Rocket U2 Web Development Environment

Application Developer’s Guide

Version 5.3.0

July 2017
WDE-530-ALL-DG-01
Notices

Edition

Publication date: July 2017
Book number: WDE-530-ALL-DG-01
Product version: Version 5.3.0

Copyright

© Rocket Software, Inc. or its affiliates 1996-2017. 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.
Corporate information

Rocket Software, Inc. develops enterprise infrastructure products in four key areas: storage, networks, and compliance; database servers and tools; business information and analytics; and application development, integration, and modernization.

Website: www.rocketsoftware.com

Rocket Global Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA

To contact Rocket Software by telephone for any reason, including obtaining pre-sales information and technical support, use one of the following telephone numbers.

<table>
<thead>
<tr>
<th>Country</th>
<th>Toll-free telephone number</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>1-855-577-4323</td>
</tr>
<tr>
<td>Australia</td>
<td>1-800-823-405</td>
</tr>
<tr>
<td>Belgium</td>
<td>0800-266-65</td>
</tr>
<tr>
<td>Canada</td>
<td>1-855-577-4323</td>
</tr>
<tr>
<td>China</td>
<td>400-120-9242</td>
</tr>
<tr>
<td>France</td>
<td>08-05-08-05-62</td>
</tr>
<tr>
<td>Germany</td>
<td>0800-180-0882</td>
</tr>
<tr>
<td>Italy</td>
<td>800-878-295</td>
</tr>
<tr>
<td>Japan</td>
<td>0800-170-5464</td>
</tr>
<tr>
<td>Netherlands</td>
<td>0-800-022-2961</td>
</tr>
<tr>
<td>New Zealand</td>
<td>0800-003210</td>
</tr>
<tr>
<td>South Africa</td>
<td>0-800-980-818</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>0800-520-0439</td>
</tr>
</tbody>
</table>

Contacting Technical Support

The Rocket Community is the primary method of obtaining support. If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Community and report a problem, download an update, or read answers to FAQs. To log in to the Rocket Community or to request a Rocket Community account, go to www.rocketsoftware.com/support.

In addition to using the Rocket Community to obtain support, you can use one of the telephone numbers that are listed above or send an email to support@rocketsoftware.com.
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 manual covers accessing RedBack objects (RBOs) from .NET and Java. It describes the core RedPages.NET API and RedBeans Java API.

This manual discusses both RedPages.NET and RedBeans alongside each other. It introduces RBO Data Sources for ASP.NET, and describes how the RedBeans Java API can be used in Java Server Pages and other Java applications.

The content in this manual provides an overview of RedPages.NET and RedBeans; it does not include installation information. For information about how to install RedPages.NET or RedBeans, see Installation and Configuration.

This documentation assumes that you know both C# and the .NET Framework (if using RedPages.NET) or Java (if using RedBeans).

RBOs

Web DE allows you to design your business rules and logic using RBOs. RBOs are object representations of U2 data and business logic.

For more information about RBOs, see Web Designer User’s Guide.

Using RedPages.NET

RedPages.NET is a collection of .NET classes that are Web DE-aware, and can communicate with RBOs on the RedBack Object Server.

For information about how to use RedPages.NET to access RBOs from C#, see Using RedPages.NET or RedBeans to access RBOs from C# or Java, on page 23.

RedPages.NET overview

The RedPages.NET API is a collection of .NET classes that provide access to the RBOs that reside in a UniData or UniVerse (U2) database. This allows access for easy building of .NET applications against a U2 database. You can use any .NET authoring tool, such as Microsoft Visual Studio, to design the applications for your web pages.

RedPages.NET API

The RedPages.NET API functionality is provided by the RedPages.NET assembly, which provides the classes required to connect and interact with RBOs on the RedBack Object Server.
In development mode, RedPages.NET should be installed in the same location where your .NET development environment or editor is installed.

For run time, you must install the RedPages.NET.dll where the required environment can access it.

RedPages.NET data source controls

In addition to the core RedPages.NET API, Web DE also includes a set of .NET data source controls that allow RBOs to participate in ASP.NET data binding.

The RedPages.NET data source controls are designed to easily bind data to the controls. Development tools such as Visual Studio have extensive tooling built around the concept of data binding, and often no code is required to link data sources to their corresponding Web controls.

The following data source controls are available:

- RBODataSource
- UQueryDataSource

The RBODataSource control provides single-record semantics for your application, and allows the data to be viewed one record at a time. This control is useful in situations where only one record per page needs to be manipulated, such as on an ASP.NET form.

The UQueryDataSource control provides record-set semantics for your application. This control is useful in situations where sets of records are bound to list-type and table-type controls, such as the ASP.NET ListView control.

Drag-and-drop operations and wizards help you bind the data Sources to the controls on the page. When you drag a control onto the Visual Studio form, it is bound to the data source and all the required ASPX code is generated automatically. You can add the RBO connection details by using the wizards that are connected to the controls.

You can also use the wizards to designate the appropriate RBO Create, Read, Update, and Delete (CRUD) operations, instead of having to add these to the code manually. For example, the CRUD operations for a uObject-based RBO might be WriteData, ReadData, WriteData, and DeleteData, respectively.

If you want to create programs that go beyond the capabilities supplied by the wizards, you can program against the events supplied by the ASP.NET infrastructure. This allows you to tie in the various phases of data binding, giving you full flexibility within your applications.

For an example of how to bind the data source to the controls, see (RedPages.NET only) Creating a RedPages.NET application, on page 38.

Using RedBeans

RedBeans is a collection of packages that contain classes which give access to RBOs that reside in a UniData or UniVerse (U2) database. This allows for easy building of Java applications against a U2 database. You can use any Java authoring tool to design the applications for web pages, Java Server Pages (JSP), or servlets, and more. If you are using web pages, these can be built using any HTML editor.

RedBeans overview

RedBeans functionality is provided by the RedBeans packages, which provide the classes required to connect and interact with RBOs on the RedBack Object Server. In development mode, the RedBeans packages must be installed in the same location that your Java IDE or editor is installed. For run time,
you must install the required RedBeans packages where the required environment can access them. For example, if using JSP, the JSP plug-in needs to know where the package resides.
Chapter 2: RedBeans and RedPages.NET APIs

The RedBeans and RedPages.NET APIs are libraries of the object classes, interfaces, and value types contained within the RedBeans jar file and RedPages.NET assembly, respectively. These are the classes that you can use within the application code to access U2 systems from within their .NET or Java application. The classes are driven by a series of methods that instruct the application on how to access and manipulate the Web DE data.

The following classes can be used in a RedBeans or RedPages.NET application:

- RedObject class, on page 8
- RecordSet class, on page 11
- RedSet class, on page 13
- RedField class, on page 15
- RbException class, on page 17
- UnidataProfiler class, on page 19

RedObject class

The RedObject class represents an RBO class instance on the client. It calls RedBack Object Server methods on behalf of its client and can store all state for that RBO class instance.

The constructors, methods, and properties of the RedObject class are described in the following tables. Properties only apply to RedPages.NET.

Table 1: RedObject constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public RedObject()</td>
<td>Creates a RedObject instance without referencing a connection or an RBO.</td>
</tr>
<tr>
<td>public RedObject(string RBOAccount, string RBOClass)</td>
<td>Creates a RedObject instance and associates a connection and an RBO with the instance.</td>
</tr>
<tr>
<td>Parameters: RBOAccount is the RBO account to which you want to connect. RBOClass is the RBO to associate with this RedObject instance (in the form of “module:RBO” where module is the module that contains the RBO).</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: RedObject methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void open()</td>
<td>Opens an RBO instance.</td>
</tr>
<tr>
<td>RbException: An RbException is thrown if open() cannot take place, indicating a connection error, RedPages.NET or RedBeans error, or server error.</td>
<td></td>
</tr>
<tr>
<td>public RedObject(string RBOAccount, string RBOClass)public void open(string account)</td>
<td>Opens an RBO instance.</td>
</tr>
<tr>
<td>Parameters: account is the account to which you want to connect.</td>
<td></td>
</tr>
<tr>
<td>RbException: An RbException is thrown if open() cannot take place, indicating a connection error, RedPages.NET or RedBeans error, or server error.</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| public void open(string account) | Opens an RBO instance.  
Parameters: `account` is the account to which you want to connect.  
RbException: An RbException is thrown if `open()` cannot take place, indicating a connection error, RedPages.NET or RedBeans error, or server error. |
| public void setProperty(string propertyName, string value) | Sets the current value for a property.  
Parameters: `propertyName` defines a property in the RBO. `value` is the current value of the property. |
| public void setProperty(string propertyName, UniDynArray value) | An alternative NLS method that sets the current value for a property name.  
Parameters: `propertyName` defines a property in the RBO. `value` is the current value of the property. |
| public string getProperty(string propertyName) | Gets the current value for a property.  
Parameters: `propertyName` is the name of the property whose value should be returned.  
Return value: Returns the current value. Null if property name is not found. |
| public UniDynArray getPropertyToDynArray(string propertyName) | An alternative NLS method that gets the current value for a property name. For RedBeans, this is the value as set by the last server call, not the value that may be held in a RedBean associated with the specified propertyName.  
Parameters: `propertyName` is the name of the property whose value should be returned.  
Return value: (RedPages.NET only) Returns the current value. Null if property name is not found. |
| public RecordSet callMethod(string methodName) | Calls the specified method for the RBO. The method call takes place on the RedBack Object Server.  
Parameters: `methodName` is the name of the RBO method to invoke.  
Return value: The return value is a RecordSet object if the method is the Select method or the PageDisp method on a uQuery. Otherwise, the return value is null. |
| public ArrayList refresh(string handle) | Refreshes the RedObject instance with RedBack Object Server state for the required session as determined by the handle passed. The session details are passed in to identify the specific session.  
If the current session details are the same as the ones passed in, the object returns the current property values. If the session details are different, a request is made to the RedBack Object Server to get the requested information.  
Parameters: `handle` is the handle obtained by using the `getRBOHandle()` method.  
Return value: Returns the current property values after refresh. |
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| public ArrayList refresh(UniDynArray handle) | An alternative NLS method that refreshes this instance with RedBack Object Server state for the required session as determined by the handle passed.  
Parameters: `handle` is the RBO handle for the RBO instance.  
Return value: Returns the current property values after refresh. |
| public void setRBOHandle(string RBOHandle) | Sets the RBOHandle on the RedObject. This handle is generated by the RedBack Object Server and can be used later to refresh a RedObject with data from the RedBack Object Server.  
Parameters: `RBOHandle` is the current handle, as returned by a previous call to `getRBOHandle()`. |
| public void setRBOHandle (UniDynArray RBOHandle) | *(RedBeans only)* An alternative NLS method that sets the RBOHandle on the RedObject as a UniDynArray.  
Parameters: `RBOHandle` is the current handle, as returned by a previous call to `getRBOHandleToDynArr()`. |
| public string getRBOHandle() | For RedBeans, gets the RBOHandle for the RBO. For RedPages.NET, gets the RBOHandle on the RedObject. This handle is generated by the RedBack Object Server and can be used later to refresh a RedObject with data from the RedBack Object Server.  
Return value: Returns current RBOHandle details for the current RBO. This handle is generated by the RedBack Object Server and can be used later to refresh a RedObject with data from the RedBack Object Server. |
| public string getRBOHandleToDynArr() | *(RedBeans only)* An alternative NLS method that gets the RBOHandle for the current RBO, as a UniDynArray.  
Return value: Returns the current RBO as a UniDynArray. |
| public string getSessionId() | Gets the SessionId for the current RBO. This session ID is generated by the RedBack Object Server and can be used later to create other RBOs as part of the same session.  
Return value: The return value is the session ID for the current RBO. |
| public string getSessionIdTo DynArr() | *(RedBeans only)* An alternative NLS method that gets the SessionId for the current RBO, as a UniDynArray.  
Return value: Returns the session ID for the current RBO as a UniDynArray. |
| public void setSessionId(string sessionId) | Sets the SessionId for the current RBO.  
Parameters: `sessionId` is the current session ID, as returned by a previous call to `getSessionId()`. |
| public void setSessionId (UniDynArray sessionId) | *(RedBeans only)* Sets the SessionId for the current RBO  
Parameters: `sessionId` is the current session ID, as returned by a previous call to `getSessionId()` |
| public UniDynArray SessionId | An alternative NLS property. |
| public string ServerAlert | *(RedBeans only)* Retrieves any alert messages from the server. This value is set after each callMethod.  
Return value: Returns the alert message from the last request to the server. |
RecordSet class

