Rocket U2 Web Development Environment

Developing RBOs with U2 Web Designer

Version 5.1.1

January 2014
WDE-511-RBO-DG-01
Notices

Edition

Publication date: January 2014
Book number: WDE-511-RBO-DG-01
Product version: Version 5.1.1

Copyright

© Rocket Software, Inc. or its affiliates 1996-2014. All Rights Reserved.

Trademarks

Rocket is a registered trademark of Rocket Software, Inc. For a list of Rocket registered trademarks go to: www.rocketsoftware.com/about/legal. All other products or services mentioned in this document may be covered by the trademarks, service marks, or product names of their respective owners.

Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

License agreement

This software and the associated documentation are proprietary and confidential to Rocket Software, Inc. or its affiliates, are furnished under license, and may be used and copied only in accordance with the terms of such license.

Note: This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Contact information

Website: www.rocketsoftware.com

Rocket Software, Global Headquarters
77 4th Avenue
Waltham, MA 02451–1468
USA
Tel: +1 781 577 4321
Fax: +1 617 630 7100
Contacting Global Technical Support

If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Customer Portal and report a problem, download an update, or find answers to in the U2 Knowledgebase. The Rocket Customer Portal is the primary method of obtaining support.

To log in to the Rocket Customer Portal or obtain a Rocket Customer Portal account, go to www.rocketsoftware.com/support

Alternatively, you can contact Global Technical Support by email or telephone:

**Email:** u2support@rocketsoftware.com

**Telephone:**

- **North America** +1 800 729 3553
- **United Kingdom/France** +44 (0) 800 773 771 or +44 (0) 20 8867 3691
- **Europe/Africa** +44 (0) 20 8867 3692
- **Australia** +1 800 707 703 or +61 (0) 29412 5450
- **New Zealand** +0800 505 515
Configuring the encoding of the JSP RBOScope testing tool................................................. 33

Chapter 6: RBO debug logging........................................................................................................ 34
  debug property............................................................................................................................ 34
  debug_id property.................................................................................................................... 34
  Debug log information............................................................................................................. 35
  Manually invoking the RBO debug process........................................................................... 36

Chapter 7: Creating HTML documentation for RBOs................................................................. 37
  Generating RBO HTML documentation.................................................................................. 37

Chapter 8: Setting up remote procedure calls.......................................................................... 38
  RPCs overview......................................................................................................................... 38
  Setting up RPCs..................................................................................................................... 38
  Writing an RPC subroutine.................................................................................................... 39
  Calling an RPC subroutine from Java or C#........................................................................... 40
Chapter 1: Rocket U2 Web Development Environment overview

Rocket U2 Web Development Environment (Web DE) is an application development toolkit for building interactive client/server applications for deployment on the World Wide Web or on a corporate intranet.

Introduction

This document gives an overview of RedBack objects (RBOs) and their function in web application design. It helps you with the basics of developing RBOs in the Eclipse-based U2 Web Designer, testing RBOs with the built-in RBOScope testing tool or the standalone JSP RBOScope testing tool, and developing server code for RBOs.

Web DE components


The following table describes the role of each Web DE component.

Table 1: Web DE components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedBack Object Server</td>
<td>The RedBack Object Server manages access to RedBack applications and implements dynamic runtime interfaces to server-side application resources.</td>
</tr>
<tr>
<td></td>
<td>The RedBack Object Server also maintains the repository that contains all the definitions and code that define an application.</td>
</tr>
<tr>
<td>Java Scheduler</td>
<td>The Java Scheduler manages data transfer between the RedBack Object Server and the web server, handling requests from RedBeans and RedPages.NET.</td>
</tr>
<tr>
<td></td>
<td>It also performs the job of load balancing, processing the queue of connection requests by allocating U2 licenses as webshares.</td>
</tr>
<tr>
<td>RedBeans and RedPages.NET</td>
<td>The RedBeans and RedPages.NET components are the Java and .NET application programming interfaces (APIs) that enable you to connect your application to RedBack Objects (RBOs). These components are the interface between the web server and the RedBack Object Server. The role of these components is to establish and maintain connection pool connections or webshare connections between the web server and the RedBack Object Server.</td>
</tr>
<tr>
<td>Web Designer</td>
<td>This suite of designer tools runs in an Eclipse development environment and provides an interface for designing RBOs.</td>
</tr>
</tbody>
</table>
Chapter 2: Application development with the RBO model

RBOs represent the logical design of the application in an object-oriented environment. The best strategy is to design the RBOs and their structure based on the requirements of your application. It is recommend that you design and plan the RBO implementation to make use of inheritance, planning which classes provide which data and methods.

The RBOs that you design and build provide an interface to your data and code. You create an object layer on top of what would normally not be viewed as an object environment. This means that your design follows object-oriented techniques rather than standard UniData or UniVerse development techniques. This is important because the RBOs will often be referenced by developers who do not understand UniData or UniVerse, or have any experience in multivalued data or UniBasic/UniVerse BASIC code.

The following sections provide an overview of developing applications by using the RBO model.

RedBack objects (RBOs) overview

Web DE allows data access to a U2 database from a web application. You use RBOs to define that access. An RBO is a kind of data envelope that moves back and forth between the middleware and the U2 data server. It moves data to the U2 data server when you call a method from a middleware routine, and returns data to the calling routine when the method is completed on the U2 data server.

Using RBOs allows support for a number of clients and middle tier interfaces while providing data abstraction and facilitating teamwork, giving you a fast and reliable data access layer.

Web DE supplies some predefined classes that you can inherit and enhance in your own business-specific object definitions. You can create and maintain your RBOs in the RBO perspective of Web Designer. In addition, the RBO perspective offers an editor tool for editing, compiling, and cataloging your UniBasic or UniVerse BASIC subroutines.

You can test your RBOs by using the RBOScope testing tool that is included with Web DE. This tool allows you to test objects before you write your own middleware calling subroutines.

RBO properties and methods

RBOs have properties and methods. The properties are data and parameter placeholders, and the methods are aliases to subroutine calls. Some of these subroutines are included with Web DE, and others you write yourself.

For example, you can create an RBO called “Order” by inheriting the stateless RBO (SLRBO) class that is included with Web DE. You then add properties, such as OrderID, Items, Qtys, CustomerName, Address, ErrorCode, and ErrorMessage, and methods, such as ReadOrder and WriteOrder, and create the subroutines that these methods reference.

In this example, there are two types of properties: those that contain database data, and those such as ErrorCode and ErrorMessage that you use for communication from the UniBasic or UniVerse BASIC programs back to the calling routines in the middle tier. For example, if a program could not open a file or was not passed mandatory input, you want to report that information back to the calling subroutine. You can accomplish this only by using the properties of the RBO through which it was called.
RBO classes

The RBO hierarchy is ordered by classes and subclasses.

The following figure shows the hierarchy and each class and subclass.

Figure 1: RBO hierarchy

RBO class

The RBO class is the superclass for all the RedBack object classes. Other RBO classes inherit from this class, and properties and methods can be added. With the RBO, you can create objects that you want to keep in a user’s session by using the U2 data server. Web DE keeps the object data in the WWSTATE file. Keeping a handle to the object enables you to request the previously used object. Every method call flushes the data in the object to the record in WWSTATE. Every open, when using the previously stored handle from a cookie, hidden field, or variable, brings back the object with any data stored previously in the object, and is flushed to the database by a method call. Because of the input/output overhead, you should use an RBO in place of an SLRBO only if you intend to make use of its statefulness. This object is stateful; a stateless, faster version of it is the SLRBO class. You typically use either the RBO or SLRBO to read or write a record.

emailObject subclass

eMailObject is a subclass of RBO. It provides properties and methods to allow email messages to be sent from the RedBack Object Server. This object runs the unix_command or nt_command you supply, so you must verify that the command works on your RedBack Object Server or U2 data server. Remember that you can use an SLRBO to do the same thing, with your method subroutine actually running the mail command. Most web applications do not send email messages from the RedBack Object Server or U2 data server, but from the middle tier. However, you might find this object useful.

