute the object within Designer to make sure everything is correct. This is also good practice in order to make sure the expected data is being returned.



   The value in the **Timeout** box allows you to limit the amount of time the object can take to execute all of its logic before it is aborted. Leave the default value of 60 seconds.

For this example, you will be testing the object using the **View Results** button. This will inform Designer to execute the object within its own environment, instead of through a browser.

Tip: You can test an object in two different ways: within Designer or in a browser. If you execute your object using the **View Results** button, Designer will execute the object within its own environment. Any errors will be reported back to Designer. You can view the result table individually from within the Designer Test window. If you choose the **Test In Browser** option, Designer will open the default browser and the result set will be displayed within the browser's window.

6. Click **View Results**.

   The object will execute and a new tab called **Table 1** will be added.

   The **Table 1** tab should now be highlighted with the following information displayed:

   You can now view the data located in each cell of **Table 1**.

   In this case, the Table Object contains a single column, called "Col1", with a single row present. You can view the string "Hello World!" that was inserted into the cell. So far, everything looks good and it appears that the object has run successfully; the data displayed is what you were expecting.

Interpreting the example

Now that you have created your first object and executed it correctly, let's go back to the beginning of this example and discuss what you were actually doing, and more importantly, why you were doing it.

The first thing you may notice is that you wrote all the script logic within a subroutine called Collect. While you are able to create as many VBScript subroutines and functions as necessary to execute your object logic, it is very

important to remember that you must have at least one subroutine called Collect when executing a Data Services object. OnWeb looks for this subroutine when it attempts to execute any Data Services object. If this subroutine is not present, an error is generated and the execution of the object is aborted.

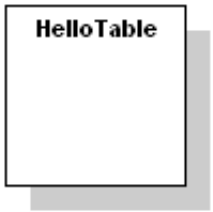
The first thing you do is to create a subroutine header called Collect.

```
Sub Collect()
```

It is important to remember that an object only returns an IObject containing Table Objects. You must place the gathered or processed data you wish to return within a Table Object, and insert it into the IObject.

Next you need to create a Table Object into which to put the "Hello World!" literal string. To create the new Table Object, you need to use the OnWeb CreateObject method (see CreateObject method in the OnWeb Scripting Help), which uses the same basic syntax as the VBScript CreateObject method. When creating a new OnWeb Table Object, you also need to include a second parameter specifying the name.

```
Set tTable = _
   Onweb.CreateObject("Onweb.IComponent.Table",_
   "HelloTable")
```



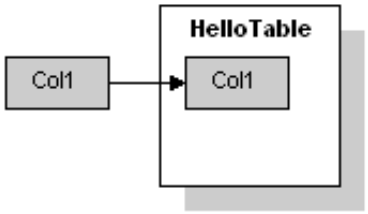
You now have an empty OnWeb Table Object that can be accessed by using the reference name tTable. At this point, you still need to add a column to the Table Object's schematics that accepts strings as input. In the same fashion as you created the new Table Object, you create a new OnWeb Column Object called cColumnOne.

```
Set cColumnOne = Onweb.CreateObject("Onweb.ColumnDef")
```

Now you need to define some default property values for the newly created Column Object before you insert it into the Table's schematics. The most important property you are required to specify is its name property. The name property is used to reference the Column Object once it has been added to the table. You also specify the type of values that the Column Object accepts; in this case you choose "String" since this is what you intend for it to contain.

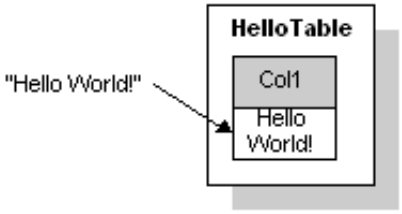
```
cColumnOne.Name = "Col1"
cColumnOne.Type = "String"
```

After creating the new Column Object, you need to add it to the Table Object's schematics collection. The Schema object (automatically created when the Table Object is created) has an Add method that is intended exactly for this purpose.




```
tTable.Schema.Add cColumnOne
```

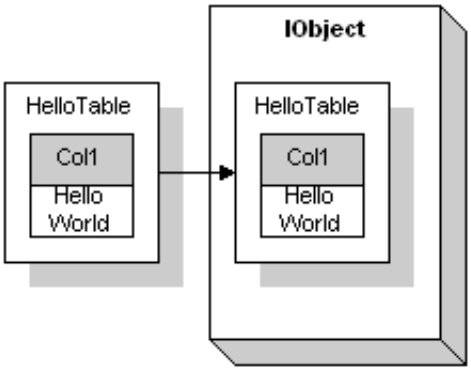
Your empty Table Object is now starting to show some form. You have a single Table Object called "HelloTable" containing a single Column Object called "Col1". Then you insert a new row into the table, which creates a cell into which to insert the literal string "Hello World!".



To add a new row, you need to use the Table Object's InsertRow method. Once this is done, the Table Object's cursor property points to the new row, so you assign the "Hello World!" string to the "Col1" column.

```
tTable.InsertRow
tTable("Col1") = "Hello World!"
```

You now have a simple OnWeb Table Object, which contains the literal string "Hello World!" within its cell.



One last step is required before the example is finished. As previously discussed, the only thing an object returns is its own IObject; at this point in the example, the IObject is empty. You must also take into consideration the fact that the IObject will contain a Table Object. Therefore, you need to include the Table Object within the object's IObject. You add the Table Object to the IObject's dictionary collection using the Add method.

```
IObject.Contents.Add tTable
```

Summary

Now you have been introduced to some of the basic concepts that form the backbone of OnWeb application development. You learned that the IObject is at the heart of each OnWeb object. You also got to see a glimpse of how OnWeb works in transporting data out of it's objects. More advanced features of the IObject are discussed in the OnWeb Scripting Help. But until you are more familiar with OnWeb development, thinking of the IObject as a simple database file will help you understand its mechanics better.

You created a simple Data Services object in order to help introduce you to the basic VBScript syntax involved with OnWeb development. It is essential that you have an understanding of the syntax and concepts introduced in this chapter. These concepts will not be fully discussed again.

Before continuing to the next chapter, you should be familiar with the following points:

- An object returns an Information Object (IObject).
- The IObject is like a database.
- IObjects contain Table Objects.
- How to create a new Data Services object.
- How to create an OnWeb Table Object.
- How to create an OnWeb Column Object.
- How data is inserted within the Table Object cells.
- How to insert a Table Object into the object's IObject.
- How to test a Data Services object from within Designer.

Creating ODBC Data Services Objects 4

OnWeb allows you to access many back-end data sources to retrieve and record data. You can create a Data Services object called ODBC Data Services object, giving you direct access to any ODBC-based data source via SQL queries. When scripting an ODBC object, you have two query methods available to you: a straight SQL statement or a standard Collect subroutine. The latter allows you to use conditional logic to create an SQL query statement based on any number of variable conditions that you choose.

The purpose of this example is to introduce you to the OnWeb Data Source object. This example will use an OnWeb object to extract a list of suppliers from the Northwind MS Access database. Most of the objects you will be creating in OnWeb will be linked to an external data source.

In order to interact with the ODBC examples shown in this chapter, you will need to acquire the Access database file called Northwind_NM.mdb. This file is available on the OnWeb CD. You can use your own database file, as long as the appropriate DSN change is made and a valid SQL statement is included in the ODBC-compliant examples.

Creating a System DNS for the Northwind database

Before accessing a database with OnWeb, you must set up a System ODBC DSN.

▶ To create the System DSN for the database

1. On the OnWeb Server machine, open the **Control Panel**.
2. Open **Administrative Tools**.
3. Open **Data Sources (ODBC)**.
4. In the ODBC Data Source Administrator dialog box, click the **System DSN** tab.
5. Click **Add**.

6. Select **Microsoft Access Driver** from the list, and click **Finish**.
7. In the ODBC Microsoft Access Setup dialog box, type *Northwind* in the **Data Source Name** box.
8. Click **Select**.
9. In the Select Database dialog box, locate the Northwind_NM database, and click **OK**.
10. In the ODBC Microsoft Access Setup dialog box, click **OK**.
11. In the ODBC Data Source Administrator dialog box, verify that Northwind is in the **System Data Sources** list, and click **OK**.

Note: This is the correct procedure for Windows 2000; for other operating systems, the procedure may vary.

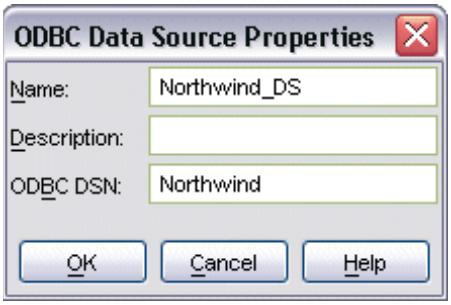
Creating an ODBC Data Source object

Before you create the ODBC-based Data Services object, you will need to create a new OnWeb ODBC-based Data Source object. This Data Source object will be the link between the ODBC-based Data Services object and the specified ODBC System DSN (data source name).

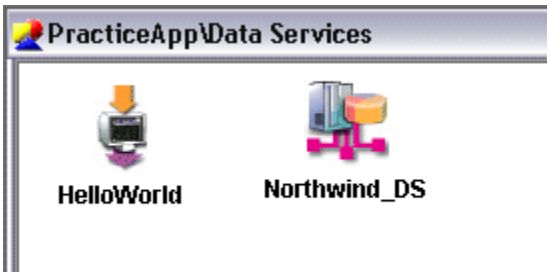
▶ To create an ODBC Data Source object

1. Double-click Data Services under the application name.
2. From the **New** menu, choose **ODBC Data Source** and click anywhere within the Data Services white-board to place your new ODBC Data Source object.

   The Create ODBC Data Source dialog box appears.
3. Click **Create Real Object** if it is not already selected. Click **OK** to continue.
4. In the ODBC Data Source Properties dialog box, type *Northwind_DS* in the **Name** box, and type *Northwind* in the **ODBC DSN** box.



5. Click **OK.**



Data Source user ID mapping

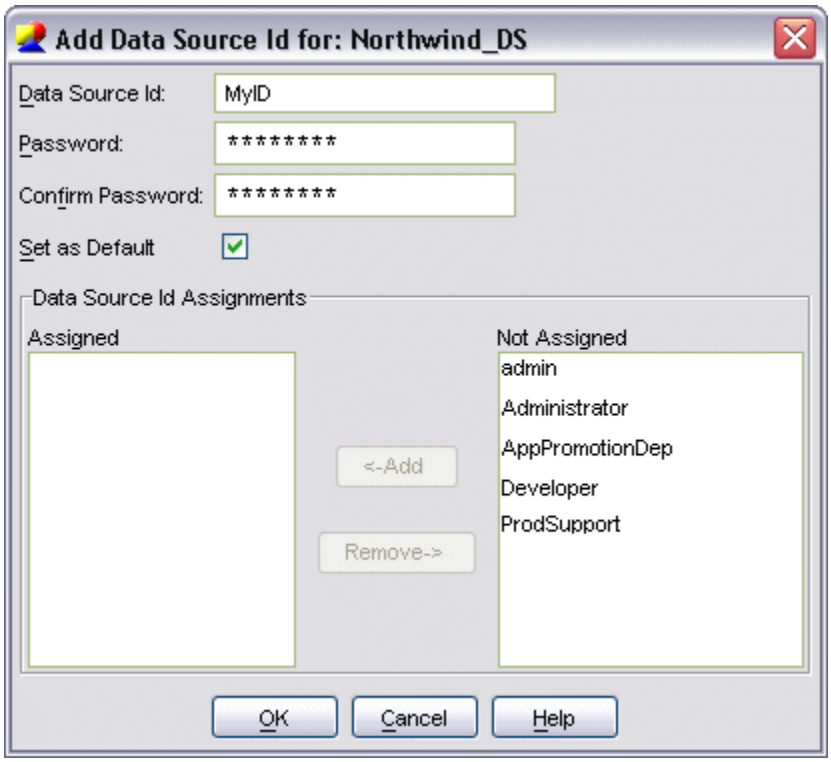
For this example, you are only using an Access database, so you do not need to worry about normal database security issues. However, you will often be performing the queries against a security-conscious database server, such as Microsoft's SQL Server. Before performing a query on these types of database servers, you usually must specify a valid user ID and password.

OnWeb allows you to map a specific user ID and password to a Data Source object. This user ID and password value will be used to log on to the database server before attempting to perform an SQL query operation.

▶ To map a user ID and password

1. Right-click the Northwind_DS object and choose **New DS User ID** from the menu. A dialog box appears, allowing you to specify a valid user ID and password.

2. Select **Set as Default** in order to inform the Data Source object to use these values.



Creating an ODBC Data Services object

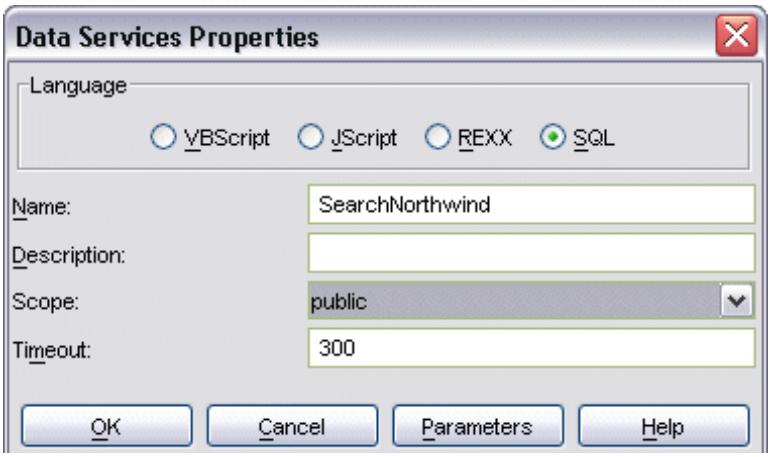
Now that you have a new ODBC-based Data Source object, you need to create a new Data Services object that will use this Data Source object.

▶ To create an ODBC Data Services object

1. Double-click Data Services under your application name to open the Data Services white-board.

2. From the **New** menu, choose **Data Services** and click anywhere within the Data Services white-board to place your new Data Services object.

3. Click **Create Real Object**, then click **OK** to continue. The Data Services Properties dialog box appears.
4. In the **Language** area, select **SQL**.
5. In the **Name** box, type *SearchNorthwind*.
6. Set the **Scope** value to public.

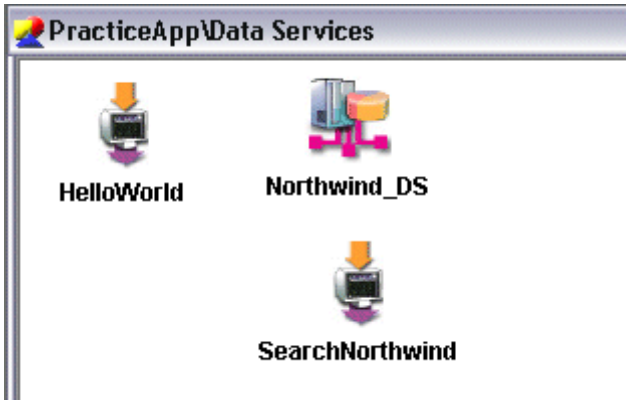


7. Click **OK** to create the object.

▶ To add parameters to the object

1. Right-click the SearchNorthwind object and choose Edit Parameters from the menu.
2. In the Edit Parameters dialog box, click **New**. A new line is added to the list.
3. In the **Name** column, type *Suppliers* and click **OK**.