The RecordSet class represents the set of rows returned from a uQuery object. It provides methods to iterate through the rows returned by the uQuery.

The methods and properties of the RecordSet class are described in the following tables. Properties only apply to RedPages.NET.

Table 4: RecordSet methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public string getProperty(string propertyName)</td>
<td>Gets the field value for the current row.</td>
</tr>
<tr>
<td></td>
<td>Parameters: propertyName is the field name to return for the current row.</td>
</tr>
<tr>
<td></td>
<td>Return value: Returns the value for the field name for the current row.</td>
</tr>
</tbody>
</table>

RecordPages.NET only

Table 3: RedObject properties for RedPages.NET

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public UniDynArray RBOHandle</td>
<td>Gets the current RBOHandle details for the current RBO. This handle is generated by the RedBack Object Server and can be used later to refresh a RedObject with data from the RedBack Object Server and to also associate other RedObjects with the current session. Parameters: RBOHandle is the current handle, as returned by a previous call to getRBOHandle(). Return value: Returns current RBOHandle details for the current RBO. This handle is generated by the RedBack Object Server.</td>
</tr>
<tr>
<td>public string ServerAlert</td>
<td>Holds any alert messages from the server. This value is set after each callMethod.</td>
</tr>
</tbody>
</table>

RedBeans only

Method Description

public RecordSet callRPC(string subroutineName, string param1, string param2) (RedBeans only) Allows you to invoke BASIC and UniVerse BASIC subroutines directly from the RedBeans API. For more information about RPC calls, see the information in Web Designer User’s Guide.

Parameters: subroutineName is the name of the subroutine to invoke using RPC, param1 is the first parameter of the subroutine to be invoked using RPC, and param2 is the second parameter of the subroutine to be invoked using RPC.

Return value: Returns a RecordSet representation of the rows returned from the RPC.

RbException: Throws an RbException if an error occurs during the call.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void move(int pos)</td>
<td>Moves the row number (cursor) to a specified position. If moved past the last row, it will set EOF to true. If moved before row 1, it will set BOF to true. If another page is required, it requests that page. If the row requested is greater than the amount of rows, the page number is set to the last page plus one and the row number is set to the last row plus one. Parameters: pos is the row number position.</td>
</tr>
<tr>
<td>public void moveFirst()</td>
<td>Moves the current row position to 1.</td>
</tr>
<tr>
<td>public void movePrevious()</td>
<td>Moves the row number (cursor) back 1. If past the initial row, it sets BOF to true. If another page is required, it requests that page to return true.</td>
</tr>
<tr>
<td>public void moveNext()</td>
<td>Moves the row number (cursor) forward 1 position. If it moves beyond the last position, it will set EOF to true.</td>
</tr>
<tr>
<td>public void moveLast()</td>
<td>Moves the current row position to the last row.</td>
</tr>
<tr>
<td>public bool movePrevPage()</td>
<td>Moves the RecordSet to the previous page. Return value: Returns true if successful, or false if already on the first page.</td>
</tr>
<tr>
<td>public bool moveNextPage()</td>
<td>Moves the RecordSet to the next page. Return value: Returns true if successful, or false if already on the last page and there are no more pages to view.</td>
</tr>
<tr>
<td>public void refresh(string objHandle)</td>
<td>Refreshes the state from RedBack Object Server and sets data to page one. You can then use the move(int pos) method to set the row number as required. Parameters: objHandle is the handle of the RBO instance from which to refresh values.</td>
</tr>
<tr>
<td>public int getAbsolutePage()</td>
<td>(RedBeans only) Gets a specific page and also sets row number and relative row number.</td>
</tr>
<tr>
<td>public int getAbsolutePosition ()</td>
<td>(RedBeans only) Gets the current row position.</td>
</tr>
<tr>
<td>public bool isBOF()</td>
<td>(RedBeans only) Returns boolean value depending on the beginning of the record set. Return value: Returns true if returned at beginning of record set (before row 1); otherwise, it returns false.</td>
</tr>
<tr>
<td>public bool isEOF()</td>
<td>(RedBeans only) Returns a boolean value depending on the end of the record set. Return value: Returns true, if at end of record set (after last row); otherwise, the return value is false.</td>
</tr>
<tr>
<td>public int getMaxPages()</td>
<td>(RedBeans only) Returns the maximum number of pages for this uQuery.</td>
</tr>
<tr>
<td>public int getNumItem()</td>
<td>(RedBeans only) Gets the count of number of items selected.</td>
</tr>
<tr>
<td>public int getPageSize()</td>
<td>(RedBeans only) Gets the number of items per page.</td>
</tr>
<tr>
<td>public int getRelativePosition ()</td>
<td>(RedBeans only) Gets the current row position relative to the current page.</td>
</tr>
</tbody>
</table>
### RedSet class

The RedSet class is used to create a storage object to save state in the .NET or Java tier. It does not communicate to the RedBack Object Server. It is passed an initial list of fields and from then on can add or set values for these fields.

The methods and properties of the RedSet class are described in the following tables. Properties only apply to RedPages.NET.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public string getRow()</code></td>
<td>(RedBeans only) Gets the data for the selected row. Return value: Returns the data for a selected row. Each column is attribute mark (AM) delimited. RbException: An RbException is thrown if at EOF, or BOG, or if it cannot decode resultant string.</td>
</tr>
<tr>
<td><code>public UniDynArray getRowToDynArray()</code></td>
<td>(RedBeans only) An alternative NLS property that gets the UniDynArray for the selected row. Return value: Returns the data for a selected row. Each column is attribute mark (AM) delimited. RbException: An RbException is thrown if at EOF, or BOG, or if it cannot decode resultant string.</td>
</tr>
</tbody>
</table>

### RedPages.NET only

**Table 5: RecordSet properties for RedPages.NET**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public int AbsolutePage</code></td>
<td>Gets a specific page and also sets row number and relative row number.</td>
</tr>
<tr>
<td><code>public int AbsolutePosition</code></td>
<td>Gets the current row position.</td>
</tr>
<tr>
<td><code>public bool BOF</code></td>
<td>Returns boolean value depending on the beginning of the record set. Returns <code>true</code> if returned at beginning of record set (before row 1); otherwise, it returns <code>false</code>.</td>
</tr>
<tr>
<td><code>public bool EOF</code></td>
<td>Returns a boolean value depending on the end of the record set. Returns <code>true</code>, if at end of record set (after last row); otherwise, the return value is <code>false</code>.</td>
</tr>
<tr>
<td><code>public int MaxPages</code></td>
<td>Returns the maximum number of pages for this uQuery.</td>
</tr>
<tr>
<td><code>public int NumItem</code></td>
<td>Gets the count of number of items selected.</td>
</tr>
<tr>
<td><code>public int PageSize</code></td>
<td>Gets the number of items per page.</td>
</tr>
<tr>
<td><code>public int RelativePosition</code></td>
<td>Gets the current row position relative to the current page.</td>
</tr>
<tr>
<td><code>public string Row</code></td>
<td>Gets the data for the selected row.</td>
</tr>
<tr>
<td><code>public UniDynArray RowToDynArray</code></td>
<td>An alternative NLS property that gets the UniDynArray for the selected row.</td>
</tr>
</tbody>
</table>
### Table 6: RedSet methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void addNew()</td>
<td>Adds a new row to the RedSet class without setting any values. Positions the row number so that it is ready for <code>setProperty</code> calls.</td>
</tr>
<tr>
<td>public void addNew(string[] fieldList, string[] values)</td>
<td>Adds values passed to corresponding field names. The field names must be one of the field names passed in the <code>open</code> method call. Parameters: <code>fieldList</code> is the name of the field to update with passed value. <code>values</code> is the value to update.</td>
</tr>
<tr>
<td>public void close()</td>
<td>Closes the storage arrays and sets all values to null.</td>
</tr>
<tr>
<td>public void delete()</td>
<td>Deletes the current row.</td>
</tr>
<tr>
<td>public void getFieldPos(string field)</td>
<td>Returns the current position within the field names array of the specified field. Parameters: <code>field</code> is the specified field name.</td>
</tr>
<tr>
<td>public string getProperty(string fieldName)</td>
<td>Returns the value for the field name for the current row. Parameters: <code>fieldName</code> is the field name to return for the current row.</td>
</tr>
<tr>
<td>public void move(int pos)</td>
<td>Moves the row number (cursor) to a specified position. If past the last row, it sets EOF to true; if before row 1, it sets BOF to true. Parameters: <code>pos</code> is the row number position.</td>
</tr>
<tr>
<td>public void moveFirst()</td>
<td>Moves the row cursor to the first position in the RedSet.</td>
</tr>
<tr>
<td>public void moveLast()</td>
<td>Moves the row cursor to the last position in the RedSet.</td>
</tr>
<tr>
<td>public void moveNext()</td>
<td>Moves the row number (cursor) forward 1. If past the last row, it sets EOF to true.</td>
</tr>
<tr>
<td>public void movePrevious()</td>
<td>Moves the row number (cursor) back 1. If past the initial row, it sets BOF to true. If another page is required, it requests that page.</td>
</tr>
<tr>
<td>public void open(string fieldList where)</td>
<td>Initializes the data stores in RedSet. Allocates a storage array for each field name passed. Parameters: <code>fieldList</code> is a comma delimited array of field names.</td>
</tr>
<tr>
<td>public void setProperty(string field, string value)</td>
<td>Sets the field to the specified value for the current row. Parameters: <code>field</code> is the field name to set. <code>value</code> is the value to set it to.</td>
</tr>
<tr>
<td>public int getAbsolutePosition()</td>
<td>(RedBeans only) Returns the current row position</td>
</tr>
<tr>
<td>public bool isBOF()</td>
<td>(RedBeans only) Returns true if the row cursor is less than one; otherwise, it returns false.</td>
</tr>
<tr>
<td>public bool isEOF()</td>
<td>(RedBeans only) Returns true if the row cursor is at a position after the last row, otherwise false.</td>
</tr>
<tr>
<td>public int NumItems()</td>
<td>(RedBeans only) Returns the number of items in this RedSet.</td>
</tr>
</tbody>
</table>
RedField class