SysInfo subclass

SysInfo is a subclass of RBO. It provides properties and methods to obtain information about the operating system, database platform, and Web DE installation.

uObject subclass

uObject is a subclass of RBO. It provides properties and methods to facilitate access to UniData and UniVerse databases with prewritten UniBasic or UniVerse BASIC methods to auto-read and auto-write records. If you use this object in a stateful manner, the object can trigger a lock or optimistic concurrency control error that you can handle by writing a subroutine for the UpdateLockError
method. This object also has PostRead, PreWrite, and PostWrite methods for which you can write subroutines. If you have complex reads or writes to run, an SLRBO class is usually faster.

**uQuery subclass**

uQuery is a subclass of uObject. It provides properties and methods to enable you to run queries against UniData and UniVerse databases. A uQuery object returns a RecordSet that you can scroll through, displaying fields from each row as you scroll. If you gather data using a single SELECT statement, you can use this object. Being a stateful object, it supports paging through the data without repeating the SELECT statement. If you cannot use a single SELECT statement to gather the data, you might want to use an SLRBO or RBO and return the records in delimited strings.

**sbObject subclass**

sbObject is a subclass of RBO. It is used to interact with SB+ applications. For information about accessing SB+ from Web DE, see *Using SB/XA with U2 Web Designer*.

**SLRBO class**

The SLRBO class is a stateless RBO, the fastest RedBack object, which puts the least load on the RedBack Object Server or U2 data server. No state for the class is stored in the RedBack Object Server, unlike all other classes. The SLRBO can be useful if the application state is to be stored elsewhere, for example in the middle tier, or where the RBO does not actually require any state to be stored. Your application might require storage of session IDs for stateless RBOs to ensure that a request is valid and allowed.

---

**Session and state overview**

The main difference between designing applications for the web and designing applications for traditional host environments is the web’s stateless nature. The web has no permanent connections. Web DE provides session and state concepts to overcome these limitations.

For example, when you click a link or enter a URL in your web browser, a new connection is made to the target web server. The target web server runs a process or script. This process sends a web page back to your web browser, and then breaks the connection. If the web page contains images or other files, multiple connections are made to download these items. In these situations, your web browser makes, and breaks, multiple connections to the target web server.

You are connected to the web server (and ultimately the RedBack Object Server) only while the request is made and resolved.

For web pages that only publish information, the stateless nature of the web does not pose any major design issues. However, for transactional applications, the lack of a permanent connection means that you can no longer use some of the traditional functionality of BASIC applications, such as COMMON memory, in the same way.

**Sessions**

Because the web is stateless, there is no concept of logging on and logging off. These concepts translate in Web DE to the concept of session. A session is user information that defines a “virtual connection.” Because there is no permanent connection, a session consists of a series of requests between your web browser and the RedBack Object Server.

Web DE maintains session information on the RedBack Object Server and in the data sent to and from your web browser. The session information maintained on the RedBack Object Server includes the
results of queries and lookups. The session information maintained on the web server identifies the user and the current session. The information used to identify the user includes the user’s name, initial IP address, and the start time and date of the current session.

**Beginning a session**

When you make your first request from your web browser, the RedBack Object Server starts to maintain session information to identify you when you make subsequent requests. Unless the RBO subclass is SLRBO, the RedBack Object Server expects to see a session ID passed with each request. If the RedBack Object Server does not receive a session ID, the RedBack Object Server assumes this is the first request, and creates a session and a session ID. Each subsequent request should pass back that session ID, so that each request can be associated with the session.

Both RedBeans and RedPages.NET provide methods to access and set the session ID. For more information see *Developing Applications with RedBeans* and *Developing Applications with RedPages.NET*.

**Ending a session**

In Web DE, a session has a “life.” Whenever the RedBack Object Server receives a request from a web browser that does not contain any session information, it creates a new session. You cannot end a session, but you can start a new session. The session information maintained on the RedBack Object Server can remain there after a session life is complete.

The RedBack Object Server automatically deletes session information on a periodic basis through the built-in garbage collection function. You can fine-tune garbage collection handling, if necessary. For more information, see the information about managing garbage collection in *Administering U2 Web DE*.

**Sessions and the RDBMS**

Because there are no permanent connections in web-based applications, fewer licenses are required for your relational database management system (RDBMS) than are required for traditional host-based applications with the same number of users. There is no direct mapping between an end-user and an RDBMS license. An RDBMS license can process the requests of many users.

**State**

Traditional host-based applications that provide constant connections can store relevant application data in COMMON memory. The concept of COMMON memory translates in Web DE to the concept of state.

As you progress through a Web DE-based application, the RedBack Object Server stores object state for RBO objects that are instantiated, with the exception of stateless RBOs (SLRBO or any of its subclasses). For example, if you select an RBO form from a menu, the RedBack Object Server maintains object state for that form, including all fields and any values that you enter. Within each object instance, the RedBack Object Server stores your unique session information.

For example, if you create RBO A, enter data, and then create RBO B, at some point the application takes you back to RBO A, and the data is redisplayed in RBO A (stored in state). However, if you re-create RBO A at some point in the application, Web DE destroys any previous data state for RBO A. You cannot have multiple instances of the same object.

The way in which your application navigates between objects is important because there is a difference between returning to and creating an object. With each request, the RedBack Object Server returns all current data for the object submitted to the RedBack Object Server and updates state. If
you need to reference a previous object, you can store the handle to the object in either RBO state or client state (or both).

It is important that you preserve the current session ID outside of the RedBack Object Server by the tier that makes the RBO request. If RBO A has state that is not required by the other RBOs, part of your design process needs to decide whether this RBO could be subclassed from SLRBO and become stateless.

Stateless RBOs do not have to store state within the RedBack Object Server. This may make sense if its state is not required at a later stage, or if the state is to be stored in a tier outside of the RedBack Object Server.

A stateless RBO cannot access data from another RBO method. The RedBack Object Server engine must preserve the current RBO state, but a stateless RBO cannot have its state preserved. This is only applicable when calling methods on the RedBack Object Server.

Understanding how session and state work together

Web DE maintains state for every property that is defined in an RBO class definition. This state information is available to RedBack Object Server code through the server API functions. If the RBO is subclassed from SLRBO (a stateless RBO), this state is available only during the request and not afterward, because the RedBack Object Server does not preserve state beyond each request.

For RBOs that are not stateless, the RedBack Object Server stores the state, and the state can be referenced by other RBOs. For this process to work, the RedBack Object Server must pass the session ID with every request. Without a session ID, the RedBack Object Server has no way of determining to which session the request belongs, and cannot reference state from that session’s other RBO instances.

The RedBack Object Server uses the session ID to validate a request and determine whether the session is still active. If you use stateless RBOs, you might want to implement your own form of session and state to validate that a user has logged in and has an active session.

To call server code, you associate a cataloged UniVerse BASIC or UniBasic subroutine call with a method on an RBO. You can call a method from the client code or from the UniVerse BASIC or UniBasic code.

Web DE associates each method with a class and interacts with an instance of that class. The method does not control the browser. By default, Web DE refreshes the current object for the initial method called with its current state.

A method can create other instances of classes and interact with those instances’ methods. Methods can also retrieve information from state for their own instance and other class instances if the handle to those instances is available and none of the RBOs involved are stateless.

For example, assume that you write a UniVerse BASIC or UniBasic subroutine called ValSal that validates an employee salary. From your Employee RBO, you add a method called ValidateSalary and associate the ValSal subroutine with that method.

The following figure shows the example of the ValSal subroutine in the Employee RBO.
When the client makes a request to call the method ValidateSalary, the RedBack Object Server engine makes a call to the ValSal subroutine.