   A new object appears in the white-board area.



Creating a relationship

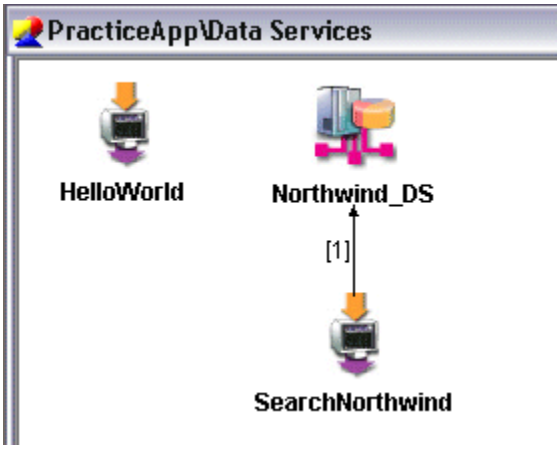
All OnWeb objects are connected together using relationships. Relationships are presented on the white-board as lines between two objects. They are used to denote two different things depending on the context that they are used in: the execution flow of an application (which objects get called in what order) and a link between any Data Services object and its respective Data Source object.

By creating a relationship between your new Data Services object and your ODBC-based Data Source object, you are informing the Data Services object to perform all of its queries against that particular ODBC-based Data Source object (which in this case, will use the Northwind ODBC System DNS).

▶ To create a relationship

1. Click the **Relationship** button  on the toolbar.
2. On the Data Services white-board, click anywhere inside the SearchNorthwind Data Services object and drag the cursor over to the Northwind_DS ODBC Data Source object that you created.

   A relationship line will appear, connecting the two objects together.



Now all SQL queries will be parsed to the Northwind_DS Data Source object. If at a later time you want to perform your SQL queries on a different ODBC database, it is as simple as creating a new ODBC Data Source object. Then all that is needed is to reconnect your relationship to your new Data Source object. There is no need to rewrite the SQL logic contained within your Data Services object.

Adding an SQL statement

You will now need to write the actual SQL statement that will perform the query.

▶ To add an SQL statement to the ODBC Data Services object

1. Right-click the SearchNorthwind Data Services object and choose **Edit Script** from the menu.
2. In the Create New Script dialog box, select **Use empty script**, and click **OK**.
3. In the script editor, enter the following code:

```
SELECT CompanyName, City FROM Suppliers
```

4. Save the script and close the script editor.

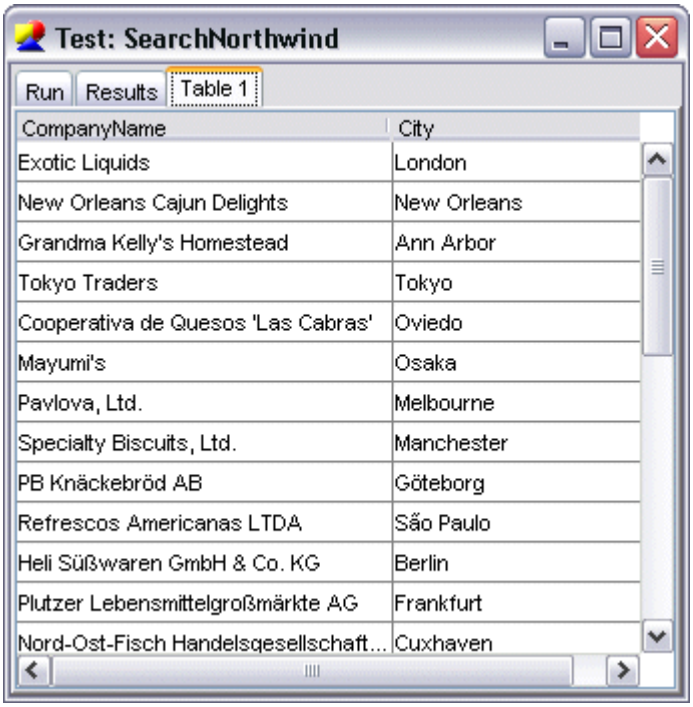
Testing the script

You should now test the script.

▶ To test your script

1. Right-click the SearchNorthwind object and choose **Test** from the menu.
2. In the Test dialog box, click **View Results**.

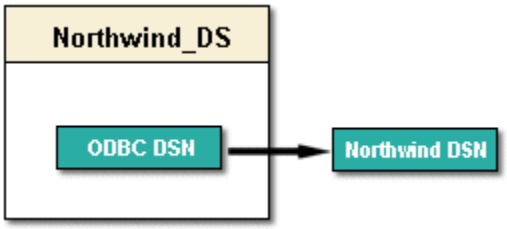
The following results should appear:



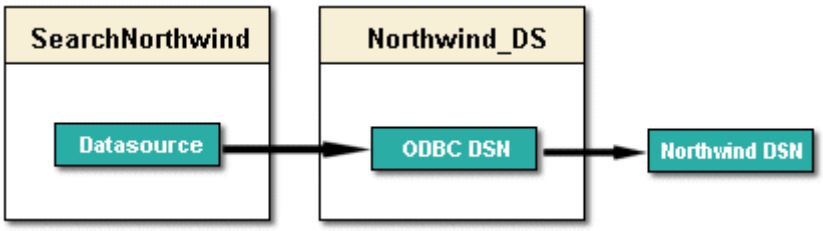
Interpreting the example

First, you create an OnWeb Data Source object, since the Data Services object is going to be collecting its data externally.

You create an OnWeb ODBC Data Source object, and specify a predetermined System DSN for it to use. This is the data source link to the Northwind_NM Access database file. When an SQL query is made through this Data Source object, it is routed through the System DSN entry (Northwind) to establish a physical link to the database file.



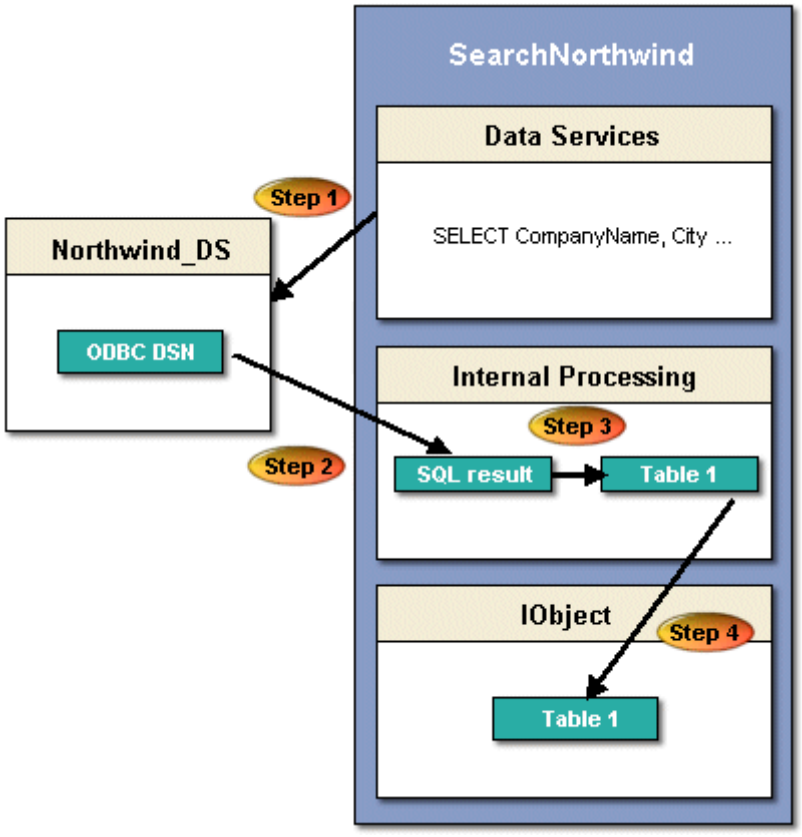
After creating the ODBC-based Data Source object, you are ready to create an ODBC-based Data Services object, which contains the actual SQL logic to be executed.



You are now ready to add some business logic. For simplicity's sake and the fact that you are not performing any advanced queries, you decide to write the logic using a standard SQL statement.

After you complete the rule logic, you test it in order to make sure everything executes correctly. The Data Services object returns its IObject with a single Table Object in it, which contains the SQL result set. Unlike the previous example, where you had to actually create your own Table Object, fill it with data, and insert it within the Data Services object's IObject, OnWeb has already done this for you! ODBC-based Data Services objects are the only objects for which OnWeb automates this process for you. Your only concern, when writing the collect logic, is to make sure you form a valid SQL query statement, and OnWeb will do the rest.

The following diagram illustrates what the SearchNorthwind Data Services object is doing.



Step 1: The SQL select statement is sent out to the OnWeb Northwind_DS data source.

Step 2: The returning result set is passed back to the SearchNorthwind object.

Step 3: OnWeb creates a new OnWeb table based on the incoming SQL result set.

Step 4: OnWeb automatically inserts the newly created table object into the Information Object.

Summary

You have now learned how to specify an incoming parameter into an ODBC-based object, and how you can easily and quickly incorporate it into SQL statements. Parameters are a very important part of any business logic, and you will use them more extensively in upcoming chapters.

Before continuing to the next chapter, you should be familiar with the following points:

- An ODBC-based Data Source object is OnWeb's link to an external database.
- How to create an ODBC Data Source object.
- How to link the Data Source object to a Data Services object.
- How to add an ODBC-based object parameter.
- How to use an incoming parameter in an SQL language statement.
- How to write a standard SQL query using an ODBC-based object.
- How to test an ODBC-based object.
- OnWeb automatically generates a table from the incoming results and inserts it into the IObject.

Creating Terminal Data Services Objects 5

In this chapter you will create a Terminal Data Services object for the Landmark host AS/400®.

OnWeb allows you to access host AS/400 systems to retrieve and record data. You can create a Terminal Data Source object, which will give you direct access to a terminal-based session.

A terminal-based Data Source is designed to work with a terminal-based session, similar to navigating an AS/400 host with a Telnet client. Any data from the host can easily be collected and returned in the Data Services object's IObject. You can issue commands and type syntax programmatically to the terminal session, as well as extract any portion of the display from the terminal screen. This allows you to create some advanced automated sequences of actions, such as login, host screen navigation, data entry, recording, and logoff.

This example will be similar to your ODBC-based supplier search. Instead of searching a database, however, you will search the Landmark AS/400 system for supplier information. This will introduce you to OnWeb Terminal Data Source object and the Data Services object's unique host navigation syntax.

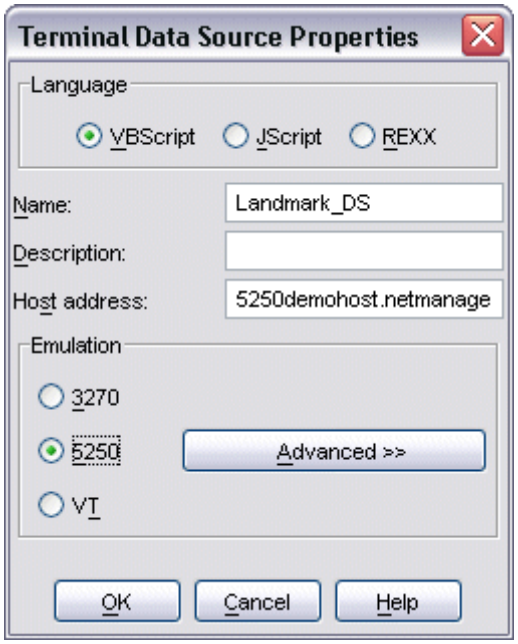
Creating a Terminal Data Source object

As in the previous chapter, you will create a new Data Source object. However, instead of creating an ODBC-based Data Source object, you will build a Terminal Data Source object. Similar to the ODBC Data Source object, you will fill out some basic properties, the most important being the IP address of the terminal host. You will also write logon and logoff scripts within the Data Source object.

Note: In order to establish a connection, the host must support the IP protocol. If the remote host does not support IP, you will need to use a gateway like Microsoft's SNA server to do so.

▶ To create a Terminal Data Source object

1. Double-click Data Services under your application name to open the Data Services white-board.
2. From the **New** menu, choose **Terminal Data Source** and click anywhere on the white-board to place your new Terminal Data Source object.
3. In the Create Terminal Data Source dialog box, select **Create Real Object**, and click **OK**.
4. In the Terminal Data Source Properties dialog box, select **VBScript**, and type *Landmark_DS* in the **Name** box.
5. In the **Host address** box, type *5250demohost.netmanage.com*.
6. In the Emulation section, select **5250**.



7. Click **OK** to create the object.

Adding the script

Now that you have specified the Data Source properties, you need to write the actual script that will initiate the connection to the remote host. You will need Logon and Logoff scripts.

▶ To create a logon script

1. Double-click the Landmark_DS object on the white-board.
2. In the Edit File dialog box, select Logon from the list, and click **OK**.
3. In the Create New Script dialog box, select **Use empty script** and click **OK**.
4. Type this script in the script editor:

```
Sub Logon()
    Connect
    WaitFor "Sign On"
End Sub
```

5. Save the script and close the script editor.

▶ To create a logoff script

1. Double-click the Landmark_DS object on the white-board.
2. In the Edit File dialog box, select Logoff from the list, and click **OK**.
3. In the Create New Script dialog box, select **Use empty script** and click **OK**.
4. Type this script in the script editor:

```
Sub Logoff()
   Disconnect
End Sub
```

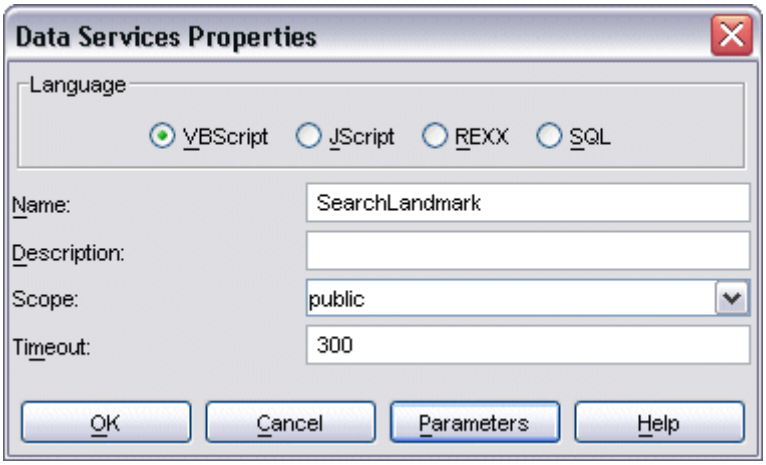
5. Save the script and close the script editor.

Creating a Terminal Data Services object

Now that you have a new Terminal Data Source object, you can create a new Data Services object that will use this Data Source object.