The RedField class provides dynamic string handling. The RedField string supports value mark (VM) and subvalue mark (SVM) delimited arrays.

The methods and properties of the RedField class are described in the following tables.

Table 7: RedSet properties for RedPages.NET

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public int AbsolutePosition { get; }</td>
<td>Returns the current row position.</td>
</tr>
<tr>
<td>public bool BOF { get; }</td>
<td>Returns true if the row cursor is less than one; otherwise, it returns false.</td>
</tr>
<tr>
<td>public bool EOF { get; }</td>
<td>Returns true if the row cursor is at a position after the last row, otherwise false.</td>
</tr>
<tr>
<td>public int NumItems { get; }</td>
<td>Returns the number of items in this RedSet class.</td>
</tr>
</tbody>
</table>

Table 8: RedField methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public int count()</td>
<td>Returns the number of values separated by a value mark in this instance of a RedField object.</td>
</tr>
<tr>
<td></td>
<td>Return value: Returns an integer that represents how many time values separated by a value mark were found.</td>
</tr>
<tr>
<td>public int count(int vm_c)</td>
<td>Returns the number of values separated by a subvalue mark at the position specified.</td>
</tr>
<tr>
<td></td>
<td>Parameters: vm_c (an integer) is the value mark position for extraction.</td>
</tr>
<tr>
<td></td>
<td>Return value: Returns an integer that represents how many time values separated by a value mark were found.</td>
</tr>
<tr>
<td>public void del(int vm_c)</td>
<td>Deletes a dynamic string element in the current RedField string at the position specified.</td>
</tr>
<tr>
<td></td>
<td>Parameters: vm_c (an integer) is the value position for deletion.</td>
</tr>
<tr>
<td>public void del(int vm_c, int svm_c)</td>
<td>Deletes a dynamic string element in the current RedField string at the position specified.</td>
</tr>
<tr>
<td></td>
<td>Parameters: vm_c (an integer) is the value position for deletion. svm_c (an integer) is the subvalue position for deletion.</td>
</tr>
<tr>
<td>public int getLength()</td>
<td>Returns the number of characters in this instance of a RedField object.</td>
</tr>
<tr>
<td></td>
<td>Return value: Returns an integer that represents the number of characters.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>public int getLength(int vm_c)</td>
<td>Returns the number of characters at the position specified. Parameters: vm_c (an integer) is the value mark position for extraction. Return value: Returns an integer that represents the number of characters.</td>
</tr>
<tr>
<td>public string getValue()</td>
<td>Gets the string representation of the RedField.</td>
</tr>
<tr>
<td>public string getValue(int vm_c, int svm_c)</td>
<td>Extracts a dynamic string element from the current RedField string at the position specified. Parameters: vm_c (an integer) is the value position for extraction. svm_c (an integer) is the subvalue position for extraction. Return value: Returns a string value.</td>
</tr>
<tr>
<td>public string getValue(int vm_c)</td>
<td>Extracts a dynamic string element from the current RedField string at the position specified. Parameters: vm_c (an integer) is the value mark position for extraction. Return value: Returns VM the character.</td>
</tr>
<tr>
<td>public void ins(int vm_c, string pval)</td>
<td>Inserts a string into the current RedField string at the position specified. Parameters: vm_c (an integer) is the value position for insertion that can be set to -1 to indicate ‘insert at end,’ but only if am_c is &gt; 0. pval (a string) is the string value that is to be inserted at the specified position.</td>
</tr>
<tr>
<td>public void ins(int vm_c, int svm_c, string pval)</td>
<td>Inserts a string into the current RedField string at the position specified. Parameters: vm_c (an integer) is the value position for insertion that may be set to -1 to indicate ‘insert at end,’ but only if am_c &gt; 0. svm_c (an integer) is the subvalue for insertion that can be set to -1 to indicate ‘insert at end,’ but only if am_c and vm_c values are &gt; 0. pval (a string) is the string value that is to be inserted at the specified position.</td>
</tr>
<tr>
<td>public RedField()</td>
<td>Constructs a new RedField class with no characters.</td>
</tr>
<tr>
<td>public RedField(string sval)</td>
<td>Constructs a new RedField object, setting its initial value to that of the passed string. Parameters: sval is a string.</td>
</tr>
<tr>
<td>public void replace(int vm_c, string pval)</td>
<td>Replaces a dynamic string element in the current RedField string at the position specified. Parameters: vm_c (an integer) is the value position for deletion. pval (a string) is the string value that replaces the string at the specified position. If the position does not exist in the target string, this string is inserted.</td>
</tr>
<tr>
<td>public void replace(int vm_c, int svm_c, string pval)</td>
<td>Replaces a dynamic string element in the current RedField string at the position specified. Parameters: vm_c (an integer) is the value position for replacement. svm_c (an integer) is the subvalue for replacement. pval (a string) is the string value that replaces the string at the specified position. If the position does not exist in the target string, this string is inserted.</td>
</tr>
</tbody>
</table>
Handling multivalue data in an NLS or I18N configuration

When you create a multivalue field in a UniVerse NLS or UniData I18N environment, you need to obtain a UniDynArray from a UO session object so that the proper marks as defined on the server are used.

You can create an instance of a UniDynArray to handle multivalue fields by calling the `getUniDynArray` method of the U2 Web DE connection class. This will allow you to create a multivalue field on your Web DE client application.

For example:

```java
// Create connection object to the rbexamples account
Connection oConn = new Connection("rbexamples");

// From connection object obtain a UniDynArray to handle Multi-value fields.
UniDynArray oDynArray = oConn.getUniDynArray();
```

The UniDynArray class is used just like the UniDynArray `getProperty` method in the RedObject class. For more information, see RedObject class, on page 8.

### RbException class

The RbException class represents the exception generated from RedPages.NET or RedBeans at various classes and methods.

The properties, methods, and constants of the RbException class are described in the following tables and section.
RedPages.NET only

Table 10: RbException properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public string Message</td>
<td>The current exception message.</td>
</tr>
<tr>
<td>public int Type</td>
<td>The type of exception. Return value: This property returns the following exception types:</td>
</tr>
<tr>
<td></td>
<td>• Connection error = 1;</td>
</tr>
<tr>
<td></td>
<td>• Server error = 2;</td>
</tr>
<tr>
<td></td>
<td>• RedPages.NET error = 3;</td>
</tr>
<tr>
<td>public string TypeDesc</td>
<td>This method gets the description of the exception type. Return value: This property returns the following exception types:</td>
</tr>
<tr>
<td></td>
<td>• Connection error = 1;</td>
</tr>
<tr>
<td></td>
<td>• Server error = 2;</td>
</tr>
<tr>
<td></td>
<td>• RedPages.NET error = 3;</td>
</tr>
</tbody>
</table>

RedBeans only

Table 11: RbException methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public string Message</td>
<td>The current exception message.</td>
</tr>
<tr>
<td>public int Type</td>
<td>The type of exception. Return value: This property returns the following exception types:</td>
</tr>
<tr>
<td></td>
<td>• Connection error = 1;</td>
</tr>
<tr>
<td></td>
<td>• Server error = 2;</td>
</tr>
<tr>
<td></td>
<td>• RedBeans error = 3;</td>
</tr>
<tr>
<td>public string TypeDesc</td>
<td>This method gets the description of the exception type. Return value: This property returns the following exception types:</td>
</tr>
<tr>
<td></td>
<td>• Connection error = 1;</td>
</tr>
<tr>
<td></td>
<td>• Server error = 2;</td>
</tr>
<tr>
<td></td>
<td>• RedBeans error = 3;</td>
</tr>
</tbody>
</table>

RbException constants

The RbException class supports the following constants:
• public const int CONNECTIONERROR = 1;
• public const int SERVERERROR = 2;
• (RedPages.NET) public const int REDPAGESERROR = 3;
• (RedBeans) public const int REDBEANSERROR = 3;
UnidataProfiler class

The UnidataProfiler class represents an RBO class instance on the client. It calls RedBack Object Server methods on behalf of its client and can store all state for that RBO class instance.

The constructors, methods, and properties of the UnidataProfiler class are described in the following tables. Properties only apply to RedPages.NET.

