



Advertisement: Support JavaWorld, click here!

**SUN JAVA™ STUDIO CREATOR.**  
GET THE BUNDLE: JAVA STUDIO CREATOR + SUN™  
DEVELOPER NETWORK + SUN PRESS BOOK FOR A GREAT  
LOW PRICE! [BUY NOW AND SAVE >>>](#)



January 2005

[HOME](#)[FEATURED TUTORIALS](#)[COLUMNS](#)[NEWS & REVIEWS](#)[FORUM](#)[JW RESOURCES](#)[ABOUT JW](#)

# A Java proxy for MS SQL Server Reporting Services

Explore the URL interface of a new contender on the enterprise reporting tool market

## Summary

Generating high-quality print output from a Web application is often difficult because browsers provide only limited control over a document's layout. Adjusting page margins or positioning document elements at a precise location on the page proves impossible when printing from the browser. When preprinted forms are being used or when page flow must be controlled tightly, you must use a server-based document generation engine. Many such technologies and products are available. A solution should provide good report development productivity, a GUI report designer, be highly scalable, perform well (especially when rendering PDF documents for concurrent sessions), and come with an attractive price tag. We tested and benchmarked several products and settled for Microsoft SQL Server Reporting Services (RS).

RS can easily be integrated and deployed in a Microsoft-centric environment, but making it accessible to a Java Web application requires some work. This article provides some guidance in integrating RS with your Java Web application. (2,100 words; **January 10, 2005**)

By **Christian Donner** and **Iliia Papas**

A Java Web application often uses a SQL Server database. SQL Server now comes bundled with a reporting tool, Reporting Services. RS has generated a lot of attention among developers since it first came on the market about a year ago. It is a full-featured reporting tool that includes a WYSIWYG report editor with scripting/programming capabilities, a delivery engine, and a powerful Web services interface. Using RS makes sense—not only from a cost perspective, but also because of the reduced development and deployment complexities compared with using a third-party vendor. This article explores such an integration scenario.

RS can be added to any SQL Server installation with no additional license fees (the current distributions of SQL Server 2000 Enterprise, Standard, or Developer Edition already include RS). RS requires that IIS (Internet Information Services) run on the server as well. The Report Developer client is a Visual Studio plug-in and installs as part of the RS client setup. The most recent version is 1.0; however, Microsoft has made available Service Pack 1 for download. Considering that the product is relatively young, it provides a complete and consistent development experience.

**Note:** In this article, the terms "report" and "document" are used interchangeably, assuming that RS is used for the generation of any print output.

## Reporting Services overview

The solution described in this article enables numerous users to generate individual documents; that is, reports are generated exclusively on an ad-hoc basis. This article focuses specifically on ad-hoc generation and delivery of reports via HTTP streams to a Web browser; other approaches are not covered.

Our Web application issues a rendering request to RS after collecting all the necessary input for a report, either by storing control data in the database or by passing parameters. The result in the requested format subsequently appears on the user's screen.

There are two fundamentally different approaches for integrating RS with a Web application.

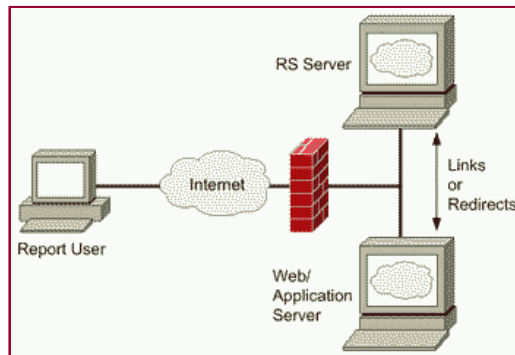
First, users can access the report server directly. Therefore, RS must be exposed to the external network. The application would typically link or redirect to the reports served by the RS server.

Second, RS can be wrapped by the Web application. The report server should not be directly accessible. The Web application issues requests to the RS server and streams report data back to the user.

Let's look closer at these alternatives.

## RS server exposed to users

When users access the RS server directly, the application can simply link or redirect to the reports (see Figure 1). While this scenario at first looks attractive because it is simple, it often results in an undesirable implication.



**Figure 1.** In this configuration, users access RS directly. Click on thumbnail to view full-sized image.

RS supports only Windows Integrated Authentication (formerly called NTLM, or NT Lan Manager). By default, the RS install script configures IIS so that Integrated Authentication is the only allowed option. This means that Internet Explorer automatically authenticates using the current user's Windows credentials, while other browsers cannot achieve access. Once basic authentication is enabled, non-IE browsers will challenge a user (some features of RS do not work with non-IE browsers, though—more about browser limitations later). Whatever form of authentication is used, the user must have access to the report server, either through a domain account or a local account.