▶ To create a Terminal Data Services object

1. Double-click Data Services under your application name to open the Data Services white-board.
2. From the **New** menu, choose **Data Services** and click inside the white-board to place your new Data Services object.
3. In the Create Data Services Object dialog box, select **Create Real Object**, and click **OK**.
4. In the Data Services Properties dialog box, select **VBScript**, and type *SearchLandmark* in the **Name** box.
5. Select Public from the **Scope** list.



6. Click **OK** to create the object.

▶ To add parameters to the object

1. Right-click the SearchLandmark object and choose Edit Parameters from the menu.

2. In the Edit Parameters dialog box, click **New**. A new line is added to the list.

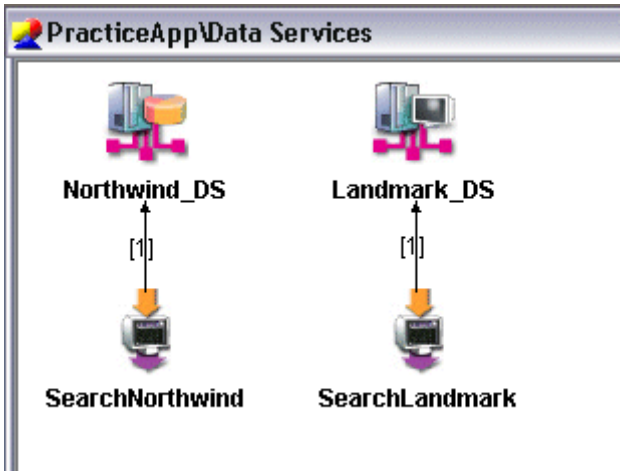
3. In the **Name** column, type *Suppliers* and click **OK**.

   Now, you need to link this Terminal-based Data Services object with the appropriate Terminal-based Data Source object. Like you did in the previous example, you need to use a **Relationship** to connect the two objects together.

4. Click the **Relationship** button on the toolbar.

5. In the Data Services white-board, click anywhere inside the SearchLandmark Data Services object and drag the cursor over to the Landmark_DS Data Source object that you created.

   A relationship line will appear, connecting the two objects together.



Adding the script

You now need a script that will navigate the Landmark host, search for the specified supplier information, and return the first 16 titles it finds. To do this, you'll need to write a script that does the following:

- Define a Collect subroutine.
- Write navigation commands for the host.
- Handle an error page that may occur after login to the host.
- Define a table object to hold the supplier information.
- Define a column for the table.
- Define a name for the column.
- Define a column type for the column (string).
- Add the column object to the table's schema.
- Define rows for the column object (one for each line of data).
- Assign the text to the column (specifying the starting point and length of the text string).
- Add the table to the IObject.

The following VBScript satisfies these requirements:

```
Sub Collect(Datasource)
    Datasource.Type "USER01"
    Datasource.Press "Tab"
    Datasource.Type "USER01"
    Datasource.Press "Tab"
    Datasource.Type "landmark"
    Datasource.Press "Enter"

    Index = Datasource.WaitFor(Array("allocated","===>"))
    If Index = 0 Then
        Datasource.Press "Enter"
        Datasource.WaitFor "===>"
    End If

    Datasource.Type "1"
    Datasource.Press "Enter"
    Datasource.WaitFor "===>"
```

```
    Datasource.Type "5"
    Datasource.Press "Enter"
    Datasource.WaitFor "===>"

    Datasource.Type "3"
    Datasource.Press "Enter"
    Datasource.WaitFor "Position"

   Set tTable = Onweb.CreateObject _
     ("Onweb.IComponent.Table","LandmarkTable")
   Set cCol = Onweb.CreateObject("Onweb.ColumnDef")
    cCol.Name = "Title"
    cCol.Type = "String"


    tTable.Schema.Add cCol

    For RowIndex = 7 To 15
     tTable.InsertRow
     tTable("Title") = _
        Datasource.Screen.Text(CInt(RowIndex), 21, 25)
    Next

    IObject.Contents.Add tTable
End Sub
```

▶ To add the script to the Data Services object

1. Right-click the SearchLandmark Data Services object and choose **Edit Script** from the menu.
2. In the Create New Script dialog box, select **Use empty script**, and click **OK**.

   The script editor window opens.
3. Enter the script as shown above.
4. Save the script, and close the script editor.

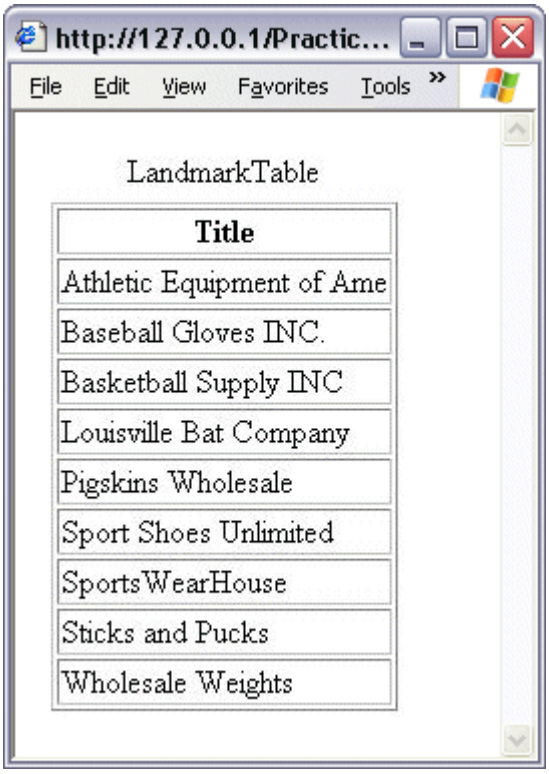
Testing the script

You should now test the script.

▶ To test your script

1. Right-click the SearchLandmark Data Services object and choose **Test** from the menu.
2. In the Test dialog box, click **Test In Browser**.

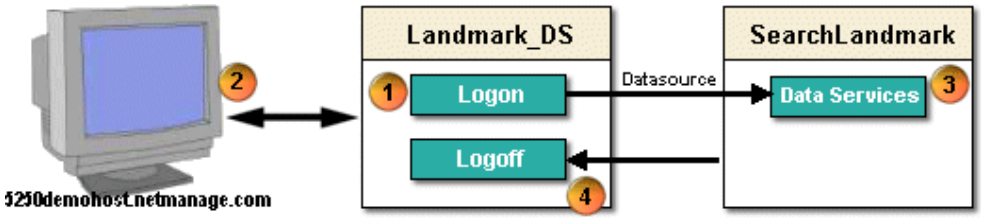
   The following result should appear:



Note: You may notice that it takes longer to execute a Terminal-based object. Remember, there are many variables that affect the execution speed of a terminal-based object, such as Internet connection speed and the relative congestion of the host. If the object does not complete its execution within the "Timeout" period, it will cancel itself.

Interpreting the example

Before you examine the example code, it is important that you understand the sequence of events that occur when you are working with Terminal-based Data Services objects and their respective Data Source objects.



When you test the SearchLandmark Data Services object:

1. The logon script of the Data Source object is executed.
2. When the logon script initiates the Connect method, OnWeb establishes a connection between the Data Source object and the remote host.
3. Once the logon script has finished processing, it passes a reference of the Data Source object to the Data Services object's script. The SearchLandmark Data Services object is then executed.
4. Once the script has finished, executing control is then passed back to the Data Source object and the logoff script is executed. This routine proceeds to disconnect the connection to the remote host.

You have now seen an overview of how the example ran; now you will examine each script that was just executed.

For the first part of the example, you create a Terminal-based Data Source object that acts as the gateway to the remote host, Landmark. After entering some default rule properties, you create a logon script that would programmatically initiate the connection to the remote host and wait for the first screen to appear.

```
Connect

WaitFor "Sign On"
```

After executing the *Connect* method, you call the *WaitFor* method, which causes the script execution to suspend until the string “Sign On” is found on the screen or until the default 30 seconds has expired.

Note: The Data Source object's WaitFor method examines the incoming data from the host waiting for the specified string to appear. If the string does not appear within the specified timeout period, the method returns with a fail code. You should always use the WaitFor method before attempting to type anything to the host screen to make sure you have successfully navigated to the correct screen.

After the logon script has completed, a reference to an OnWeb Data Source object (representing the connection to the remote host) is passed on to the script of the calling SearchLandmark Data Services object. When your Terminal-based Data Services object is connected to a Terminal-based Data Source object, you must make sure to specify a variable for the connected Data Source object as an incoming function parameter. In this example, you name the parameter *Datasource*.

```
Sub Collect(Datasource)
```

Next, you perform some basic terminal navigation in order to get to the supplier screen. You use the incoming Data Source object's methods to programmatically perform the host navigation.

```
Datasource.Type "user01"
Datasource.Press "Tab"
Datasource.Type "user01"
Datasource.Press "Tab"
Datasource.Type "landmark"
Datasource.Press "Enter"
Datasource.WaitFor "COMMAND?"
```

Type, *Press*, and *WaitFor* are the three most commonly used methods for remote navigation that you will use when working with Terminal-based Data Services objects. You can read about the other data source methods in the OnWeb Scripting Help.

It is important that you determine whether Landmark has returned the expected terminal screen (main menu screen) or if Landmark is already being accessed by USER01. If there is indeed someone already logged on as USER01, an information screen appears, and you will need to enter a "RETURN" in order to view the main menu screen.

If you use a separate terminal emulator to monitor your progress (and you should always do this), you can see whether or not the extra information screen appears. Note that you can force the screen to appear by logging on twice concurrently. Based on this information, you can create an algorithm to determine which terminal screen was returned. If the string "allocated" is

found on the screen, you just need to enter a “RETURN” to get back to the main menu screen.

After performing some additional navigation, you reach the suppliers screen.

```
Index =
Datasource.WaitFor(Array("allocated","===>"))
If Index = 0 Then
   Datasource.Press "Enter"
   Datasource.WaitFor "===>"
End If
```

Next, you build an OnWeb Table Object and insert a single Column Object, which will be used to store the supplier information.

Since Landmark contains the supplier information on nine separate lines, you need to use a basic loop to iterate through each row on the terminal screen that contains a supplier and extract the line and store it in a separate row within the Table Object.

```
For RowIndex = 7 To 15
   tTable.InsertRow
   tTable("Title") = Datasource.Screen.Text _
   (CInt(RowIndex),21,25)
Next
```

You now have a table with nine separate rows, each containing a supplier. Again, since the rule will only return its own IObject, you need to insert the Table Object within the rule's IObject collection.

```
IObject.Contents.Add tTable
```

Once the collect script finishes its execution, control is then transferred back to the Landmark_DS Data Source object and the logoff script is initiated.

The only line of script contained within the logoff subroutine, Disconnect, essentially just closes the physical connection to the remote host. While this subroutine is optional, it is considered good programming practice to clean up all unused host connections.

Summary

You have now completed the Terminal-based example. You examined how a Terminal-based object operates, and saw it in action. You created your own Terminal-based Data Source object and the simple logon and logoff scripts to manage a connection to a remote host. You also examined how OnWeb performs navigation programmatically.

Before continuing to the next chapter, you should be familiar with the following points:

- How a Terminal-based Data Services object works.
- How to create a Terminal-based Data Source object.
- How to write a Data Source object logon script.
- How to write a Data Source logoff script.
- How to access the data out of the parameter collection.
- How to navigate the terminal host programmatically.
- How to extract strings from the Data Source object.
- How to test a Terminal-based Data Services object.
- Understand the flow of what is happening when a Terminal-based Data Services object is executed.

Creating Business Logic Objects 6

OnWeb's Business Logic is used to manipulate and/or unite the data collected from any number of Data Services objects that reside within OnWeb's Data Services tier. Business Logic allows the OnWeb application developer to encapsulate and modify data from individual Data Services objects. This logic can be used to help form a larger, coherent business rule or application.

Remember the internal OnWeb model, where the Data Services objects reside in the Data Services tier because they perform the actual data collection. Objects that reside within the Business Logic tier are called Business Logic objects. Relationships are created from Business Logic objects to existing Data Services objects residing within the Data Services tier. Relationships can also be created to link other existing Business Logic objects. In the Business Logic and User Services tiers, Relationships are used to denote order of execution.

In this chapter you will create a new Business Logic object. This Business Logic object will invoke your previously created SearchNorthwind object and receive its IObject, which will contain the result of your search. Your Business Logic object will then proceed to sort the suppliers in ascending order.

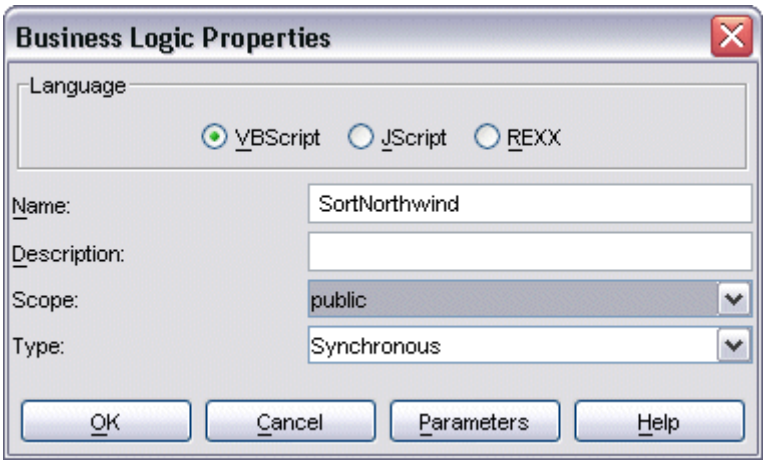
While this example is straightforward, it does demonstrate the tremendous scalability options available to developers. You could just as easily create a Business Logic object that lists only high volume suppliers, then all an administrator would have to do is to link to this rule to award bonuses, and when the bonuses have been awarded, just unlink it -- simple and flexible.

Creating a Business Logic object

For this example, you will be working within the Business Logic white-board. This is where all your Business Logic objects reside.

▶ To create a Business Logic object

1. Double-click Business Logic under your application name to open the Business Logic white-board.
2. From the **New** menu, choose **Business Logic** and click anywhere on the white-board to place your new Business Logic object.
3. In the Create Business Logic Object dialog box, select **Create Real Object**, and click **OK**.
4. In the Business Logic Properties dialog box, select **VBScript**, and type *SortNorthwind* in the **Name** box.
5. From the **Scope** list, select Public.



6. Click **OK**.

To add parameters

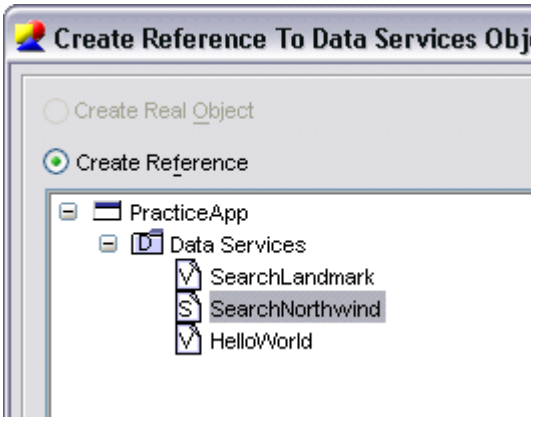
1. Right-click the object and choose Edit Parameters from the menu.
2. In the Edit Parameters dialog box, click **New**. A new line is added to the list.
3. In the **Name** column, type *Suppliers* and click **OK**.