Table 12: UnidataProfiler constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public UnidataProfiler()</td>
<td>Creates a UnidataProfiler instance without referencing a connection or an RBO.</td>
</tr>
<tr>
<td>public UnidataProfiler(string RBOAccount, string RBOClass)</td>
<td>Creates a UnidataProfiler instance and associates a connection and an RBO with the instance. Parameters: RBOAccount is the RBO account to which you want to connect. RBOClass is the RBO to associate with this UnidataProfiler instance (in the form of &quot;module:RBO&quot; where module is the module that contains the RBO).</td>
</tr>
</tbody>
</table>

Table 13: UnidataProfiler methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void open()</td>
<td>Opens an RBO instance. RbException: An RbException is thrown if open() cannot take place, indicating a connection error, RedPages.NET or RedBeans error, or server error.</td>
</tr>
<tr>
<td>public UnidataProfiler(string RBOAccount, string RBOClass) public void open(string account)</td>
<td>Opens an RBO instance. Parameters: account is the account to which you want to connect. RbException: An RbException is thrown if open() cannot take place, indicating a connection error, RedPages.NET or RedBeans error, or server error.</td>
</tr>
<tr>
<td>public void open(string account)</td>
<td>Opens an RBO instance. Parameters: account is the account to which you want to connect. RbException: An RbException is thrown if open() cannot take place, indicating a connection error, RedPages.NET or RedBeans error, or server error.</td>
</tr>
<tr>
<td>public void setProperty(string propertyName, string value)</td>
<td>Sets the current value for a property. Parameters: propertyName defines a property in the RBO. value is the current value of the property.</td>
</tr>
<tr>
<td>public void setProperty(string propertyName, UniDynArray value)</td>
<td>An alternative NLS method that sets the current value for a property name. Parameters: propertyName defines a property in the RBO. value is the current value of the property.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>public string getProperty(string propertyName)</code></td>
<td>Gets the current value for a property.  &lt;br&gt;<strong>Parameters:</strong> <code>propertyName</code> is the name of the property whose value should be returned.  &lt;br&gt;<strong>Return value:</strong> Returns the current value. Null if property name is not found.</td>
</tr>
<tr>
<td><code>public UniDynArray getPropertyToDynArray(string propertyName)</code></td>
<td>An alternative NLS method that gets the current value for a property name. For RedBeans, this is the value as set by the last server call, not the value that may be held in a RedBean associated with the specified <code>propertyName</code>.  &lt;br&gt;<strong>Parameters:</strong> <code>propertyName</code> is the name of the property whose value should be returned.  &lt;br&gt;<strong>Return value:</strong> <em>(RedPages.NET only)</em> Returns the current value. Null if property name is not found.</td>
</tr>
<tr>
<td><code>public RecordSet callMethod(string methodName)</code></td>
<td>Calls the specified method for the RBO. The method call takes place on the RedBack Object Server.  &lt;br&gt;<strong>Parameters:</strong> <code>methodName</code> is the name of the RBO method to invoke.  &lt;br&gt;<strong>Return value:</strong> The return value is a RecordSet object if the method is the Select method or the PageDisp method on a uQuery. Otherwise, the return value is null.</td>
</tr>
<tr>
<td><code>public ArrayList refresh(string handle)</code></td>
<td>Refreshes the UnidataProfiler instance with RedBack Object Server state for the required session as determined by the handle passed. The session details are passed in to identify the specific session.  &lt;br&gt;If the current session details are the same as the ones passed in, the object returns the current property values. If the session details are different, a request is made to the RedBack Object Server to get the requested information.  &lt;br&gt;<strong>Parameters:</strong> <code>handle</code> is the handle obtained by using the <code>getRBOHandle()</code> method.  &lt;br&gt;<strong>Return value:</strong> Returns the current property values after refresh.</td>
</tr>
<tr>
<td><code>public ArrayList refresh(UniDynArray handle)</code></td>
<td>An alternative NLS method that refreshes this instance with RedBack Object Server state for the required session as determined by the handle passed.  &lt;br&gt;<strong>Parameters:</strong> <code>handle</code> is the RBO handle for the RBO instance.  &lt;br&gt;<strong>Return value:</strong> Returns the current property values after refresh.</td>
</tr>
<tr>
<td><code>public void setRBOHandle(string RBOHandle)</code></td>
<td>Sets the RBOHandle on the UnidataProfiler. This handle is generated by the RedBack Object Server and can be used later to refresh a UnidataProfiler with data from the RedBack Object Server.  &lt;br&gt;<strong>Parameters:</strong> <code>RBOHandle</code> is the current handle, as returned by a previous call to <code>getRBOHandle()</code>.</td>
</tr>
<tr>
<td><code>public void setRBOHandle(UniDynArray RBOHandle)</code></td>
<td><em>(RedBeans only)</em> An alternative NLS method that sets the RBOHandle on the UnidataProfiler as a UniDynArray.  &lt;br&gt;<strong>Parameters:</strong> <code>RBOHandle</code> is the current handle, as returned by a previous call to <code>getRBOHandleToDynArr()</code>.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>public string getRBOHandle()</td>
<td>For RedBeans, gets the RBOHandle for the RBO. For RedPages.NET, gets the RBOHandle on the UnidataProfiler. This handle is generated by the RedBack Object Server and can be used later to refresh a UnidataProfiler with data from the RedBack Object Server. Return value: Returns current RBOHandle details for the current RBO. This handle is generated by the RedBack Object Server and can be used later to refresh a UnidataProfiler with data from the RedBack Object Server.</td>
</tr>
<tr>
<td>public string getRBOHandleToDynArr()</td>
<td>(RedBeans only) An alternative NLS method that gets the RBOHandle for the current RBO, as a UniDynArray. Return value: Returns the current RBO as a UniDynArray.</td>
</tr>
<tr>
<td>public string getSessionId()</td>
<td>Gets the SessionId for the current RBO. This session ID is generated by the RedBack Object Server and can be used later to create other RBOs as part of the same session. Return value: The return value is the session ID for the current RBO.</td>
</tr>
<tr>
<td>public string getSessionIdToDynArr()</td>
<td>(RedBeans only) An alternative NLS method that gets the SessionId for the current RBO, as a UniDynArray. Return value: Returns the session ID for the current RBO as a UniDynArray.</td>
</tr>
<tr>
<td>public void setId(string sessionId)</td>
<td>Sets the SessionId for the current RBO. Parameters: sessionId is the current session ID, as returned by a previous call to getSessionId().</td>
</tr>
<tr>
<td>public void setId(UniDynArray sessionId)</td>
<td>(RedBeans only) Sets the SessionId for the current RBO Parameters: sessionId is the current session ID, as returned by a previous call to getSessionId()</td>
</tr>
<tr>
<td>public UniDynArray SessionId</td>
<td>An alternative NLS property.</td>
</tr>
<tr>
<td>public string ServerAlert</td>
<td>(RedBeans only) Retrieves any alert messages from the server. This value is set after each callMethod. Return value: Returns the alert message from the last request to the server.</td>
</tr>
<tr>
<td>public RecordSet callRPC(string subroutineName, string param1, string param2)</td>
<td>(RedBeans only) Allows you to invoke BASIC and UniVerse BASIC subroutines directly from the RedBeans API. For more information about RPC calls, see the information in Web Designer User’s Guide. Parameters: subroutineName is the name of the subroutine to invoke using RPC, param1 is the first parameter of the subroutine to be invoked using RPC, and param2 is the second parameter of the subroutine to be invoked using RPC. Return value: Returns a RecordSet representation of the rows returned from the RPC. RbException: Throws an RbException if an error occurs during the call.</td>
</tr>
</tbody>
</table>
RedPages.NET only

Table 14: UnidataProfiler properties for RedPages.NET

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public UniDynArray</td>
<td>Gets the current RBOHandle details for the current RBO. This handle is generated by the RedBack Object Server and can be used later to refresh a UnidataProfiler with data from the RedBack Object Server and to also associate other UnidataProfilers with the current session.</td>
</tr>
<tr>
<td>RBOHandle</td>
<td>Parameters: RBOHandle is the current handle, as returned by a previous call to getRBOHandle().</td>
</tr>
<tr>
<td></td>
<td>Return value: Returns current RBOHandle details for the current RBO. This handle is generated by the RedBack Object Server.</td>
</tr>
<tr>
<td>public string</td>
<td>Holds any alert messages from the server. This value is set after each callMethod.</td>
</tr>
<tr>
<td>ServerAlert</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 3: Using RedPages.NET or RedBeans to access RBOs from C# or Java

The following sections provide step-by-step examples of how to access RBOs from C# (for RedPages.NET) or Java (for RedBeans).

Creating and using a RedObject

A Java class or .NET application can communicate with an RBO through the com.rs.u2.wde.redbeans.RedObject class for RedBeans, or U2.WDE.RedPages.RedObject class for RedPages.NET. The RedObject class provides methods that enable you to set and get values from the properties of its RBO, and to call RedBack Object Server methods.

The following steps provide an example of how to create and use an instance of RedObject to access RBOs on the host computer.

1. Create an instance of RedObject, for example,
   For RedPages.NET:
   ```csharp
   using U2.WDE.RedPages;
   RedObject currObj = new RedObject();
   ```
   For RedBeans:
   ```java
   import com.rs.u2.wde.redbeans.*;
   RedObject currObj = new RedObject();
   ```

2. Specify the details of the U2 Web DE-enabled account, for example,
   For RedBeans: `currObj.setRBOAccount(String accountname);`
   For RedPages.NET: `currObj.RBOAccount ="accountname";

   where `accountname` denotes the account name used by Web DE to represent a U2 account that has been enabled for Web DE. The `accountname` corresponds to an entry in the rgw5.ini file that maps the name to an account on the backend, such as rbexamples. For more information about the rgw5.ini file, see `rgw5.ini file, on page 46`.

