Crystal Reports XI Release 2

Migrating from the RDC to the JRC

Overview

This technical document discusses how to migrate your Report Designer Component (RDC) 8.5 application to the Java Reporting Component (JRC) in Crystal Reports XI Release 2.

This document includes a migration example, considerations for moving existing RDC applications to J2EE, and code samples that describe how various important scenarios implemented with the RDC are now implemented with JRC.

Contents

Introduction

The purpose of this technical document is to discuss the benefits of using the Java Reporting Component (JRC) for integrating reporting functionality into web and desktop applications, as well as to help you migrate applications from the Report Designer Component (RDC) to the JRC.

This document includes a migration example, considerations for moving existing RDC applications to J2EE, and code samples that describe how various important scenarios implemented with the RDC are now implemented with JRC.

History

Crystal Reports has been a market-leading reporting tool since it was first included with Microsoft Visual Basic, with millions of licenses shipped. It has kept pace with technological innovation by giving Visual Basic developers new ways to integrate reporting into their applications. In June 1998, Crystal Reports was launched with the Report Designer Component (RDC), a tool designed specifically to create, view, and modify reports within the Visual Basic integrated development environment (IDE).

Benefits of upgrading to Crystal Reports

The Developer Edition of Crystal Reports XI Release 2 is a feature rich upgrade that offers web reporting, dynamic cascading prompting, dynamic image location, customizable report preview, improved data connectivity, easier deployment, and additional export formats. Reports are now easier to create and require less code to expose these features in an application.

Crystal Reports XI Release 2 Developer Edition provides the following benefits:

- **Java Server Faces DHTML Report Page Viewer.** The Java Server Faces (JSF) DHTML Report Page Viewer enables you to easily integrate a Crystal Reports viewer into any J2EE web application that is implemented using the JSF framework. The new viewer provides the same level of functionality as the DHTML Report Page Viewer, and is supported for use with the JRC, Report Application Server, and Page Server. It includes a custom set of JSF tags to add the viewer into Java Server Pages (JSP).
- **Report Parts Viewer support.** The JRC supports the Report Parts Viewer. You can view individual parts of a report, such as chart, text, or field objects.
- **Crystal Reports export format.** With the JRC, you can export your Crystal reports to the .rpt file format. You can also save a copy of

your reports locally by using the export function in the Crystal Reports Viewer.

- **XML data source support.** The JRC supports the new XML data driver for reporting from web services and XML files.
- **Java User Function Libraries.** Java User Function Libraries (UFLs) allows you to extend the already rich set of functions available in the Formula Workshop (located within the Crystal Reports Designer). With the Crystal Reports Designer, you can add these custom user functions into reports. Java UFLs are supported by the JRC and in the Crystal Reports Designer.
- **Custom tag library.** Reduces the amount of code required to embed report templates into your JSP files.
- **Cross-platform support.** The JRC takes full advantage of Java, including portability across operating systems and hardware platforms.

In addition, Crystal Reports XI Release 2 provides the following new features:

- **Dynamic and cascading prompts.** Now that report prompts can be based on dynamic values, you no longer have to maintain static prompt value lists within individual reports. Instead, you can reuse existing prompts stored in the repository.
- **HTML preview.** The iterative report design/view process is streamlined, with a new HTML preview that allows you to see how reports will look when published to the web.
- **Dynamic image location.** Pictures and graphics can now be placed in a report through a link in a database so that it is no longer necessary to store images in the database. This supports the common practice of storing images on the web server and storing references to those images in the database.
- **Editable RTF format.** This new feature is ideal for report export editing. It delivers reports to end users in a new editable RTF format to enable them to make their own modifications.
- **Intelligent charting.** New drag and drop charting and cross-tabs allow for intelligent charting. Variables are approximated when a chart is dropped into a section. Chart design is now faster and easier because charts are updated automatically when new variables are added.
- **Updated drivers.** Crystal Reports XML, JDBC, IBM DB2, and Exchange data drivers have been updated to give you an incredible range of data access options.
- **Additional export formats.** Reports can be exported to a number of popular formats such as Adobe Acrobat (.pdf), rich text format (.rtf), or character-separated values (.csv). This makes the distribution of information easier and allows you to manipulate report data with familiar tools.

Example of porting an RDC application to J2EE

This section discusses the process of porting a simple RDC application to a J2EE web application. A brief description of the sample application is provided, followed by a description of the RDC implementation. Next, the steps required to port this application to J2EE are detailed. Finally, the resulting J2EE implementation is examined.

General description of the example application

A report with a subreport is created using the sample xtreme.mdb database. The main report contains the Customer table and a parameter field. The subreport contains the Orders table and a formula field.

The application consists of two forms. The first form, frmMain, loads the report and provides three command buttons to preview, print, and export the report. The second form, frmPreview, contains the Report Viewer for previewing the report.

Below is a summary of the forms:

frmMain

Form Load

- The report is opened.
- The location of the database in the main report is changed.
- The parameter in the main report is set.
- The subreport is opened.
- The location of the database in the subreport is changed.
- A string is passed to the formula field in the subreport.

