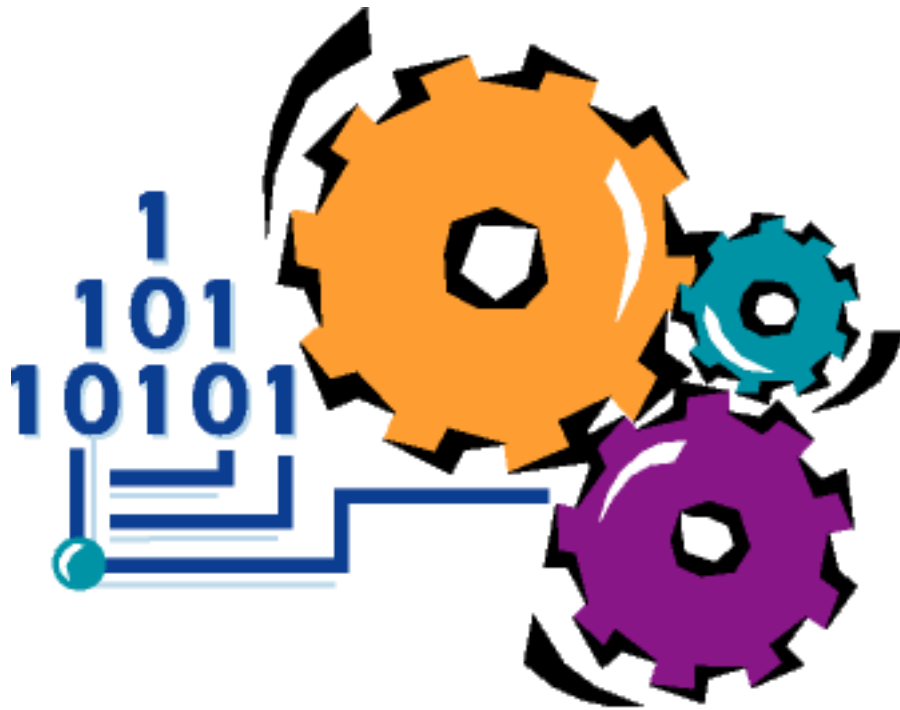


ReportBuilder

Server Edition



Developer's Guide

Second Edition

Server Edition

Developer's Guide

Copyright © 2002 - 2007 by Digital Metaphors Corporation

CONTENTS

SERVER FUNDAMENTALS

Report Application vs. Report Server Application	3
Socket To Me	5
Building a Report Server Application	7
Talking To The Server From A Thin Client	9
Registering Reports With The Server	11
Spelunking With The Report Explorer	15
Publishing Reports to the Web	17
Making Web Browser Content Printable With PDF	19
Troubleshooting the WebTier	21
FAQ	23

TUTORIALS

Building a Report Server Application for Reports on Forms

Overview	29
Copy Existing Report Application to a New Directory	29
Convert the Application to a Server	30
Register the Reports On the Server	30
Make the Data Access Components Thread-Safe	30
Run the Server Application	31
Create a Thin-Client Application	31

Building a Report Server Application for Reports in Files

Overview	33
Copy Existing Report Application to a New Directory	33
Create a Thread-Safe Data Module	34
Register the File-Based Reports	34
Move the Report Event Handlers to the Data Module	34
Convert the Application to a Server	35
Run the Server Application	35
Create a Thin-Client Application	36

Building a Report Server Application for Reports in a Database

Overview	37
Copy Existing Report Application to a New Directory	37
Update the Path of the Database Component	37
Create a Thread-Safe Data Module	38
Register the Databased Reports	38
Move the Report Event Handlers to the Data Module	38
Convert the Application to a Server	39
Run the Server Application	39
Create a Thin-Client Application	40

Building a Report Server Application for an Explorer Database

Overview	41
Copy Existing Report Application to a New Directory	41
Create a Thread-Safe Data Module	42
Register the Databased Reports	42
Convert the Application to a Server	43
Run the Server Application	43
Create a Thin-Client Application	43

Building a Report Server Application for Report Archives

Overview	45
Copy Existing Report Application to a New Directory	45
Register the Report Archives	46
Convert the Application to a Server	46
Run the Server Application	46
Create a Thin-Client Application	47

Run a Report Server Application from a Windows Service

Overview	49
Install the Windows Service	49
Create a Service Compatible Report Server Application	50
Update the File Directory of the Report Volume	50
Designate the Report Server Application from the Windows Service	50
Run the Thin-Client Application	50

Publish Reports to the Web

Overview	53
Create a Directory for the Web Tier	53
Create a Virtual Directory on IIS	54
Create an ISAPI DLL	54
Add a Default Action to the	55
WebModule	55

Scaling the Web Tier with a Server Farm

Overview	57
Create a Report Server Application	57
Copy the Report Server Application to a Network Folder	57
Deploy the Report Server Application	58
Check Each Server via Thin Client	58
Configure a Web Tier Using Round Robin	59
Test via Web Browser	59
Configure a Web Tier Using Minimum Load	60

SERVER FUNDAMENTALS

Report Application vs. Report Server Application 3

Socket To Me 5

Building A Report Server Application 7

Talking To The Server From A Thin Client 9

Registering Reports With The Server 11

Spelunking With The Report Explorer 15

Publishing Reports To The Web 17

Making Web Browser Content Printable With PDF 19

Troubleshooting the WebTier 21

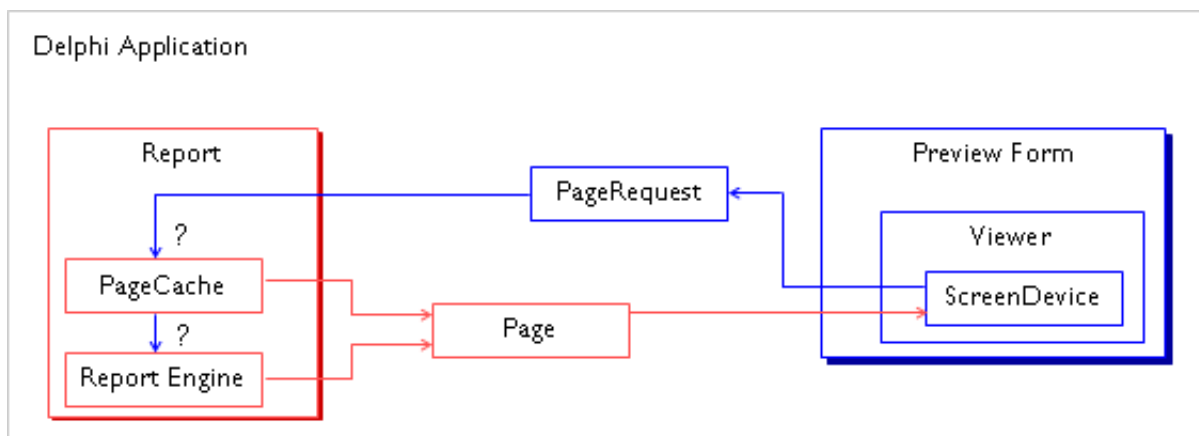
FAQ 23

Report Application vs. Report Server Application

ReportBuilder Server Edition makes it easy to create a report server application. A report server application is quite different from a standard Delphi report application. The key differences are:

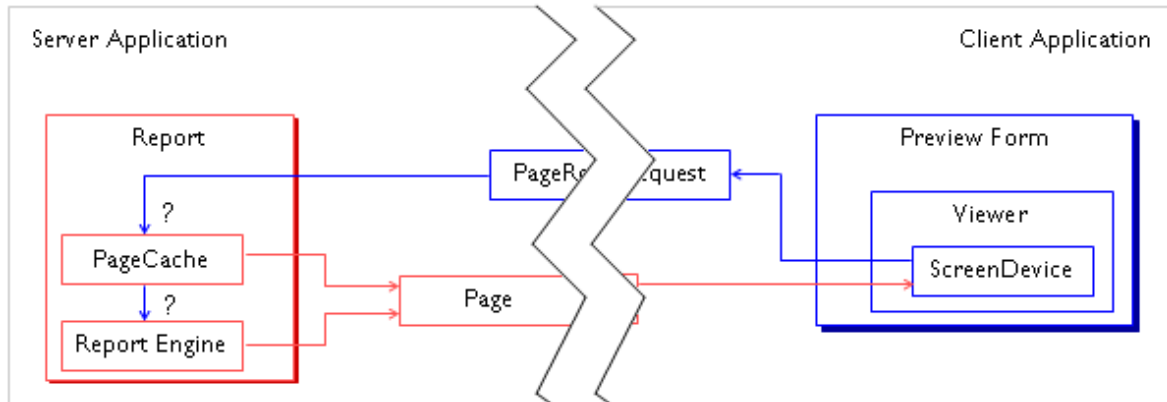
- 1 The server application must be able to communicate across the network in order to satisfy the requests of client applications. In a traditional application, objects may pass freely through the system, with “communication” taking the form of method calls. In a server application, objects or data must be converted to a network compatible format, sent across the network to the client and then converted back to objects or data by the client.
- 2 The server application must be able to run each report within the context of a thread. This means that the data access components and the report components must be thread-safe: that is, multiple instances of these objects must be able to co-exist and run “side-by-side”, without any conflicts created by the use of shared resources. This is not a requirement for a standard reporting application, which usually generates reports one at a time.

The diagram below depicts a typical report application. When the Preview Form is displayed, the first page of the report is requested by the Screen Device via a call to the RequestPage method. The report object checks a cache for the requested page, and if the page cannot be found, passes the request along to the report engine. The report engine attempts to generate the requested page, and then triggers an OnSendPage event, which returns the page to the report. The report then calls the ReceivePage method of the Screen Device, causing the page to be displayed. Notice that all of the objects exist in the same memory space: method calls and events are used to pass information from object to object.



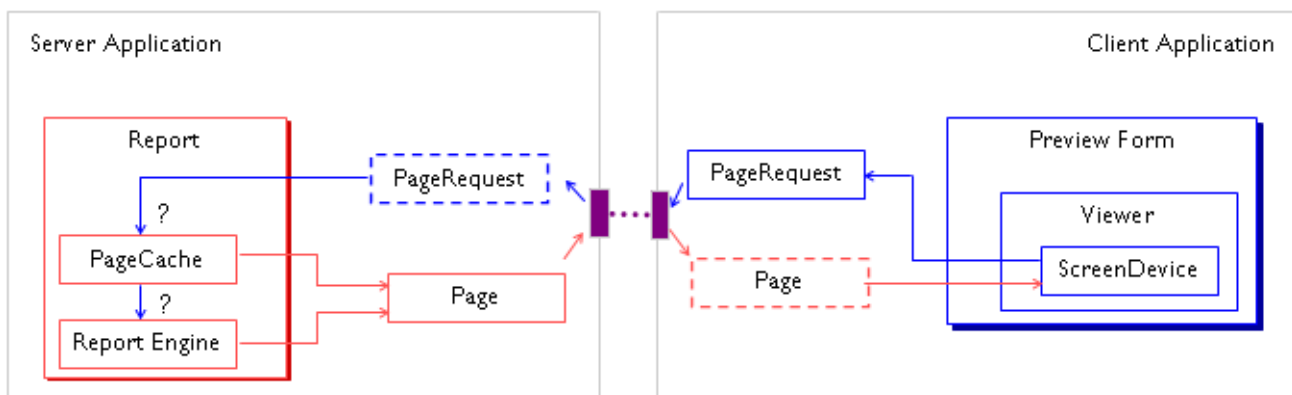
In ReportBuilder Server Edition, this application is divided into two parts: client and server. The server portion of the application will contain the data access components and the reports. The client portion of the application will contain the Preview Form and any devices necessary to process the pages.

Any objects which were passed freely between these two areas in the standard reporting application will now be passed across the network.



Obviously the complexity of this reporting solution is much greater than the complexity of the standard application; two applications must communicate in order for this solution to function effectively.

The Server Edition makes this conversation happen via a technology known as sockets (depicted in the diagram below); a topic we discuss briefly in the next section.



Socket To Me

In the Server Edition communication between the server and client application occurs via sockets. A low level feature of the Windows operating system, sockets were invented by the team which modified Unix in order to create the early internet. To function properly, sockets require a machine address (IP address like 192.168.1.120) and a port (number between 1 – 65535.) Given these values, two-way communication can be established between applications.