3. Specify an RBO class for this object, for example,
   `currObj.RBOClass(String RBOClass);`

   where `RBOClass` denotes an RBO class in the format module:rboclass, for example,
   `currObj.RBOClass("EXMOD:Employee");`

   In RedPages.NET, the RedObject needs to know which RBO to access, and you must specify which module and RBO you will be accessing.
Chapter 3: Using RedPages.NET or RedBeans to access RBOs from C# or Java

Tip:

You can combine the first three steps into one step as follows:
RedObject currObj = new RedObject("rbexamples", "EXMOD:Employee");

You can alternatively create a RedObject without the rgw5.ini file by instantiating a RedObject programmatically with the rgw5.ini file parameters in it, for example:
RedObject currObj = new
RedObject("rbexamples",server, port, min, max, connectionString, "EXMOD:EmpReader");

4. If the RBO is stateful, open the RBO.
   With a stateful RBO, this step is necessary in order to create the RBO instance within the RedBack Object Server. Opening the RBO creates initial state for the RBO, as well as session details (if a new session), for example,
currObj.open();
open() throws RbException if an error occurs in the open. The following will catch the RbException.

   try {
currObj.open();
} catch(RbException rbe) {
   ...}

5. (RedPages.NET only) Get property values of the RBO, for example,
   String val = currObj.getProperty(String propname)
   where propname denotes a property name from the current RBO, for example,
   String lastName = currObj.getProperty("LastName");

6. Set property values of the RBO, for example,
currObj.setProperty(String propname, String value);
   where propname denotes a property name from the current RBO, for example,
currObj.setProperty("EmpId", "1014");

7. Call a method on the RBO, for example,
   RecordSet rs = currObj.callMethod(String methodName);
   If the method is a “Select” against a uQuery object, it returns a RecordSet object. Otherwise, a null object pointer is returned.

   try{
currObj.callMethod("ReadData");}
catch(RbException rbe)
{...}
   or
try {
RecordSet rs = currObj.callMethod("Select");
} catch(RbException rbe) {
   ...
}

The callMethod() throws an RbException on error when calling a method.
Creating and using a RecordSet object

8. (RedBeans only) Get property values of the RBO, for example,
   
   ```java
   String val = currObj.getProperty(String propname)
   ```
   
   where `propname` denotes a property name from the current RBO, for example,
   
   ```java
   String lastName = currObj.getProperty("LastName");
   ```

Creating and using a RecordSet object

If the RBO is a uQuery or RPC call, Web DE creates a `com.rs.u2.wde.redbeans.RecordSet` object for RedBeans, or `U2.WDE.RedPages.RecordSet` object for RedPages.NET. The RecordSet is used to return rows of data. A RecordSet allows code to move through the rows of data returned from the server and get properties for the current row. Users can retrieve pages of data and scroll through them to display in a report or lookup. For further information about RPCs, see *Web Designer User’s Guide*.

The following steps provide an example of how to create and use an instance of a RecordSet object.

1. Create a RedObject and set the U2 account name and class details.

   **Note:** The account must be a Web DE-enabled UniData or UniVerse account.

   For example, for RedPages.NET:
   ```java
   using U2.WDE.RedPages...
   RecordSet rs = null;
   RedObject currObj = new RedObject("rbexamples", "EmployeeList");
   ```

   For RedBeans:
   ```java
   import com.rs.u2.wde.redbeans.*;
   RecordSet rs = null;
   RedObject currObj = new RedObject("rbexamples", "EXMOD:EmployeeList");
   ```

2. Open the RBO, for example,

   ```java
   try {
   currObj.open();
   } catch(RbException rbe) {
   ...
   }
   ```

3. Call the `Select()` method on the RBO, creating a RecordSet object, for example,

   ```java
   try {
   rs = currObj.callMethod("Select");
   } catch(RbException rbe) {
   ...
   }
   ```

4. Scroll through the RecordSet rows, extracting column data as require, for example,

   For RedPages.NET:

   ```java
   //Now that we have the first page (returned by the "Select" call)
   //we can get and set values for that page that tell us how many items were
   //selected and what the page size is (determined by the RBO from the class
   //definition).
   int pageSize = rs.PageSize;
   string maxRows = currObj.getProperty("MaxRows");
   int maxPages = rs.MaxPages;
   ```
Chapter 3: Using RedPages.NET or RedBeans to access RBOs from C# or Java

/*This section of code will loop through each item on a page, printing the details to the console. The movement through each row is done with rs.MoveNext().
When a page is at the end then rs.EOF() will be set to true.
rs.moveFirst(); // set the row cursor at the first row
for (int z = 1; z <= pageSize; z++)
{
if (rs.EOF)
return;
String s = "";
if (rs.EOF)
s = "";
else
{
s = rs.getProperty("EMP.ID") + " " + rs.getProperty("FIRST.NAME")
+ " " + rs.getProperty("LAST.NAME");
// do something with data;
Console.Out.WriteLine(s);
} rs.moveNext();
}

For RedBeans:

/*Now that we have the first page (returned by the "Select" call) we can get and set values for the page that tell us how many items were selected and what the page size is (determined by the RBO from the class definition).*/
int pageSize = rs.getPageSize();
string maxRows = currObj.getProperty("MaxRows");
int maxPages = rs.getMaxPages();
/*This section of code will loop through each item on a page, displaying the details. The movement through each row is done with rs.MoveNext(). When a page is at the end, then rs.isEOF() will be set to true.
In the following example, data from each row’s details are extracted from the row Data and placed into a string variable.*/
rs.moveFirst(); // set the row cursor at the first row
for (int z = 1; z <= pageSize; z++)
{
if (rs.isEOF())
return;
String s = "";
if (rs.isEOF())
s = "";
else
{
s = rs.getProperty("EMP.ID") + " " +
rs.getProperty("FIRST.NAME") +
" " + rs.getProperty("LAST.NAME");
// do something with data
} rs.moveNext();
}

Creating and using a RedSet object

You can use a RedSet object to store rows of data. For example, you might need to store a shopping cart. You can use a RedSet to store rows of data within RedPages.NET or Java.
Creating and using a RedSet object

The following steps provide an example of how to create and use an instance of a RedSet object.

1. Create an instance of a RedSet object. This instance creates an empty shell for the new RedSet object, for example,
   RedSet redSet = new RedSet();
2. Call the open() method, passing a list of property names.
   Before you can do anything with the object, open the object and define the properties that it supports. In calling the open() method on the RedSet object, pass a list of its properties as a parameter. Opening a RedSet object defines the fields to be supported in its internal arrays, for example,
   redSet.open(fieldlist);
   where fieldlist denotes a comma-separated list of names of properties the object supports.
3. Add new records to the object.
   To save data in the RedSet object, you must first add a new record. You do this by calling the addNew() method. After calling the addNew() method, the new record becomes the current record, for example,
   redSet.addNew();
   or
   redSet.addNew(String[] fieldNames, String[] values);
   where fieldNames denotes an array of field names; each field name must exist in the initial list passed in the open() method call. And values denotes an array of corresponding values.
4. Select the current record.
   This syntax moves to the first, last, next, or previous record in a specified RedSet object, and makes that record the current record, for example,
   RedSet.{MoveFirst() | MoveLast() | MoveNext() | MovePrevious()}
5. Get property values of the RedSet object, for example,
   variable = redSet.getProperty(String propname);
   where propname denotes a field name that must exist in the initial list passed in the open() method call.
6. Set the property values of the RedSet object, for example,
   redSet.setProperty(String propname, String value);
   where propname denotes a field name that must exist in the initial list passed in the open() method call.
7. Optional: Clear the RedSet object, for example,
   redSet.close();
   This closes an open object and any dependent objects, making the RedSet object available for reuse.

Example

The following example shows the RedSet code.

For RedPages.NET:

using U2.WDE.RedPages;
//1. Create a RedSet object instance that will be used to manipulate our strings.
RedSet oRedSet = new RedSet();
//2. Define "field name"
   oRedSet.open("firstName,lastName,dept");

//3. Assign the rows of data.
String[] fields = new String[] { "firstName", "lastName","dept"};
String[] values = { "Mike", "Kontorovich", "Eng"};
oRedSet.addNew(fields, values);
values[0] = "Jurgen";
values[1] = "Joarder";
values[2] = "Eng";
oRedSet.addNew(fields, values);
values[0] = "Simon";
values[1] = "Caddick";
values[2] = "Support";
oRedSet.addNew(fields, values);
values[0] = "Nghia";
values[1] = "La";
values[2] = "Eng";
oRedSet.addNew(fields, values);

//4. Now scroll through the RedSet getting values.
oRedSet.moveFirst();
while (!oRedSet.EOF)
{
   Console.Out.WriteLine(oRedSet.getProperty("firstName"));
   Console.Out.WriteLine(oRedSet.getProperty("lastName"));
   Console.Out.WriteLine(oRedSet.getProperty("dept"));
oRedSet.moveNext();
}

//5. Finished, so can get rid of all data - this leaves the object available
   for re-use oRedSet.close();

For RedBeans:

   import com.rs.u2.wde.redbeans.*;
    /*1. Create a RedSet object instance that will be used to manipulate
       our strings.*/ 
    RedSet oRedSet = new RedSet();

    /*2. Define "field name".*/
    oRedSet.open("firstName,lastName,dept");

    /*3. Assign the rows of data.*/
    String fields[] = { "firstName" , "lastName" , "dept"};
    String values[] = { "Mike", "Kontorovich", "Eng" }; 
    oRedSet.addNew(fields, values);
    values[0] = "Jurgen";
    values[1] = "Joarder";
    values[2] = "Eng";
    oRedSet.addNew(fields, values);
    values[0] = "Simon";
    values[1] = "Caddick";
    values[2] = "Support";
    oRedSet.addNew(fields, values);
    values[0] = "Nghia";
    values[1] = "La";
    values[2] = "Eng";
    oRedSet.addNew(fields, values);
Creating and using a RedField object

Creating and using a RedField object

A RedField object is a nested data structure that can represent UniData or UniVerse multivalues and multi-subvalues.

Note:
The RedField class is not NLS or I18N aware. If you are writing an application that requires NLS or I18N, use the UniDynArray class instead. See Handling multivalue data in an NLS or I18N configuration, on page 17.

CHAR(253) and CHAR(252) are the default values for a value mark and subvalue mark, respectively.
You change these values by using the SetVM() and SetSVM() methods.

Access to multivalues and multi-subvalues uses a syntax similar to that found in the UniVerse BASIC or UniBasic language, using the getValue() method.

The following steps provide an example of how to create and use an instance of a RedField object.

1. Create an instance of a RedField object.
The following syntax creates an empty RedField object.
RedField rf = new RedField();

2. Assign the RedField a string value.
You can assign the RedField a string value in a variety of ways by using the Ins(), Replace(), and Append() methods, for example,

For RedPages.NET: rf.StringValue = “dynastring”;
For RedBeans: rf.setStringValue(dynastring);

3. Add, delete, or replace individual elements in the string value.
For example, execute the following to insert an element at Value position 2:
rf.ins(2,stringValue);

Enter the following code to delete SubValue 3 of Value 1:
rf.del(1,3);

Enter the following code to add a new SubValue to the end of Value 2:
For RedPages.NET: rf.ins(2,rf.count(2) +1, "");
For RedBeans: rf.ins(2,rf.count(2)+1);

4. After you manipulate the string by using the RedField, you can extract it again as a dynamic string, for example,
rf.getValue();

5. **(RedPages.NET only)** When you finish with any object, close or reset the object to release any resources that it might be consuming by using the following code:

```
rf = null;
```

**Example**

The following example shows the RedField code.