Each method uses one or more of the server API functions to interact with RBO State for its instance of the RBO class. For example, the ValSal subroutine looks at the salary entered, and then checks that the salary falls within the allowed salary ranges for the employee’s department. The code creates an instance of the Department RBO and calls its CheckSalary method. If the salary is outside of the salary range, you want the browser to display an error message.

Do not put business logic code, such as file opens, reads, calculations, and writes in your methods called directly through your RBOs. Instead, use your method subroutines to call business logic subroutines, receiving data and status or error messages. All RBO.getProperty and RBO.setProperty commands should be in the method subroutine that you call directly through the RBOs; none should be in the business logic subroutines. This makes testing and debugging of your business logic subroutines possible outside of Web DE.
Chapter 3: Developing RBOs

The following sections provide information about using the RBO perspective of the Web Designer.

Getting started with Web Designer

After you install Web DE, you can develop RBOs in the Web Designer. The Web Designer has an intuitive user interface to help you design and modify RBOs quickly. The components of the Web Designer workspace are defined by the perspective that is selected. A perspective is a collection of views. The views in the RBO perspective of Web DE enable you to build, maintain, and test RBOs for your web application.

1. Open the Web Designer by clicking Start > All Programs > Rocket U2 > Web Development Environment > U2 Web Designer.
2. By default, the perspective opens to the RBO perspective. If the Web Designer does not open to this by default, select Window > Open Perspective > RBO Perspective.

Developing RBOs

Do the following tasks before you create or edit RBOs. These tasks are described in more detail in Administering U2 Web DE.

- Create a U2 server
- Connect to a U2 server
- Enable an account for U2 Web DE
- Create a module

The following sections describe how to create RBOs in Web Designer. For information about how to test RBOs, see Testing RBOs, on page 30.

Creating or editing RBOs

You can create an RBO, edit an existing one, or create one from a U2 database file.

The following sections describe each of these processes.

- To create an RBO in the Web Designer, follow the steps in Creating an RBO, on page 13.
- To edit an existing RBO, follow the steps in Editing an RBO, on page 17.
- To create an RBO from a U2 database file, follow the steps in Creating an RBO from a U2 database file, on page 18.

Creating an RBO

After you connect to a U2 server, enable an account for U2 Web DE, and create a module, you can create an RBO for the module. A module is a repository for all of the RBO class definitions that relate to a convenient grouping. The RBOs are therefore relevant to each module.

1. In the Web Designer, expand the U2 Web DE Elements tree, right-click a module, and select New RBO.
2. In the Select Inheritance window, select the class from which the RBO inherits properties and methods, and click OK.
There are two tabs available to view the available classes, the **Hierarchy View** and **Flat View**. Each view shows all of the possible superclasses, including system classes, classes in the SHARED module, and all classes in the current module that are not final classes. For more information about these types of classes, see RBO classes, on page 8.

**Note:** RBOs cannot inherit from final classes; therefore, final classes are not displayed in the Select Inheritance window. RBOs can, however, inherit from system objects, but you cannot modify system objects. Custom-developed objects do not have the word ‘SYSTEM’ next to their title, but they have the word “SHARED” next to their title if they were created in the SHARED module. You cannot add your own system objects. However, if you add a class to the SHARED module, classes in other modules can inherit from it.

After you click **OK**, the new RBO appears in the RBO Editor in edit mode. The default name of the RBO is “new.” Inherited properties and methods are populated in the RBO Editor tabs, as shown in the following figure.

![Figure 3: New RBO](image)

3. Add identifying information, as described in Entering identifying information for the new RBO, on page 15.
4. Add properties, as described in Defining properties of the RBO, on page 15.
5. Add methods, as described in Defining methods of the RBO, on page 17.
6. After you add properties and methods, click the **Save** icon or select **File > Save**, and enter a unique name for the new RBO.

Web Designer stores the RBO definition in the U2 database. The following reserved names for RBOs must not be used for the user-defined RBOs:

- rbo
- sysinfo
- emailobject
- uobject
- sbobject
- uquery
- logon
Entering identifying information for the new RBO

When you have a new RBO in edit mode, you must enter some identifying information such as a name and description to identify the object.

1. From the RBO Editor pane, in the **Description** field, enter a description to identify the RBO.
2. Click the ellipsis button (…) to enter a long description and to specify the group.

   The long description appears in the HTML documentation generated by the RBO HTML documentation tool. The long description can contain HTML tags that help the RBO HTML documentation tool display this description. For more information, see Creating HTML documentation for RBOs, on page 37.

   The **Group** field allows you to group RBOs for documentation purposes only. For example, if you have an RBO related to updating accounts, you could define a group as UpdateAccounts and assign it to each new RBO that is related to updating accounts.

3. If you want to prevent this class from being used as a superclass for any new classes, select the **Final Class** check box.

Defining properties of the RBO

In the **Properties** tab of the RBO Editor, you can create new properties or edit existing properties where allowed.

1. To the right of the **Properties** tab, click the **Insert Row** icon (▲).

   **Note:** If you insert a row that you no longer need, click the **Delete Row** icon (▼) to remove it.

2. In the **Name** cell of the new row, enter a name for the property that you are defining for the RBO. To change the name for a user-defined property, click the cell to enable edit mode.

3. In the **Default Value** cell, set the default value for the property, if applicable.

   The property must have Create (C) access to allow setting a default value. The value that you set here is assigned to the property by default when a user creates an instance of the RBO at run time.

   To change the default value at any point for a user-defined property, click the cell to enable edit mode.

4. In the **Type** cell, select a property type.

   Use the following table to decide which property type to choose.

   **Table 2: Property types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Denotes that the value for the property is a string.</td>
</tr>
<tr>
<td>mvString</td>
<td>Denotes that the value for the property is a string that contains multivalued data (U2 values, subvalues, or both).</td>
</tr>
<tr>
<td>File</td>
<td>Denotes that the value for the property is the name of a file.</td>
</tr>
<tr>
<td>LogicalFile</td>
<td>Denotes that the value for the property is the name of a logical file.</td>
</tr>
</tbody>
</table>
### Type and Description

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ref</td>
<td>This is the RBO object handle created by a BASIC method call on the server. From your BASIC code, you can store the object handle in an RBO property of type Ref. This handle is then used by client code, such as RedPages.NET code, to interact with the object.</td>
</tr>
<tr>
<td>mvRef</td>
<td>Denotes that the value for the property contains the names of a number of Refs.</td>
</tr>
<tr>
<td>Field</td>
<td>Denotes that the value for the property is a field from the UniData or UniVerse dictionary file defined for this RBO. The dict_file or logical_file property determines the dictionary for this RBO.</td>
</tr>
<tr>
<td>mvField</td>
<td>Denotes that the value for the property is a field containing multivalued data from the UniData or dictionary file defined for this RBO. The dict_file or logical_file property determines the dictionary for this RBO.</td>
</tr>
<tr>
<td>FieldNames</td>
<td>Denotes that the value of the property is a list of selected field names from the UniData or UniVerse dictionary file defined for this RBO. The dict_file or logical_file property determines the dictionary for this RBO.</td>
</tr>
</tbody>
</table>

**Note:** The assignment of a property type is for documentation purposes. You can run the RBO HTML Documentation tool (detailed in Creating HTML documentation for RBOs, on page 37) from Web Designer and give the output to RedPages.NET or RedBeans developers so that they know whether to expect a string or some other type. Web Designer also uses the type internally to determine, for example, whether to display a field column for an RBO, depending on whether any of its properties are of the field type.

5. In the **Description** cell, click the ellipsis button (...) and enter a short and long description and a group, if applicable.

6. The **Field** cell is populated for the RBO if any of its properties are of the type “Field.” If the property type is Field or mvField, select a database field name.