For our example, your Business Logic object will be connected to the SearchNorthwind Data Services object, which will collect the supplier data.

First, you must create references to the SearchNorthwind Data Services object on the Business Logic white-board.

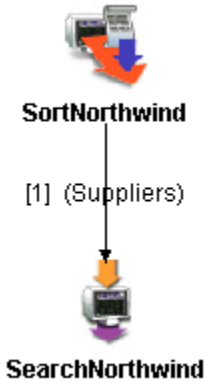
To connect objects

1. From the **New** menu, choose **Data Services** and click anywhere on the white-board to create a reference to a Data Services object.
2. In the Create Reference To Data Services Object dialog box, select **Create Reference**.
3. Expand the items in the tree-list area, and select SearchNorthwind, then click **OK**.



4. Click the **Relationship** button on the toolbar.
5. Click anywhere inside the SortNorthwind Business Logic object and drag the cursor to the SearchNorthwind Data Services object.
6. In the Parameter Mappings dialog box, ensure 'Suppliers' maps to 'Suppliers', and click **OK**.

A relationship line will appear, connecting the two objects.



The Parameter Mappings dialog box allows you to match up the parameter values to each other. When connecting two objects together with a Relationship you are actually specifying the executing order for your objects.

In our example, when a user executes your SortNorthwind Business Logic object, OnWeb will first follow the execution order specified by the connected Relationships. In this case, the SearchNorthwind Data Services object will execute first and return its IObject when it is finished. Then, since there are no other connected Relationships, the SortNorthwind Business Logic object will execute its own business logic.

Now, you have to remember that the user is running your SortNorthwind Business Logic object directly and not the SearchNorthwind Data Services object. So, when the user passes the *Suppliers* value to search for, your SortNorthwind Business Logic object needs to pass the relevant parameters on to the SearchNorthwind Data Services object, and this is done through the Parameter Mappings dialog box.

You may have noticed that we said “relevant” parameters; this is because your Business Logic objects could possibly expect any number of parameter values. Yet when calling your SearchNorthwind Data Services object, your Data Services object is only expecting a single *Suppliers* parameter. So, by using the Parameter Mappings dialog box you have the option of specifying which of your possible multiple parameters to pass on.

Adding the script

You now need a script that will accept data from the SearchNorthwind object, and then sort it alphabetically.

▶ To add a script to the Business Logic object

1. Right-click the SortNorthwind Business Logic object and choose **Edit Script** from the menu.
2. In the Create New Script dialog box, select **Use empty script**, and click **OK**.
3. In the script editor, enter the following script:

```
Sub Refine()
   Set ioObject = IObject.Contents.Item(0)
   Set tblTable = ioObject.Contents.Item(0)
   Set tblSortedTable = _
         Onweb.TableServices.Sort(tblTable,0)
   IObject.Contents.Add tblSortedTable

End Sub
```

4. Save the script, and close the script editor.

Testing the script

You should now test the script.

▶ To test the script

1. Right-click in the SortNorthwind Business Logic object and choose **Test** from the menu.

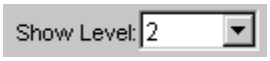
2. In the Test dialog box, click **View Results**.

   The following results appear:

If you compare these results with the ones received in the Data Services - ODBC example, you will notice that the exact same suppliers were found. The only difference is that the SortNorthwind Business Logic object results are sorted in ascending order, as expected.

Displaying all objects on the white-board

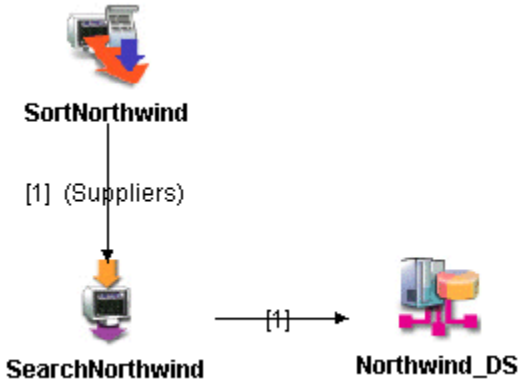
Let's examine a handy feature of Designer that allows you to view the complete execution order of your objects. The Show Level drop-down list on the toolbar allows you to control the depth to which the execution tree is displayed.



Change the level to 2, since our SortNorthwind Business Logic object only goes one level deeper.

You should now see the complete execution order for your SortNorthwind Business Logic object. The SortNorthwind Business Logic object calls the

SearchNorthwind Data Services object, which is linked to the Northwind_DS Data Source object.



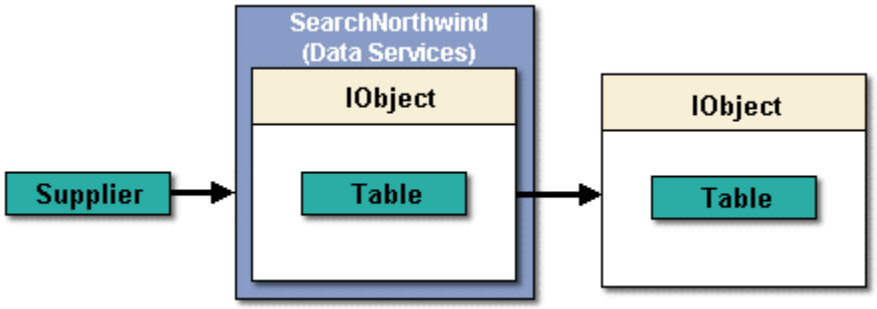
Interpreting the example

Now let's go back through the example and try to see exactly what the SortNorthwind Business Logic object is doing when it's executing.

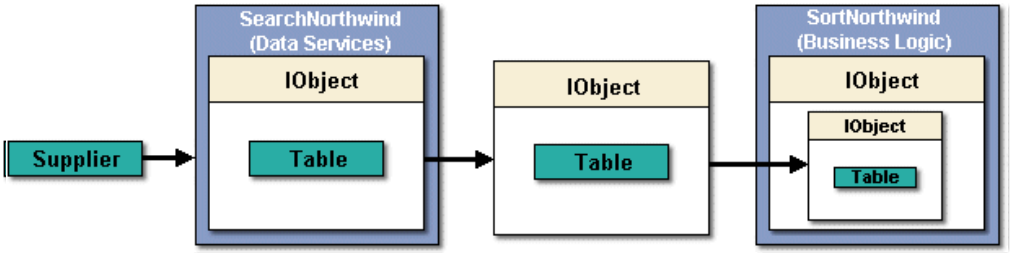
When you run the SortNorthwind Business Logic object, the first task it accomplishes is to capture the incoming parameter, *Suppliers*. After acquiring a reference to all incoming parameters, it proceeds to pass the appropriately mapped parameters to the SearchNorthwind Data Services object.

After the SearchNorthwind Data Services object has executed successfully, it returns its own IObject (full of suppliers) to the SortNorthwind Business Logic object. Remember from the Data Services - ODBC example, that the returning IObject contains the result Table Object from the Suppliers SQL logic you generated.

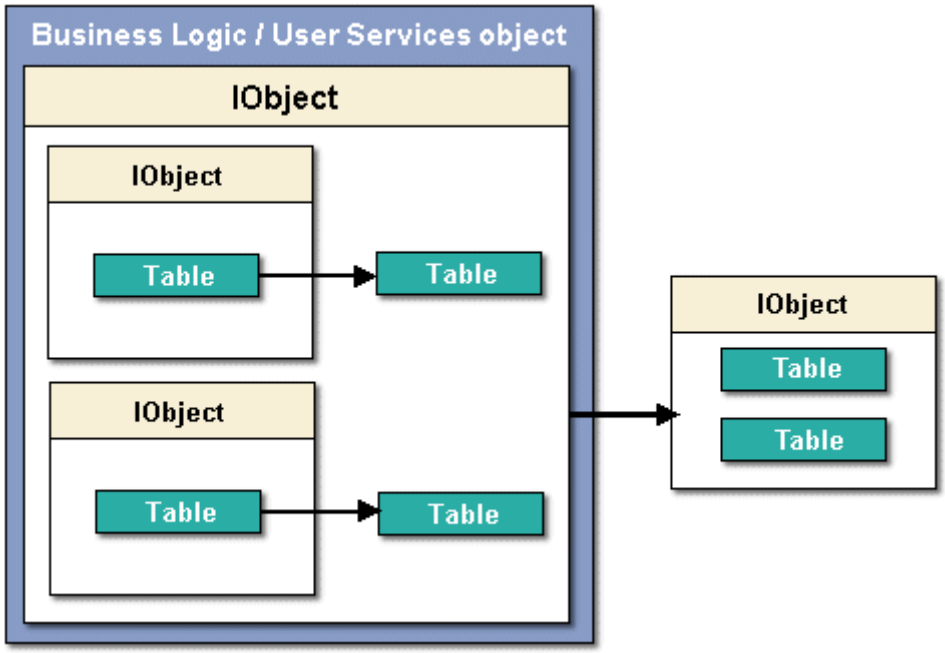
At this point in the SortNorthwind execution, the returning IObjects from the SearchNorthwind Data Services object have been inserted into the SortNorthwind Business Logic object's own IObject.



Now one of two actions can occur depending on whether the presence of a script exists within the Business Logic object. If a script exists, it will execute its business logic and then return an IObject. When a script is present, it is important to note that it is your responsibility to programmatically remove the Table Objects from each incoming IObject and insert the ones you want returned into the Information Builder's IObject. However, if a script does not exist, OnWeb will automatically reference all the Table Objects from each incoming IObject and place them within the Information Builder's IObject.



You did not include any business logic script logic for this example. OnWeb performs the action of copying each Table Object into the Information Builder's IObject automatically. While business logic script logic is not discussed in this guide, it is important that you gain a basic understanding of what is happening to the Table Objects before continuing to the next chapter.



The last step in the Object's execution is the act of returning its own IObject.

If your Business Logic has no script, OnWeb will automatically create a copy of each table that exists in the Information Objects that were returned from the executed component rules and insert them into the Business Logic object's own IObject.

Note: Only Table objects are returned in the IObject. The two IObjects will be removed before the Business logic object's IObject is returned.

Summary

In this chapter, you used the ODBC-based and Terminal-based objects you created in previous chapters. You saw that the Business Logic object accepted an incoming parameter, and passed it to both of its connected objects (Data Services objects in this case) to perform their respective queries. You examined how a Business Logic object can connect multiple objects together to perform business logic.

Before continuing to the next chapter, you should be familiar with the following points:

- What Business Logic object is used for.
- How to create a new Business Logic object.
- How to insert a new parameter.
- How to reference a Data Services object.
- How to connect a Relationship to a reference object.
- The returning reference IObject's are inserted into the calling object's IObject.
- If a Business Logic object script is not present, OnWeb will automatically transfer the Table Objects from the incoming IObjects into the Business Logic object's own IObject.
- Understand why the Table Objects must be transferred from the incoming IObjects to the calling object's returning IObject.

Creating User Services Objects 7

OnWeb's User Services tier is where the presentation logic will normally reside. This tier is where you will create your User Services objects, which will be used to format and display your data results to the user. User Services objects allow you to write straight HTML script, using special OnWeb tags with which to substitute data, or to create actual presentation logic to dynamically format your HTML output.

In this chapter, you will create two sample User Services objects. One User Services object will be written using straight HTML and the other will dynamically generate an HTML output string. Both User Services objects will be linked with a Relationship to both the SortNorthwind Business Logic object and the SearchLandmark Data Services object.

Creating a User Services object

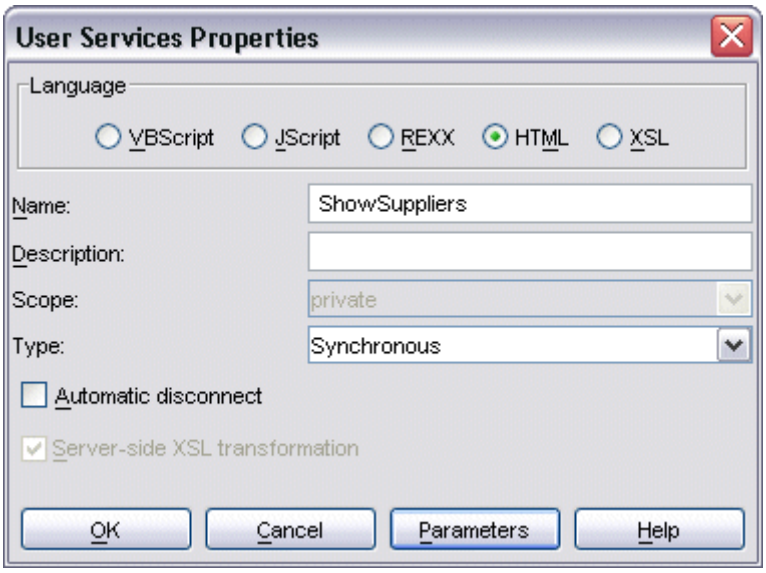
For this example, you will be working within the User Services white-board. This is where all your User Services objects reside.

▶ To create a User Services object

1. Double-click User Services under your application name to open the User Services white-board.
2. From the **New** menu, choose **User Services** and click anywhere within the white-board to place your new User Services object.

   The Create User Services Object dialog box appears.
3. Click **Create Real Object** and then click **OK** to continue.

   The User Services Properties dialog box appears, allowing you to enter the various property values for your new User Services object.
4. In the **Language** area, select **HTML**.
5. In the **Name** box, type *ShowSuppliers*.



Since your users will be executing this User Services object and passing the Suppliers parameter, you need to create a new incoming parameter for this User Services object as well.

6. Click **Parameters**.

7. In the Edit Parameters dialog box, click **New** to add a new parameter. A new line is added to the list.

8. In the **Name** column, type *Suppliers*.

9. Click **OK** two times. The ShowSuppliers object appears on the white-board.

This new ShowSuppliers User Services object will be connected to both the SortNorthwind Business Logic object and the SearchLandmark Data Services object. The order that you connect your Relationships will determine the execution order of the objects (unless the objects Type property is set to asynchronous, which will cause all connected objects to be executed simultaneously).

▶ To add reference to a Business Logic object

1. From the **New** menu, choose **Business Logic** and click anywhere on the User Services white-board.

   The Create Reference To Business Logic Object dialog box appears. Since you selected a new Business Logic object, you can only select one of your existing Business Logic objects residing in the Business Services tier.

2. Select **Create Reference**.

3. Expand the Business Logic tree view.

4. Select the SortNorthwind Business Logic object and click **OK**.

   A reference to the SortNorthwind Business Logic object appears on the User Services white-board.

Now you need to connect your ShowSuppliers User Services object to the SortNorthwind Business Logic object with a Relationship. This will denote that whenever the ShowSuppliers User Services object is executed, it will first go to the SortNorthwind Business Logic object and pass any mapped parameters to it.