**For RedPages.NET:**

```csharp
using U2.WDE.RedPages.
...

//1. Create a RedField object instance that will be used to manipulate our string. RedField oFld = new RedField();

//2. Create initial string - a list of names and item counts and convert the":" and "," separators into subvalue and value marks.
String myString = "Colin:100, Jurgen:205, Tony:400, Mike:400";
myString = RedField.swap(myString, ":", RedField.SVM.ToString());
myString = RedField.swap(myString, ",", RedField.VM.ToString());

//3. Assign the string to the RedField.
oFld.StringValue = myString;

//4. Display current contents.
Console.Out.WriteLine("RedField contains " + oFld.getValue());

//5. Now display each multi value.
Console.Out.WriteLine("<br><br>"); for (int mvpos = 1; mvpos <= oFld.count(); mvpos++) { Console.Out.WriteLine("value at " + mvpos + " = " + oFld.getValue(mvpos)); }

//6. Now display each multi value, subvalue
1. Console.Out.WriteLine("<br><br>"); for (int mvpos = 1; mvpos <= oFld.count(); mvpos++) { Console.Out.WriteLine("value at" + mvpos + ",1 = " + oFld.getValue(mvpos, 1)); }

//7. Now insert Nghia at multivalue position
3. oFld.ins(3, "Nghia"); oFld.ins(3, 2, "800");

//8. Now redisplay current values. Console.Out.WriteLine("RedField contains " + oFld.getValue()); for (int mvpos = 1; mvpos <= oFld.count(); mvpos++) { Console.Out.WriteLine("value at " + mvpos + " = " + oFld.getValue(mvpos)); }

//9. Delete Mike from the list. oFld.del(5); Console.Out.WriteLine("RedField contains " + oFld.getValue()); //10. Now convert it back into the original format.
myString = RedField.swap(oFld.getValue(), RedField.VM.ToString(), ","); myString = RedField.swap(oFld.getValue(), RedField.SVM.ToString(), ":"); Console.Out.WriteLine("myString = " + myString);
```
For RedBeans:

```java
import com.rs.u2.wde.redbeans.*;
...
/*1. Create a RedField object instance that will be used to manipulate our string.*/RedField oFld = new RedField();

/*2. Create initial string - a list of names and item counts - and convert the ":" and "," separators into subvalue and value marks.*/
String myString = "Colin:100,Jurgen:205,Tony:400,Mike:400";
myString = oFld.swap(myString, ":", "," + oFld.SVM);
myString = oFld.swap(myString, ",", "" + oFld.VM);

/*3. Assign the string to the RedField.*/
oFld.setStringValue(myString);

/*4. Display current contents.*/
System.println("RedField contains "+oFld.getValue());

/*5. Now display each multivalue.*/
System.println("<br><br>");
for(int mvpos = 1; mvpos <= oFld.count(); mvpos++) {
    System.println("value at "+mvpos+" = "+oFld.getValue(mvpos));
}

/*6. Now display each multivalue, subvalue 1.*/
System.println("<br><br>");
for(int mvpos = 1; mvpos <= oFld.count(); mvpos++) {
    System.println("value at "+mvpos+",1 = "+oFld.getValue(mvpos,1));
}

/*7. Now insert nghia at multivalue position 3.*/
oFld.ins(3, "Nghia");
oFld.ins(3,2, "800");

/*8. Now redisplay current values.*/
System.println("RedField contains "+oFld.getValue());
for(int mvpos = 1; mvpos <= oFld.count(); mvpos++) {
    System.println("value at "+mvpos+" = "+oFld.getValue(mvpos));
}

/*9. Delete Mike from the list.*/
oFld.del(5);
System.println("RedField contains "+oFld.getValue());

/*10. Now convert it back into the original format.*/
myString = oFld.swap(oFld.getValue(), "," + oFld.VM, ":");
myString = oFld.swap(oFld.getValue(), ":", "," + oFld.SVM);
System.println("myString = " + myString);
```

(RedBeans only) Creating and using an RbEvent object

RbEvent is a class in the com.rs.u2.wde.redbeans package that extends EventObject. Use the RbEvent object to inform interested classes about RedBeans debugs.

The following steps provide an example of how to create and use an instance of an RbEvent object.

1. **Invoke the `getType()` method on a selected RbEvent object, for example,**
   ```java
   public int getType()``
This returns the string value associated with the type:

\[ \text{RbEvent.DEBUG} \]

2. Invoke the `getText()` method on the RbEvent object, for example,

```java
public String getText()
```

This returns the string value associated with the type.

(\text{RedBeans only}) Creating and using an RbEventListener object

To make your class a listener for RbEvents, you need to implement `RbEventListener`. The following steps provide an example of how to create and use an instance of an `RbEventListener` object.

1. Invoke the `receiveRbEvent()` method on a selected RbEvent object, for example,

```java
public void receiveRbEvent(RbEvent e)
```

2. Add the class as a listener, for example,

```java
thisObj = new RedObject("rbexamples", "EXMOD:Employee");
// add ourselves as a listener for debugs
thisObj.addDebugListener(this);
```

(\text{RedBeans only}) Implementing debug logging

You can switch on debug logging for an instance of RedObject. This results in the RedObject launching an RbEvent for which your code can listen.

The following steps provide an example of how to implement debug logging.

1. Invoke the `setDebug()` method on a RedObject object, for example,

```java
currObj.setDebug(boolean flag);
```

where `flag` is either true or false; true enables logging, false disables it.

2. Implement RbEventListener to listen to debugs, for example,

```java
// receive an rbevent and display information
public void receiveRbEvent(RbEvent e) {
    // get type and display information
    if(e.getType() == RbEvent.DEBUG) {
        System.out.println(e.getText());
    }
}
```

Error and alert handling

The RedBack Object Server with RedPages.NET or RedBeans can generate errors. The RedBack Object Server itself can also generate alerts.

**Errors**

Errors are thrown as RbException when RedObject methods `open()` and `callMethod()` are called.

```java
try {
```
empObj.open();
}  
catch(RbException rbe) {//deal with error.
System.out.println(rbe.getType() + rbe.getMessage());return;
}

The types of exceptions are:
- RbException.CONNECTIONERROR
- RbException.SERVERERROR
- For RedPages.NET: RbException.REDPAGESERROR
- For RedBeans: RbException.REDBEANSERROR

Alerts
If you want to look for alerts generated by the RedBack Object Server callMethod() method call, use:

getServerAlert();
If(this.empObj.getServerAlert().length()>0)
System.out.println(empObj.getServerAlert());

Profiling RBOs in UniData

As of version 5.2.0, you can profile your BASIC subroutine calls for stateful and stateless RBOs. A new
class, UniDataProfiler, has been added to RedPages.NET and RedBeans to call RBO methods. This
allows users to generate execution profiles that track call counts and execution times for UniBasic
programs, internal subroutines, and program calls. You can use these profiles to identify sections
of your UniBasic application that are called most frequently, and then focus optimization efforts in those
areas.

Prerequisites
You cannot utilize profiling with the Java Scheduler; you must use connection pools or use standard
connection by setting the minimum and maximum connection pool to zero. If you are using the Java
Scheduler, you need to comment out the SchedulerPort parameter in the rgw5.ini file. Set your
credentials and working directory appropriately, and set the minimum and maximum pool size zero.

Procedure
1. Compile your UniBasic programs with the -G option.
2. Write your code to call your own RBOs with the UnidataProfiler class and Profile method instead
   of the RedObject class and callMethod method.
   You must have your own program outside of the application – you cannot profile your
   application. RBOs must be called outside of the application, or the application must be modified.
   The following is an example of C# program.

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using U2.WDE.RedPages;
using IBMU2.UODOTNET;
namespace ProfileTest
{
    class Program
    {
```
static void Main(string[] args)
{
    Connection newconn = new Connection("rbexamples");
    long cpend, cpstart, cpwait;

    UnidataProfiler oEmployee = new UnidataProfiler(newconn,
            "EXMOD:Employee");

    try
    {
        oEmployee.open();
        oEmployee.setProperty("EmpId", "1012");
        oEmployee.Profile("ReadData");

        Console.WriteLine("Status: "+oEmployee.getProperty("Status"));
        Console.WriteLine("StatusText: "+oEmployee.getProperty("StatusText"));
        Console.WriteLine("First Name: "+oEmployee.getProperty("FirstName"));
        Console.WriteLine("LAST.NAME: "+oEmployee.getProperty("LastName"));

    } catch (Exception e){
        Console.WriteLine("Exception caught: " + e.Message);
    }
}

When complete, profile files are created under the account that contains your profiled RBO.

Example: Session IDs

When you make an initial request to the RedBack Object Server, Web DE creates a session ID. You should pass this session ID with every subsequent request so the RedBack Object Server can associate all stateful RBO instances created for a specific session.

Each RBO created on the RedBack Object Server is given a unique RBO handle. The RedBack Object Server uses a unique session ID to identify each request from a client.

The program that makes the request to the RedBack Object Server ensures that the session ID is passed as required. You can also use the RBO handle to make sure that a current state for an RBO is used rather than a new instance of that RBO.

For example, a program might need to:

1. Create an instance of an Employee RBO as the first request for a user. The RedBack Object Server creates a session ID and state for the user and also creates state for the Employee RBO. The state and session ID are returned to the program that made the initial request.

2. The program then makes a request to create an instance of the Department RBO, read the specific department record for the Employee (as stored in Employee RBO), and then update the department record with details of that Employee.

The session ID passed in step 1 must be associated with the RBO in step 2. Otherwise, Web DE creates a new session ID on the RedBack Object Server, because none was passed with the open() method,
and any method called from the Department RBO on the RedBack Object Server will not have access to state created for the session from the Employee RBO.

Always associate session IDs with all stateful RBOs that a specific user creates.

The following example does not use stateless RBOs. Stateless RBOs do not store state and do not use session IDs.