**Note:** You must define either the dict_file property or logical_file property before fields become available to assign to properties of type “Field.” The fields in the selection list come from the dictionary specified in the dict_file or logical_file property. If both the dict_file and logical_file properties are defined, the logical_file property determines the dictionary fields and the dict_file property is ignored.

7. The **Superclass** cell displays the name of the class from which the property was inherited. The property’s superclass cannot be changed. If you are defining a new property of the RBO, superclass is not applicable.

8. In the **Access** cell, select a setting to indicate whether the value of the property can be created at design time or can be read or written at run time. The default is **CRW**, which allows create, read, and write access.

The following table describes each value.

**Table 3: Access values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Create. You can set a default value for the property at design time in the RBO Editor. This default value is assigned to the property when an object of this class is instantiated at run time.</td>
</tr>
<tr>
<td>R</td>
<td>Read. The value of the property can be read at run time using the RBO.getProperty() function call.</td>
</tr>
</tbody>
</table>
Defining methods of the RBO

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>Write. The value of the property can be set at run time using the RBO.setProperty() function call.</td>
</tr>
</tbody>
</table>

**Tip:** You can sort a column by clicking the column heading. If you have large lists, performance can be improved by clicking the Disable drop lists icon (▲). Once disabled, the icon will change to have a red line through it, indicating that drop lists are disabled.

**Defining methods of the RBO**

In the **Methods** tab of the RBO Editor, you can create new methods or edit existing methods where allowed.

1. To the right of the **Methods** tab, click the **Insert Row** icon (▲).
2. In the **Name** cell of the new row, enter a name for the method you are defining for the RBO.
   To change the name for a user-defined property, click the cell to enable edit mode.
3. In the **Description** cell, click the ellipsis button (…) and enter a short and long description, and select the input and output properties.
   Input properties are the properties that you must set before calling this method, and output properties are the properties that will be set by the method. Setting these properties provides more information to the RedPages.NET or RedBeans programmers.
4. In the **Cataloged Name** cell, select the name of the BASIC subroutine that has been cataloged in the UniData or UniVerse database.
   **Note:** Do not define any parameters for subroutines that are intended to become methods.
   To pass arguments into a BASIC program, you use properties. Assign values to the required properties before calling the method and then ‘get’ the values of these properties once in the BASIC subroutine.
5. The **Superclass** cell displays the name of the class from which the method was inherited. The method’s superclass cannot be changed. If you are defining a new method of the RBO, superclass is not applicable and input is disabled.
6. Select **File > Save** to save the new RBO with its added properties and methods.

**Editing an RBO**

You can edit an RBO that already exists in the Web Designer.

1. In the RBO List, U2 Resource, or RBO Hierarchy view, expand the U2 Web DE Elements tree and a module, and double-click an existing RBO to edit.
   **Tip:** Select **Window > Show View > RBO Hierarchy** to display all RBOs in a hierarchical structure. In this view, all objects supplied with Web Designer are marked SYSTEM to distinguish custom-developed object definitions from system-supplied objects.

   2. Switch to edit mode by clicking the **Lock** icon (▲).
      For more information about why and how you must lock an RBO to edit it, **Locking and unlocking an RBO**, on page 18.
   3. Edit identifying information, as described in **Entering identifying information for the new RBO**, on page 15
   4. Edit properties, as described in **Defining properties of the RBO**, on page 15.
5. Edit methods, as described in Defining methods of the RBO, on page 17.
6. After you edit properties and methods, click the Save icon or select File > Save.
7. Click the Unlock icon to release the lock.

Creating an RBO from a U2 database file

After you connect to a U2 server and enable an account for Web DE, you can use the Database Files node to create an RBO from a U2 database file.

1. Expand an account and the Database Files node.
2. Right-click the name of the U2 database file that you want to use to create an RBO, and select Create RBO from selection.
3. In the Choose a parent module wizard, select the parent module to contain the new RBO, and click Next.
4. Select the RBO’s superclass and in the RBO Name field, enter a unique name. Click Next.
   
   **Note:** The name is subject to the restrictions detailed in the last step of Creating an RBO, on page 13.

5. Select the check box next to the dictionary field definitions that you want to use as properties of the RBO.
   
   If you selected the Query RBO superclass on the previous wizard page, continue with the following substeps; otherwise proceed to step 6.
   
   a. Change the order of the selected items by dragging the fields to a different location above or below their current positions. Click Next.
   
   b. Select the fields to determine the sorting of your query results. You can drag the fields to different locations.

6. Click Finish.
   
The RBO is added to the list of RBOs for the parent module that you selected in step 3.

Deleting an RBO

You can delete an RBO by using the Web Designer. If other objects inherit from an object that you delete, they become orphaned, and can be identified in the RBO Hierarchy view under the Orphaned Classes node. Note that an orphaned object cannot be instantiated.

1. In the RBO List, U2 Resource, or RBO Hierarchy view, right-click the RBO that you want to delete, and select Delete.
2. From the Delete RBO dialog box, click Yes.

Locking and unlocking an RBO

You must lock an RBO to edit its attributes, properties, or methods. Only one user can lock an RBO at a time. This prevents multiple developers from overwriting each other’s changes. Locking is based on the full path of the object, meaning that application IDs and account names can be duplicated on a server, but the object can still be unique. When you lock an object, the definition is downloaded and an entry is created in the WWLOCKS file to indicate that you have opened the object. The RedBack Object Server timestamps the entry in the WWLOCKS file. The entry in the WWLOCKS file remains until you close the object or unlock it. When you close the Web Designer, all open objects are closed, and their entries are removed from the WWLOCKS file.
Maintaining logical file definitions

1. From the RBO List, U2 Resource, or RBO Hierarchy view, right-click the name of an RBO and select **Lock RBO**.
   
   If using the U2 Resource view, you can also press **Ctrl** to select multiple RBOs to lock.

   Alternatively, a **Lock** icon ((pf) is available on the toolbar.

   **Note:** You cannot lock an RBO that is already locked. You can only access a read-only version of the object definition.

2. Make any changes to the RBO by double-clicking it and editing the details.

3. To save the changes, select **File > Save**.

4. To unlock the RBO, right-click it and select **Unlock RBO**.

   Alternatively, an **Unlock** icon (pf) is available on the toolbar.

**Maintaining logical file definitions**

You can combine the field definitions from a number of physical files into a single ‘logical’ file definition. This logical file definition is used during development and at run time as if it were a single physical file. You can specify the name of the logical file definition in the logical_file property of a uObject class definition.

1. Expand the U2 Web DE Elements tree, right-click a module, and select **New Logical File**.

2. From the New Logical File view, in the **Description** field, enter an appropriate description for the logical file.

3. If you are sure that the only way to update the primary file in the logical file definition is through the logical file, select the **Lock Only Primary Item** check box.

   If you select this check box, the RedBack Object Server locks only the primary file. If you do not select this check box, the RedBack Object Server locks each file that is defined in the logical file definition, which decreases performance.

4. Under **Source Files**, define the files that you want to include in the logical file definition:

   a. Click the **Insert Row** icon (pf).

      **Note:** The file on the first row is treated as the primary file in the logical file definition. To move the rows, drag them to the desired location.

   b. In the Source Files area, enter information for each physical file to be included in the logical view. The following table describes the source files.