When creating a report server application, the Port property of the rsServer component is used to establish the port to which the application will “listen” when communicating with clients. Of course, you should avoid port numbers which conflict with other applications that may be running on the server machine.

Ports for typical services appear in the table below.

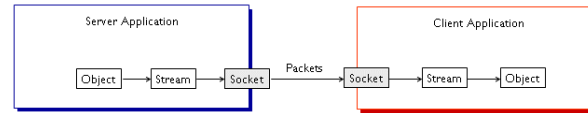
ftp	21
telnet	23
smtp	25
http	80
pop3	110
nntp	119

The machine address (IP address) is not set on the server component, because the address used for the socket is always the address of the machine on which the server application is running.

The client application is in a rather different situation. It must specify both the machine address and the port for the server to which it will connect.

These values are set on the ServerConnection property of the rsClientReport component. The machine address can either be a set of four numbers, or a valid DNS name (i.e. www.mycompany.com) Armed with these two values, the client application can initiate a conversation with the server.

The nature of this conversation is as follows:



Sockets give us an excellent pipeline for bytes – but objects are hardly bytes. Before we can send an object over the wire, we must convert it to a stream. This is done using a customized version of Delphi streaming. Once an object has been converted to a stream, it is compressed, converted to alphanumeric characters (Base64), wrapped in the body of a SOAP message and sent, piece-by-piece through the socket to the client. The client reassembles the stream, extracts the element from the SOAP message and then uses the same customized Delphi streaming logic to instantiate an exact replica of the original object. When the client needs to send a message to the server, the same sequence occurs in the opposite direction.

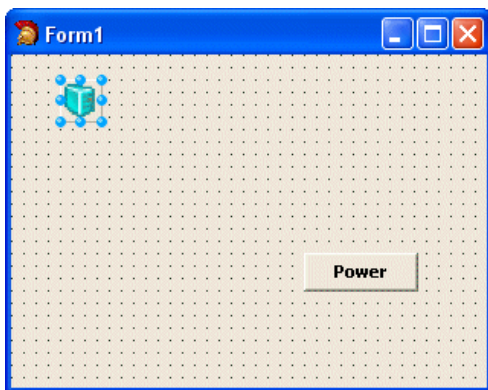
Building a Report Server Application

Our basic goal is to create a Delphi application that will respond from the host machine on a certain port. Any application which wishes to talk to the report server, must know the IP address of the server machine and the port of the server application. Armed with these two pieces of information, a client can begin a conversation with the server.

In the Server Edition, we specify the port in the Server component. This non-visual component serves as the heart of the application. The most important setting is the Port property, and we will accept the default value: 1333. This value was chosen as the default because it does not conflict with any of the port values used by common operating system software.

The screen shot below shows a TrsServer component on a Delphi form. In order to get the server running, we have added a “Power” button to the form. When we run the application and click the button, the following code fires:

```
rsServer1.Active := spbPower.Down;
```



Once the server is active, it will begin listening for requests. At this point the server does not have access to any reports and so can't do much except respond with “I got nuthin'.”

Actually the server will respond with something a little more concise than that, it will say: “No volumes exist.” In the context of the Server Edition, a volume is a collection of reports in the Report Catalog. Volumes are the natural consequence of registering reports with the server – they are ReportBuilder's way of attaching a name and an access mechanism to a group of reports.

So let's create a volume. Assume we've got a report component on a Delphi form. All of the data access components for the report are on the form as well. We add this form to our Report Server project, open it and place the following code at the bottom of the unit:

```
initialization
  TrsReportCatalog.RegisterReport('Accounting',
                                  'By Quarter',
                                  'rptByQuarter',
                                  frmByQuarter);
end;
```

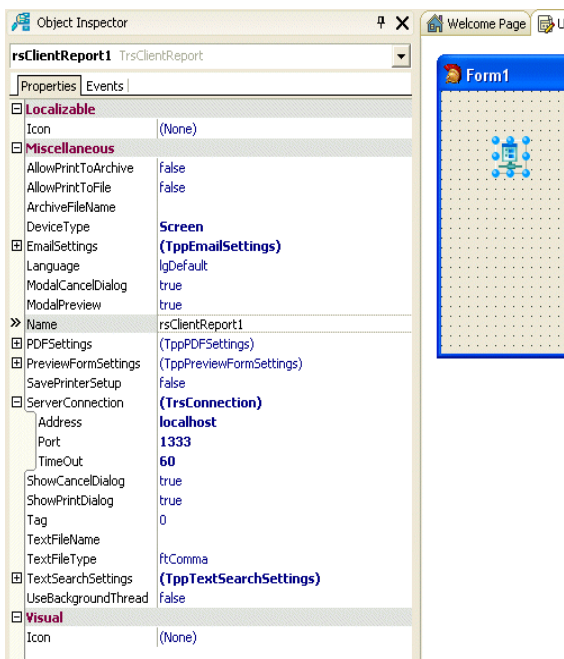
The first parameter is the volume name, the second is the report alias (the name the user will see from the client), the third is the name of the actual report component (i.e. the value of the TComponent.Name property) and the fourth is the name of the form. When the application runs, this initialization section will register the report with the server.

Now, we've almost got enough here to serve this report, but we need two more things. This form is perfectly good for running a report in a Delphi application. But in the context of the server, this form will actually be running within a thread. The BDE-based data access components on our form will not like this; they will blow unless we add a TDatabase object – specifying where the database is located and a TSession object.

The `AutoSessionName` property of the session must be set to `True`, so that when this form is instantiated, the session will be uniquely named within the context of the entire application. With these two details out of the way, we are now ready to serve the report. We run the application and click the “Power” button. Great, now, how do we tell if the server is actually working? The answer: create a thin-client application.

Talking To The Server From A Thin Client

The easiest way to verify that our server is actually serving this report is to create a thin-client application. To do that, we'll need to create a new application and add a `TrsClientReport` component to it. The `ClientReport` component allows us to work with a given report as if it were actually inside the client application. We can use it to “load” reports – though this process is slightly different from the `LoadFromFile` or `LoadFromDatabase` techniques you are used to, and we can use it to preview reports as well. The `ClientReport` component has a lot of network communication stuff crammed inside of it - but we can mostly ignore that and get our app working by configuring the `ServerConnection` and then setting the `VolumeName` and `ReportName` properties at run-time.



If we expand the `ServerConnection` property in the Object Inspector, we can see several properties embedded within it. Address refers to the location of the server: the IP address. Since we are running the server application on the same machine as the client, the default (`localhost`) will do nicely.

Password we can ignore, since we have not implemented a security scheme on the server. Port has defaulted to the correct value: 1333 and Timeout is set to 60 seconds. Timeout refers to the amount of time the client will wait for a server response before giving up on the request and displaying an error message.

The next thing we need to add is a button. To display the report in the `OnClick` event, we'll code this:

```
rsClientReport.VolumeName := 'Accounting';
rsClientReport.ReportName := 'By Quarter';
rsClientReport.Print;
```

We know this report exists on the server and so we just hard-code the values. When the `Print` method is called, the `ClientReport` component will communicate with the server, asking for the first page of the report named “By Quarter” in the “Accounting” volume. Using the information we supplied in the `RegisterReport` call, the server will instantiate an instance of our report form and run the report.

Once the first page is generated, the report server will send it across the wire to `ClientReport`, where it will then be displayed in the standard `Print Preview` form. There is, of course, a lot going on behind the scenes to make all of this possible, but that is the gist of it.

So, we've got a form based report running across the wire. That's great and all, but it's also pretty limited. We don't want to be recompiling and re-deploying our server application every time we want to add or change a report. Realistically we need to consider other report configurations: reports located in template files, reports saved in end-user database tables and reports in archive files. How do we serve these types of reports?

Registering Reports With The Server

Registering Report Templates

If you've ever worked with report template files, then you probably know that there are some important caveats to consider, the first and foremost of which is: where are the data access components going to reside? Without data access components most reports will not run. Second, where are the event handlers going to reside? Delphi event handlers exist within the executable, and any template which uses them must be loaded in a way that allows the events to successfully reconnect. Each of these issues has two basic resolutions:

- 1 When creating the report, use the Data tab (DADE) and the Calc tab (RAP) to configure the data access components and create the event handlers.

With this approach the data access components and the event handlers are saved in the report, providing maximum portability. However, even this configuration requires a database connection and RAP pass-thru function availability.

- 2 When creating the report, use standard Delphi data access components and Object Pascal event handlers.

Here the entire supporting cast is stored in a Delphi form or data module, with the report connected to them. In order to successfully load such a template, we must make sure that the Delphi form or data module is the owner of the report component used for the loading process (or at least the data module is instantiated and "used by" the form or data module containing the report.)

You can certainly view these two alternatives from the perspective of portability, where the first is the maximum portability and the second is the minimum (actually form-based reports should probably be considered the minimum, since they are "locked" into the executable.) Either way, the report server needs to successfully load the template and run the report. To make sure that happens, we need to use a ReportTemplateVolume component, which let's us specify an operating system directory where template files reside. And we need to place that volume in a Delphi form or data module which provides a complete, functional "habitat" in which the report can generate. If you have a functioning report application, then you already have such a habitat somewhere in your app – but it usually makes sense to break this out as a separate data module, so that the server does not have to include your entire application. Let's create a typical habitat for set of highly portable (i.e. DADE and RAP based) report templates.

The first thing we do is open the Server application and add a new data module to it. Make sure the data module is auto-created. We create a ReportTemplateVolume component and set the FileDirectory property to the location of our templates. At this point we can run the server and any templates which appear in the directory will be accessible from the client app. The problem is, they will not run, because we have not provided a database connection. Assuming the reports are BDE based, we add a TDatabase component and TSession component to the form. We set the AutoSessionName property of the TSession to True, so the session name will remain unique, even in a multi-threaded environment.

Next we add the name of the unit containing the DADE plug-in used for these reports to the uses clause of our data module (in this case, daDBBDE.) We compile the Report Server application and run. The reports should now function properly from the client.

That is, unless some of them call RAP pass-thru functions. If this is the case, we need to add the unit containing the pass-thru functions to our application. If the functions are registered in the initialization section of their unit (which is usually the case) then we are done. Recompile and run.

You'll notice that registering reports is more about creating a habitat conducive to report generation, than it is about understanding the ins and outs of the volume components. The volume components are actually quite simple, representing little more than a location (and the associated access mechanism) for a given set of reports. No volume illustrates this more clearly than the ReportArchiveVolume.

Registering Report Archives

Report Archives are reports which have been generated and saved to a file or to a database BLOB field. Because the generation cycle is complete, no database objects are needed when previewing the reports. This greatly simplifies the task of serving report archives. Assuming we have a system directory stocked with report archive files, we can register these by creating a new data module in the Server Application, adding a ReportArchiveVolume and setting the FileDirectory property. Once we compile and run these reports will be available from the client app. Also, any archive files we add to this directory will appear on the client the next time the client takes a fresh look at the given volume. In other words the directory is now "hot", and anything we add to it will be served to clients.

As you can see, there is not much to serving report archives, it's by far the easiest way to get a server up and running. Archives are the best way to serve reports when lots of users want to access the same report and interactive search criteria (where the user enters the search criteria) are not needed.

Registering End-User Reports

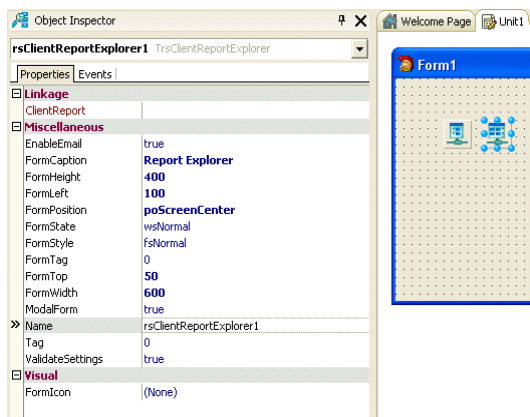
The end-user solution provided by ReportBuilder organizes reports in a folder tree structure. This structure is stored in two database tables: a folder table and an item table. Almost all items are reports (though some may be exported data modules or code modules.) The trick, of course, is how to serve this more complex structure. It's not simply a directory with templates or archives in it. This problem is so fundamental, that the concept of folders became "the way" to model all volumes. Both the TemplateVolume and ArchiveVolume can handle subfolders, and automatically display subfolders when they are configured. The end-user folder structure can be served via the ReportExplorerVolume. The name refers to the Report Explorer, which normally displays the folders and items in an end-user application. The same configuration conditions apply to end-user reports as to any template based reports: a database connection is needed, any RAP pass-thru functions which have been used in the reports are needed, and the unit name for the DADE plug-in must be included in the uses clause. So let's configure a ReportExplorerVolume and see just how easy (or difficult) it is.