▶ To connect the objects

1. Click **Relationship** on the toolbar.

2. Click inside the ShowSuppliers User Services object and drag the cursor over to the SortNorthwind Business Logic object reference.

   As you have seen before, since both objects contain parameters, the Parameter Mappings dialog box automatically appears.

3. Click **OK** to accept the default mapping displayed.

Now, you will add another reference object to the User Services white-board and connect it to the ShowSuppliers User Services object.

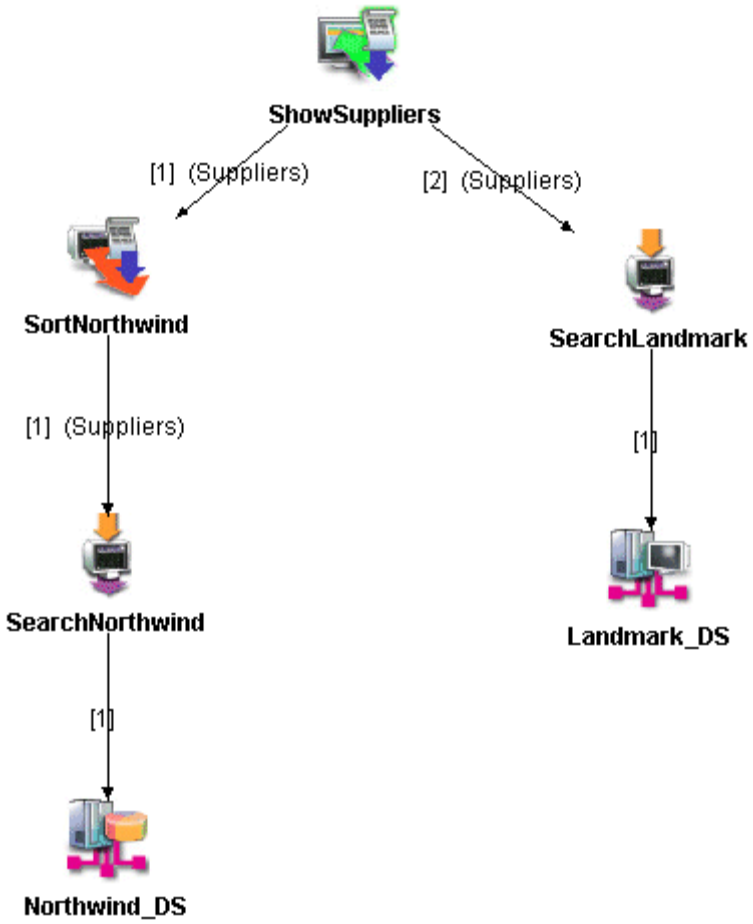
▶ To add another reference object

1. Add a reference to the SearchLandmark Data Services object to the User Services white-board, as you did with the SortNorthwind Business Logic object.

2. Create a Relationship to connect the ShowSuppliers User Services object and the SearchLandmark Data Services object. Keep the default parameter mappings.

In the **Show Level** list on the toolbar, select 3 to see the complete execution order of your ShowSuppliers User Services object.

Your User Services white-board should now look similar to the following picture, depending on how you have organized your objects.



You can see from this view that when your ShowSuppliers User Services object is executed, it will first call the SortNorthwind Business Logic object, passing it the supplied Suppliers parameter. The SortNorthwind object will then immediately call the SearchNorthwind Data Services object, before executing its own script again, passing the Suppliers parameter value.

When the SortNorthwind Data Services object is finished executing, it returns its own IObject (containing all the suppliers) back to the calling SortNorthwind Business Logic object. The SortNorthwind Business Logic object then proceeds to execute its script (sorts the incoming suppliers list) and returns only its own IObject (containing a sorted supplier list).

Once the ShowSuppliers User Services object receives the returning IObject from the SortNorthwind Business Logic object, it proceeds to execute the next object within its execution order (dictated by the connecting Relationships).

The SearchLandmark Data Services object is now called and the Suppliers parameter value is passed to it. When it finishes executing, it also returns its own IObject to ShowSuppliers.

Now that all connecting Relationships have been executed, the ShowSuppliers User Services object executes its own script or HTML display logic. In this example, ShowSuppliers will just display the first five suppliers from both queries using straight HTML with the use of OnWebImport tags.

Using OnWebImport tag

If you plan to write your User Services objects using straight HTML, you will probably make use of the OnWebImport tag. This tag allows you to easily extract any data from any Table Objects contained within the User Services object's own IObject. The OnWebImport tag allows you to define how you would like the tables of data to be presented in HTML.

The standard OnWebImport syntax is as follows:

```
OnWebImport(table, row, column)
```

For example:

- `OnWebImport()` inserts all of the data contained within all of the tables.
- `OnWebImport(2)` inserts all the data contained within the second table.
- `OnWebImport(1, ,3)` inserts the rows of data found in column 3 from table 1.
- `OnWebImport(1,2,3)` inserts cell data found at row 2, column 3, of table 1.

Adding the script

You now need a script that will accept data from the SortNorthwind and SearchLandmark objects, and present it using the OnWebImport tags.

▶ To add the script to the User Services object

1. Right-click the ShowSuppliers User Services object and choose **Edit Script** from the menu.

2. In the Create New Script dialog box, select **Use empty script**, and click **OK**.

3. Enter the following script:

```
<HTML>
<BODY BGCOLOR=LIGHTBLUE>
<CENTER>
<H2>Top 5 Suppliers From NorthWind</H2>
<HR>
OnWebImport(1,1,1)<BR>
OnWebImport(1,2,1)<BR>
OnWebImport(1,3,1)<BR>
OnWebImport(1,4,1)<BR>
OnWebImport(1,5,1)<BR>
<HR>
<H2>Top 5 Suppliers From LandMark</H2>
<HR>
OnWebImport(2,1,1)<BR>
OnWebImport(2,2,1)<BR>
OnWebImport(2,3,1)<BR>
OnWebImport(2,4,1)<BR>
OnWebImport(2,5,1)<BR>
</CENTER>
<HR>
</BODY>
</HTML>
```

4. Save the script, and close the editor.

Testing the script

You should now test the script.

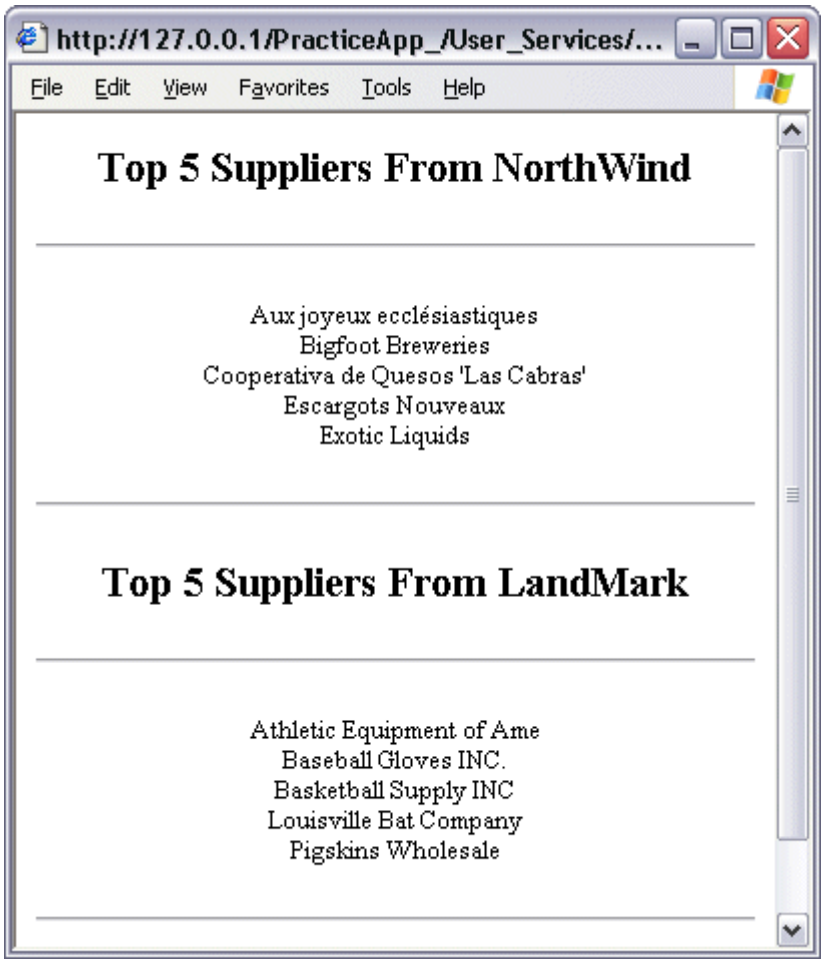
▶ To test the script

1. Right-click the ShowSuppliers User Services object and choose **Test** from the menu.

2. In the Test dialog box, click **Test in Browser**.

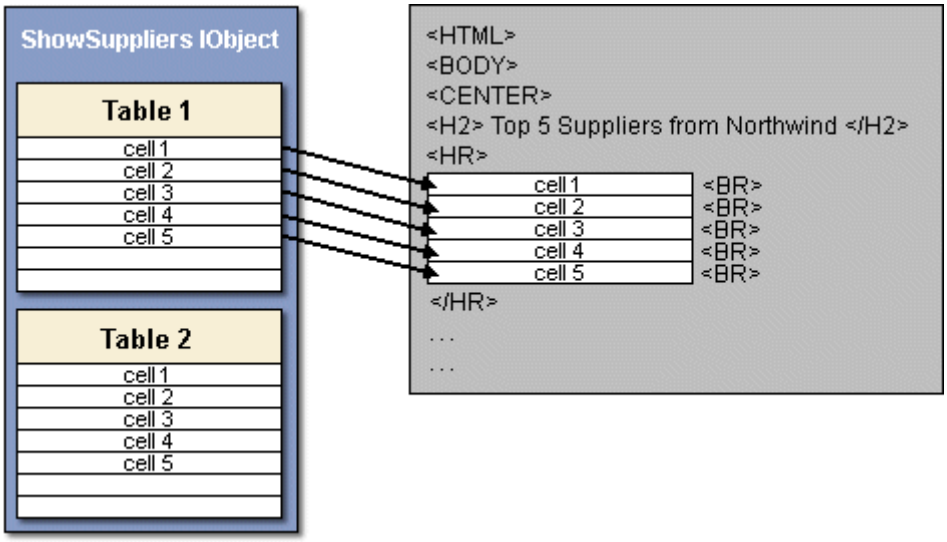
Note: When you test a User Services object with the **View Results** button, only its connected Data Services and Business Logic objects will be executed. If you wish to test your presentation layer, you will need to test it with the **Test In Browser** button.

The browser will display the following page:



Interpreting the example

If you examine the information layout in the browser window, you can see that OnWeb displays the HTML code as well as the first five suppliers returned.



When a User Services object is ready to display its data from an HTML script, OnWeb performs an extra step first. It translates all of the OnWebImport tags that exist within the HTML script into the actual table cell values returned from the User Services object's IObject.

After OnWeb has finished executing the User Services object, it sends the new HTML code to the browser.

Creating a User Services object using dynamic HTML

OnWeb also allows you to use dynamic HTML code to display the host data in the browser. This is called presentation script.

In the HTML code in the previous example, you used OnWebImport tags to get the first five table cell values and displayed these values in the browser using standard HTML. This is acceptable if you know exactly what is to be displayed; however, what if you don't know the exact number of rows that are going to be returned?

In the User Services object, you can write a script to cycle through the first five table cells and send the values as HTML strings to the browser. This method allows you to modify the actual values inside the HTML strings as much as you like.

Creating a second User Services object

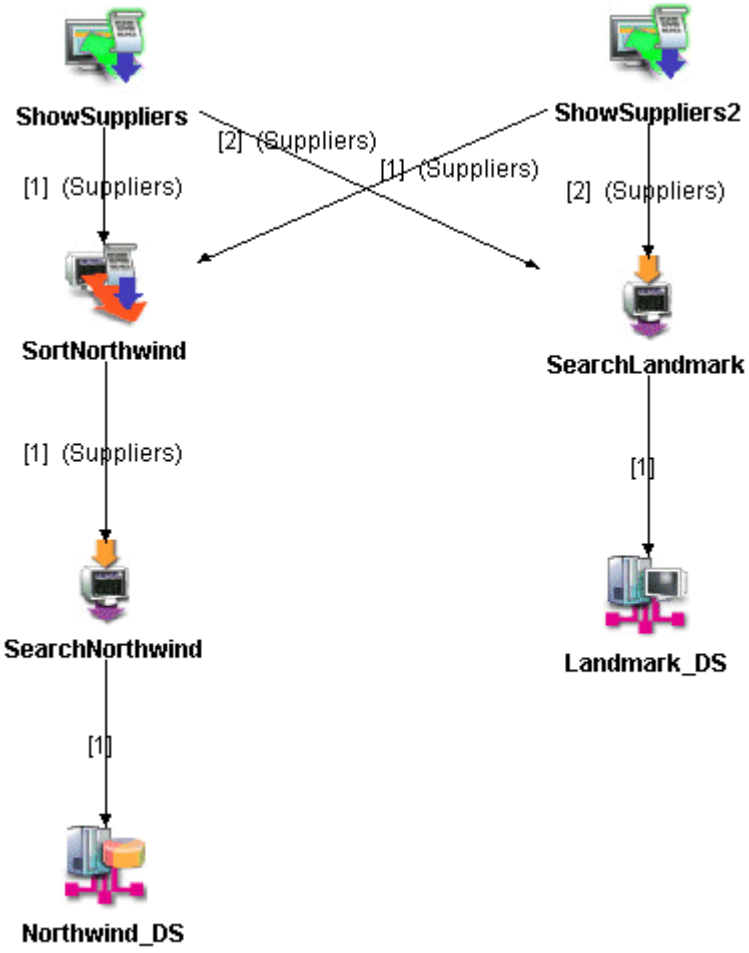
Now you will create the dynamic HTML example. You will recreate the same HTML output format as you did in the previous example; this time, however, you will do it programmatically.

▶ To create a User Services object with dynamic HTML

1. Double-click the User Services to open the User Services white-board, if it is not already displayed.
2. From the **New** menu, choose **User Services** and click anywhere within the white-board to place the object.

   The Create User Services Object dialog box appears.
3. Click **Create Real Object**, then click **OK**.
4. In the User Services Properties dialog box, select **VBScript**.
5. In the **Name** box, type *ShowSuppliers2*.
6. From the **Scope** box, choose **Public**.
7. Click **Parameters**. The Edit Parameters dialog box appears.
8. Click **New** to add a new parameter.
9. In the **Name** column, type *Suppliers*.
10. Click **OK** two times.

11. Click the **Relationship** button on the toolbar.
12. Click inside the ShowSuppliers2 User Services object and drag the cursor over to the SortNorthwind Business Logic object reference.
13. Click **OK** to specify the default mapping displayed.
14. Create a Relationship to connect the ShowSuppliers2 User Services object and the SearchLandmark Data Services object. Keep the default parameter mappings.



