Business Objects

Crystal Reports XI Release 2

Java Reporting Component Deployment Guide

Overview

This document discusses how to deploy the Crystal Reports XI Release 2 Java Reporting Component in web and desktop (thick-client) environments.

Contents

Introduction

The Java Reporting Component (JRC) in Crystal Reports XI Release 2 supports the ability to deploy applications in web and desktop environments. The JRC also includes APIs for server-side exporting and printing of reports, as well as for accessing information about a loaded report such as parameters and database information.

It is possible to deploy the JRC in one of two ways:

1. Embedding the JRC into a desktop Java application. This method requires deployment of the JRC runtimes and thick-client Swing viewer libraries in a desktop application, which is installed on client computers.

2. Embedding the JRC into a J2EE web application. This method involves the deployment of the JRC runtimes with the zero-client DHTML Viewer in a web application archive (WAR) file.

This document first discusses the common deployment steps between these methods before discussing the specifics of each method. Before proceeding, review the Crystal Reports XI Release 2 supported platforms document that contains information on the supported application servers, operating systems, and database drivers for the JRC. The supported platforms document (platforms.txt) is available on the Crystal Reports XI Release 2 Developer Edition CD.

Background

The JRC is a 100% pure Java reporting engine that allows developers to embed reporting functionality directly into desktop and web applications. The JRC is best suited for delivering simple on demand reports to small departments or workgroups. For serving reports to larger user bases over the web with security, and scalability and fault tolerance, the Crystal Reports Server or BusinessObjects Enterprise Java SDK is recommended.

Common deployment steps

This section discusses the common deployment steps for embedding the JRC into a desktop or web application.

IMPORTANT	▪ For desktop Java applications, the CLASSPATH refers to the application's CLASSPATH. ▪ For web applications, the CLASSPATH refers to the /WEB-INF/lib and /WEB-INF/classes folder of the application.

Including required run-time libraries

The first step when embedding the JRC into a desktop application is including the required run-time libraries.

Core libraries

Table 1 lists the required JAR files with their corresponding source folders. Include these files in your application's CLASSPATH.

Table 1 – Core Library JAR Files	
Source Folder	**JAR File**
\Program Files\Business Objects\common\3.5\java\lib folder	CrystalCommon.jar CrystalDatabaseConnectors.jar CrystalReportingCommon.jar CrystalContentModels.jar CrystalFormulas.jar CrystalQueryEngine.jar CrystalReportEngine.jar keycodeDecoder.jar MetafileRenderer.jar rpoifs.jar jrcerom.jar jrcadapter.jar rascore.jar rasapp.jar serialization.jar
\Program Files\Business Objects\common\3.5\java\lib\external	log4j.jar icu4j.jar Concurrent.jar xercesImpl.jar xml-apis.jar

Charting, exporting, and report viewer libraries

If your application or report requires charting, exporting, or report viewing functionality, add these JAR files to your CLASSPATH:

- ReportPrinter.jar (required for server-side printing)
- CrystalCharting.jar
- CrystalExportingBase.jar and CrystalExporters.jar (these files are required for any exporting, including exporting from the JRC viewers or server-side exporting)

Database driver libraries

Depending on the type of database connectivity your report is using, other JAR files may be required in your application's CLASSPATH. For example, if your report connects to an Oracle database using JDBC, include the file ojdbc14.jar in the CLASSPATH.

Libraries for native XML-based reports

If your application includes a report that uses the native XML driver, include the JAR files listed in Table 2.

Table 2 – Native XML Driver JAR Files

Source Folder	JAR Files
\Program Files\Business Objects\common\3.5\java\lib\	CRDBJavaServerCommon.jar CRDBXMLServer.jar
\Program Files\Business Objects\common\3.5\java\lib\external	CRDBXMLExternal.jar axis-ant.jar axis.jar commons-discovery.jar commons-logging.jar jaxrpc.jar log4j-1.2.8.jar pullparser.jar saaj.jar wsdl4.jar xbean.jar

Including configuration files

Once the required libraries have been included in the application's CLASSPATH, the next step is to include the CRConfig.xml and log4j.properties files in the CLASSPATH. These files are located in the \Program Files\Business Objects\common\3.5\java folder.

Configuring CRConfig.xml

The CRConfig.xml file contains a number of configuration tags for the JRC that may need to be set for your application. For more detailed information on the properties in the CRConfig.xml file for Crystal Reports XI Release 2, refer to Finding more information.

<Keycode> tag

Set the value of the key code tag with a Crystal Reports XI or XI Release 2 Developer Edition key code to enable the JRC to process reports. If you attempt to use a key code from the Standard or Professional editions of Crystal Reports XI or earlier, an "exceeded license count" error will be thrown when you run your JRC application as per knowledge base article c2017244.