First, we open the Server Application and add a new data module to the project. We place a ReportExplorerVolume in it. Then we open the main form containing our end-user solution. From this we copy the data access components for the folder and item tables.

We paste these into the data module. Next we add database connection objects: for this example we need a TDatabase and a TSession object. We set the TSession.AutoSessionName property to True so that it is thread-safe. We add the unit name for the DADE plug-in to the uses clause (in this case daDBBDE) and that's it. We're ready to go. The steps necessary to configure the database connection for any given database (and database connectivity product) will vary. See the examples in RBServer\Demos\Explorer Databases for an example of your database/data connectivity product.

Spelunking With The Report Explorer

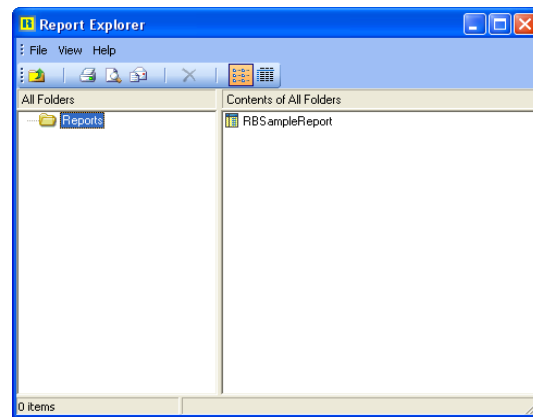
We have created quite a collection of reports on the server with all of this registration activity. We have a report on a form, a set of reports in an operating system directory, a set of report archives in an operating system directory and a set of reports in an end-user database. While the report catalog and volume naming scheme ensures that we do not have a report name collision on the server, things are about as clear as mud on the client side. There are far too many reports to use the simplistic hard-coding technique we used earlier. Further, giving a simple report name will not be sufficient for the end-user reports – we need a way to specify folder names too. Fortunately, there is a component which will give us a nice “bird’s eye view” of all reports on the server, without forcing us to hard-code anything: the ClientReportExplorer.



Let’s modify the thin-client application we created earlier by adding a ClientReportExplorer component to it. This component does not contain a ServerConnection object, but instead relies on a ClientReport to do its connecting. That means we simply assign the ClientReport property and away we go. The ClientReportExplorer works much like a Delphi System Dialog component; you call the Execute method and a dialog is displayed. Let’s code a Launch button:

```
rsClientReportExplorer1.Execute;
```

The screen-shot below shows what happens when we run this application and click the Launch button. It’s the same old Report Explorer from the end-user application, only this explorer is running across the wire, and is displaying reports from a lot more sources than just the end-user database tables.

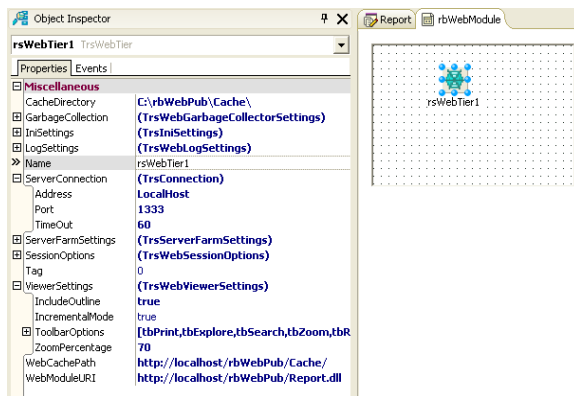


A full-blown Report Explorer may be more than you really need. If this is the case then rest assured that the classes used by the Report Explorer to query the server, discover volumes, folders and reports are fully available to you. If you need to implement a custom UI in your thin client app, check out the TrsClientReportCatalog class for an easy way to get started.

Publishing Reports to the Web

The Server Edition has a special component, called the WebTier, which converts the information provided by the server into an HTML/JavaScript application. Let's configure a web tier and get a feel for how it works. We'll assume that our web tier will reside on a PC running IIS, and that we will deploy the web application as an ISAPI DLL.

First we create a new Web Server application from within Delphi. To do this, select File | New | Other... and then select Web Server Application. This will create a WebModule, which is a special kind of data module that knows how to talk to an IIS web server. In the web module we place a WebTier component.



Assuming the web server is running on the same machine as the report server application, we set the following three property values:

```
CacheDirectory   C:\myWebTier\Cache\
ServerConnection {use the default values}
WebCachePath    http://localhost/myWebTier/Cache/
WebModuleURI    http://localhost/myWebTier/myWebTier.dll
```

The CacheDirectory names the system directory where the WebTier will store the output generated for a given session (folder/tree structures, page object files, and images.)

The WebCachePath is the URI to the cache directory. We will create a virtual directory on the server which maps to the CacheDirectory on the PC. The virtual directory describes a path which will be accessible from a web browser (i.e. you can type it as an address in a web browser and the server will return the content of the directory.)

The WebModuleURI is an HTTP address which describes the exact location of the web tier on the server. This is the address you type into the browser to invoke the web tier.

Next we need to process the requests received from web browsers. To do this, we need to hook the Web Module to the Web Tier:

- 1 Right-click over the Web Module and select "Action Editor."

- 2 Add an action to the list.

- 3 Use the Object Inspector set the Default property of the Action to True.

- 4 Code the OnAction event of the Action as:

```
Response.Content := rsWebTier1.ProcessWebRequest (
    Request.QueryFields,
    Request.Content);
```

The web tier handles all possible requests through the method: ProcessWebRequest, and so we pass http requests straight through to the web tier, which in turn provides the appropriate response. For more on Web Modules and Actions, see the Delphi help.

We can now compile myWebTier and copy the resulting DLL to C:\myWebTier. That's all there is to it!

We may be done generating the DLL, but IIS isn't yet configured to use it. We need to create a virtual directory, so that IIS can make our new DLL available from a web browser. To do this:

- 1** Select Start | Control Panel and double-click Administrative Tools.
- 2** Double-click Internet Information Services (if you can't find this icon, you probably do not have IIS installed. Double-click Add/Remove Programs and then click Add/Remove Windows Components – from there you can install IIS.)
- 3** Expand the “Local Computer” entry and find the Default Web Site entry under Web Sites.
- 4** Right-click over Default Web Site and select New | Virtual Directory. The Virtual Directory Creation Wizard will be displayed.
- 5** Enter an alias of “myWebTier.”
- 6** Click Finish, find the new virtual directory and select it.
- 7** Right-click over the virtual directory and select Properties.
- 8** Locate the Application Settings section at the bottom of the first tab.
- 9** Set Application Protection to High (Isolated) (This will keep the web tier from crashing the web server and other ISAPI DLLs from crashing the web tier.)
- 10** Click OK to exit the dialog.
- 11** Close the Internet Information Services and Administrative Tools windows.

We're now ready to test the web tier! Launch your web browser and enter the following address:

<http://localhost/myWebTier/myWebTier.dll>

If we've configured everything properly, you should see the HTML version of the report explorer in your web browser. Click on a report and the HTML version of the report viewer should be displayed.

Making Web Browser Content Printable With PDF

If you're previewing a report in the browser, how do you print it? Well, we can certainly right-click over the frame which contains the page, select Print and we will get "something." But honestly, HTML just doesn't print very well. To get the high-quality output we normally expect from ReportBuilder, we'll need to harness yet another "web technology," PDF and the Acrobat Reader.

ReportBuilder Server Edition includes web adapter classes that enable the WebTier to use ReportBuilder's built-in PDF device as well as third-party add-ons available from Gnostice, Pragmaan, and Waler. (Note that the third-party web adapters require that you have the relevant product installed.)

To add PDF support to your web application, add the appropriate unit reference to the uses clause of your web module

PDF Device	Web Adapter Unit
ReportBuilder	rsWebAdapaterPDF
Gnostice	rsWebAdapaterPDFGnostice
Pragmaan	rsWebAdapaterPDFPragmaan
Waler	rsWebAdapaterPDFWaler

After you have done this, recompile the project and copy the resulting web tier DLL to the virtual directory. Test the new DLL from your web browser. You should now see a printer icon on the far left of the report viewer. Clicking on this icon launches a separate browser window with the entire report in PDF format (of course the Adobe Acrobat Reader plug-in must be installed in order to view the report.)

Troubleshooting the WebTier

I receive the message “Unable to connect to server.”

Either the server was not started successfully or the server has crashed. In the latter case, if the server is not being deployed as a Windows service it will have to be manually restarted.

I receive the compile error “Could not create output file C:\Inetpub\wwwroot\rbbin\Report.dll” when trying to build the web tier project.

Most likely the Report.dll has been loaded by IIS. Access the IIS manager, right click on the virtual directory where Report.dll resides, select Properties and click Unload. If that fails, restart the web server.

I have LogSettings.EnableLogging set to true on the web tier but the log file is not being generated.

Make sure that the user account which is being used to execute the web tier has permission to write to the directory specified by LogSettings.Location. The default user account used by IIS is IWAM_Machinename.

When I try to bring up the report explorer the browser cannot load the frames.

There are a few reasons this could be happening. First, check that the WebCachePath property on the web tier has been set correctly (should have machine IP address instead of “localhost”.) Second, check if the desired path is set up correctly in the web server. Third, make sure that the cache directory is set up for web access. This is usually done by adding the “Internet Guest Account (IUSR_MachineName)” account to the access list for the cache directory.

Where’s my print button?

To use ReportBuilder's built-in PDF support, add rsWebAdapterPDF to the uses clause of the web module. To use a PDF add-on product from Gnostice, Pragmaan, or Waler, first install the add-on product and then add rsWebAdapterPDFGnostice, rsWebAdapterPDFPragmaan, or rsWebAdapterPDFWaler to the uses clause of the web module.

I have to wait a really long time to see my report.

There are a number of reasons this could be happening. The first and foremost is that the requested report is very complex. That is it either has a query that takes a long time to execute or the report has numerous calculations and takes a long time to process. There are numerous ways to improve the execution of the report itself which will not be discussed in this article. There are, however, a number of options which you could control on the web tier to help improve performance. Setting the viewer to incremental mode (ViewerSettings.Incremental Mode) and setting garbage collection to run in a separate thread (GargabeCollection.SeparateThread) could potentially reduce the latency time for receiving the first page of the report.

The text in the outline is all garbled.

This is simply due to how your browser wraps text. Resizing the outline frame should fix the problem.

When I execute a new report, some of my old reports are wiped from the cache.

The web tier has a setting which determines the maximum number of reports which could be present in the cache simultaneously per session (SessionOptions.MaxReportCount). When the specified number of reports in the cache has been reached, the next incoming request from the same session will cause the least recently viewed report be wiped from cache.

After having run a report once my data has changed, but when I return to my report from the report explorer I see the same old report. How can I see the new data?

The web tier caches report data to avoid having to rerun the report and give you better performance. To force a report to be regenerated click the Refresh button on the navigation toolbar.

I receive a message telling me that logging has been disabled. What does this mean?

This is a non-critical error which means that logging has been enabled on the web tier but the web tier was not able to successfully log its activity in the specified location. You can continue working normally but the error will recur every time the web tier is reloaded. The most likely cause of this problem is that the directory specified for the web log either does not exist or the web tier does not have permission to write to it.

In the WebTier Cache directory, none of the sessions are ever deleted.

Session folders should be deleted when the session timeout (WebTier.SessionOptions.SessionTimeout) is exceeded (i.e. no activity for the session has occurred in the last five minutes.) However, the WebTier.dll must have read/write access to the cache directory. The WebTier.dll runs under the account IWAM_<MachineName>. Right-click over the cache directory, access Properties... and click the Security tab. Select IWAM_<Machine-User> and give this user Full Control.