   **Table 4: Source files and descriptions**

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dict File</td>
<td>Select the dictionary file from the Dict File list. If you define a dictionary file used to describe data from a file of a different name, you must also enter the data file name in the next column.</td>
</tr>
<tr>
<td>Data File</td>
<td>Select the data file from the Data File list.</td>
</tr>
</tbody>
</table>
### Chapter 3: Developing RBOs

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Source Field</td>
<td>If a primary file, the field is not editable, as the primary file is not accessed through a key source file. The RedBack Object Server accesses the primary file by using the key data from the RBO class definition. As with other application objects, the key data passed to the read method describes a single physical item.</td>
</tr>
<tr>
<td></td>
<td>For any physical file other than the primary file, enter the key source field. The RedBack Object Server uses the key source field to access the physical file. The value must be an existing field name in the collection of selected field names that you defined for the files above this one in the grid.</td>
</tr>
<tr>
<td></td>
<td>The key source field name can refer to a physical location in a database item, or can be a virtual field that derives a value. For example, consider a file with a multivalued list of product numbers. The key to a second file can be made up of the key to the first file, an asterisk, and then the product number. You can create a virtual field formula to represent this.</td>
</tr>
<tr>
<td>MV Key</td>
<td>If a primary file, the field cannot have a multivalued key. For other physical files, if the data in the key source field is multivalued, you must click the cell to read “Yes.”</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If the MV Key field is “Yes,” and the key value contains multiple entries, the RedBack Object Server reads each individual key value. In this way, the RedBack Object Server can read multiple physical records to construct multivalued fields in the logical record. Only a single value is taken from each physical item read. If the field is “No,” the RedBack Object Server performs a single read.</td>
</tr>
<tr>
<td>Fields</td>
<td>Select the fields from the primary dictionary that you want to include in the logical file definition. Field names can refer to a physical location in a record, or can be virtual fields that derive a value.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Selected field names must be unique across the logical file definition. That is, you cannot select the CUSTOMER field from one file, and then another field called CUSTOMER from a second file.</td>
</tr>
</tbody>
</table>

c. Continue inserting rows and entering information in the grid as necessary. As you enter rows, you can select a row, and then click the **Insert Row** icon to add the row below your currently selected one.

5. After you enter information for all physical files to be included in the logical file definition, click the **Save** icon or select **File > Save**, and enter a unique name for the new logical file.

## Creating a subroutine

You can use the Web Designer code editor to create and edit the code for subroutines that are stored on the RedBack Object Server.

### Prerequisite
- Defining properties of the RBO, on page 15
- Defining methods of the RBO, on page 17
Editing a subroutine

Procedure

1. In the RBO List, U2 Resource, or RBO Hierarchy view, expand the U2 Web DE Elements tree and a module, and double-click an existing RBO to edit.

2. Switch to edit mode by clicking the Lock icon.
   For more information about why and how you must lock an RBO to edit it, Locking and unlocking an RBO, on page 18.

3. Click in the Description cell, and click the ellipsis button (...).

4. Set the Input Properties and Output Properties of the subroutine as appropriate. Click OK.

5. Click in the Cataloged Name cell of the method for which you are creating a subroutine.

6. Type a descriptive name for the new subroutine or select a name from the list. This subroutine is the BASIC program that is run when the RBO method call is made. In the following example, a cataloged program called “Validate” runs when RedPages.NET or RedBeans code calls the PostRead method.

   Figure 4: Validate cataloged program

7. Click the Edit Basic Source Code icon.

8. On the tab that opens, add code to the basic source code as required.
   Web Designer automatically inserts comments into the code page based on the entries you make. Web Designer automatically inserts a $INCLUDE statement to the code listing. Web Designer provides a template text file, includetemplate.txt, in the directory in which the product was installed. Modifying this template determines the contents that appear when creating a subroutine.

9. When finished, click the Compile U2 BASIC Subroutine for Method icon. This action saves, compiles, and catalogs the UniVerse BASIC or UniBasic subroutine on the RedBack Object Server.

   Note: The RBO Editor compiles the subroutine in Pick flavor and directly catalogs the subroutine. You might want to add a compile directive in your program to override this default and manually recatalog the program from the command line prompt.

Editing a subroutine

You can edit an existing subroutine by using the code editor.
Prerequisite

- Defining properties of the RBO, on page 15
- Defining methods of the RBO, on page 17
- Creating a subroutine, on page 20

Procedure

1. In the RBO List, U2 Resource, or RBO Hierarchy view, expand the U2 Web DE Elements tree and a module, and double-click an existing RBO to edit.

2. Switch to edit mode by clicking the Lock icon ( 

   For more information about why and how you must lock an RBO to edit it, Locking and unlocking an RBO, on page 18.

3. In the Cataloged Name column, click the name of the subroutine you want to edit.

4. Click the Edit Basic Source Code icon ( 

5. On the tab that opens, edit code to the basic source code as needed.

6. When finished, click the Compile U2 BASIC Subroutine for Method ( 

   This action saves, compiles, and catalogs the UniVerse BASIC or UniBasic subroutine on the RedBack Object Server.
Chapter 4: Developing server code for RBOs

The following sections provide an overview of developing server side basic subroutines.

Developing RBOs involves specifying the contents of a subroutine to gain access to the properties, and then setting properties to return to the calling client application, invoking business logic code. This interaction with the RBO structure is achieved by calling RBO API function.

Server API functions

The following sections describe the server API functions.

RBO.createObject()

**Syntax**

```
Status = RBO.createObject(className, handle)
```

**Description**

The RBO.createObject function creates an instance of the class you specify and returns a handle to the object created. If an error occurs, the function passes the error back to that Status. If no error occurs, the function sets Status to 0 or RBO.NO ERROR.

```
Status = RBO.createObject("Department", deptHandle)
IF Status # RBO.NO.ERROR THEN .......... (error in creating instance)
```

When you have a handle to the instance, in this case deptHandle, you can access properties and methods of that class.

---

**Note:** The className must be the RBO name and not the module:RBO name. You can only reference RBOs within the same class in this manner.

You cannot create an SLRBO from UniBasic or UniVerse BASIC because it is stateless.

RBO.getProperty()

**Syntax**

```
Status = RBO.getProperty(handle, propName, value)
```

**Description**

The RBO.getProperty function returns the value of a property that you specify for the object referenced by handle. If the property is from the current object from which Web DE calls this method, you do not pass anything in handle. If an error occurs, the function passes the error back to Status. If no error occurs, the function sets Status to RBO.NO.ERROR.
In the following example, the RBO.getProperty function returns the current salary in the argument value. If an error occurs, the RedBack Object Server sets Status to the appropriate error value.

```c
Status = RBO.getProperty('', "Salary", value)
```

In the following example, RBO.setProperty gets the value of the property SalaryStatus that exists in the OFFICE:Department RBO, using the current instance of the class referenced by deptHandle.

```c
Status = RBO.getProperty(deptHandle, 'SalaryStatus', value)
```

## RBO.setProperty()

**Syntax**

```c
Status = RBO.setProperty(handle, propName, value)
```

**Description**

The RBO.setProperty function sets the value of a property that you specify for the object referenced by handle. If the property is from the current object from which the method is called, you do not pass anything in handle. If an error occurs, the function passes the error back to Status. If no error occurs, the function sets Status to RBO.NO.ERROR.

The following example sets the value of the ServerStatus property to 1.

```c
Status = RBO.setProperty('', 'ServerStatus', 1)
```

## RBO.callMethod()

**Syntax**

```c
Status = RBO.callMethod(handle, methodName)
```

**Description**

The RBO.callMethod function calls the method that you specify for the object referenced by handle. If the method is from the current object, you do not pass anything in handle. If an error occurs, the function passes the error back to Status. If no error occurs, the function sets Status RBO.NO.ERROR.

The following example calls the CheckSalary method in the class referenced by the deptHandle handle.

```c
Status = RBO.callMethod(deptHandle, 'CheckSalary')
```
RBO.setError()

Syntax

RBO.setError(text)

Description

The RBO.setError function sets the error message that is passed back to the client for display purposes. The following example sets the error message to the text string that you specify.

RBO.setError("U1")

RBO.getDBVals()

Syntax

Status = RBO.getDBVals(names, values)

Description

The RBO.getDBVals function enables you to retrieve the field values that are received from the U2 data server before assigning them to the associated property of an RBO. You can call GetDBVals only from a postRead, preWrite, or postWrite method, and you can use only valid database field names (not RBO property names). You can retrieve multiple property names, delimited by @AM.