<ReportLocation> tag

The report location tag specifies the location of the Crystal Report (.rpt) files. This location is a relative path to the location of the JRC engine libraries. By default, the value of this tag is set to "../..". That is, the JRC searches for reports in the folder that is two physical directory levels above the JRC engine libraries folder. Table 3 provides an example.

Table 3 - <ReportLocation> Tag Configuration Example

Location of JRC Engine Libraries	Value of <Report Location> tag	Report Search Folder
\myapp\resources\lib	../..	\myapp
\myapp\resources\lib	..\reports	\myapp\resources\reports

When using the JRC to open the report file with this method only the report file name should be specified. For example:

```
ReportClientDocument.open("myreport.rpt", 0);
```

Alternatively, the report location tag can be commented out or removed from the CRConfig.xml, allowing the JRC to open the report files using as Java resources, or reports can be opened using an absolute path:

If the reports are to be opened as Java resources, then place the .rpt files in a folder that is listed in the CLASSPATH or place the .rpt files in a JAR file and then add the JAR file to the CLASSPATH.

In this case, when using the JRC to open a report as a Java resource, only the report file name should be used. For example:

```
ReportClientDocument.open("myreport.rpt", 0);
```

If the reports are to be opened using absolute paths, place the .rpt files in the desired folder, and when opening the report in the JRC, use the full path to the .rpt file. For example:

```
ReportClientDocument.open("C:\\reports\\myreport.rpt"
, 0);
```

<Timeout> tag

The timeout tag specifies the number of minutes for which a report source object can be inactive until its associated temporary files, database connections, and other resources objects and files are released by the JRC. The default value is 10 minutes.

This timeout can be particularly useful in web applications to ensure that a report is released after a period of inactivity, and that an inactive report source and other objects and files are cleaned up on the server. If the report source has been released by the JRC in a web application, the error "*The Report Source Has Expired*" is thrown by the viewer.

For desktop applications, if a user attempts to navigate to the next page of a report, export the report, or interact with the report in any way after the specified timeout period has elapsed, then the error "*Due to inactivity, the report document timed out and has already been closed*" will be thrown.

If a desktop application must maintain a report source for an indefinite amount of time, the timeout tag can be set to 0 minutes, in which case, the report source is never automatically released by the JRC engine. However, you will need to ensure that the report source is released if the report is no longer in use by calling the **close** method:

```
ReportClientDocument.close();
```

It is good practice to use this method when the report is no longer needed, but essential when the timeout tag has been set to 0 minutes.

Configuring Log4.properties

The log4j.properties file shows how to use the Log4j logging system for troubleshooting and error detection. The JRC uses Log4j to capture runtime information from the JRC engine to help troubleshoot JRC errors.

In order for the log4j loggers and appender configuration options to be read by the JRC, add the folder containing the log4j.properties file to the application's CLASSPATH. Typically for web applications, the log4j.properties file is included in the \WEB-INF\classes folder. For desktop applications, ensure that the folder containing the log4j.properties is included in the application's CLASSPATH.

The log4j.properties configuration file located in the \Program Files\Business Objects\common\3.5\java folder provides a sample of how to enable various log4j logging options for the JRC engine. Of particular interest are the options for configuring the location and name of the log file:

```
log4j.appender.fileAppender.file=c:\\temp\\logfile.log
```

Also, there is an option to limit or expand the amount of logging information written. For example, the following sets JRC print engine messages to the 'DEBUG' level and writes them to the Console:

```
log4j.logger.com.crystaldecisions.reports.printengine
=DEBUG, MyConsoleAppender
```

NOTE	**MyConsoleAppender** in this example should be already defined. Refer to the log4j documentation in the Finding more information section for more information on creating appenders in log4j.

The main Crystal JRC loggers are listed below:

- com.crystaldecisions.reports
- com.crystaldecisions.threedg
- com.crystaldecisions.common
- com.businessobjects.reports.sdk.JRCCommunicationAdapter

For information on using log4j, refer to Finding more information.

Additional steps for Java desktop applications

This section discusses additional required steps specific to deploying the JRC in a desktop application.

One of the major new enhancements to the JRC in Crystal Reports XI Release 2 is the ability to embed it in a desktop application. In particular, with the thick-client ReportViewerBean viewer, you can embed report viewing functionality into a Swing application.

To use the new thick-client Java viewer into a desktop application, include the ReportViewer.jar file at the beginning of your classpath (as per knowledge base article c2018313). For more information on getting started with ReportViewerBean in a desktop Java application, refer to Finding more information.

Additional steps for J2EE web applications

This section discusses additional required steps specific to deploying the JRC in a J2EE web application.