FAQ

Do I have to change all of my reports to get them to work on the server?

No. But you do have to provide thread-safe containers for the database connection components needed by your reports. That may sound scary, but it's really no big deal.

The documentation for the Server Edition will include directions for creating thread-safe containers for all of the popular database/connectivity product combinations. These containers can be Delphi data modules or invisible forms and should be configured with the appropriate Volume components or direct registration calls. Through the registration process you associate containers with reports so that the reports can successfully connect to a database when loaded. Report related event-handler code must also be thread-safe. However, unless you are referencing global objects or resources that exist outside the context of the container form or datamodule, then the code is probably thread-safe as is.

How hard is it to get the server up and running?

It is very easy. The Server component encapsulates all of the functionality required to communicate with clients and execute reports on their behalf. Your primary task will be registering your reports with the server's report catalog. If the reports are deployed as components on Delphi forms or in data modules, this requires a simple registration call to the report catalog. If the reports are deployed as report templates, as report archives, or as report templates in the end-user database structure it is even simpler. You can connect one of the Volume components to the file directory structure or the database tables and it will dynamically generate catalog items for all of the reports residing in the structure. Using this latter scheme you can add and remove reports without bringing the server down.

How many users can practically be served at once?

It depends. A report based on a simple query, which retrieves few records and generates few pages can be run by many users at once. A report which utilizes multiple complex queries, pulls millions of records and generates tens of thousands of pages may be all the server can handle at any one time. The best way to determine the answer to this question is to create a report server with your reports and see how it goes. You have lots of options regarding live report vs. archive report, full generation vs. incremental generation and caching configuration options on both the report server and the web tier. There is also multiple CPU and multiple server configurations to consider.

How fast is the server?

Just as fast as ReportBuilder, but certainly no faster. Remember that directory structures (folder trees), search criteria, and page objects must be sent across the wire to the client. This takes additional time. Yes, the messages are compressed. Yes, multi-threading means that the messages will be processed in a timely fashion. But there are many, many factors which control the performance any given user may experience. The best bet is to test the server using your reports and your client application.

How do I stop a "runaway report" on the server?

The `Volume.PublishingOptions` can be used to specify the maximum number of pages and maximum number of seconds for report execution. These values can be used as the defaults for all reports in the volume. The `Volume.OnGetPublishingOptions` event can be used to dynamically set the publishing options for individual reports.

Proper use of the Volume.PublishingOptions should prevent runaway reports. However, in the event of a runaway report, you will need to stop the service or (if you are running the server in stand-alone mode) shut down the server application. A future version of the Server Edition may contain an administrative application where you can stop individual sessions – but for now we are focused on solidifying a fast, stable and scalable core.

What happens if the server crashes?

When configured correctly, the server will simply clean up after itself and automatically restart. This is why you should deploy your server application using the Windows Service scheme. Under this scheme if the server crashes, the Windows Service can automatically restart the application. The server application is running in its own process space – which means that other applications on the same machine should not be affected if the server crashes. For reasons of both performance and stability, you should also run your web tier as an Isolated Process. This keeps other ISAPI DLLs on your web server from crashing your web tier and vice versa. Of course, when a crash does occur, the exception will be logged – so you should be able to isolate the report causing the problem.

Does the server do any kind of logging?

Yes. Both the report server application and the web tier will automatically log any exceptions (unless logging is turned off.) The web tier also has a “verbose” mode where it logs every request, every response, every run of the garbage collector and every exception. By default this verbose mode is turned off, because logging adversely affects performance. The log format can be set to ASCII text or XML.

Can I hot-swap reports running on the server?

Yes. The Report Server can dynamically discover the existence of new folders, reports, and archives. Thus you can add or replace reports and archives that are stored in files or a database. Of course changing reports which are compiled directly into the server application requires that the server be stopped and replaced with a new executable containing the new reports. That is why the TemplateVolume, ArchiveVolume, and ExplorerVolume components are preferred - they specify a location where reports are stored.

Can I administer the server remotely?

Yes, using Windows remote desktop or a product such as PCAnyWhere, you can run the RB Services Manager. The RB Services Manager is available from the Start Menu, under ReportBuilder | Services Manager. It is a system tray based application which allows you to start/stop the report server or change the location of the server app (if you need to move the app to a different directory.) The RB Services Manager is available only when the server is deployed as a Windows Service.

How is printing handled from the web browser?

The Web Report Previewer includes a Print button on the toolbar. Pressing the Print button generates a PDF that is opened on the user's machine by Adobe Acrobat Reader.

Does the server support https?

Yes. Not only can you secure the web tier so that it is only accessible via https, you can configure RB thin-client components so that they talk to the web tier instead of the report server, making thin-client communications secure.

Does the server provide encryption?

The first version of the server does not include encryption. One solution is to use HTTPS to communicate between clients and the WebTier. Then deploy the Server and the WebTier on the same machine or perhaps run the Server application on a separate machine behind your firewall.

Can I provide a login and user name before users get access to the reports?

Yes, an example is installed with RB Server Edition. The example also shows how to allow different users to see different sets of reports based upon login credentials.

Can Server run on Linux?

No. While the report engine inside the server application is functioning without a user-interface, many dependencies on the Windows Operating System (and the associated API) still exist. We consider a Linux server to be a mammoth undertaking and are not committing to such a project at this time.

Can I use the Server Edition to build ISAPI, CGI, ASP, and Apache applications?

The Server Edition leverages Delphi's ability to build several types of web applications. The WebTier component can be included in a Delphi WebBroker application, WebSnap application, or Active Server Object. The Server Edition includes examples of using each of these technologies to build web applications.

Can I use the Server Edition with WebBroker, WebSnap, or IntraWeb?

Yes. We have examples of using the WebTier with WebBroker, ASP, and ASP.NET. We do not currently have an example of using IntraWeb, however, the WebTier can be included in any web application as long you forward the web report request to the WebTier by calling its ProcessRequest method. The WebTier can handle the rest.

Is the Server Edition royalty-free?

No. The Server Edition provides deployment licenses which allow you to run a server application on a single CPU. If you decide to run the application on a machine with multiple CPUs then you will need to purchase one deployment license for each CPU. The Server Edition includes one server development license and one server deployment license. Each deployment license is “unlimited-user,” you can run as many users against any given server as you like.

Do I need to buy both the Server Edition and the Enterprise Edition?

No. The Server Edition includes the Enterprise Edition, so you’ll have all of the tools you need to both create reports and deploy them to the web.

How much does it cost?

The Server Edition is US \$999 - and that includes everything in the Enterprise Edition, the Server Edition components, one server development license, and one server deployment license. The Enterprise Edition already costs \$749 – so you are basically getting all of the Server Edition technology for \$250. Considering that additional deployment licenses are only \$249, and that each license allows you to run unlimited users against the server, it’s probably the most cost effective solution out there.

TUTORIALS

- Building a Report Server Application for Reports on Forms 29
- Building a Report Server Application for Reports in Files 33
- Building a Report Server Application for Reports in a Database 37
- Building a Report Server Application for an Explorer Database 41
- Building a Report Server Application for Report Archives 45
- Run a Report Server Application from a Windows Service 49
- Publish Reports to the Web 53
- Scaling the Web Tier with a Server Farm 57

Building a Report Server Application for Reports on Forms

Overview

This tutorial will show you how to:

- Configure Form-based reports for use in a Report Server Application
- Preview Reports from a Thin-Client Application

Copy Existing Report Application to a New Directory

1 From the Windows desktop, launch the Windows Explorer.

2 Create the directory:

C:\My RB Tutorials\01. Form-Based Report Server

3 Copy all of the files in:

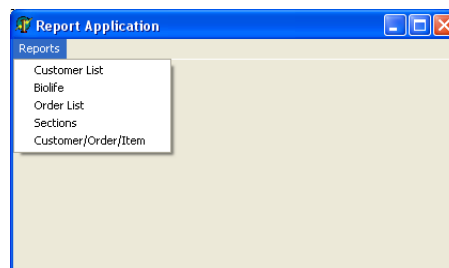
*...RBServer\Tutorials\Server Projects
\01. Form-Based Report Application*

to the new directory.

4 Select File | Open Project from the Delphi menu, locate the project in the new directory and open it.

5 Select File | Save Project As from the Delphi menu and save the project under the name “rbServer.”


6 Run the project. You should see a single menu item named “Reports.” Under this menu there should be five reports listed. You should be able to preview any of these reports.



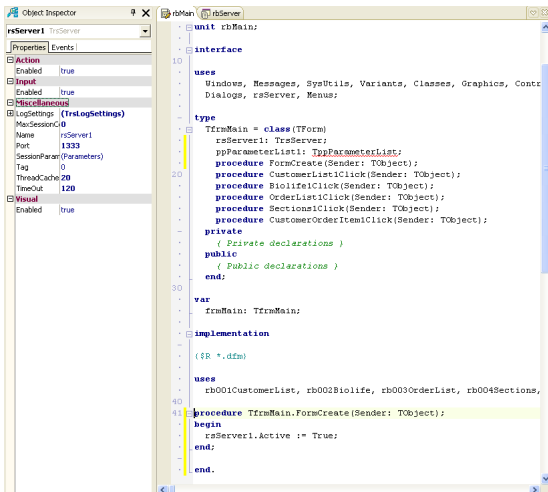
7 Close the application and return to the Delphi IDE.

Convert the Application to a Server

- 1 Select View | Project Manager from the Delphi menu.
- 2 Double-click the form named rbMain.
- 3 Change the caption of the form to:

Form-Based Report Server
- 4 Select the RBServer tab on the Component Palette.
- 5 Add an rsServer  component to the form.
- 6 Select the form.
- 7 Code the OnCreate event of the form as:

```
rsServer1.Active := True;
```
- 8 Press F12, to display the form.
- 9 Delete the menu component from the form.
- 10 Use the Code Editor to remove all event handlers associated with the menu (This should leave no code in the main form at all, except the server activation in the FormCreate method.)



- 11 Close the Form and the associated unit, saving all changes.

Register the Reports On the Server

- 1 Select View | Project Manager from the Delphi menu.
- 2 Double-click the form named “rb001CustomerList.”
- 3 Scroll to the bottom of the unit, and add the following code:

```
uses
  rsReportCatalog;


initialization
  TrsReportCatalog.RegisterReport('Examples',
    'CustomerList',
    'ppReport1',
    Tfrm001CustomerList);
```
- 4 Close the form, saving all changes.
- 5 Add a registration call to each of the remaining forms:


- rb002Biolife
- rb003OrderList
- rb004Sections
- rb005CustomerOrderItem

Note: For reports which contain event handlers, the uses clause will need to appear after the implementation keyword as opposed to directly before the initialization keyword.

- 6 Compile the project to make sure all code is correct.

Make the Data Access Components Thread-Safe

- 1 Select View | Project Manager from the Delphi menu.
- 2 Double-click the form named “rb001CustomerList.”
- 3 Select the BDE tab of the component palette.
- 4 Place a TDatabase  component on the form.
- 5 Set the AliasName to “DBDemos.”

- 6 Select all dataset components on the form and set the DatabaseName to “Database1.”
- 7 Place a TSession  component on the form.
- 8 Set AutoSessionName to True.
- 9 Select the TDatabase and TSession components and copy them to the clipboard.
- 10 Close the form saving all changes.
- 11 Open each of the following forms, and paste the Database and Session components into them.

rb002Biolife
 rb003OrderList
 rb004Sections
 rb005CustomerOrderItem

- 12 Select File | Close All from the Delphi menu, saving all changes.



Run the Server Application

- 1 From the Windows desktop, launch a Windows Explorer.
- 2 Locate the rbServer.exe in the directory you created for this tutorial.
- 3 Double-click the EXE to launch the application.



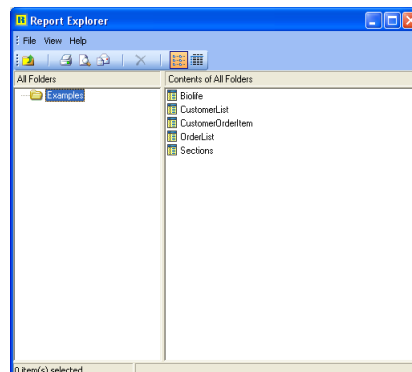
- 4 Minimize the application.