Command1

- The report is previewed on the screen.

Command2

- The printer is selected.
- The report is printed.

Command3

- The export options are set to export the report to rich text format.
- The report is exported.

frmPreview

Report View

- The source of the Report Viewer is set to the main report.

- The report is viewed.

Form Resize

- The Report Viewer is resized to the dimensions of the preview form.

Report Designer Component application

This section examines the Visual Basic 6 application that uses the Crystal Reports 8.5 RDC.

frmMain

```
'Declare the application object used to open the rpt file
Dim crxApplication As New CRAXDRT.Application
'Declare the report object
Public Report As CRAXDRT.Report

Private Sub Form_Load()

'Declare a DatabaseTable Object
Dim crxDatabaseTable As CRAXDRT.DatabaseTable
'Declare a Report object to set to the subreport
Dim crxSubreport As CRAXDRT.Report

'Open the report
Set Report = crxApplication.OpenReport _
  (App.Path & "\RDC_To_CR.Net.rpt", 1)

'Use a For Each Loop to change the location of each
'DatabaseTable in the Reports DatabaseTable Collection
For Each crxDatabaseTable In Report.Database.Tables
  crxDatabaseTable.Location = App.Path & "\xtreme.mdb"
Next crxDatabaseTable

'Pass the Parameter value to the first parameter field in
'the ParameterFields collection of the Report
Report.ParameterFields.Item(1) _
  .AddCurrentValue "Main Report Parameter"

'Set crxSubreport to the subreport 'Orders' of the main
'report. The subreport name needs to be known
'to use this method.
Set crxSubreport = Report.OpenSubreport("Orders")
```

```
'Use a For Each loop to change the location of each
'DatabaseTable in the Subreport Database Table Collection
For Each crxDatabaseTable In crxSubreport.Database.Tables
  crxDatabaseTable.Location = App.Path & "\xtreme.mdb"
Next crxDatabaseTable

'Pass the formula's text to the first formula field
'in the FormulaFields collection of the subreport
crxSubreport.FormulaFields.Item(1).Text = _
  "'Subreport Formula'"

End Sub

Private Sub cmdPreview_Click()
  'Call frmPreview to preview the Report
  frmPreview.Show
End Sub

Private Sub cmdPrint_Click()
  'Select the printer for the report passing the
  'Printer dRiver, Printer Name and Printer Port
  Report.SelectPrinter "HPPCL5MS.DRV", _
    "HP LaserJet 4m Plus", "\\Vanprt\v1-1mp1s-ts"
  'Print the Report without prompting user.
  Report.PrintOut False
End Sub

Private Sub cmdExport_Click()

'Set the report to be exported to Rich Text Format
Report.ExportOptions.FormatType = crEFTExactRichText

'Set the destination type to disk
Report.ExportOptions.DestinationType = crEDTDiskFile

'Set the path and name of the exported document
Report.ExportOptions.DiskFileName = _
  App.Path & "\RDCExport.rtf"

'Export the report without prompting the user
```

```
Report.Export False

End Sub

Private Sub Form_Unload(Cancel As Integer)
  Set crxApplication = Nothing
  Set Report = Nothing
End Sub
```

frmPreview

```
Private Sub Form_Load()

  'Set the report source for the Report Viewer
  CRViewer1.ReportSource = frmMain.Report

  'View the Report
  CRViewer1.ViewReport

End Sub

Private Sub Form_Resize()
  'This code resizes the report Viewer control to
  'frmPreview's dimensions
  CRViewer1.Top = 0
  CRViewer1.Left = 0
  CRViewer1.Height = ScaleHeight
  CRViewer1.Width = ScaleWidth
End Sub
```

Java Reporting Component application

Porting the Visual Basic 6 RDC application to the JRC involves the following:

1. Creating a JSP file for each form.
2. Porting each RDC method to the JRC equivalent.

Creating a Java Server Pages file for each form

A JSP file is created to represent each form in the RDC application. Index.jsp and previewReport.jsp are created to represent frmMain and frmPreview, respectively. The file exportReport.jsp contains the logic to export a report. In general, a JSP file must be created for each Visual Basic form.

Porting each method to the Java Reporting Component equivalent

A J2EE application controls a report's data through the JRC. The same general programming concepts used to design programs with the RDC also apply to the JRC. Programming language differences will also be encountered as you port from Visual Basic to Java.

The approach to porting the application is to translate each VB statement into its Java equivalent line by line as you migrate each VB form to a JSP equivalent.