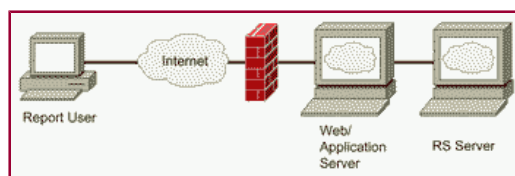
Thus, direct access to RS proves impractical if you must authenticate a large user base that does not have Windows domain accounts. In our case, the Web application uses IBM's Tivoli Access Manager and an LDAP (lightweight directory access protocol) directory to manage user permissions. We were unwilling to develop a custom security extension to RS for this integration. By the way, this extensibility feature is not available in the Standard Edition—it requires a more expensive SQL Server Enterprise Edition license.

Configuring RS for anonymous access is possible, but, for obvious reasons, should be avoided on a publicly accessible server.

Because of these concerns about the limited out-of-the-box ability to directly authenticate users, the direct-access option is not attractive for our purposes.

### RS server wrapped by a Java servlet

When users don't need to access the report server directly, a service account can be used. With a service account, all requests to the report server are made with the same user account, regardless of who is signed on to the Web application. A Java servlet that manages calls to RS and streams report data back to the client eliminates the need for individual user accounts on the RS server. In such a scenario, the Web application must provide a domain name and can then authenticate with IIS even in the default configuration, regardless of the user's browser type.



**Figure 2.** In this configuration, RS is wrapped by the application server. Click on thumbnail to view full-sized image.

RS (i.e., IIS on the RS server) should be configured to run on a nonstandard port blocked to outside users.

We decided to use this approach because the application's large user base of more than 10,000 made authentication through a service account mandatory.

### Report access methods

Up to this point, we have talked about calling RS without detailing how a report can be invoked. Reports can be accessed through three approaches: through Report Manager, through a Web service interface, and through a URL interface.

#### Report Manager

The Web management interface for RS is called Report Manager. It comes with its own (albeit customizable) look and feel, requires users to be authenticated (enabling anonymous access in IIS results in error messages), and does not work properly in Netscape 7 despite that Microsoft claims otherwise in the documentation.

Report Manager provides access to all the features of RS. It can be used to upload and run reports, but because of its administration

features, it is not the right tool for users and cannot be easily integrated with another Web application.

### Web service interface

RS's Web service interface is complete and exposes virtually all of its features. Report Manager itself is implemented as a Web services client. You could access the RS Web service interface from a Java/Apache Axis client if you wish. However, in this article, we focus on the URL interface.

### URL interface

By using the URL interface, it is possible to get a good level of access to RS functions while still using a simple API. Reports are generated by sending an HTTP request to an RS URL. The query string can contain many parameters that detail such items as the report name and the output format. The report server streams the response back to the requestor in the specified format.

A PDF report, for instance, would consist of only one binary stream, while an HTML report would require a separate stream for the page and each embedded image. Because the servlet streams the HTML to the user's browser, which does not maintain a session with the report server, the user would not see such images, but instead receive an authentication error from the report server. The servlet therefore must parse the HTML stream for image references and replace them with the URL of a second servlet that can simulate the original session to RS for requesting the image streams.

## URL interface wrapper servlets

The remainder of this article shows you how to implement a set of servlets that act as a URL interface client to RS.

### ReportRequest.java

The `ReportRequest` is the main servlet that makes the initial report request and processes the report stream. The `ReportRequest` servlet is designed to act as a mediator between the J2EE infrastructure and a Microsoft Reporting Services Web server. It is accessible from the Internet, but the reporting server itself is not exposed for security purposes.

The tasks that this servlet performs include authenticating to the reporting server, passing a form post containing report parameters, and then returning the requested response.

### Authentication

`ReportRequest` authenticates to IIS using NTLM via Java's `Authenticator` class (NTLM requires Java 1.4.2 or later). A custom class called `ReportAuthenticator` extends `java.net.Authenticator` and handles the transmission of the encrypted username and password. To specify the user's realm or domain name, use the following format:

```
DOMAIN\username
```

or:

```
SERVER\username
```

To set the callback object, `ReportRequest` makes the following call:

```
Authenticator.setDefault(new ReportAuthenticator());
```

Once the URL connection is made, the server uses the credentials defined in `ReportAuthenticator` to negotiate authentication.

### Receiving the request

Requests are sent to `ReportRequest` as HTTP form posts containing the report parameters loaded into an `Enumeration`. The servlet calls the `GetAvailableParameters` class, which queries the database and populates an `ArrayList` with the parameters that users are allowed to pass to the report server. The parameters are filtered by this list and parsed into a string. This way, only approved predefined parameters are included in the request. The name of the report being generated is extracted into a separate string to be used later in forming the URL to be posted to the report server.

### Posting to the report server

The next step is to create the connection to the report server using an `URLConnection` to mimic a browser's request. When requesting an HTML report, the report server references auxiliary items such as images by using a stream ID stored in a cookie that is returned to the client in the HTML stream's header. This stream ID identifies the initial report session.