Create a Thin-Client Application

- 1 Select File | New | Application from the Delphi menu.
- 2 Select File | Save Project As... from the Delphi menu.
- 3 Name the form’s unit frmThinClient and save it in the directory with the Server application.
- 4 Name the project rbThinClient and save it in the same directory.
- 5 Select the RBServer tab of the Component Palette.
- 6 Place a rsClientReport  component on the form.
- 7 Place a rsClientReportExplorer  component on the form.
- 8 Set the ClientReport property to “ClientReport1.”
- 9 Code the OnCreate event of the form as:

```
rsClientReportExplorer1.Execute;
```

- 10 Select File | Save All from the Delphi menu.
- 11 Run the application. You should see the Report Explorer with all of the registered reports in the “Examples” folder. You should be able to preview the reports.



Building a Report Server Application for Reports in Files

Overview

This tutorial will show you how to:

- Configure File-based Reports for use in a Report Server Application
- Preview Reports from a Thin-Client Application
- Create a Thread-Safe Data Access Configuration
- Use Delphi Event Handlers with Server-based Reports

Copy Existing Report Application to a New Directory

1 From the Windows desktop, launch the Windows Explorer.

2 Create the directory:

C:\My RB Tutorials\02. File-Based Report Server

3 Copy all of the files in:

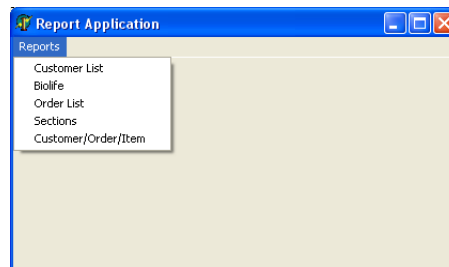
...RBuilder\Tutorials\Server Projects\02. File-Based Report Application

to the new directory.

4 Select File | Open Project from the Delphi menu, locate the project in the new directory and open it.



5 Select File | Save Project As... from the Delphi menu and save the project under the name “rbServer.”

6 Run the project. You should see a single menu item named “Reports.” Under this menu there should be five reports listed. You should be able to preview any of these reports.




7 Close the application and return to the Delphi IDE.

Create a Thread-Safe Data Module

- 1 Select File | New | Data Module from the Delphi menu.
- 2 Save the data module as dmMain.
- 3 Select View | Project Manager and open the rbMain form.
- 4 Cut the Database  component from the form and paste it into the data module.
- 5 Select the BDE tab of the component palette.
- 6 Add a TSession component  to the data module.
- 7 Set the AutoSessionName property to True.
- 8 Press Ctrl-S to save all changes.

Register the File-Based Reports

- 1 Select the RBServer tab of the component palette.
- 2 Add an rsReportTemplateVolume  component to the data module.
- 3 Set the FileDirectory property to:


```
C:\My RB Tutorials\02. File-Based Report
Server
```
- 4 Set the VolumeName to “Examples.”
- 5 Press Ctrl-S to save all changes.

Move the Report Event Handlers to the Data Module

- 1 Use the Delphi Code Editor to access the unit for the main form.
- 2 Locate the following event handlers in the class declaration for the form:

```
{rb004Sections}
procedure ppEmpSalesGroupFooterBand1BeforePrint(Sender: TObject);
procedure ppLabelContinuedPrint(Sender: TObject);
procedure ppStockListDetailBeforePrint(Sender: TObject);

{rb005CustomerOrderItem}
procedure varItemTotalCalc(Sender: TObject; var Value: Variant);
procedure varOrderTotalCalc(Sender: TObject; var Value: Variant);
procedure ppDetailBand1BeforePrint(Sender: TObject);.
```

- 3 Select and cut these event handler declarations into your clipboard.
- 4 Click the dmMain tab of the Code Editor and paste the declarations into the published section of the data module class declaration.
- 5 Click the rbMain tab of the Code Editor and scroll down to the implementations of these methods. Select and cut the implementations into your clipboard.
- 6 Click the dmMain tab of the Code Editor and paste the code into the implementation section of the data module.
- 7 Scroll to the first method in the data module and place the cursor directly in front of TfrmMain class name.
- 8 Press Ctrl-R; the Replace Text dialog is displayed.
- 9 Enter “TDataModule1” into the Replace edit box.

10 Click the “All” button.

11 Move the following supporting method from the main form to the private section of the data module:

```
function GetDataPipelineForName(aList: TList;
                               const aComponentName: String
                               ): TppDataPipeline;
```

12 Declare the following implementation uses in the data module, so that the report event handlers will compile:

```
uses
  Graphics,
  ppClass, ppCtrls, ppVar, ppBands, ppReport, daDataModule,
  daDBBDE;
```

13 Scroll to the top of the unit and add “ppDB” to the interface uses clause.

14 Right-click over the dmMain tab of the Code Editor and close the unit, saving all changes.

Convert the Application to a Server

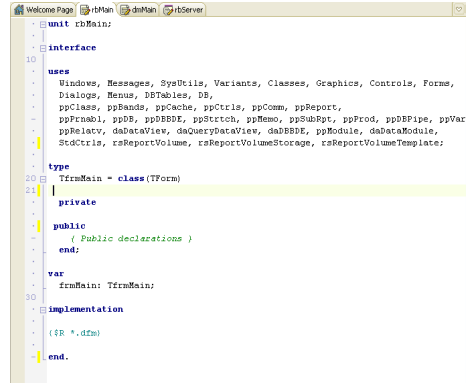
1 Select View | Project Manager from the Delphi menu.

2 Double-click rbMain to display the form.

3 Delete the Menu component.

4 Remove all event handlers associated with the menu.

5 Delete the FReport and FPath private variables and any code associated with them from the main form. The unit should look like this:



6 Change the Caption of the form to:

File-Based Report Server

7 Add an rsServer  component to the form.

8 Code the OnCreate event of the form as:

```
rsServer1.Active := True;
```

9 Run the application. It should compile and run with no errors.

10 Shut-down the application.

11 Select File | Close All from the Delphi menu, saving all changes.

Run the Server Application



1 From the Windows desktop, launch a Windows Explorer.

2 Locate the rbServer.exe in the directory you created for this tutorial.

3 Double-click the EXE to launch the app.

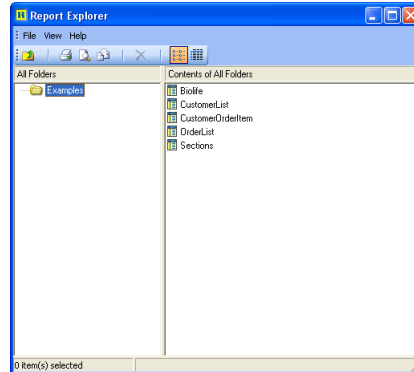
4 Minimize the app.

Create a Thin-Client Application

- 1 Select File | New | Application from the Delphi menu.
- 2 Select File | SaveProject As... from the Delphi menu.
- 3 Name the form's unit frmThinClient and save it in the directory with the Server application.
- 4 Name the project rbThinClient and save it in the same directory.
- 5 Select the RBServer tab of the Component Palette
- 6 Place a rsClientReport  component on the form.
- 7 Place a rsClientReportExplorer  component on the form.
- 8 Set the ClientReport property to "ClientReport1."
- 9 Code the OnCreate event of the form as:

```
rsClientReportExplorer1.Execute;
```
- 10 Select File | Save All from the Delphi menu.

- 11 Run the application. You should see the Report Explorer with all of the registered reports in the "Examples" folder. You should be able to preview the reports.



Building a Report Server Application for Reports in a Database

Overview

This tutorial will show you how to do the following:

- Configure Databased Report Templates for use in a Report Server Application
- Preview Reports from a Thin-Client Application
- Create a Thread-Safe Data Access Configuration
- Use Delphi Event Handlers with Server-Based Reports

Copy Existing Report Application to a New Directory

1 From the Windows desktop, launch the Windows Explorer.

2 Create the directory:

C:\My RB Tutorials\03. Databased Report Server

3 Copy all of the files in:

...RBuilder\Tutorials\Server Projects\03. Databased Report Application

to the new directory.

4 Select File | Open Project from the Delphi menu, locate the project in the new directory and open it.

5 Select File | Save Project As... from the Delphi menu and save the project under the name "rbServer."

Update the Path of the Database Component

1 Double-click the Database2 database component. Locate the PATH setting at the top of the Parameter overrides list. We need to update this path with the new location of the reporting tables.

2 From the Windows desktop, launch the Windows Explorer.

3 Navigate to the directory for the application:

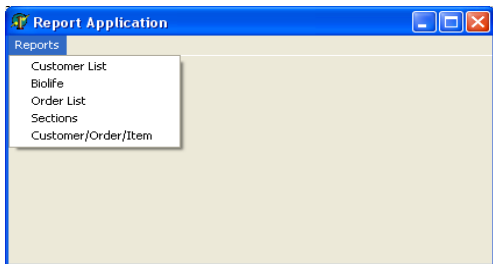
C:\My RB Tutorials\03. Databased Report Server

4 Copy the path from the Address box of the Windows Explorer.

5 Return to the Delphi IDE and replace the database path with the path in your clipboard.

6 Close the Database component window.

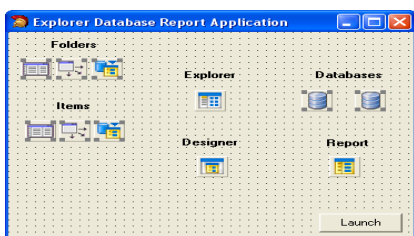
- 7 Select the Table1 component and set Active to True. This tests the new database path setting.
- 8 Run the project. You should see a single menu item named “Reports.” Under this menu there should be five reports listed. You should be able to preview any of these reports.



- 9 Close the application and return to the Delphi IDE.

Create a Thread-Safe Data Module

- 1 Select File | New | Data Module from the Delphi menu.
- 2 Save the data module as dmMain.
- 3 Select View | Project Manager and open the rbMain form.
- 4 Select the two database components, the table, datasource and data pipeline components.



- 5 Cut the selection and paste it into the data module.
- 6 Select the BDE tab on the Delphi component palette.
- 7 Place a TSession component in the data module.

- 8 Set the AutoSessionName property of the Session component to True.
- 9 Press Ctrl-S to save all changes.

Register the Databased Reports

- 1 Select the RBServer tab of the Component Palette.

- 2 Add an rsReportTemplateVolume component to the data module.

- 3 Expand the DatabaseSettings property and configure:

DataPipeline	ppDBPipeline1
BLOBField	Template
NameField	Name

- 4 Set Storage Type to stDatabase.
- 5 Set the VolumeName to “Examples.”
- 6 Press Ctrl-S to save all changes.

Move the Report Event Handlers to the Data Module

- 1 Use the Code Editor to access the unit for the main form.
- 2 Locate the following event handlers in the class declaration for the form:

```

{rb004Sections}
procedure ppEmpSalesGroupFooterBand1BeforePrint(Sender: TObject);
procedure ppLabelContinuedPrint(Sender: TObject);
procedure ppStockListDetailBeforePrint(Sender: TObject);

{rb005CustomerOrderItem}
procedure varItemTotalCalc(Sender: TObject; var Value: Variant);
procedure varOrderTotalCalc(Sender: TObject; var Value: Variant);
procedure ppDetailBand1BeforePrint(Sender: TObject);
    
```

- 3 Select and cut these event handler declarations into your clipboard.
- 4 Click the dmMain tab of the Code Editor and paste the declarations into the published section of the data module class declaration.
- 5 Click the rbMain tab of the Code Editor and scroll down to the implementations of these methods. Select and cut the implementations into your clipboard.