Since the initial release of the JRC, it has been possible to embed simple report viewing and exporting functionality into a J2EE web application. The JRC is ideal for embedding small scale report processing for viewing and exporting reports in a web application. For serving larger or more complex reports to a wide user base, use the Crystal Reports Server or BusinessObjects Enterprise Java SDK.

For quick start instructions on embedding the JRC into a web application archive (WAR) file, refer to the following knowledge base article:

http://support.businessobjects.com/library/kbase/articles/c2018318

The following sections further discuss the deployment instructions in this knowledge base article.

DHTML Viewer libraries

The following libraries are required to view reports in the zero-client DHTML Viewer or to export using the **ReportExportControl** method:

- webreporting.jar (required for the DHTML **CrystalReportViewer** or **ReportExportControl** method)
- webreporting-jsf.jar (required for the Java Server Faces (JSF) DHTML Viewer)

crystalreportviewers115 folder

The crystalreportviewers115 folder contains utility files for the DHTML Viewer including the toolbar button images, and the ActiveX Print Control that can be enabled in the DHTML Viewer for client-side printing. For J2EE web applications that use the DHTML Viewer SDK, include the crystalreportviewers115 folder as follows:

1. Copy the crystalreportviewers115 folder from the …\Program Files\Business Objects\common\3.5\ to the root of the WAR file. Table 4 lists of the subfolders included under the crystalreportviewers115 folder and their descriptions.

Table 4 – Contents of the crystalreportviewers115 Folder

Directory	Description
root	Contains crystalimagehandler.jsp that obtains charts and other images generated for the DHTML Viewer.
/ActiveXControls	Contains the ActiveX Print Control that can be enabled for client-side printing from the zero-client DHTML Viewer. Also contains the ActiveX Viewer component (not supported by JRC).
/css	Contains style sheets for DHTML Viewer. The CSS can be modified to provide a different look and feel for the DHTML Viewer.
/html	Contains HTML utility files for the DHTML viewer's parameter prompt, print, and export

Table 4 – Contents of the crystalreportviewers115 Folder	
	functionality.
/images	DHTML Viewer toolbar and control images. The images can be modified to provide a different look and feel for the DHTML Viewer.
/js	JavaScript utility files for the DHTML Viewer controls (modification not supported).
/prompting	Utility files for the DHTML Viewer's parameter prompt page (modification not supported).

2. The next step is to add the crystal_image_uri context parameter to the web.xml file to enable the DHTML Viewer to obtain its utility and resource files from the crystalreportviewers115 folder. If this context parameter value is not set or incorrectly set, the DHTML viewer's toolbar images will not appear and graphical objects embedded into the report will not be rendered.

   To add the context parameter, insert the following code below the <display-name> and <description> tags but within the <webapp> tags in your web.xml file:

```
<!-- Context Param -->
<context-param>
<param-name>crystal_image_uri</param-name>
<param-value>crystalreportviewers115</param-value>
</context-param>
<!-- Context Param End -->
```

   The value of the **crystal_image_uri** parameter is relative to the URI of the servlet, or JSP file that displays a report by calling the **CrystalReportViewer.processHttpReques**t method.

   In the above example, the **crystalreportviewers115** directory will need to be located at the same folder level in the web application as the servlet or JSP file that calls the **processHttpRequest** method.

   If the relative URI locations differ, for example, if the crystalreportviewers115 directory is located in the root of the web application but the JSP files that invoke the DHTML Viewer's **processHttpRequest** method are located in various subdirectories of the web application context, then pre-append a slash to the crystal_image_uri context parameter value:

```
…

<param-value>\crystalreportviewers115</param-value>
…
```

Furthermore, if needed, you can specify a particular web application context for the URI:

…

```
<param-value>\mywebapp\crystalreportviewers115</param-value>
```

…

NOTE	The URI value may differ between J2EE application servers. To display the crystal_image_uri value, right-click a red 'X' that appears in place of an image object (a toolbar image or graphical report objects such as a chart or static picture), and then click **Properties**.

Finding more information

JRC XI Release 2 Developer Guide and API Reference

Go to Start > Programs > BusinessObjects XI Release 2 > Crystal Reports > Java Developer Documentation

JRC XI Release 2 desktop samples

http://support.businessobjects.com/communityCS/FilesAndUpdates/crxi_r2_jrc_desktop_samples.zip.asp

JRC XI Release 2 web samples

http://support.businessobjects.com/communityCS/FilesAndUpdates/crxi_r2_jrc_web_samples.zip.asp

Crystal Reports XI Release 2 CRConfig.xml configuration

http://support.businessobjects.com/communityCS/TechnicalPapers/cr_xir2_crconfigxml.pdf.asp

Apache log4j

http://logging.apache.org/log4j/docs/