Adding the script

You now need the code that will accept data from the SortNorthwind and SearchLandmark object, and present it using the Cell method of the Table object..

▶ To add the script to the User Services object

1. Right-click the ShowSuppliers2 User Services object and choose **Edit Script** from the menu.
2. In the Create New Script dialog box, select **Use empty script**, and click **OK**.

   The script editor opens.
3. Enter the following script:

```
Sub Present()

   Set tNorthwind = Contents.Item(0)
   strHTML = "<html><body><center><h2>Top 5 Suppliers _
        From" & " NorthWind</h2><hr>"

   For RowIndex = 0 To 4
        strHTML=strHTML&tNorthwind.Cell_
             (CInt(RowIndex),0)&"<br>"
   Next

   strHTML = strHTML & "<hr>"

   Set tLandmark = Contents.Item(1)
   strHTML = strHTML & "<center><h2>Top 5 Suppliers _
        From " & "Landmark</h2><hr>"

   For RowIndex = 0 To 4
        strHTML = strHTML & tLandmark.Cell_
             (CInt(RowIndex),0)&"<br>"
   Next

   strHTML = strHTML & "<hr>"

   'Remove all contents from the IObject
   Contents.RemoveAll

    'Set OutputFormat to:HTML
   this.OutputFormat = "HTML"
```

```
    'Set PresentationTemplateFile
    this.PresentationTemplateFile = ""

    'Set ServerSideXSLT
    this.ServerSideXSLT = false

    'Set DisconnectSession
    Session.DisconnectSession = true
    this.Output = strHTML

End Sub
```

4. Save the script, and close the editor.

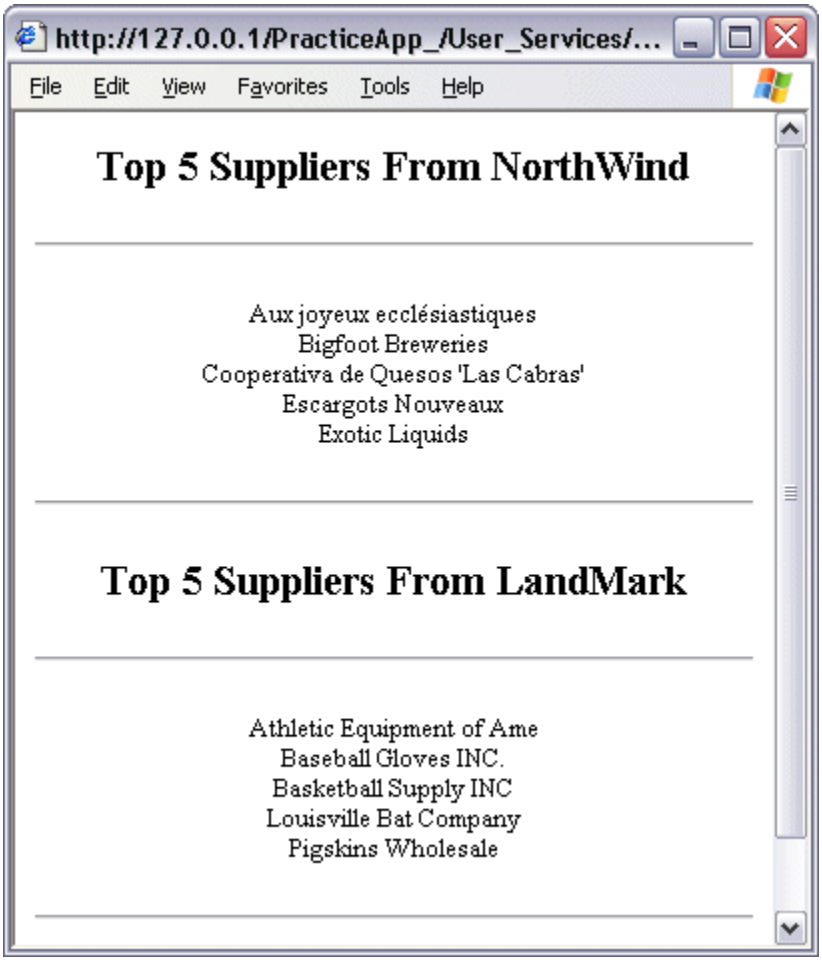
Testing the script

You should now test the script.

▶ To test the script

1. Right-click the ShowSuppliers2 User Services object and choose **Test** from the menu.
2. In the Test dialog box, click **Test In Browser**.

The default browser will open and the HTML output will be displayed.



Interpreting the example

We already discussed what is happening during the execution flow for the ShowSuppliers User Services object and since the new ShowSuppliers2 User Services object follows the same flow, we won't look at it. However, let's examine the script that you just entered and discuss what is happening.

The first thing you do is create a new reference object to the first Table Object contained within the object's IObject. You will be using the cell values later.

```
Set tNorthwind = Contents.Item(0)
```

Note: You may have noticed that you did not include a reference to the IObject when you extracted the Table Object. The reason for this is that the Data Services and Business Logic objects are executed within the context of the OnWeb object (IRule). However, the present script is executed within the context of the Information Object (IObject). Our presentation subroutine is already within the context of the IObject; there is no need to reference it when using its methods and properties.

Next, you generate the standard HTML string to be sent to the browser. You use a VBScript For/Next loop to cycle through the first 5 cells of the SearchNorthwind Table Object that you previously referenced. You use the Table Object's Cell method to extract the data values from their respective cells.

```
For RowIndex = 0 To 4
   strHTML = strHTML&tNorthwind.Cell_
         (CInt(RowIndex),0)&"<br>"
Next
```

So far we have accounted for the Northwind data. The following accounts for the Landmark data in much the same way:

```
Set tLandmark = Contents.Item(1)
strHTML = strHTML & "<center><h2>Top 5 Suppliers _
   From" & " Landmark</h2><hr>"

For RowIndex = 0 To 4
strHTML = strHTML &_
   tLandmark.Cell(CInt(RowIndex),0)&"<br>"
Next

strHTML = strHTML & "<hr>"
```

You now have an HTML string generated and stored within the variable strHTML.

In this example, you have no need for the existing Table Objects currently residing within the object's IObject (you already extracted the data that you require to generate the dynamic HTML string), so you just remove them completely from the IObject.

```
Contents.RemoveAll
```

The OutputFormat indicates whether the IObject output is in HTML or XML format. In this case, we set it to HTML.

```
this.OutputFormat = "HTML"
```

The PresentationTemplateFile property sets the file name of the presentation template applied to the IObject output.

In this example, we have generated our dynamic HTML in strHTML variable, so we won't need to use an external template file.

```
this.PresentationTemplateFile = ""
```

This property is used to indicate whether OnWeb Server should perform XSL transformation. The default value for this property is true and since you set HTML as the OutputFormat in our example, you must set the ServerSideXSLT to false.

```
this.ServerSideXSLT = false
```

You must specify whether to keep or disconnect the current user session after the results of our presentation are sent to the browser. The default value is False, which means that the session will not be disconnected.

In this example, you set the value to true to disconnect the user session.

```
Session.DisconnectSession = true
```

Finally, your last step in the presentation script is to output your dynamically generated HTML string to the browser.

Using the Output property, you can output the HTML generated by your Present script.

```
this.Output = strHTML
```

The present subroutine logic is now complete. You have dynamically generated the HTML string using the data values from the returning IObject of the SortNorthwind Business Logic object.

With the present logic completed, OnWeb takes care of the details of transmitting HTML code to the browser.

Summary

In this chapter, you used OnWebImport tags in an HTML template to present the contents of an IObject in a web browser. You now have an understanding of the methods OnWeb uses to control the presentation of Information Objects.

Before continuing to the next chapter, you should be familiar with the following points:

- How the User Services tier works.
- The different HTML presentation options available.
- How to create an OnWeb HTML User Services object using OnWebImport tags.
- How to create a User Services object script to dynamically output HTML.

Handling Errors

8

Proper error handling is an important aspect of good OnWeb development. It is important that the application is able to handle bad user input and errors produced by specific function calls. Therefore, you should design the application to handle errors from the start.

So far, we have not talked about or used any error checking in the examples. While not a good practice, this was done to simplify the examples as much as possible in order to concentrate on the fundamentals of OnWeb development.

Regardless of the level of protection you wish to provide, detecting error conditions is the first step in error handling. While we will not examine the different methods of error checking in-depth, we will illustrate some of the most common schemes of employing error checking in the scripts. While it is often difficult to determine what level of error checking is enough, this section will present some basic techniques that can be used alone or in combination to provide varying degrees of error checking.

Adding error checking codes

You will start by adding error checking to the first example script you created (in the simple Data Services object in Chapter 3, “Creating Data Services Objects”).

You will modify the script and add some error checking codes into it.

```
Sub Collect()
   Set tTable = _
        Onweb.CreateObject("Onweb.IComponent.Table", _
        "HelloTable")

   If IsObject(tTable) Then
      Set cColumnOne = _
        Onweb.CreateObject("Onweb.ColumnDef")

      If IsObject(cColumnOne) Then
```

```
            cColumnOne.Name = "Col1"
            cColumnOne.Type = "String"

            tTable.Schema.Add cColumnOne
            tTable.InsertRow
            tTable("Col1") = "Hello World!"

            IObject.Contents.Add tTable
         Else
                  Call SetOnWebError( Err, this.Name & _
                       ":Collect", Err.Description)
            Exit Sub
         End If
      Else
         Call SetOnWebError( Err, this.Name & ":Collect", _
           Err.Description)
         Exit Sub
      End If
End Sub

Sub SetOnWebError( eCode, eType, eDesc )
   IObject.Errors.Post eCode, eType, eDesc
   IObject.ReturnCode = eCode

End Sub
```

The first thing you probably notice is that you added a new subroutine called SetOnWebError. This subroutine adds a new Error Object (see Appendix A, "OnWeb Object Reference") to the IObject's error collection.

```
IObject.Errors.Post eCode, eType, eDesc
```

In order to insert a new Error Object, you must use the Error collection's Post method. An error code value, an error type, and a simple description must be supplied as parameters.

After including the new Error Object, you proceed to set the current IObject's ReturnCode property to the specified error code.

Another form of error checking is to make sure that each object has been created correctly.

```
If IsObject(tTable) Then
```

In this case, the IsObject VBScript function is useful. In this example, the IsObject function will return True if the Table Object has been created correctly. If this condition fails, you call the SetOnWebError subroutine and exit the Collect subroutine.

Handling the errors

Once an error is caught, it is advisable to display a message to the end user, explaining what has occurred or the phone number of the customer support center. You can create a custom HTML error page to accompany every object you create. This page will be displayed when an error occurs within an object.

To create a custom error message for an object, create an HTML page with the same name as the object. Give it a .err extension. Save this file in the \Server\HTML_Pages\Templates directory. For example, if you wanted to create an error HTML page for the “HelloWorld” example, you would name it HelloWorld.err.

Summary

It is imperative that you use proper error handling during OnWeb development. Remember to plan error-handling mechanisms in the design phase, not at the end of the implementation phase. Designing code that performs a task is only half the battle. Customers want applications that meet their list of functional requirements, but they also want applications that don't fall apart every time an incorrect key is pressed.

After reading this chapter, you should be familiar with the following points:

- Different OnWeb scripting errors that can occur.
- How to detect OnWeb errors.
- How to handle OnWeb errors.
- How to add error handling into the IObject's error collection.

Creating Conditional Objects 9

After going through the previous chapters, you now have a basic knowledge of how to use OnWeb Designer and how to create OnWeb applications. In this chapter, you will learn how to use conditional objects.

A conditional object is an enhanced Business Logic object that provides the OnWeb application developer with flexibility in deciding how and when rules are executed. Using an enhanced Business Logic object, you can control, at design time, which object (rule) to run under different circumstances, how many times a rule is run, which rule to run when an error occurs, or if to run a rule at all.

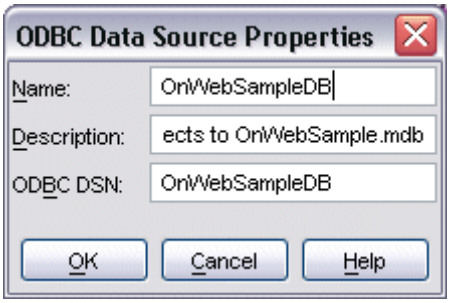
Creating a conditional object

In this section, we will create a sample application that centers on a conditional object.

▶ To create the OnWebSampleDB Data Source object

1. Start OnWeb Designer.
2. From the **New** menu, choose **Application** to create a new OnWeb application.
3. In the Create Application dialog box, type *CB* as the application name.
4. In the **Type** area, select **Host Integration**.
5. Click **OK**.
6. Open the Data Services white-board and create a new ODBC Data Source.
7. In the ODBC Data Source Properties dialog box:
   - In the **Name** box, type *OnWebSampleDB*.
   - In the **Description** box, type *Connects to OnWebSample.mdb*.

› In the **ODBC DSN** box, type *OnWebSampleDB*.



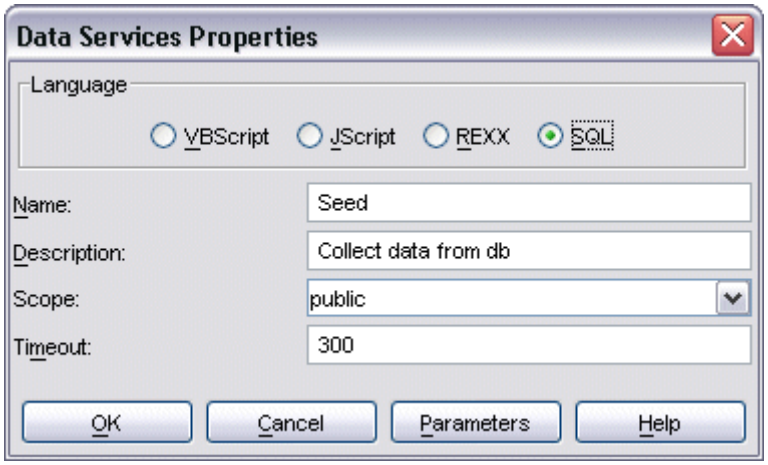
8. Click **OK**.

Note: OnWeb installation automatically creates the OnWebSampleDB DSN. If you want to connect to any other system DSN, you must create it first.

▶ To create the Seed Data Service object

The Seed object will connect to the OnWebSampleDB database and attempt to return a record set matching a parameter entered by the user.