6 Click the dmMain tab of the Code Editor and paste the code into the implementation section of the data module.

7 Scroll to the first method in the datamodule and place the cursor directly in front of TfrmMain class name.

8 Press Ctrl-R; the Replace Text dialog is displayed.

9 Enter “TDataModule1” into the Replace edit box.

10 Click the “All” button.

11 Move the following supporting method from the main form to the private section of the data module:

```
function GetDataPipelineForName(aList: TList;
                               const aComponentName: String;
                               ): TppDataPipeline;
```

12 Declare the following implementation uses in the data module unit, so that the report event handlers will compile:

```
uses
  Graphics,
  ppClass, ppCtrls, ppVar, ppBands, ppReport, daDataModule
  daDBBDE;
```

13 Right-click over the dmMain tab of the Code Editor and close the unit, saving all changes.

Convert the Application to a Server

1 Select View | Project Manager from the Delphi menu.

2 Double-click rbMain to display the form.

3 Change the Caption of the form to:

Databased Report Server

4 Add an rsServer  component to the form.

5 Replace the existing OnCreate event handler with:

```
rsServer1.Active := True;
```

6 Press F12 to display the form.

7 Delete the menu component from the form.

8 Use the Code Editor to remove all event handlers associated for the menu .

9 Delete the FReport private variable and any code associated with it from the main form.

This should leave no code in the main form at all, except the server activation in the FormCreate method.



10 Run the application. It should compile and run with no errors.

11 Shut-down the application.

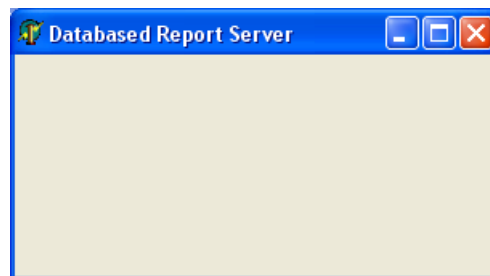
12 Select File | Close All from the Delphi menu, saving all changes.

Run the Server Application

1 From the Windows desktop, launch a Windows Explorer.



2 Locate the rbServer.exe in the directory you created for this tutorial.

3 Double-click the EXE to launch the app.

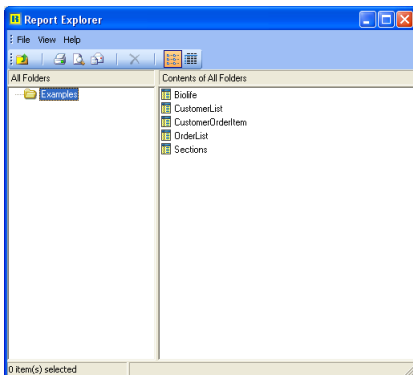


4 Minimize the app.

Create a Thin-Client Application

- 1 Select File | New | Application from the Delphi menu.
- 2 Select File | Save Project As... from the Delphi menu.
- 3 Name the form's unit frmThinClient and save it in the directory with the Server application.
- 4 Name the project rbThinClient and save it in the same directory.
- 5 Select the RBServer tab of the Delphi component palette
- 6 Place a rsClientReport  component on the form.
- 7 Place a rsClientReportExplorer  component on the form.
- 8 Set the ClientReport property to "ClientReport."
- 9 Code the OnCreate event of the form as:

```
rsClientReportExplorer1.Execute;
```
- 10 Select File | Save All from the Delphi menu.
- 11 Run the application. You should see the Report Explorer with all of the registered reports in the "Examples" folder. You should be able to preview the reports.



Building a Report Server Application for an Explorer Database

Overview

This tutorial will show you how to do the following:

- Configure Explorer Reports for use in a Report Server Application
- Preview Reports from a Thin-Client Application
- Create a Thread-Safe Data Access Configuration
- Use RAP Event Handlers with Server-Based Reports.

Copy Existing Report Application to a New Directory

1 From the Windows desktop, launch the Windows Explorer.

2 Create the directory:

C:\My RB Tutorials\04. Explorer Database Report Server

3 Copy all of the files in:

...RBuilder\Tutorials\Server Projects\04. Explorer Database Report Application

to the new directory.

1 Select File | Open Project from the Delphi menu, locate the project in the new directory and open it.

2 Select File | Save Project As... from the Delphi menu and save the project under the name "rbServer."

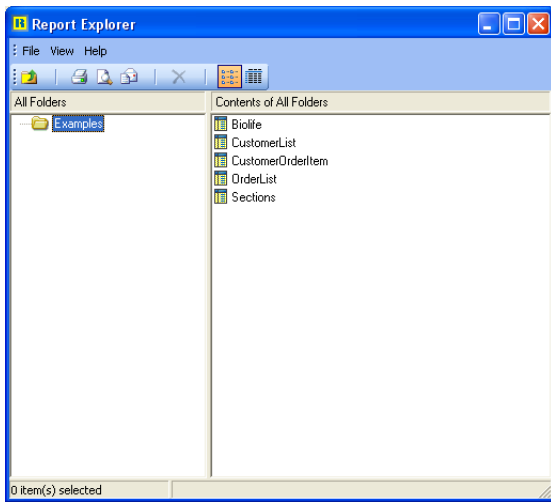
3 Double-click the dbSupportingTables database component. Locate the PATH setting at the top of the "Parameter overrides" list. We need to update this path with the new location of the reporting tables.

4 From the Windows desktop, launch the Windows Explorer.

5 Navigate to the directory for the application:

C:\My RB Tutorials\04. Explorer Database Report Server

- 6 Copy the path from the Address box of the Windows Explorer.
- 7 Return to the Delphi IDE and replace the database path with the path in the clipboard.
- 8 Close the Database component window.
- 9 Select the “tblFolder” component and set Active to True. This tests the new database path setting.
- 10 Run the project, and click the “Launch” button. You should see the Report Explorer and be able to preview any of the listed reports.

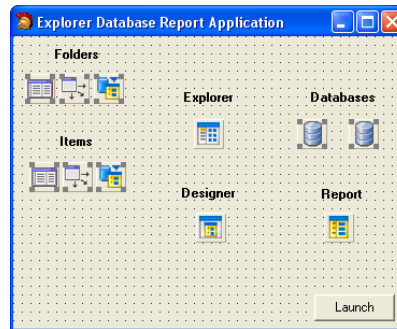


- 11 Close the application and return to the Delphi IDE.

Create a Thread-Safe Data Module

- 1 Select File | New | Data Module from the Delphi menu.
- 2 Save the Data Module as dmMain.
- 3 Select View | Project Manager and open the rbMain form.

- 4 Select all of the components in the groups labeled Folders, Items, and Databases.



- 5 Cut the selection and paste it into the data module.
- 6 Select the BDE tab on the Delphi component palette.
- 7 Place a TSession component in the data module.
- 8 Set the AutoSessionName property to True.
- 9 Press Ctrl-S to save all changes.

Register the Databased Reports

- 1 Select the RBServer tab of the Delphi component palette.
- 2 Add an rsReportExplorerVolume component to the data module.
- 3 Set the FolderPipeline property to plFolder.
- 4 Expand the FolderFieldNames property and make sure each field name is set as follows:


FolderId	FolderId
Name	Name
ParentId	ParentId

- 5 Set the ItemPipeline property to plItem data.
- 6 Expand the ItemFieldNames property and make sure each field name is set as follows:

Deleted	Deleted
FolderId	FolderId
ItemId	ItemId
ItemType	ItemType
Name	Name
Size	Size
Template	Template

- 7 Close the data module and the associated unit, saving all changes.

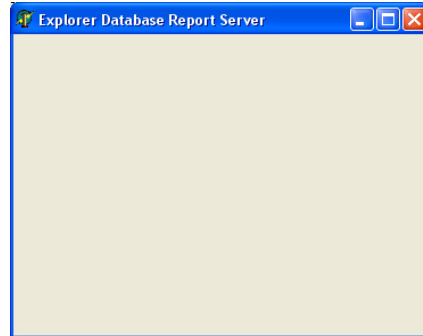
Convert the Application to a Server

- 1 Select View | Project Manager from the Delphi menu.
- 2 Double-click rbMain to display the form.
- 3 Change the Caption of the form to:
Explorer Database Report Server
- 4 Delete all of the components from the form. The form should now be blank.
- 5 Select the RBServer tab of the Delphi component palette.
- 6 Add a rsServer  component to the form.
- 7 Code the OnCreate event handler of the form as:

```
rsServer1.Active := True;
```

Delete the event handler for the “Launch” button.

- 8 Run the application. It should compile and run with no errors.





- 9 Shut-down the application.
- 10 Select File | Close All from the Delphi menu, saving all changes.

Run the Server Application

- 1 From the Windows desktop, launch a Windows Explorer.
- 2 Locate the rbServer.exe in the directory you created for this tutorial.
- 3 Double-click the EXE to launch the app.
- 4 Minimize the app.

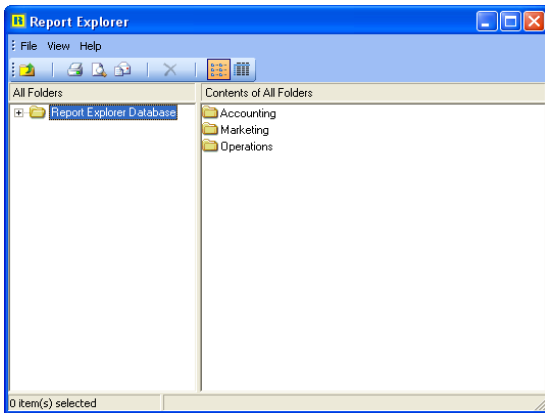
Create a Thin-Client Application

- 1 Select File | New | Application from the Delphi menu.
- 2 Select File | Save Project As... from the Delphi menu.
- 3 Name the form’s unit frmThinClient and save it in the directory with the Server application.

- 4 Name the project rbThinClient and save it in the same directory.
- 5 Select the RBServer tab of the component palette.
- 6 Place an rsClientReport  component on the form.
- 7 Place an rsClientReportExplorer  component on the form.
- 8 Set the ClientReport property to “ClientReport1.”
- 9 Code the OnCreate event handler for the form as:

```
rsClientReportExplorer1.Execute;
```

- 10 Select File | Save All from the Delphi menu.
- 11 Run the application. You should see the Report Explorer with all of the registered reports in the “Examples” folder. You should be able to preview the reports.



Building a Report Server Application for Report Archives

Overview

This tutorial will show you how to do the following:

- Access Report Archives from a Report Server Application
- Preview Reports from a Thin-Client Application

Copy Existing Report Application to a New Directory

1 From the Windows desktop, launch the Windows Explorer

2 Create the directory:

C:\My RB Tutorials\05. Archive Report Server

3 Copy all of the files in:

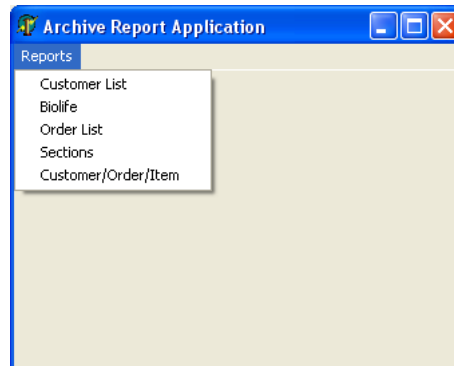
...RBuilder\Tutorials\Server Projects\05. Archive Report Application

to the new directory.

4 Select File | Open Project from the Delphi menu, locate the project in the new directory and open it.


5 Select File | Save Project As... from the Delphi menu and save the project under the name "rbServer."

6 Run the project. You should see a single menu item named "Reports." Under this menu there should be five reports listed. You should be able to preview any of these reports.



7 Close the application and return to the Delphi IDE.

Register the Report Archives

- 1 Select File | New | Data Module from the Delphi menu.
- 2 Save the data module as dmMain.
- 3 Select the RBServer tab on the Delphi component palette.
- 4 Add an rsReportArchiveVolume  component to the data module.
- 5 Set the FileDirectory property to:

C:\My RB Tutorials\05. Archive Report Server

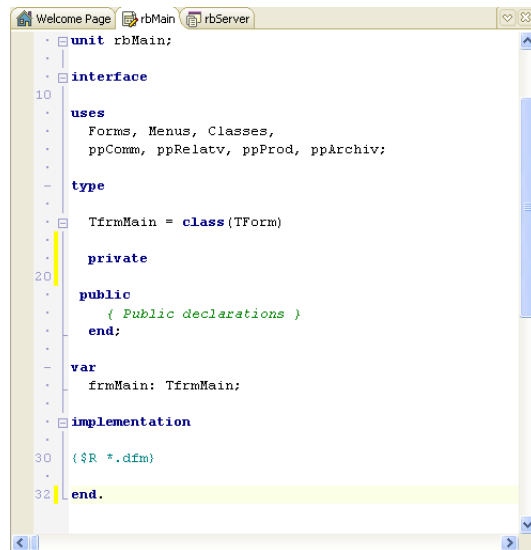
- 6 Set the VolumeName property to “Examples.”
- 7 Close the data module and the associated unit, saving all changes.