RBO.setDBVals()

Syntax

RBO.setDBVals(names, values)

Description

The RBO.setDBVals function enables you to manipulate field values that are received from the U2 data server before assigning those values to the associated property value of an RBO. You can call setDBVals from only a postRead, preWrite or postWrite method, and you can use only valid field names (not RBO property names). You can set multiple property names, delimited by @AM.

STATUS = RBO.setDBVals("EMP.ID", EMPID);
**WW.FILEVAR.S**

**Syntax**

`WW.FILEVAR.S(fname, mode, f.var, err)`

**Description**

The WW.FILEVAR.S subroutine opens and closes files, and maintains the list of open file names and associated file handles. This subroutine provides additional error handling options and a logical close operation, which is not supported by the WW.OPEN.FILE function.

The following table describes the parameters.

**Table 5: WW.FILEVAR.S parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `fname`  | The name of the file to open or close.  
           | -1 clears the file handle secondary cache. |
| `mode`    | 0 – Opens the file and reports and errors  
           | 1 – Opens the file, but does not report errors  
           | 2 – Closes the file  
           | 3 – Forces the file to open  
           | 4 – Forces the file to open, but does not report any errors  
           | 5 – Invalidates cache entry for `fname`  
           | A forced open means that Web DE ignores any existing handle for the file. Web DE effectively closes the file, and then opens it. Forcing a file to open is useful for opening remote file pointer items. |
| `f.var`   | The file variable if `mode` is 0, 1, 3, or 4. |
| `err`     | The file operation was successful.  
           | 1 – The file operation failed. |

**Options for setting text with RBO.setMessage and RBO.setError**

You can set the text for RBO.setMessage and RBO.setError by using the following syntax.

`ERROR message displays error messages as specified in message, which can have one of the following formats:`

- `n`
  - where `n` is an attribute in item ERRORS in WWTEXT.
- `Un`
  - where `n` is an attribute in item ERRORS in SHAREDTEXT or in xxTEXT.
- `string`
  - where `string` is the text to display.
- `n:VM:daata:VM:daata2:VM`...
Caching and storing file handles

You can reduce the number of times Web DE needs to open files by caching or storing file handles. You can write a subroutine that opens files and stores their handles in COMMON, and call the subroutine on connection start. You can also use the file open API, WW.FILEVAR.S, which returns a handle of the file that you open. This handle comes from the file cache.

Be aware that using the WW.FILEVAR.S subroutine has implications of logical file opens on the application. For example, consider the following sequence of events:

1. You write a program to process a list of files. For each file name in a list, the program creates a remote file pointer (Q-Pointer) called QFILE and writes it to the VOC file. The program now calls WW.FILEVAR.S to open the file. The program returns the file handle and the name 'QFILE' is stored in the cache of 'open file names'.

2. A problem now exists. If you call WW.FILEVAR.S by using mode 0 or 1, each call after the first call returns the file handle of the first file. This is because the same file name is being used for different files. In this type of situation, the file handle cache should not be searched when requesting a file open action. Use of modes 3 or 4 would give the required result.

The implication of the preceding example is that applications should not reuse file names unless you open the file by using mode 3 or 4.

For file opens in your application that are called from RBO methods, the WW.FILEVAR.S function might not be relevant. It might be more appropriate to use existing code and call it during a connection startup.

Essentials to add to a BASIC subroutine

Every UniVerse BASIC or UniBasic subroutine should include the following statement to ensure that you have access to property names, which makes your code and error status codes easier to read:

```
$INCLUDE WWINSERT RBO.H
```

Example

The following example illustrates a method that is called from an RBO subroutine to validate a salary. It gets the current value for the Salary and Department properties, and then calls the CheckSalary method in the OFFICE:Department RBO.

```
SUBROUTINE ValSal
* called to validate salary as the ValidateSalary method from OFFICE:Employee
*
$INCLUDE WWINSERT RBO.H
$INCLUDE WWINSERT STD.H
```
* now get the salary entered
Status = RBO.getProperty('', 'Salary', SALARYVALUE)
IF Status # RBO.NO.ERROR THEN
    Status = RBO.setError("CANNOT GET SALARY")
    RETURN
END

* now get the department entered
Status = RBO.getProperty('', 'Department', DEPTVALUE)
IF Status # RBO.NO.ERROR THEN
    Status = RBO.setError("CANNOT GET DEPARTMENT")
    RETURN
END

* now check if we have an instance to the department RBO and if not then create one
Status = RBO.getProperty('', 'depObj', DEPTHANDLE)
IF Status # RBO.NO.ERROR THEN
    Status = RBO.setError("CANNOT GET DEPARTMENT HANDLE")
    RETURN
END

IF DEPTHANDLE = "" THEN
    Status = RBO.createObject("Department", DEPTHANDLE)
    IF Status = RBO.NO.ERROR THEN
        Status = RBO.setProperty('', 'depObj', DEPTHANDLE) ; * set handle for subsequent calls
        IF Status # RBO.NO.ERROR THEN
            Status = RBO.setError("CANNOT SET DEPARTMENT HANDLE")
            RETURN
        END ELSE RETURN
    END ELSE RETURN

* now set the salary value and call the check salary method in department instance
RBO.setProperty(DEPTHANDLE, "Salary", SALARYVALUE)
* now set the current department for this employee
RBO.setProperty(DEPTHANDLE, "CheckDept", DEPTVALUE)
* now check the salary
RBO.callMethod(DEPTHANDLE, "CheckSalary")
* now check the status
RBO.getProperty(DEPTHANDLE, "SalaryStatus", SALSTATUS)
IF SALSTATUS = 0 THEN ; * ok so leave
    RETURN
END

* if here then salary is not ok!
Status = RBO.setProperty('', "ServerStatus", SALSTATUS)
Status = RBO.setError("INVALID SALARY ENTERED")
RETURN

Automatically calling subroutines

You can call specific subroutines for a variety of session actions. To do this, specify the subroutines that you want to call in the SESSION.EVENTHANDLERS item in the WWCONTROL file. All Web DE-enabled U2 accounts share the WWCONTROL file.

The fields of the SESSION.EVENTHANDLERS item are as described in the following table.
Table 6: RBO function status values

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Att1</td>
<td>Reserved.</td>
</tr>
<tr>
<td>Att2</td>
<td>Reserved.</td>
</tr>
<tr>
<td>Att3</td>
<td>Reserved.</td>
</tr>
<tr>
<td>Att4</td>
<td>Reserved.</td>
</tr>
<tr>
<td>Att5</td>
<td>Subroutine to call on connection start (applicable only when WWCONTROL is used).</td>
</tr>
<tr>
<td>Att6</td>
<td>Subroutine to call on connection stop (applicable only when WWCONTROL is used).</td>
</tr>
</tbody>
</table>

Status values returned from RBO function calls

The following table describes the Status values returned by RBO functions. These values are equated in WWINsert RBO.H.

Table 7: RBO function status values

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBO.NO.ERROR</td>
<td>No errors occurred.</td>
</tr>
<tr>
<td>RBO.BAD.ARRAYSIZE</td>
<td>Invalid parameter array size.</td>
</tr>
<tr>
<td>RBO.BAD.OBJECT</td>
<td>Object not found.</td>
</tr>
<tr>
<td>RBO.BAD.CLASS</td>
<td>Class not found.</td>
</tr>
<tr>
<td>RBO.BAD.PROPNAME</td>
<td>Property not found.</td>
</tr>
<tr>
<td>RBO.BAD.PERMS</td>
<td>Permission denied.</td>
</tr>
<tr>
<td>RBO.ERROR</td>
<td>Generic error code.</td>
</tr>
</tbody>
</table>
Chapter 5: Testing RBOs

This chapter provides an overview of using the RBOScope testing tool to test RBOs.