1. Create a new Data Service object.
2. In the Create Data Services Object dialog box, select **Create Real Object**.
3. Click **OK**.
4. In the Data Services Properties dialog box, select **SQL** in the **Language** area and enter the following:



5. Click **Parameters,** then click **New**.

6. In the **Name** column, type *BookTitle*.
7. Click **OK** two times.
8. Right-click the Seed object and choose **Edit Script** from the menu.
9. In the Create New Script dialog box, select **Use empty script** and click **OK**.
10. In the script editor, enter the following code:

```
SELECT Title, ISBN
FROM Titles
WHERE Title LIKE '%{OnWeb,BookTitle}%'
```

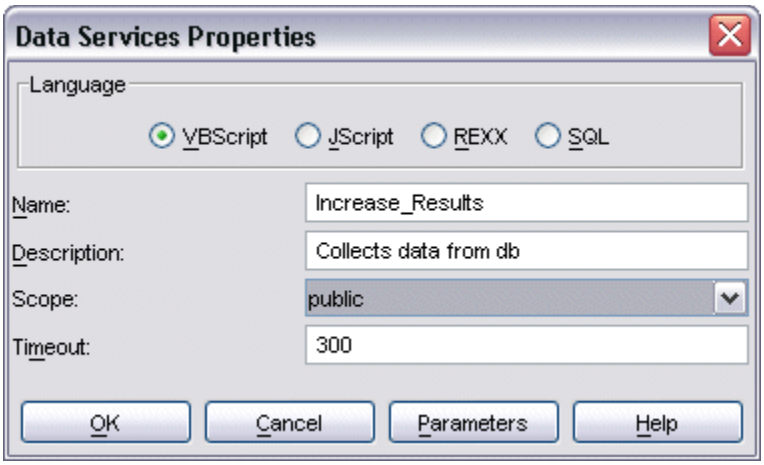
11. Save the script and close the editor.
12. Connect the Seed data service to the OnWebSampleDB data source by using a Relationship.

▶ To create the Increase_Results Data Service object

The Increase_Results object will connect to the OnWebSampleDB database only if the Plan script determines that the Seed object did not return a large enough result set. If the Increase_Results object is executed, it will modify the parameter entered by the user and attempt to return a larger number of results.

1. Create the Increase_Results Data Service object.

2. In the Data Services Properties dialog box, select **VBScript** in the **Language** area and enter the following:



3. Click **Parameters**, then click **New**.

4. In the **Name** column, type *BookTitle*.

5. Click **OK** two times.

6. Right-click the Increase_Results object and choose **Edit Script** from the menu.

7. In the Create New Script dialog box, select **Use empty script** and click **OK**.

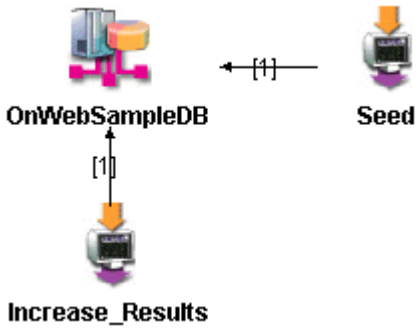
8. In the script editor, enter the following code:

```
'************************************************
' Name       : CB.Increase_Results
' Purpose    : To increase the results of the BookTitle
'                search
' Parameters : BookTitle
' Returns    : Record Set
'************************************************
Sub Collect()
   'Capture the incoming parameter
   Parm = Parameters("BookTitle").Value

   'Use the LEFT function to extract the first two letters
   'of the parameter
   Output = left(Parm, 2)

   'Create the message table
   'Create a table object with no title
```

```
    Set tblTable =_
       OnWeb.CreateObject("OnWeb.IComponent.Table", "")

    'Create the first column object
    Set colColumnOne = OnWeb.CreateObject("OnWeb.ColumnDef")

    'Set the column name and the type to String
    colColumnOne.Name = "Search Results: Not enough " & _
       "book titles found."
    colColumnOne.Type = "String"

    'Add the first column to the table schema
    tblTable.Schema.Add colColumnOne

    'Create a new row in table and insert a message
    tblTable.InsertRow
    tblTable("Search Results: Not enough book " & _
       "titles found.") = "The search for a book title " & _
       "containing the word, """ & Parm & """, " & _
       "did not return sufficient results." & _
       " The search has been changed to " & _
       "look for the first two letters: " & _
       """" & Output & """, " & _
       "and returned the following results."

    'Add the table to the IObject
    IObject.Contents.Add tblTable
    'Generate SQL Query based on the new shortened search
    'criteria
    SQL = "SELECT Title, ISBN FROM Titles WHERE" _
       &"Title LIKE '%" & Output & "%'"
End Sub
```

9. Save the script and close the editor.

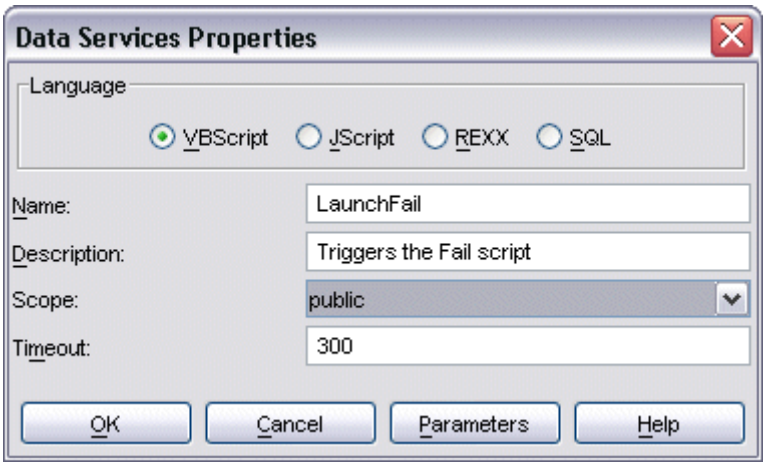
10. Connect the Increase_Results data service to the OnWebSampleDB data source by using a Relationship.



▶ To create the LaunchFail Data Service object

The LaunchFail object is used to demonstrate how to trigger a Fail script. It will cause the Fail script (within the conditional object) to be executed.

1. Create the LaunchFail Data Service object.
2. In the Data Services Properties dialog box, select **VBScript** in the **Language** area and enter the following:



3. Click **OK** to save the changes.
4. Right-click the LaunchFail object and choose **Edit Script** from the menu.
5. In the Create New Script dialog box, select **Use empty script** and click **OK**.
6. In the script editor, enter the following code:

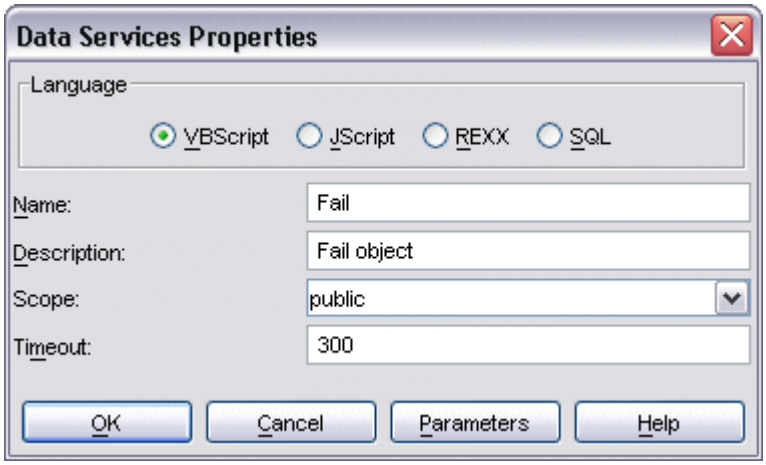
```
'************************************************
' Name        : CB.LaunchFail
' Purpose     : Used to trigger the fail script
' Parameters  : None
' Returns     : Nothing
'************************************************
Sub Collect()
   'Triggers the Fail Object
   this.Failed = 1
End Sub
```

7. Save the script and close the editor.

▶ To create the Fail Data Service object

The Fail object will run when a fail condition has been triggered.

1. Create the Fail Data Services object.
2. In the Data Services Properties dialog box, select **VBScript** in the **Language** area and enter the following:



3. Click **OK** to save the changes.
4. Right-click the Fail object and choose **Edit Script** from the menu.
5. In the Create New Script dialog box, select **Use empty script** and click **OK**.
6. In the script editor, enter the following code:

```
'*****************************************************
' Name       : CB.Fail
' Purpose    : Creates a single table
' Parameters : None
' Returns    : One table containing a message
'*****************************************************
Sub Collect()
   'Create the message table
   'Create a table object with no title
   Set tblTable = _
      OnWeb.CreateObject("OnWeb.IComponent.Table", "")

   'Create the first column object
   Set colColumnOne = OnWeb.CreateObject("OnWeb.ColumnDef")


   'Set the column name and the type to String
   colColumnOne.Name = "Search Results"
   colColumnOne.Type = "String"
```

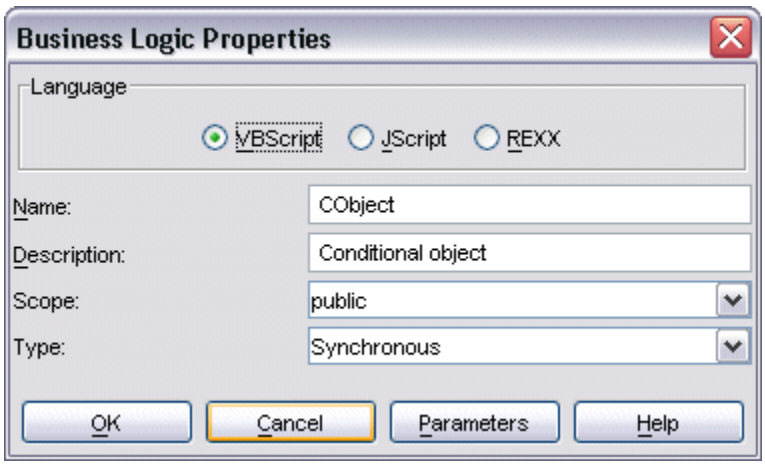
```
    'Add the first column to the table schema
    tblTable.Schema.Add colColumnOne

    'Create a new row in table and insert a message
    tblTable.InsertRow
    tblTable("Search Results") = "The search did not" & _
       "return any records; hence the fail script was" & _
       "launched. Please try another book title."

    'Add the table to the IObject
    IObject.Contents.Add tblTable

End Sub
```

7. Save the script and close the editor.

▶ To create the CObject Business Logic object

Now that you have created the Data Source object and the Data Service objects, you are ready to create the Business Logic object and connect it to the Data Service objects.

1. Open the Business Logic white-board and create a new Business Logic object.
2. In the Create Business Logic Object dialog box, select **Create Real Object**.
3. Click **OK**.
4. In the Business Logic Properties dialog box, select **VBScript** in the **Language** area and enter the following:



Note: The conditional object's properties do not become available until the object has been connected to at least one more object in a relationship. Once connected, you can return to the Properties dialog box to set the conditional object properties.

5. Click **Parameters.** Then click **New**.
6. In the **Name** column, type *BookTitle*.
7. Click **OK** two times.

▶ To create the relationship to the Data Services objects

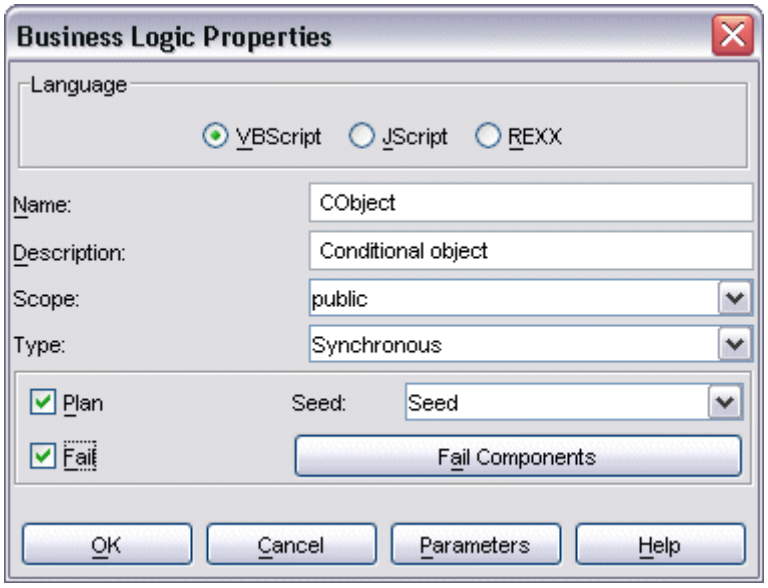
1. Right-click the Business Logic white-board, point to **New**, and choose **Data Services** from the menu.
2. Click anywhere within the white-board to place your new Data Services object.
3. In the Create Reference To Data Services Object dialog box, the **Create Reference** option will be selected. Expand the CB application tree structure and select the Seed Data Services object.
4. Repeat steps 1-3 for the Increase_Results, LaunchFail, and Fail objects.
5. Connect CObject to each of the Data Service objects by using a relationship. Remember to click **OK** to accept the default parameter mapping that appears automatically once the relationship has been created.

▶ To set the conditional object properties

Now that CObject is connected to one or more components, we can set the conditional object specific properties.

1. Right-click CObject, and choose **Edit Properties** from the menu.
2. Select the **Plan** option to indicate that the conditional object will have a Plan script.
3. To specify the seed object, select the Seed object from the **Seed** list. This object will execute first when we run the sample application.

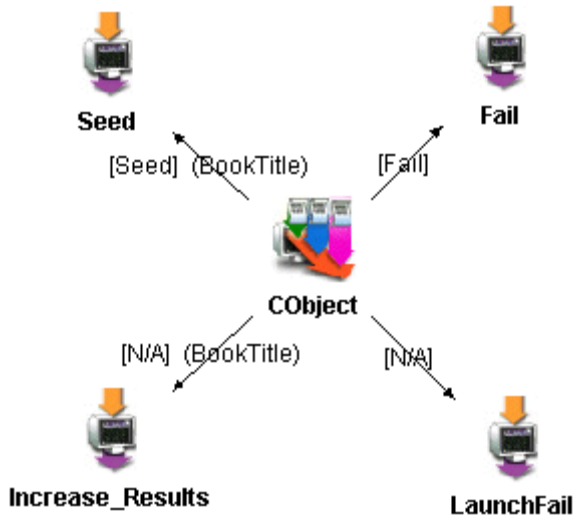
4. Select the **Fail** option to indicate that you want to have a Fail script.



5. Click **Fail Components**.

6. Select **Fail** as the fail object and click **Close**.

7. Click **OK**. The graphical representation of the sample application on the white-board should look like this:



▶ To add the Plan, Fail, and Refine scripts to the conditional object

1. Right-click the CObject object and choose **Edit Script** from the menu.
2. In the Edit File dialog box, select **Refine** from the list and click **OK**.
3. In the Create New Script dialog box, select **Use Empty Script** and click **OK**.