Convert the Application to a Server

- 1 Select View | Project Manager from the Delphi menu.
- 2 Double-click rbMain to display the form.
- 3 Change the Caption of the form to:

Archive Report Server
- 4 Delete the Menu and Archive Reader components from the form. The form should now be blank.

- 5 Remove all of the event handlers associated with the Menu from the form’s unit.



```

unit rbMain;
interface
uses
  Forms, Menus, Classes,
  ppComm, ppRelatv, ppProd, ppArchiv;
type
  TfrmMain = class(TForm)
  private
  public
    { Public declarations }
  end;
var
  frmMain: TfrmMain;
implementation
  {$R *.dfm}
end.
    
```

- 6 Select the RBServer tab of the Delphi component palette.

- 7 Add an rsServer  component to the form.

- 8 Code the OnCreate event of the form as:

```
rsServer1.Active := True;
```

- 9 Run the application. It should compile and run with no errors.



- 10 Shut-down the application.

- 11 Select File | Close All from the Delphi menu, saving all changes.

Run the Server Application

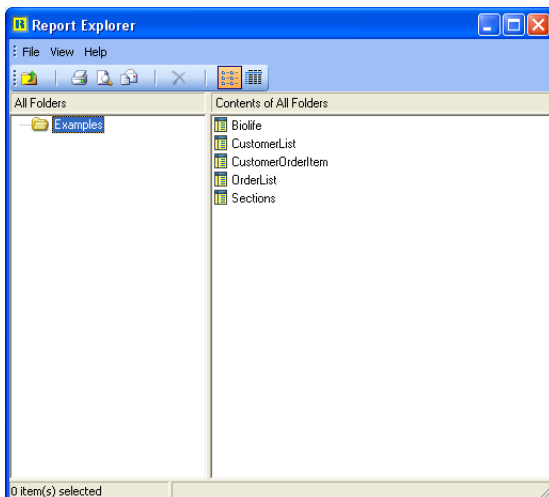
- 1 From the Windows desktop, launch a Windows Explorer.
- 2 Locate the rbServer.exe in the directory you created for this tutorial.
- 3 Double-click the EXE to launch the app.
- 4 Minimize the app.

Create a Thin-Client Application

- 1 Select File | New Application from the Delphi menu.
- 2 Select File | Save Project As... from the Delphi menu.
- 3 Name the form's unit frmThinClient and save it in the directory with the Server application.
- 4 Name the project rbThinClient and save it in the same directory.
- 5 Select the RBServer tab of the Component Palette
- 6 Place an rsClientReport  component on the form.
- 7 Place an rsClientReportExplorer  component on the form.
- 8 Set the ClientReport property to "ClientReport1."
- 9 Code the OnCreate event handler of the form as:

```
rsClientReportExplorer1.Execute;
```

- 10 Select File | Save All from the Delphi menu.
- 11 Run the application. You should see the Report Explorer with all of the registered reports in the "Examples" folder. You should be able to preview the reports.



Run a Report Server Application from a Windows Service

Overview

This tutorial will show you how to do the following:

- Install a Windows Service
- Use the RB Services Administrator to Designate a Report Server Application

In previous tutorials we built a Report Server application as a stand-alone EXE. While this is certainly a valid way to run a report server, the Server Edition provides for a two-piece server architecture, where a Windows Service hosts a report server application. This scheme is more robust; if the report server crashes, the Windows Service can restart it. This tutorial will show you how to put such an architecture in place.

Install the Windows Service

1 From the Windows Desktop, launch a Windows Explorer.

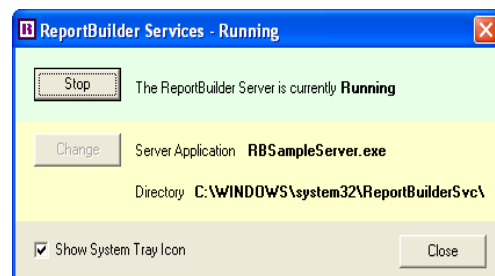
2 Locate the RBWinService.exe in:

```
C:\Program Files\Boland\Delphi6-  
\RBServer\Windows Service
```

3 Run this program and accept the default location for the service application. Once the installation process is complete, you should see the following icon on your system tray:



4 Double-click this icon to display the RB Services dialog:



This dialog shows that the service is up and running, and that a sample Report Server application is being hosted from the given location.

5 Close the dialog.

Create a Service Compatible Report Server Application

1 From the Windows Desktop, launch a Windows Explorer and create a new directory:

```
C:\My RB Tutorials\  
06. Windows Service-Based Server
```

2 Copy the contents of:

```
...RBServer\Tutorials\Complete\01. Build a  
Report Server Application\05. Reports in  
Archives
```

to the new directory.

3 Launch Delphi and open the rbServer project in the new directory.

4 Access the unit for the rbMain form.

5 In the interface section add “rsServerActiveX” to the uses clause.

6 Select Run | Parameters from the Delphi menu.

7 In the parameters edit box enter:

```
/regserver
```

8 Click OK to close the dialog.

9 Press F9 to run the project. If the COM server is registered successfully, no message will be displayed.

Update the File Directory of the Report Volume

1 Select View | Project Manager from the Delphi menu.

2 Double-click dmMain to display the data module.

3 Select the rsReportArchiveVolume1 component.

4 Set the FileDirectory property to:

```
C:\My RB Tutorials\  
06. Windows Service-Based Server
```

5 Select Project | Compile from the Delphi menu.

6 Select File | Close All from the Delphi menu, saving all changes.

Designate the Report Server Application from the Windows Service

1 Double-click the ReportBuilder Services  icon in the system tray.

2 Click the Stop button. The top part of the dialog should change to light red, indicating the service is stopped.

3 Click the Change button and designate the rbServer application in:

```
C:\My RB Tutorials\06. Windows  
Service-Based Server
```

4 Click the Start button. The top area of the dialog should turn light green, indicating the server is running.

5 Close the dialog.

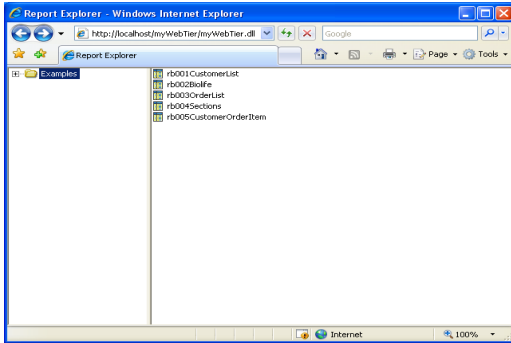
Run the Thin-Client Application

1 Return to the Delphi IDE.

2 Select File | Open Project from the Delphi Menu.

3 Locate rbThinClient.dpr in the directory for this tutorial and open it.

4 Run the application. You should see the Report Explorer with all of the registered reports in the “Examples” folder. You should be able to preview the reports.



Publish Reports to the Web

Overview

This tutorial will show you how to do the following:

- Build a WebTier as an ISAPI DLL
- Configure a Virtual Directory on IIS

For this tutorial, you will need a machine with IIS 5 installed. You will also need a report server application running. Therefore, completing the previous tutorial is recommended before beginning this tutorial.

Note: This tutorial was created using IIS 5, however the steps for IIS 4 are similar. Depending on your knowledge of IIS, you may be able to successfully use this tutorial with IIS 4.

Create a Directory for the Web Tier

1 From the Windows Desktop launch the Windows Explorer.

2 Create the following directory:

`C:\My RB Tutorials\07. WebTier\Cache`

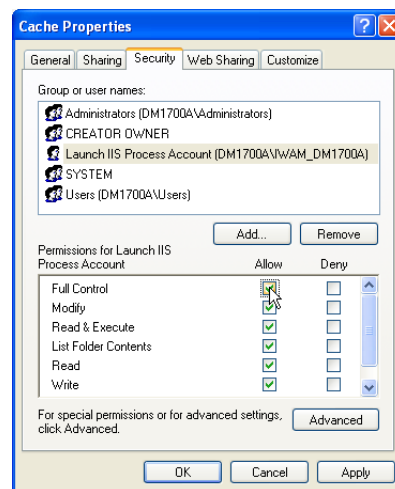
3 Select the “Cache” directory.

4 Right-click and select Properties.

5 If you see the Security tab, select this tab and continue with steps 6-8. If you do not see the Security tab, jump to the next section entitled “Create a Virtual Director on IIS.”

6 Select IWAM_<MachineName> user.

7 Click the “Full Control” checkbox.

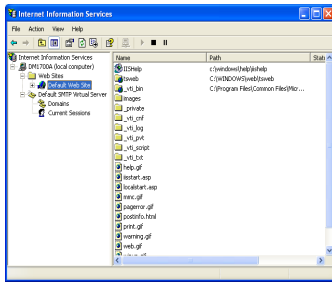


8 Close the dialog.

Note: The ISAPI DLL we are about to create needs full read/write access to the Cache directory in order to store and remove files as reports are generated.

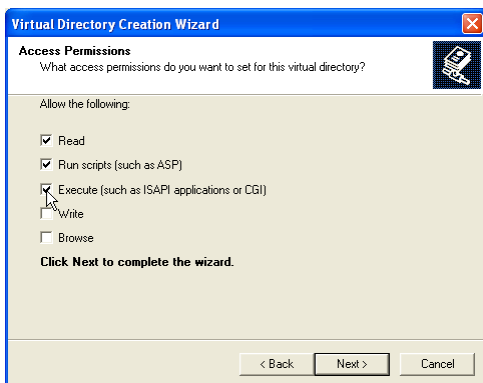
Create a Virtual Directory on IIS

- 1 Access the Control Panel and double-click Administrative Tools.
- 2 Double-click Internet Information Services.
- 3 Expand “local computer,” “Web Sites” and locate the “Default Web Site” entry.



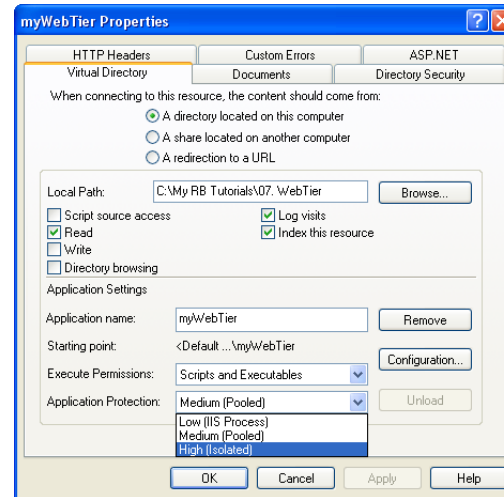
- 4 Right-click “Default Web Site” and select New | Virtual Directory.
- 5 Click Next.
- 6 Enter “myWebTier” as the alias for the directory.
- 7 Click Next.
- 8 Set the directory to:

C:\My RB Tutorials\07. WebTier
- 9 Click Next.



- 10 Check the box labeled “Execute (such as ISAPI applications or CGI).”
- 11 Click Next.

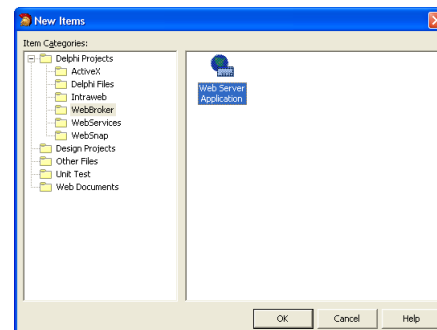
- 12 Click Finish. A new virtual directory will be created and selected.
- 13 Right-click over the new directory and select Properties.
- 14 Locate the “Application Protection” drop-down list at the bottom of the dialog.