Web DE provides two versions of the RBOScope testing tool:

- Accessing the RBOScope testing tool from Web Designer, on page 30
- Accessing the stand-alone JSP RBOScope testing tool, on page 32

RBO testing overview

The RedBack Object Server processes method calls on objects on request. Using this messaging system, Web DE provides an RBO testing tool that you use to test your RBOs in isolation, without the need to build an application by using a RedBeans or RedPages.NET form.

Testing your method calls with the RBOScope testing tool provides a mechanism for isolating problems. If there is a problem when calling a method from a RedBeans or RedPages.NET form, but the method call works from the RBOScope testing tool, the problem points to the client code rather than the RBO or the RedBack Object Server code.

The RBOScope testing tool exercises the RBO in the same way that a Java program would if running on the client. It uses a “live” RedBack object from the RedBeans.jar API and requires a valid setting in the rgw5.ini configuration file.

An Eclipse-based version of the RBOScope testing tool is built into Web Designer. Using this tool, you can test RBOs from within Web Designer.

A JSP version of the RBOScope testing tool is also available in Web DE. Using this tool, developers can test RBOs without accessing Web Designer.

Accessing the RBOScope testing tool from Web Designer

The RBOScope testing tool is a view within Web Designer. In this view, you can test your RBOs without leaving the development environment. When you open the RBOScope testing tool in Web Designer, Web Designer uses the user ID and password you entered in the User Credentials window of the RedBack Object Server Connection wizard to access the RedBack Object Server. If your connection times out, right-click the connection in the U2 Resource view and select Refresh.

1. To open the RBOScope testing tool, select Window > Show View > RBOScope Testing Tool.
2. From the RBO List, U2 Resource, or RBO Hierarchy view, right-click the name of the RBO you want to test, and click Test RBO.
Testing an RBO

1. To open the RBOScope testing tool, select **Window > Show View > RBOScope Testing Tool**.

2. In the RBO List, U2 Resource, or RBO Hierarchy view, right-click the RBO you want to test, and select **Test RBO**. You can also click the **RBOScope Testing Tool** icon.

   **Tip:** You can double-click the view to maximize it so you can work in the RBOScope testing tool easier, as shown in the following figure.

3. Next to the property that you want to include in the method, enter a value in the **Value** column.

4. From the Methods list, click the link provided for the method name that you want to test.

   The RBOScope testing tool passes the values to the method, and then displays the current values for the RBO properties. If the method call affected any of the RBO’s properties, the updated values are displayed in the **Value** column.

5. To monitor the performance of method calls to the RBO, click **Start Performance Monitor**. This opens a table that collects timing results for the methods you test.

6. To switch to a different RBO, select the RBO that you want to test from the **RBO Quick Changer** list.

   The **Open in new tab** check box is selected by default. With this check box selected, the tab for the current RBO remains, and the selected RBO opens in a new tab. If you want the selected RBO to replace the current RBO in the same tab, clear the check box.
To test an RBO outside of Web Designer, see Testing RBOs with the stand-alone JSP RBOScope testing tool, on page 32.

**Accessing the stand-alone JSP RBOScope testing tool**

You can use the stand-alone JSP RBOScope testing tool to test RBOs outside the development environment.

**Prerequisite**

- “Installing the JSP RBOScope testing tool” in *Getting Started*.

**Procedure**

1. Open a web browser.
2. In the web browser, navigate to http://localhost:8080/rboscope, which redirects the browser to: http://localhost:8080/rboscope/RBOScopeLogin.jsp

   **Note:** The URL used to navigate to the stand-alone JSP RBOScope testing tool can be different depending on your installation. These steps assume you have chosen the default paths during installation.

3. Select an account, enter the credentials, and then click **Connect**.

   The account list is populated with accounts from the `rgw5.ini` file. The default user name of the administrator is `rbadmin` and the default password is `redback`.

**Testing RBOs with the stand-alone JSP RBOScope testing tool**

After you access the JSP RBOScope testing tool, you can use it to test RBOs outside of Web Designer.

**Prerequisite**

- Accessing the stand-alone JSP RBOScope testing tool, on page 32

**Procedure**

1. From the **Module Name** droplist, select a module that contains the RBO to test.
2. From the **Class Name** droplist, select an RBO to test.
   The selected RBO is displayed in the RBO testing grid.
Configuring the encoding of the JSP RBOScope testing tool

You can configure how the JSP RBOScope testing tool handles encoding by updating the `rgw5.ini` file. When configured to a specific encoding type, the JSP RBOScope testing tool will support any session encoding and display the information correctly.

1. Open the `rgw5.ini` file in a text editor.
2. In the `[accountname]` section, add the following:
   ```ini
   encoding=type
   ```
   where `type` is the encoding type. For example, UTF-8, or ISO 8859-1.
3. Save and close the file.
Chapter 6: RBO debug logging

You can use RBO debug logging to track values of RBO properties between method calls, as well as profile the time taken to run a method. Web DE provides a way to automatically log property values as well as profile method execution times, but you can also do this manually.

Two properties of the default RBO class, debug and debug_id, provide debug logging capabilities; these properties are inherited by every class. The RBO uQuery class also logs the actual UniBasic or UniVerse BASIC SELECT verb that the RedBack Object Server uses to perform a selection.

debug property

The debug property is of type String. You can set the debug flag, which is used to automatically call the RedBack Object Server internal debug logging program from a method. The RedBack Object Server places all debug information in the WWLOG file located in the rbdefn account.

The debug flag can contain one of three values:

- 0: indicates do not log
- 1: indicates to log start and finish times only
- 2: indicates to log start and finish times, as well as RBO property values

The debug property has access codes of RW. The format of the debug file is shown in the following example:

```
001 RBO Debug
002 RBO Object Name
003 RBO Method Name
004 Session ID
005 Start timedate
006 Finish timedate
007 "Start timedate"
008 property1=value1
009 property2=value2
010 ...
01n property=value
010 "Finish timedate"
01n property1=value1
01n property2=value2
01n ...
01n property=value
```

debug_id property

The debug_id property is of type String, and allows you to specify the name of the debug log item the RedBack Object Server writes to the WWLOG file. If you do not specify a value, the key has the format of classname.methodname.sessionid. Otherwise, the RedBack Object Server saves the item as debug_id.value.methodname.

The following is an example item list from the WWLOG file:

```
LIST WWLOG 19:14:53 MAR 28 2001 1
WWLOG...................................................
default_test.ReadData
Employee.Clear.12141
Employee.ReadData.12141
```

Debug log information

The RedBack Object Server saves all debug information to items in the `WWLOG` file. This file is a directory type file, allowing convenient access at the operating system level. The RedBack Object Server saves all items in the format of `classname.methodname.sessionid`, unless you specify otherwise in the debug_id property.

**Note:** Be sure to monitor the size of the items in the `WWLOG` file and clear items as required.

The following partial listing illustrates debug information obtained from a ReadData method call on the Employee RBO, with the debug property flag set to 2. This setting shows the start and finish times of the method call, as well as the property values at the start and finish times:

```
RBO Debug
Employee
ReadData
18:57:33 Mar 28 2001
   ___18:57:32 Mar 28 2001_________________________
EmpId=1006
debu=2
FirstName=
LastName=
ImageFile=
Salary=
HireDate=
Interests=
Dept=
dict_file=EMPLOYEES
data_file=EMPLOYEES
DeptCodes=
DeptDescs=
.....
   ___18:57:33 Mar 28 2001_________________________
EmpId=1006
debu=2
FirstName=Mario
LastName=Delas
ImageFile=emp_mario_delas.jpg
Salary=990.00
HireDate=05/08/1998
Interests=Music
Dept=3
dict_file=EMPLOYEES
data_file=EMPLOYEES
DeptCodes=
DeptDescs=
.....
```

Web DE also logs debug information to the COMO files. This information is a summary of the items saved to the `WWLOG` file; it includes the name of the item, the debug flag used, and the start and finish times for the method called.