For the Refine script, enter the following code:

```
'************************************************
' Name        : CB.CObject Refine
' Purpose     : Adds all of the incoming IObjects to the
'               refine's IObject
' Parameters : BookTitle
' Returns     : An IObject containing all of the incoming
'               tables
'************************************************

Sub Refine()

intItemPosition = 0     'The position of the current item
intArraySize = 0        'The current array size
Dim tblTableArray()     'Dynamic Array to hold the tables

   'Loop through each IObject inside of the global IObject
   For each ioObject in IObject.Contents

   'Get the number of tables in the current IObject and
   'increase the array size by that amount
      intArraySize = intArraySize +_
         ioObject.Contents.Count

   ' Reset the number of items in the array but preserve
   ' the contents it already contains
         Redim Preserve tblTableArray(intArraySize)

   ' Loop through each item (table) in the current
   ' IObject
         For each objItem in ioObject.Contents

   ' Make a copy of the item and place it in the array, at
   ' the item position
               Set tblTableArray(intItemPosition) = _
                     OnWeb.TableServices.Copy(objItem)

         ' Increment the item position by one
               intItemPosition = intItemPosition + 1
```

```
      Next
   Next

   ' Remove all items from the global IObject
   IObject.Contents.RemoveAll

   ' Add all items in the tblTableArray to the global
   ' IObject
   For i = 0 to (intArraySize - 1)
      IObject.Contents.Add i, tblTableArray(i)
   Next
End Sub
```

4. Save the script and close the editor.

5. Repeat steps 2 to 5 to create the Plan and Fail scripts.

For the Plan script, enter the following code:

```
'************************************************
' Name       : CB.CObject Plan
' Purpose    : Executes a Data Services object if a
'              specified condition is met. Under normal
'              conditions (no errors), this script will
'              cause the defined rule to execute
' Parameters : None
' Returns    : Nothing
'************************************************

Sub Plan()

   'Check to make sure that an IObject was returned
   If (IObject.Contents.Count > 0) Then

      'Get the first IObject in the collection
      Set ioObject = IObject.Contents.Item(0)

      If (ioObject.Contents.Count > 0) Then
      'Get a reference to the first table in the first
      'IObject
         Set tblTable = ioObject.Contents.Item(0)

         'Get the row count
         RowCount = tblTable.RowCount

         'If less than two rows are returned, call
```

```
   'Increase_Results Object
      If (RowCount < 10) Then
         If (RowCount = 0) Then
         'In VBScript, when you run a component
         'rule you must remove all of the contents
         'of the IObject if you don't want the
         'results of the seed included in the
         'Refine
            IObject.Contents.RemoveAll
            AddComponentRule("CB.LaunchFail")
         Else
         'In VBScript, when you run a component
         'rule you must remove all of the contents
         'of the IObject if you don't want the
         'results of the seed included in the
         'Refine
            IObject.Contents.RemoveAll
            AddComponentRule("CB.Increase_Results")
         End If
Else
         ' If a sufficient number of rows is
         ' returned, and you want to pass up the
         ' results to the calling rule

         ' The position of the current item
         intItemPosition = 0
         ' The current array size
         intArraySize = 0
         ' Dynamic Array to hold the tables
         Dim tblTableArray()

         ' Loop through each IObject inside of the
         ' global IObject
         For each ioObject in IObject.Contents
         ' Get the number of tables in the current
         ' IObject and increase the array size by
         ' that amount
            intArraySize = intArraySize + _
               ioObject.Contents.Count

         'Reset the number of items in the array but
         'preserve the contents it already contains
            Redim Preserve_
               tblTableArray(intArraySize)

         ' Loop through each item (table) in the
         ' current IObject
```

```
                    For each objItem in ioObject.Contents
                  ' Make a copy of the item and place it in
                  ' the array, at the item position

                    Set tblTableArray(intItemPosition) = _
                       OnWeb.TableServices.Copy(objItem)
                  ' Increment the item position by one
                    intItemPosition = intItemPosition + 1
                  Next

                  Next

                  'Remove all items from the global IObject
                  IObject.Contents.RemoveAll

                  'Create an IObject
                  Set myIObject = OnWeb.CreateObject _
                    ("OnWeb.IComponent.Iobject", "")

                  'Add all items in the tblTableArray to
                  'myIObject
                  For i = 0 to (intArraySize - 1)
                  myIObject.Contents.Add i, tblTableArray(i)
                  Next

                  'Add the local IObject to the global IObject
                  IObject.Contents.Add myIObject
               End if

            Else
               Call SetOnWebError(this.Name & ".Refine()", _
                    "Failure:Please create a reference" _
                    & " to a data service object that" _
                    & " returns an IObject with at least" _
                    & " one table.", 99999)
            End if
         Else
            Call SetOnWebError(this.Name & ".Refine()", _
               "Failure: Please create a reference" _
               & " to a data service object that" _
               & " returns an IObject with at least" _
               & " one table.", 99999)
         End if
      End Sub
```

```
'*************************************************
' Function Name: SetOnWebError
' Parameters  : strType - The type of error
'               strDescription - The description
                    of the error
'                  strCode - The error code value
' Purpose       : A standard error routine
' Returns       : Nothing
'*************************************************

Sub SetOnWebError(strType, strDescription, strCode)
   IObject.Errors.Post strCode, strType, strDescription
   IObject.ReturnCode = strCode

End Sub
```

For the Fail script, enter the following code:

```
'*************************************************
' Name        : CB.CObject Fail
' Purpose     : Basic declarations required for any Fail
'               method. Under normal conditions (no fatal
'               errors), it will cause the defined rule to
'               execute
' Parameters  : None
' Returns     : Nothing
'*************************************************

Sub Fail()

   AddComponentRule("CB.Fail")

End Sub
```

Testing the conditional object

We are now ready to test the conditional object.

▶ To test the CObject conditional object

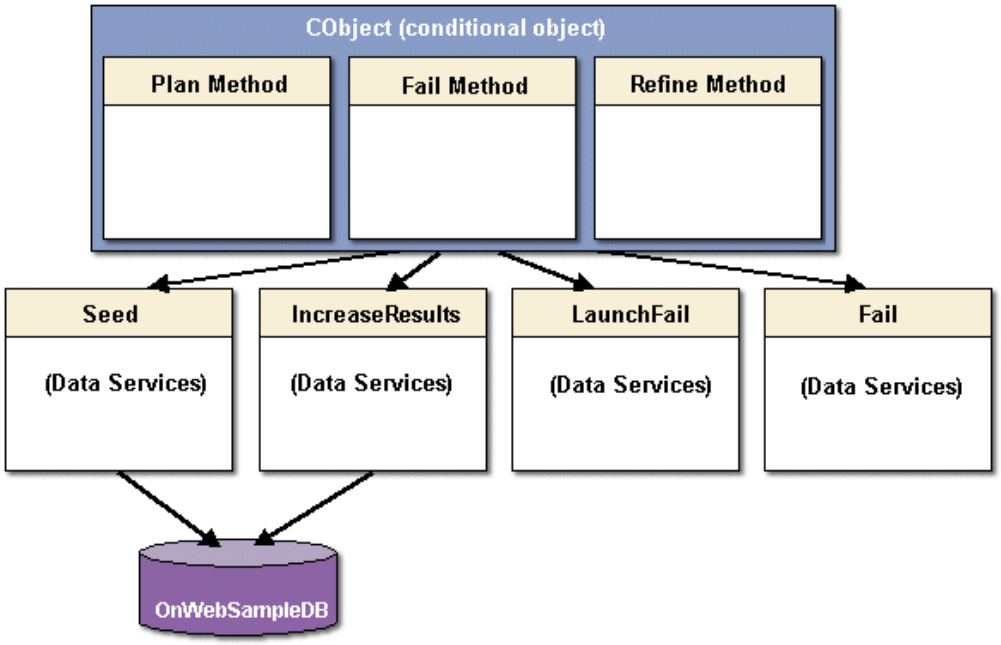
1. Right-click CObject and choose **Test** from the menu.
2. In the **Test:CObject** dialog box, type one of the following words as the **Value** of the CObject's "BookTitle" parameter: world, genetic, and jungle.
3. Click **Test**.

You will see the following test scenarios:

Word typed	Result description
world	In this scenario, there are numerous book entries in the database that contain the word "world". A sufficient number of rows is returned, and the Plan script passes the results back to the user.
genetic	In this scenario, there are less than 10 book entries in the database that contain the word "Genetic". An insufficient number of rows is returned by the search causing the Plan script to call the IncreaseResults object. IncreaseResults will alter the passed parameter to search for entries that contain "Ge", and return a sufficient number of rows.
jungle	In this scenario, there are no book entries in the database that contain the word "Jungle". Zero rows are returned by the search causing the Plan script to invoke the LaunchFail object. LaunchFail will trigger the Fail script. The Fail script will invoke the Fail object, which will create a generic table to illustrate that it has been invoked.

Interpreting the example

The following diagram illustrates the flow of the sample application.



The CObject conditional Business Logic object takes one parameter, called "BookTitle", and allows the user to search for a specific book by title.

CObject connects to several Data Services objects, some of which will access the OnWebSampleDB ODBC data source.

The Seed object collects the information and passes it up to the conditional object's (CObject) IObject.

```
SELECT Title, ISBN
FROM Titles
WHERE Title LIKE '%{OnWeb,BookTitle}%'
```

The Plan method looks at the information in the IObject to determine if it is necessary to search the data source again by easing up on the search criteria.

If it is determined that an insufficient number of results are returned by our initial search, the Plan method invokes either the IncreaseResults or the LaunchFail component rule.

```
If (RowCount = 0) then
   AddComponentRule("CB.LaunchFail")
Else
   AddComponentRule("CB.Increase_Results")
End If
```

If it is determined that a sufficient number of rows are returned by the initial search, the Plan method continues to run the remaining script. It preserves the result set from the Seed object and puts it into the global IObject.

```
Set myIObject = OnWeb.CreateObject_
   ("OnWeb.IComponent.Iobject","")
For i = 0 to (intArraySize - 1)
   myIObject.Contents.Add i, tblTableArray(i)
Next
IObject.Contents.Add myIObject
```

The conditional object's Refine script will then run and display the results.

Summary

Now you know that the enhanced capability of the conditional object is achieved by including the following scripts in the object:

- A Plan script that controls the execution of other objects.
- A Fail script that identifies which object to execute when an error occurs.

In addition to these scripts, there is also the ability to designate a seed object, whose main purpose is to generate an Information object containing some type of data that the Plan script will evaluate.

After reading this chapter, you should be familiar with the following points:

- Why and when you should use conditional objects.
- How to create a Seed object and how to write the Plan and Fail scripts.
- How the Seed object, Refine script, Plan script, and Fail script interact with each other.

OnWeb Object Reference A

The following is a quick reference to OnWeb objects. To learn more about these objects, refer to OnWeb Scripting Help.

ColumnDef Object

Represents a single column in a table. (Use the add method of the schema collection)

Methods: (none)

Properties: AcceptNulls, IsModifiable, IsUnsigned, Name, Precision, Scale, Type

Note: After a Column Object has been added to a table, you can change the Name property only. All other properties cannot be modified.

Cursor Object

Represents the current state of the cursor associated with the emulation screen (object member of the emulator object)

Methods: Set

Properties: Column, Offset, Row

Data Source Object (abstract)

Represents an OnWeb data source; ODBC-based or terminal-based.

Methods: (none)

Properties: Classname, Connected, Name, Password, UserID, Variables(collections)

Emulator Object (abstract)

Represents a Terminal-based Data Source object (VT100, VT220, 3270, and 5250)

Methods: Connect, Disconnect, Press, Type, WaitFor

Properties: Classname, Connected, Emulation, IPAddress, IPPort, Name, Timeout, TraceFile

3270 emulator object additional properties: Autoskip, Codepage, ExtendedAttributes, KeyboardLocked, Model, TTYMode

5250 emulator object additional properties: CodePage, KeyboardLocked, Model, TTYMode

Error Object

Represents an OnWeb error that has occurred. Exists only as a member of the Errors Collection of the Information Object.

Methods: (none)

Properties: Code, Type, Text

Errors Collection

Collection of Error objects. To refer to a particular error, use its ordinal number, e.g. IObject.Errors(0) or IObject.Errors.Item(0)

Methods: Items, Post, RemoveAll

Properties: Count, Item

Information Component Object (abstract)

Represents an item of information that can be passed to an object as a parameter, or created by an object.

Methods: (none)

Properties: (none)

Represented by: Information Object, String Object, Table Object, and Dictionary Collection.

Information Rule Object (abstract)

Represents an OnWeb object (Data Services, Business Logic or User Services), Terminal-based object (Data Services), and ODBC-based object (Data Services).

Methods: (none)

Properties: Classname, IObject, SQL, name, Parameters

Information Object (IObject)

Represents the result generated when OnWeb executes an object.

Methods: (none)

Properties: Classname, Contents, Errors, Parameters, ReturnCode, Variables

Onweb Object

Global object that represents the OnWeb server session. It exists while the server is active; it is available to every script.

Methods: ChangePassword, CreateObject

Properties: TableServices

Screen Object

Represents the current state of the emulation screen.

Methods: CheckFor

Properties: Attribute, Color, ColumnCount, Length, RowCount, Text

Session Object

Global object that represents a user session; it exists while the user is connected to OnWeb. It is always available to every script.

Methods: Logoff, Logon

Properties: UserID, Variables

String Object

Represents a text string.

Methods: Append

Properties: Value

Note: In addition to the one method, you can manipulate strings using VBScript.

Table Object

Represents a table of data (an Information Component)

Methods: Clear, Copy, RowFrom, DeleteRow, InsertRow, MoveFirst, MoveLast, MoveNext, MovePrevious

Properties: Cell, Classname, ColumnCount, Current, IsOutOfBounds, Name, RowCount, Schema

Table Services Object

Represents the services that the OnWeb server provides for tables.

Methods: Copy, CopySchema, Join, Project, Sort, Union

Properties: (none)

VBScript Code Samples

B

The following table lists frequently-used VBScript codes:

Functions	Codes
To access parameters	`var = Parameters("paramname").value` (if parameter is an object, use Set)
To leave the script (at any point)	`Exit Sub`
To get the number of Table objects in the IObject	`var = IObject.Contents.Count`
To access a Table object in the IObject	`Set tablevar = IObject.Contents.Item(num)`
To create a new Table object	`Set tablevar = Onweb.CreateObject _ ("Onweb.IComponent.Table", "tablename")`

To create a new Column object	`Set columnvar = Onweb.CreateObject _ ("Onweb.ColumnDef")`
	`columnvar.Name = "columnname"`
	`columnvar.Type = "string"`
	`columnvar.AcceptsNulls = "true"(or false)`

To add a Column object to a Table object	`tablevar.Schema.Add columnvar`

Functions	Codes
To add a Table object to the IObject	`IObject.Contents.Add tablevar`

To navigate table rows	`tablevar.MoveFirst`
	`Do While Not tablevar.IsOutOfBounds`
	`tablevar.MoveNext`
	(also MovePrevious or MoveLast)
	`Loop`

To get data from a Table cell	`var = tablevar(num).value` (where num is the column number: i.e. 0,1,2 etc.)

To enter data into a Table	`tablevar.InsertRow`
	`tablevar(0) = "the data to enter for this column"`
	`tablevar("columnname") = "the data to enter for this column"`

To put terminal screen into a string	`Set var = Datasource.Screen.Text` (can specify Row, Columns, and Length)
To position the cursor on the screen	`Datasource.Screen.Cursor. Setrownum, colnum`
To get the number of Columns	`var = tablevar.Schema.Count`
To get the Column name	`var=tablevar.Schema.Item(num).Name` (where num is the column number)

Index

R

S

T

U