- 15 Change the setting from “Medium (Pooled)” to “High (Isolated)”
- 16 Click Apply.
- 17 Click OK to close the dialog.
- 18 Close the Internet Information Services and Administrative Tools windows.

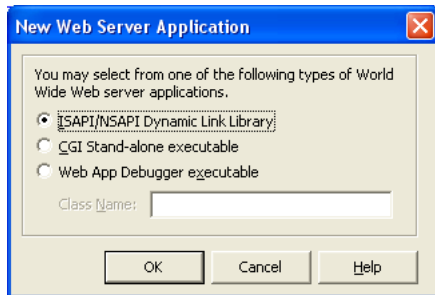
Create an ISAPI DLL

- 1 Launch Delphi.
- 2 Select File | New | Other... The following dialog will be displayed:



3 Scroll to the bottom row of icons and double-click “Web Server Application.”

A dialog will be displayed requesting the type of web application you want to create:



4 Click OK to accept the default. A new web module will be created.

5 Select File | Save Project As... from the Delphi menu.

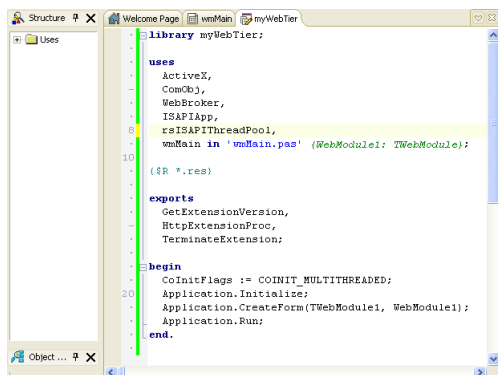
6 Save the web module under the name “wmMain.”

7 Save the project under the name “myWebTier.”


8 Select View | Project Manager from the Delphi Menu.

9 Right-click myWebTier.dll and select View Source.

10 Change “ISAPIThreadPool” to “rsISAP-IThreadPool.” This will allow the ISAPI dll to utilize a more optimized thread pool.



11 Select the RBServer tab on the Delphi component palette.

12 Place an rsWebTier  component in the web module.

13 Configure as follows:

CacheDirectory	C:\My RB Tutorials\ 07.WebTier\Cache\
WebCachePath	http://localhost/ myWebTier/Cache/
WebModuleURI	http://localhost/ myWebTier/myWebTier.dll

Add a Default Action to the WebModule

1 Right-click over the Web Module and select “Action Editor.”

2 Add an Action to the list.

3 Use the Object Inspector to set the Default property of the Action to True.

4 Select the Events tab of the Object Inspector.

5 Double-click the OnAction event and code the following:

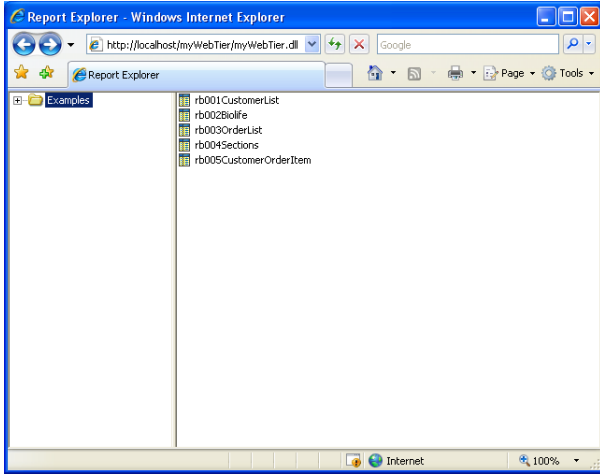
```
Response.Content := rsWebTier1.ProcessWebRequest(
    Request.QueryFields,
    Request.Content);
```

6 Compile the application. You should now see myWebTier.dll in the project directory.

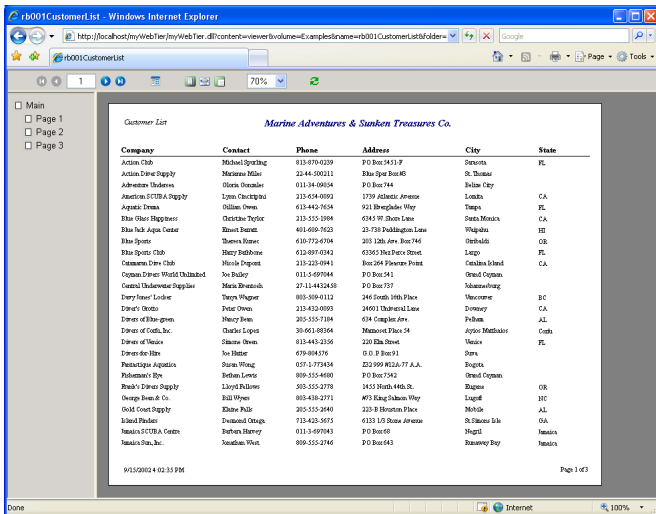
7 Launch Internet Explorer and enter the following address:

http://localhost/myWebTier/myWebTier.dll

8 Press enter. The Report Explorer is displayed:



9 Single click on a report to preview.



Scaling the Web Tier with a Server Farm

Overview

This tutorial will show you how to do the following:

- Use a ServerFarm to increase the number of users which can be supported by a WebTier.
- Use “Round Robin” or “Minimum Load” load balancers to assign servers to client sessions.
- Track which server has been assigned to a given client session.

Prerequisites: This tutorial assumes you have completed at least one of the report server application tutorials, the Windows service tutorial and the “Publish Reports to the Web” tutorial.

Hardware needed: You will need access to at least three machines: Two to be used as report servers and one to be used as the web server/client testing machine. On the first PC (which we will call the Web Server), you’ll need , IIS 5 (or 4), Delphi and ReportBuilder Server Edition installed. The two report server machines need access to the same network as the Web Server machine.

Create a Report Server Application

1 From the Web Server machine, launch Delphi and select File | Open Project... from the main menu.

2 Open the rbServer project in the following directory:

*C:\Program Files\Borland\Delphi6-
\RBServer\Tutorials\Complete-
\01. Build a Report Server Application-
\05. Reports in Archives*

3 Compile the project.

4 Select File | Close All from the Delphi menu.

Copy the Report Server Application to a Network Folder

1 From the Windows Desktop, launch a Windows Explorer.

2 Locate the executable you just created in:

*C:\Program Files\Borland\Delphi6-
\RBServer\Tutorials\Complete-
\01. Build a Report Server Application-
\05. Reports in Archives*

3 Select the following files and copy them to your clipboard:

rbServer.exe
rb001CustomerList.raf
rb002Biolife.raf
rb003OrderList.raf
rb004Sections.raf
rb005CustomerOrderItem.raf

4 Paste these files into a network folder which is accessible from the two report server machines.

5 Copy RBWinService.exe and ReadMe.doc from:

*C:\Program Files\Borland\Delphi6-
\RBServer\Windows Service*

To the same network folder in which you saved the report server application files.

Deploy the Report Server Application

1 Move to the first report server machine. (You'll need to be at the keyboard for this computer.)

2 From the Windows Desktop launch a Windows Explorer.

3 Create the directory:

C:\Report Server

4 Copy the following files from the network folder into this directory:

rbServer.exe
rb001CustomerList.raf
rb002Biolife.raf
rb003OrderList.raf
rb004Sections.raf
rb005CustomerOrderItem.raf

5 Locate RBWinService.exe in the network folder.

6 Run this program, accepting the default entries (this will install the Windows service that hosts the report server application.) After the installation is complete, you should see a new icon on your System Tray:



7 Double-click this icon, to display the Report-Builder Services dialog.

8 Click the Stop button to stop the sample server.

9 Click the Change button and select rbServer.exe in C:\Report Server.

10 Click the Start button; the top section of the dialog should turn green after a second or two, indicating that the server is running.

11 Close the dialog.

12 From the Windows Desktop select Start | Programs | Accessories | Command Prompt.

13 At the command prompt, type "ipconfig" and press enter.

14 Write down the IP address of the machine.

15 Repeat steps 1-14 on the second report server machine. You should now have two active report server machines and the IP address of each.

Check Each Server via Thin Client

1 Return to the Web Server machine.

2 From Delphi, open the project in:

*C:\Program Files\Borland\Delphi6-
\RBServer\Demos\Clients-
\01. Client Explorer*

3 Select the "cbxAddress" combo box.

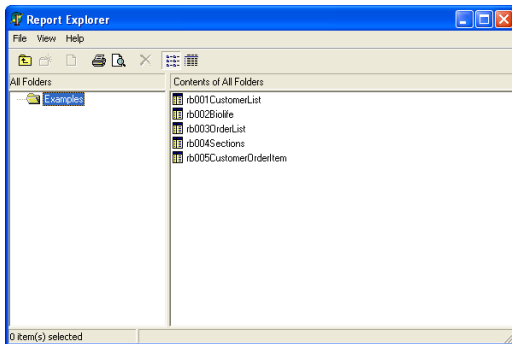
4 Use the Object Inspector to enter the two IP addresses for the server machines in the Items property.

5 Press Ctrl-S to save your changes.

6 Run the application.

7 Select the IP address for the first server and click the "Explorer" button.

8 You should see the main window for the ClientReportExplorer:



9 Close this window and select the IP address for the second server.

10 Click the “Explorer” button; you should see the same ClientReportExplorer window. This confirms that both report servers are up and running.

11 Close the application and return to Delphi.

Configure a Web Tier Using Round Robin

1 From Delphi, open the myWebTier.dpr project in:

```
C:\Program Files\Borland\Delphi6-
\RBServer\Tutorials\Complete-
\03. Publish Reports via a Web Server
```

2 Select View | Project Manager and double-click “wmMain.”

3 Select the WebTier component.

4 Expand the ServerFarmSettings property in the Object Inspector.

5 Launch the property editor for ServerAddresses.

6 Enter the IP address for each report server on a separate line.

7 Close the String List Editor.

8 Check to make sure that the LoadBalancer property is set to “Round Robin.”

9 Press Ctrl-S to save your changes.

10 Select Project | Build All Projects from the Delphi menu.

11 Use the Windows Explorer to copy myWebTier.dll to the myWebTier virtual directory. (If you do not understand this step, please complete the “Publish Reports to the Web” tutorial before continuing.)

Note: If myWebTier.dll will not copy successfully, use Internet Information Services to unload the existing DLL.

Test via Web Browser

1 From the Windows Desktop, launch an Internet Explorer.

2 Enter the following address:

```
http://localhost/myWebTier/myWebTier.dll
```

You should see the ClientReportExplorer (only this time as a JavaScript/HTML application.)

Note: It would be a good idea to add this link as one of your “favorite” links, otherwise you will end up entering it over and over again in the remainder of this tutorial.

3 Preview a few of the reports.

4 Launch another Internet Explorer and preview reports with it as well. When you run the second browser, you’ll actually be running reports on the second server. However, from the browser you can’t really tell; it all looks the same. Let’s track which server is generating the pages for a given browser.

5 Return to Delphi and reopen the “myWebTier” project.

6 Select the WebTier component and expand the ServerFarmSettings property.

7 Set ColorizedPages to True.

8 Recompile and redeploy myWebTier.dll.

9 Launch two new web browsers and preview reports in each. You should notice that the report page color in the first web browser is light red and the report page color in the second web browser is light green. This verifies that the reports have been generated by different servers.

Configure a Web Tier Using Minimum Load

1 Return to Delphi and reopen the “myWebTier” project.

2 Select View | Project Manager and double-click “wmMain.”

3 Select the WebTier component and expand the ServerFarmSettings property.

4 Set the LoadBalancer property to “Minimum Load.”

5 Recompile and redeploy myWebTier.dll.

6 Launch a web browser and begin previewing a report. The report page color should be light red. Return to the Report Explorer and launch another report in the same web browser. Continue previewing different reports for about sixty seconds.

7 Launch another web browser and preview a report. If your activity on the first browser generated enough load, the report page color in this second browser will be light green (indicating that the second server was used to generate the pages.)

8 Experiment with creating load on a given server and then launching a new web browser to see if the other server is used. You should be able to get browsers with reports generated by both servers.