The following example illustrates a COMO file listing the debug information:

```
COMO UO.SPIDER_393310 established 15:21:39 19 Nov 2010
```
Manually invoking the RBO debug process

Although Web DE provides functionality to automatically log property values as well as profile method execution times, you can also directly call the Web DE debug subroutine from your own methods. This enables you to specifically debug isolated sections of your method. Web DE saves all debug information to the WWLOG file and summaries to the COMO file.

Use the following call to manually invoke RBO debugging within a method:

CALL WW.RBO.DEBUG(RBO.DBG.FLAG)

where RBO.DBG.FLAG has four options:

- 1: Log method started
- 2: Log method started and includes RBO’s properties
- 3: Log method completed
- 4: Log method completed and includes RBO’s properties

The following example illustrates a user method that calls to WW.RBO.DEBUG with options 2 and 4. It starts and completes log properties.

```plaintext
SUBROUTINE RBO.TEST2
  *
  $INCLUDE WWINSERT STD.H
  $INCLUDE WWINSERT SB.COMMON
  * Log method started
  CALL WW.RBO.DEBUG('2')
  * Body of Method
  CALL WW.SB.COMMON(1, ERR)
  USERDATA(1) = "UserData1!"
  CALL WW.SB.COMMON(4:AM:6, ERR)
  * Log method completed
  CALL WW.RBO.DEBUG('4')
  *
  RETURN
  *
END
```
Chapter 7: Creating HTML documentation for RBOs

An automatic documentation tool is available from the Tools menu in Web Designer. This tool parses all class definitions for all RBOs in a module and produces HTML pages that show the inheritance, properties, and methods of each class in the module.

An example of the HTML documentation is available in your installation directory in the U2 account server. For example, /U2/U2WDE511/UVserver/rbexamples. This location contains a Documenta000 folder that contains all the HTML files for modules, classes, and more that the rbexamples account uses. The index-all.html file organizes all these .html files for you. When you generate your own HTML documentation, the files will be similar but placed in a location of your choosing.

Generating RBO HTML documentation

To generate RBO HTML documentation, make sure that a Web DE-enabled U2 account is connected to the RedBack Object Server, and any changes to the RBOs are saved.

For information about how to edit RBOs, see Developing RBOs, on page 13.

1. From the U2 Resource view, select a module for which you want to generate HTML documentation.
2. Select Tools > Generate RBO HTML Documentation.
3. In the Generate RBO HTML Documentation window, enter the name of the host directory where you want the tool to store the generated HTML documentation, and click OK.
Chapter 8: Setting up remote procedure calls

The following sections describe Web DE remote procedure calls (RPCs), an alternative to using RBOs. You can use RPCs to call a UniVerse BASIC or UniBasic subroutine directly and receive a RecordSet.

RPCs overview

The RPC protocol is an alternative to using RBOs. Web DE RPC allows you to call a UniVerse BASIC or UniBasic subroutine directly from the web.

You might want to use Web DE RPC for the following reasons:

▪ To replace existing middleware that works in the same way, minimizing the changes necessary to implement Web DE.
▪ For maximum performance when generating reports that return a large number of records.
▪ To receive a RecordSet back from a UniVerse BASIC or UniBasic subroutine, where a uQuery select cannot gather the data, and you do not want to use an SLRBO to return delimited strings that represent records.

The result of an RPC request is a RecordSet in ASP.NET or Java.

A disadvantage of using RPCs is that they provide less data abstraction than RBOs.

Note: Web DE RPCs are stateless. If session state is required, you must maintain it in some part of your application outside of Web DE.

Setting up RPCs

To set up an RPC, you must create a pointer.

Procedure

1. Enable the UniData or UniVerse account for RPC access.
2. Log on to the account as the RedBack user.
3. Create a pointer as follows:
   ▪ If the account is a UniVerse account, create an F-pointer in the VOC file to WWPROGS in the rbdefn account.
   ▪ If the account is a UniData account, create a DIR-type item in the VOC file pointing to WWPROGS in the rbdefn account.
4. Enter the following command:
   :RUN WWPROGS ENABLE.RBO

Results

The WW.RPC program reads the first parameter in your request, which indicates which subroutine to call. After calling the subroutine, the program passes your second and third parameters as the first and second parameters to your subroutine. Web DE uses a third parameter to receive the attribute mark-delimited list of data from the subroutine. The RedBack client components convert this attribute mark-delimited list of data into the appropriate record and set of records for the calling program.
Writing an RPC subroutine

Web DE checks for the first repeating name in the list to determine where to break the returning list of data into records. For example, assume the program returns the following string:

```
FIRSTNAME=BOB@AMLASTNAME=SMITH@AMFIRSTNAME=JOHN@AMLASTNAME=DOE
```

Web DE locates FIRSTNAME, and breaks the string into two records, each with two fields, FIRSTNAME and LASTNAME.

Writing an RPC subroutine

The following sections describe the server API functions.

An RPC subroutine must conform to the following requirements:

- Have three parameters: two input and one output.
- Cannot process screen input, as this causes the RPC call to hang.
- Processing of the RPC must finish in less time than it takes for the web request to time out.
- Must return the third parameter response as attribute mark delimited “name=value” pairs, as shown in the following example:

```
FNAME=BOB@AMLASTNAME=SMITH@AMFNAME=JOHN@AMLASTNAME=DOE
```

Example

Web DE includes an example RPC subroutine, EMP.RPC, in the EXMODLIB directory of the rbexamples account. The following example shows a simplified version of the source code:

```fortran
SUBROUTINE EMP.RPC(PARAMS, RECORD, RESPONSE)
*Note we are using PARAMS to get our input (we could also have used RECORD)
*We are using RESPONSE to return an @AM delimited string of name=value pairs
  PROG.ID = ’EMP.RPC’
  METHOD = PARAMS<1>
  FNAME = PARAMS<2>
  ID = PARAMS<3>
  STATUS = 0
  MESS = ’’
  RESPONSE = ’’
  BEGIN CASE
  CASE METHOD = ’READ’ ; GOSUB 1000
  CASE METHOD = ’SELECT’ ; GOSUB 5000
  END CASE
  RESPONSE<-1> = ”SERVERSTATUS=”:STATUS
  RESPONSE<-1> = ”SERVERMESSAGE=”:MESS
  RETURN
  **********************
  * Read a Record
  **********************
  1000*
  OPEN ’’,FNAME TO F.FILE THEN
  READ RECORD FROM F.FILE,ID THEN
  GOSUB 1500 ;* Format Record Set
  END ELSE
  STATUS = 1
  MESS = ID:’ NOT FOUND!’
  END
  END ELSE
  STATUS = 1
  MESS = FNAME:’ NOT OPENED!’
```
Calling an RPC subroutine from Java or C#

You can call an RPC subroutine from Java or C# using the RedObject.CallRPC API call. For more information about using Java or C# with Web DE, see Developing Applications with RedBeans and Developing Applications with RedPages.NET.

Example

The following example illustrates a Web DE RPC subroutine from Java or C#.

```java
Connection oConn = new Connection("rpcaccountname");
```
RedObject oCust = new RedObject();
oCust.setConnection(oConn);
// set param1 as required by subroutine you are going to call
// this could be a delimited string of several params in one
String param1 = “myparam1”;
// set param2 as required by subroutine you are going to call
String param2 = “param”;
// call subroutine, getting back a recordset or null object
RecordSet rs = oCust.callRPC(“mysubroutinename”, param1, param2);
// you can then loop thru your recordset so:
for(int z =1; z <=10; z++) {
  if(rs.isEOF()) { return; }
  String s =””;
  if(!rs.isEOF()) {
    s = rs.getProperty(“EMP.ID”)+””+ rs.getProperty(“FIRST.NAME”) + 
    “ “ + rs.getProperty(“LAST.NAME”);
    // do something with s
    // ...
  }
  rs.moveNext();
}