For RedPages.NET:

```csharp
using U2.WDE.RedPages;
...
String accountname = "rbexamples";
String sessionID = "";
RedObject empObj = new RedObject(accountname, "EXMOD:Employee");
try
{
    empObj.open();
    // Preserve session ID for subsequent RBOs.
    sessionID = empObj.getSessionId();
} catch (RbException RBE)
{
}
// Now create another RBO part associate the current session ID.
RedObject deptObj = new RedObject(accountname, "EXMOD:Department");
depObj.setSessionId(sessionID);
// Now when the RBO is opened, the RedBack object server will
// associate it with the current session id.
```

For RedBeans:

```java
import com.rs.u2.wde.redbeans.*;
...
String sessionID = "";
RedObject empObj = new RedObject(accountname, “EXMOD:Employee”); 
try {
    empObj.open();
    // Preserve session ID for subsequent RBOs
    sessionID = empObj.getSessionId();
} catch (RbException RBE){
...
}

// Now create another RBO part associate the current session ID
RedObject deptObj = new RedObject(accountname, “EXMOD:Department”);
depObj.setSessionId(sessionID);
// Now when the RBO is opened, the RedBack object server will
// associate it with the current session ID.
```

It is important that your program preserve the current session ID so that it can be used with all stateful RBOs. If the session is no longer available on the RedBack Object Server, for example, if it has expired, an RbException will be thrown on the open() method.

Example: RBO handles

Web DE assigns a unique handle to each RBO that is created on the RedBack Object Server. Web DE passes and returns the handle with every request made on that RBO using RedObject.
It is common to create an RBO in one page, or application component, and to enter data in that RBO. You might require that data later in the application processing from another page or application component.

If the data is held in the RedObject, you must preserve the RedObject instance somewhere (depending on your environment) so that you can retrieve it later. If the data is held on the RedBack Object Server in state, which it would be if you made a call to any RedBack Object Server method from the RedObject, you can preserve the RBO handle for that RBO, and then refresh its state into any RBO.

For example, you can create an instance of the Employee RBO from a JSP or an ASP.NET page, set some properties of that RBO, and then call an Employee method (an Employee update method for RedBeans) on the RedBack Object Server. You can then move to another JSP or ASP.NET page that might need to access some of the properties for the Employee RBO. You can preserve the Java or .NET instance of the RedObject somewhere, or you could preserve the RBO handle.

The following example shows how to use an RBO handle.

**For RedPages.NET:**

```csharp
using U2.WDE.RedPages;
...
String empHandle = "";
RedObject empObj2 = new RedObject(accountname, "EXMOD:Employee");
try {
    empObj.open();
    empHandle = empObj2.RBOHandle;
    // Do stuff
    empObj.callMethod("update"); // server side method
} catch (RbException rbe) {
    // Now before we leave this page/code, make sure empHandle is preserved
    // in a later page/code. Need to get handle of current employee RBO
    RedObject newObj = new RedObject();
    newObj.RBOAccount = accountname;
    try {
        newObj.refresh(empHandle); // where empHandle was preserved
        // newObj will now contain the state from the RedBack object
        // server for the Employee RBO.
    } catch (RbException rbe) {
        //
    }
}
```

**For RedBeans:**

```java
import com.rs.u2.wde.redbeans.*;
...
String empHandle = "";
RedObject empObj = new RedObject(accountname, "EXMOD:Employee");
try {
    empObj.open();
    empHandle = empObj.getRBOHandle();
    // Do stuff
    empObj.callMethod("update"); // server side method
} catch (RbException rbe) {
    //
} catch (RbException rbe) {
    //
} /*Now before we leave this page/code, make sure empHandle is preserved
```
Example: RBO handles

```java
RedObject newObj = new RedObject();
newObj.setRBOAccount(accountname);
try {
    newObj.refresh(empHandle); // where empHandle was preserved
    /*newObj will now contain the state from the RedBack object server
     * for the Employee RBO./*
} catch (RbException rbe){}
```

There are different ways to store session IDs and RBO handles (both values are strings).

Where you store these values depends on your runtime platform, application architecture, and design.
Some of the places you can store them include:

- Cookies
- Hidden fields on page
- As part of a query string

If you do not store and associate a session ID with RBOs, you create new session state within the RedBack Object Server, preventing you from accessing previous RBO state information.

If you do not need to access previous RBO state, consider using stateless RBOs for at least part of your application. They are referenced in the same way by RedBeans or RedPages.NET, but do not require an open call, and do not have a session ID. You cannot refresh a stateless RBO later because server state is not preserved. For further information about stateless RBOs, see Web Designer User’s Guide.
Chapter 4: (RedPages.NET only) Creating a RedPages.NET application

You can use any .NET authoring tool to create a web application, but the examples in these sections are created using C# in Microsoft Visual Studio 2010.

To create a project in Microsoft Visual Studio, complete the following steps:

1. Opening a new project in Visual Studio, on page 38
2. Setting the page properties, on page 39
3. Building the application, on page 40
4. Linking the controls, on page 43

Opening a new project in Visual Studio

Using Visual Studio, you can open a new project to create a web application.

1. From Visual Studio, select File → New → Web Site.
2. From the New Web Site window, select ASP.NET Web Site.

Figure 1: New Web Site window
3. Click **Browse**, and navigate to the directory where you want your project to reside. Provide an appropriate name for the project, for example, **EmployeeList**, and then click **OK**.

The new project opens in the Microsoft Visual Studio window.

**Figure 2: New project**

---

**Setting the page properties**

1. In the Solution Explorer pane, right-click the **C:|..\EmployeeList** project, and select **Add Reference**.
2. Click the **Browse** tab, and navigate to the Web DE installation path.
3. Select all of the available DLLs, and click **OK**.
4. In the Visual Studio toolbox, right-click an empty section and select **Add Tab**.
5. In the dialog box, name the tab appropriately, for example, **WebDE**. Click **OK**.
6. Right-click inside the WebDE toolbox tab and select **Choose Items**.
7. In the Choose Toolbox Items window, select **Browse**, and navigate to the Web DE installation path, for example, **C:\U2\U2WDE521\DataSource**.
8. Select **U2.WDE.WebControls.dll** and click **Open**.
9. Click **OK** to close the Choose Toolbox Items window.
10. In the Visual Studio toolbox, right-click an empty section and select **Show All** to see the added WebDE section.

**Note:** At this point, on some versions of Visual Studio, you need to run **Build → Build Solution** from the menu to make Visual Studio aware of the DLLs. Omitting this step can result in errors appearing on the design surface when the Data Source controls are dragged to the page.
Building the application

1. Drag a Label onto the form and set the Text Property to Employee List.
2. Drag a UQueryDataSource control onto the form and position it under the Employee List label. You can change the name of the control in the Properties pane. In this example, the default name of UQueryDataSource1 is accepted.
   The UQueryDataSource Tasks menu opens. If the tasks menu does not open automatically, click the arrow button on the upper right corner of the control.
3. On the Tasks menu, click Set Credentials.
4. Enter your user name and password, and click OK.
5. Select the following connection details for the control:
   - Account: rbexamples
   - Module: EXMOD
   - RBO: EmployeeList

![Figure 3: Connection details](image)

Click the arrow on the upper right side of the control to minimize the Tasks menu.
6. Drag a GridView control onto the form, and position it below the UQueryDataSource control. The GridView Tasks menu opens.

![Figure 4: GridView Tasks](image)
7. On the Tasks menu, click **Set Credentials**.
8. Enter your user name and password, and click **OK**.
9. From the Choose Data Source options, select **UQueryDataSource1**, and from the GridView Tasks menu, select the **Enable Selection** check box.
10. From the GridView Tasks menu, select **Edit Columns** and change how the information is displayed:

   Figure 5: Fields dialog box

   ![Fields dialog box](image)

   a. From the **Selected Fields** list, select an item.
   b. In the **BoundField properties** section, edit the properties as follows:
      - Set the HeaderText for the EMP.ID field to **Employee ID**.
      - Set the HeaderText for the FIRST.NAME field to **First Name**.
      - Set the HeaderText for the LAST.NAME field to **Last Name**.
      - Set the HeaderText for the DEPT field to **Dept**.
   c. When finished, click **OK**.
11. Drag a **Label** onto the form, and set the Text Property to **Employee Information**.
12. Drag an **RBODatasource** control onto the form. Position the control below the **Employee Information** label.

The **RBODatasource Tasks** menu opens. If the tasks menu does not open automatically, click the arrow button on the upper right corner of the control.

Select the following connection details for the control:

- **Account**: rbexamples
- **Module**: EXMOD
- **RBO**: Employee
- **ReadMethod**: ReadData
- **UpdateMethod**: WriteData
- **InsertMethod**: WriteData
- **DeleteMethod**: DeleteData

![Figure 6: RBODatasource Tasks menu](image)

Click the arrow on the upper right side of the control to minimize the **Tasks** menu.

13. Drag a **Data DetailsView** control onto the form, and position it below the **RBODatasource** control.

14. On the **DetailsView Tasks** menu, from the Choose Data Source options, select **RBODatasource1**.

15. From the **DetailsView Tasks** menu, select **Edit Columns** and change how the information is displayed:
   a. From the **Selected Fields** list, select an item.
   b. In the **BoundField properties** section, edit the properties.
   c. Delete unnecessary fields by highlighting the item in the **Selected Fields** list, and then clicking “X.” The fields used in this example are **Select**, **Empid**, **FirstName**, **LastName**, **Salary**, **Dept**, and **Interests**.
   d. Change the **HeaderText** property for the selected fields as follows:
      - Set the **HeaderText** for the EMP.ID field to **Employee ID**.
      - Set the **HeaderText** for the FIRST.NAME field to **First Name**.
      - Set the **HeaderText** for the LAST.NAME field to **Last Name**.
   e. When finished, click **OK**.
Linking the controls

1. In the GridView properties pane, select **DataKeyNames** and then click the ellipsis (...) button.

2. From the Data Fields Collection Editor dialog box, select **EMP.ID** from the available data fields list, and click the right arrow (>) to move it to the selected data fields list. Visual Studio pulls information from this field when you select an employee name from the list. Click **OK**.

3. In the RBODataSource control property pane, select **Select Parameters** from the property window and then click the ellipsis (...) button.

4. From the Parameter Collection Editor window, click **Add Parameter**. In the text box that appears in the Parameters section, type **EmpId**. EmpId is the key field property of the Employee RBO.

5. From the **Parameter Source** menu, select **Control**.

6. From the **ControlID** menu, select **GridView1**. The row selected from the Employee List grid will now drive the Employee Details control. Click **OK**.