By default, `ReportRequest` posts to the server with the User-Agent header set to *Java 1.4.2*, which the report server believes does not accept cookies and thus disables the stream. To keep the references active, it is necessary to spoof the header using the following line:

```
repCon.setRequestProperty("User-Agent", "Mozilla/5.0");
```

### Forwarding the response to the client

If the requested report is a format other than HTML, all that is required is a buffered binary stream from the `URLConnection`'s input stream to the client response's output stream. Before streaming the data, you must mimic the server stream's content type and disposition headers on the client using the following code:

```
String contentType = repCon.getContentType();
clientResponse.setContentType(contentType);
clientResponse.setHeader("Content-disposition",
    repCon.getHeaderField("Content-disposition"));
```

If the report is in HTML format, the stream must be modified to accommodate any external items, such as images, that may be included in the report. The `ReportRequest` servlet parses the HTML stream character by character, searching for any reference to the reporting server—which will be replaced with a link to the `GetReportItem` servlet—including the cookie originally returned with the stream. So, an image tag that looks like this:

```

```

becomes:

```

```

As each character is processed, it is then passed to the client output stream.

### GetReportItem.java

The `GetReportItem` servlet takes the parameters that would normally be passed back to the report server to retrieve an auxiliary item along with the original cookie provided with the report. `GetReportItem` extracts the query string from the calling URL, removes the cookie parameter, and uses it to set the header of the HTTP GET request it makes to the report server, forwarding the remaining parameters. The cookie tells the report server to which stream the servlet is referring, which is required for the server to pass back the requested object.

The content type and disposition is then set with the same method used in `ReportRequest.java`, and then the data is passed to the client using buffered binary streams.

The following sequence diagram illustrates the interaction between the user's browser and the servlets, and the servlet interaction with RS.

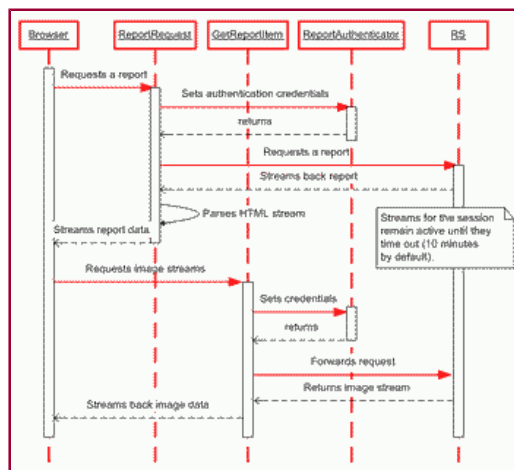



Figure 3. Sequence diagram. Click on thumbnail to view full-sized image.

## Conclusion

Using the `GetReportItem` and `ReportRequest` servlets together successfully allows users to integrate MS SQL Server Reporting Services into a J2EE environment without compromising security policies or exposing the server to the Internet. Also, full reporting functionality is maintained, as the `ReportRequest` servlet can be posted to via a form or the traditional URL method.

Since the main `ReportRequest` servlet also parses out each submitted parameter, implementing additional security checks and restrictions is relatively simple. Unfortunately, at this time, the servlet does not support interactive HTML features such as those found in drill-down reports, but perhaps with further development, this functionality could be added. 

### About the author

[Christian Donner](#) is a senior consultant and software architect at [Molecular](#). His interests include enterprise application and data integration, and strategies for introducing new technologies to large corporations. Donner is a Sun Certified Enterprise Architect for the Java 2 Platform. He currently lives south of Boston.

[Ilija Papas](#) is an associate software engineer at [Molecular](#). He has worked with Web applications for four years, and lives west of Boston.

### Resources

- Download the source code that accompanies this article:  
<http://www.javaworld.com/javaworld/jw-01-2005/sqlrs/jw-0110-sqlrs.zip>
- Download the RS Trial Version and Service Pack 1:  
<http://www.microsoft.com/sql/reporting/downloads/default.asp>
- Also by Christian Donner: "Java and .Net Interoperability Using CORBA" (*JavaWorld*, May 2004):  
<http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-java.net.html>
- For more articles on Java development tools, browse the **Development Tools** section of *JavaWorld's* Topical Index:  
[http://www.javaworld.com/channel\\_content/jw-tools-index.shtml](http://www.javaworld.com/channel_content/jw-tools-index.shtml)
- For more articles on servlets, browse the **Servlets** section of *JavaWorld's* Topical Index:  
[http://www.javaworld.com/channel\\_content/jw-servlets-index.shtml](http://www.javaworld.com/channel_content/jw-servlets-index.shtml)



Advertisement: Support JavaWorld, click here!

<b>THE SHARPER IMAGE</b>	<b>Special Offer: 50% OFF</b> with any other purchase <a href="#">shop now</a>	
----------------------------------	--	--

[HOME](#) | [FEATURED TUTORIALS](#) | [COLUMNS](#) | [NEWS & REVIEWS](#) | [FORUM](#) | [JW RESOURCES](#) | [ABOUT JW](#) | [FEEDBACK](#)

Copyright © 2004 JavaWorld.com, an IDG company