7. Run the application by selecting **Debug → Start Without Debugging**. An Explorer browser opens. Your application should look similar to the following example:

![Figure 7: ASP.NET application example](image)

**Binding multivalued properties to a list type control**

Properties with multivalues can be bound to list type controls. Multivalued properties can be bound individually or together with other associated properties. The following steps bind the multivalued-delimited Interests property to a Repeater control.
1. Select the DetailsView control that you added to the form earlier. Click the arrow on the upper right corner of the control to open the DetailsView Task menu, and then select **Edit Fields**.

2. On the Field dialog box, from the Selected fields section, select **Interests**. Click **Convert this field into a TemplateField**, and then click **OK**.

3. Open the **DetailsView Task** menu, and select **Edit Templates**. The DetailsView menu opens in Template Editing Mode, as shown in the following figure.

4. From the ItemTemplate section, delete **[Label1]**.

5. Drag a **Repeater** control from the Data toolbox tab to the ItemTemplate section.

6. Click the arrow to open the **Repeater Tasks** menu and from the Choose Data Source options, select **RBODataSource1**.

7. With the Repeater control still selected, go to the Properties window and set theDataMember field of the Repeater to **Interests**.

   **Note:** When binding associated multivalued properties, set the **DataMember** field of the data bound control to a comma-separated list of RBO property names (for example, “CardNumber,ExpirationDate,SecurityCode”).

8. The Repeater control requires you to switch to **Source View** to enter the markup that will be repeated for each row of data. Inside the existing `<asp:Repeater>` tag, enter `<ItemTemplate>`. Increase the width of the DetailsView to accommodate the **Interests** field.
9. Select **Debug → Start without Debugging** from the Visual Studio menu to run the application. The individual values of the multivalued Interests field are displayed on their own lines, as shown in the following figure:

Figure 9: ASP.NET Application

![Figure 9: ASP.NET Application](image-url)
Appendix A: Configuration files

The following sections describe the content of the Web DE configuration files.

rgw5.ini file

Configuration of RedPages.NET and RedBeans connections is controlled through the rgw5.ini file, which is installed on Windows by default in the C:\U2\U2WDEnnn directory, where nnn is the version number you installed. This path is set in the environment variable %U2WDE%. On UNIX systems, the rgw5.ini must be manually copied to your desired location, for example, the /etc directory. The $U2WDE environment variable should be created and the value set to point to the path to the rgw5.ini file. The gateway configuration file, rgw5.ini, is in the directory specified by the U2WDE environment variable. The RedBack gateway components, which run on the web server or the client, use the rgw5.ini file for global parameters and as a reference to the location at which each U2 account resides.

Format

The rgw5.ini file format is:

```
[SectionName]
parameter_1=value_x
parameter_2=value_y
...
```

Each section contains a set of related parameters. Web DE searches the rgw5.ini file for predefined and user-defined section names, and updates internal parameters based on the values set in the file. Most of the section names are predefined, so you cannot change them. However, you can define the names of additional sections.

Example rgw5.ini file

The following example rgw5.ini file contains the default sections and settings that show the type of data required for each parameter:

```
[LogLevel]
panic=1
err=1
inf=0
init=1
times=1
wrn=1
trace=1

[Default]
perfstats=0

[rbexamples]
workdir=C:\U2\U2WDE521\UDserver\rbexamples
MinimumPoolSize=2
MaximumPoolSize=2
ConnectionString=udcs
userId=cbrown
password=2spwV0hJequEefsSBqK1KQ==
server=localhost
logpath=C:\U2\U2WDE521
```
Sections of rgw5.ini file

The rgw5.ini file contains three default sections:

▪ [LogLevel] section
▪ [Default] section
▪ [accountname] section

[LogLevel] section

This section of the rgw5.ini file contains parameters for the types of information to write to the client log files for Web DE-enabled U2 accounts. The log level parameters are global; they apply to all Web DE-enabled U2 accounts on the RedBack Object Server computer.

The client log files are:

▪ RedBeans_accountname.log
▪ RedPages_accountname_processid.log, where processid is the ID of the Windows process with RedPagesNet.dll loaded.

The following example shows the [LogLevel] section:

```
[LogLevel]
panic=1
err=1
inf=0
init=1	
times=1
wrn=1
trace=1
```

The following table lists the log level parameters. Each parameter specifies whether to write details for a type of information to the client log files.

[Default] section

The parameters in this section define Web DE system defaults. They apply to all Web DE-enabled U2 accounts within the system. You can override the defaults within each [accountname] section.

The following example shows the [Default] section:

```
[Default]
perfstats=0
```

The following table lists the default global parameters at the Web DE system level. You can add other default parameters in this section, if applicable.
Appendix A: Configuration files

Table 15: [Default] section parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>perfstats</td>
<td>Specifies whether to log performance statistics.</td>
</tr>
<tr>
<td></td>
<td>• 0 – Do not log performance statistics.</td>
</tr>
<tr>
<td></td>
<td>• 1 – Log performance statistics.</td>
</tr>
<tr>
<td></td>
<td>The Perfstats.log file includes the same statistics available from the Web DE performance monitor API, such as wait time and number of requests processed for each account.</td>
</tr>
<tr>
<td></td>
<td>The Perfstats.log file is written to the C:\U2\U2WDEnnn directory by default, where nnn is the version number you installed.</td>
</tr>
</tbody>
</table>

[accountname] section

The rgw5.ini file contains a section for each account. The name of the section is the connection name of the account, used as a reference by RedBeans or RedPages.NET.

In each account section, you must define at least the mandatory parameters for the account. You can specify additional parameters. If you specify a parameter in the [accountname] section that is also in the [Default] section, the value you set for the account overrides the system default value.

The following example shows the [accountname] section:

```
[rbexamples]
workdir=C:\U2\U2WDE521\UDserver\rbexamples
MinimumPoolSize=2
MaximumPoolSize=2
ConnectionString=udcs
userId=cbrown
password=2spwV0hJequEefsSBqK1KQ==
server=localhost
logpath=C:\U2\U2WDE521
;SchedulerPort=7070
como=1
usingssl=1
PoolingDebug=1
IdleRemoveThreshold=15000
IdleRemoveExecInterval=1000
OpenSessionTimeout=1000
unirpcTimeout=500
```

The following table describes the parameters in the [accountname] section:

Table 16: [accountname] section parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workdir</td>
<td>The physical path to the U2 account on the computer on which the RedBack Object Server is installed.</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MinimumPoolSize</strong></td>
<td>The minimum number of connections maintained in the connection pool. Setting this value and the MaximumPoolSize value to 0 enables the developer mode, which allows you to execute newly compiled code through Web DE processes. Instead of using a cached version of the object code, the developer mode allows you to pick up the new object code as soon as it is available by causing the associated database processes to restart after each method call. While allowing newly compiled code to be picked up, this mode carries a performance overhead as the associated database processes terminate and are recreated after each method call and is not recommended for “live” production use for this reason. The SchedulerPort must also be commented out to bypass the connection to the Java Scheduler.</td>
</tr>
<tr>
<td><strong>MaximumPoolSize</strong></td>
<td>The maximum number of connections maintained in the connection pool. Setting this value and the MinimumPoolSize value to 0 enables the developer mode. See the description in MinimumPoolSize for information about the developer mode.</td>
</tr>
<tr>
<td><strong>ConnectionString</strong></td>
<td>Specifies whether the connection is a UniData (udcs) or UniVerse (uvcs) connection.</td>
</tr>
<tr>
<td><strong>userId</strong></td>
<td>The operating system-level user ID for the RedBack Object Server computer.</td>
</tr>
<tr>
<td><strong>password</strong></td>
<td>The password associated with the user ID specified in the userId parameter. The password is encrypted when you save the JavaScheduler.ini file in Web Designer. If you edit and save the rgw5.ini file in a text editor, the password is saved in plain text. To maintain server security, open and save the rgw5.ini file in Web Designer to encrypt the password.</td>
</tr>
<tr>
<td><strong>server</strong></td>
<td>The name or IP address of the computer on which the RedBack Object Server is installed.</td>
</tr>
<tr>
<td><strong>logpath</strong></td>
<td>The full path of the directory in which to store client log files for the account on the client computer. The client log files are RedBeans_accountname.log and RedPages_accountname_processid.log, where processid is the ID of the Windows process with RedPagesNet.dll loaded. The directory must already exist; Web DE does not create a new directory for you. If the specified logpath directory does not exist, the application fails at run time.</td>
</tr>
<tr>
<td><strong>SchedulerPort</strong></td>
<td>The port number on which the Java Scheduler accepts requests from this account. If this parameter is commented out, and the MinimumPoolSize and MaximumPoolSize parameters are set to 0, the Java Scheduler connection is bypassed.</td>
</tr>
</tbody>
</table>
| **como**              | Specifies whether to save the server logs to the `account accountname` file in UniData or the &COMO& file in UniVerse.  
  - 0 – Do not save the logs.  
  - 1 – Save the logs.  |
### Appendix A: Configuration files

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>usingssl</strong></td>
<td>Specifies whether to use the SSL protocol when connecting to the U2 server.</td>
</tr>
<tr>
<td></td>
<td>• 0 – Do not use SSL.</td>
</tr>
<tr>
<td></td>
<td>• 1 – Use SSL.</td>
</tr>
</tbody>
</table>

| **PoolingDebug** | Specifies whether to keep a log of UOJ processes for the account for debugging purposes.                                                      |
|                  | • 0 – Do not keep a log of UOJ processes.                                                                                                     |
|                  | • 1 – Keep a log of UOJ processes.                                                                                                             |

| **encoding**     | Allows support of any session encoding for connections using RedPages.NET or RedBeans.                                                       |

| **IdleRemoveThreshold** | The number of milliseconds that a connection pool thread can remain idle before it is flagged for removal.                              |

| **IdleRemoveExecInterval** | The number of milliseconds of the interval at which UOJ runs an executable to remove from a connection pool any threads that have exceeded the idle threshold. |

| **OpenSessionTimeout**  | The number of milliseconds for which UOJ attempts to acquire a session from the connection pool before timing out.                          |

| **unirpcTimeout**       | Optional parameter that specifies the maximum amount of time the client (RedBeans or RedPages.NET) is going to wait for a response from the Java Scheduler before it considers the request as timed out. Default is 300 seconds. |