Identifying differences between the applications

The major differences between the two applications include the following areas:

- **OS Support:** Windows only versus platform independence
- **Technology:** Microsoft COM versus Java
- **Delivery Mode:** A desktop application versus a web application
- **Object Model Differences:** RDC versus JRC
- **Presentation Layer:** Visual Basic forms versus JSP files

J2EE application

The following code is a ported J2EE application of the RDC application:

index.jsp

```
<%
//Define report here
String reportName = "RDC_to_JRC.rpt";

try
{
  ReportClientDocument oReportClientDocument =
    new ReportClientDocument();

  //Open report.
  oReportClientDocument.open(reportName, 0);

  //Get the report source
  Object reportSource =
    oReportClientDocument.getReportSource();

  //Cache report source.
  session.setAttribute("reportSource", reportSource);
}
catch (ReportSDKException e)
{
  out.print(e);
}
%>

<html>
  <head>
    <title>Preview Export Sample</title>
  </head>
  <body>
    <input type="button" value="Preview"
           onClick="location.href='previewReport.jsp'">
    <input type="button" value="Export"
           onClick="location.href='exportReport.jsp'">
  </body>
</html>
```

previewReport.jsp

```
<%
  //Get the IReportSource object from session and
  //pass to the viewer
  IReportSource reportSource =
    (IReportSource) session.getAttribute("reportSource");

  //create the CrystalReportViewer object
  CrystalReportViewer oCrystalReportViewer =
    new CrystalReportViewer();

  //set the reportSource property of the viewer
  oCrystalReportViewer.setReportSource(reportSource);
  //set viewer attributes
  oCrystalReportViewer.setOwnPage(true);
  oCrystalReportViewer.setOwnForm(true);

  //set the CrystalReportViewer print mode
  //oCrystalReportViewer.setPrintMode(CrPrintMode.ACTIVEX);
  oCrystalReportViewer.setPrintMode(CrPrintMode.PDF);
  //process the report
  oCrystalReportViewer.processHttpRequest
    (request, response,
     getServletConfig().getServletContext(), null);
%>
```

exportReport.jsp

```
<%
  //Get the IReportSource object from session and
  //pass to the viewer
  IReportSource reportSource =
    (IReportSource) session.getAttribute("reportSource");

  //Set export options to export in the format of choice.
  ExportOptions oExportOptions = new ExportOptions();
  oExportOptions.setExportFormatType
    (ReportExportFormat.RTF);
  //oExportOptions.setExportFormatType
  //  (ReportExportFormat.crystalReports);
  //oExportOptions.setExportFormatType
  //  (ReportExportFormat.PDF);

  ReportExportControl oReportExportControl =
    new ReportExportControl();

  //Pass the report source to ReportExportControl and
  //set ExportOptions
  oReportExportControl.setReportSource(reportSource);
  oReportExportControl.setExportOptions(oExportOptions);

  //ExportAsAtttachment(true) prompts for open or save;
  //false opens report in specified format after exporting.
  oReportExportControl.setExportAsAttachment(true);

  //Export the report
  oReportExportControl.processHttpRequest
    (request, response,
     getServletConfig().getServletContext(), null);
%>
```

Configuration of the J2EE application for deployment

To configure the web application for deployment on a J2EE web application server, follow these steps:

1. Create a lib folder in the web application or WAR file's WEB-INF folder.
2. Copy the JAR files from these folders to the application or WAR file's WEB-INF\lib folder:
3. Copy the JAR files from these folders to the WEB-INF\lib folder:
   - ...\Program Files\Common Files\Business Objects\3.0\java\lib
   - ...\Program Files\Common Files\Business Objects\3.0\java\lib\external
4. Create a classes folder in the web application or WAR file's WEB-INF folder.
5. Copy the CRconfig.xml file from ...\Program Files\Common Files\Business Objects\3.0\java to the WEB-INF\classes folder.
6. Copy the report file RDC_to_JRC.rpt to the root folder of the web application or WAR file.
7. Copy the entire crystalreportviewers115 folder from ...\Program Files\Common Files\Business Objects\3.0 to the application or WAR file's root folder.

The web application or WAR file is now ready for deployment on a J2EE web application server.

Configuration of the report data source

The data source for the report comes from the file xtreme.mdb. To set up an ODBC connection to this database, complete these steps:

1. Go to Control Panel > Administrative Tools > Data Sources (ODBC).
2. Click the **System DNS** tab.
3. Click the **Add** button.
4. Click **Microsoft Access Driver (*.mdb)** and then click **Finish**.
5. Type "Xtreme Sample Database" in the **Data Source Name** box.
6. Click the **Select** button and browse to the xtreme.mdb file.
7. Click **OK** to finish.

A connection is created for the JRC to connect to and query the database for report data.

Summary

The previous sections demonstrate how to port a simple RDC application to J2EE. Porting the example application required creating a JSP for each Visual Basic form, porting the code from Visual Basic to Java, and using the JRC API. The steps required to configure the web application for deployment on a web application server were also listed.

Key considerations when migrating to J2EE

Applications built using the RDC are COM-based and designed for the Windows platform only. Migrating to JRC provides a wider range of deployment platforms without additional coding. This section discusses other key considerations when migrating to a Java-based Crystal Reports solution.

IDE integration

Crystal Reports XI offers integration into a number of popular Java integrated development environments (IDEs): IBM Rational Application Developer, Borland JBuilder, and BEA WebLogic Workshop. The Crystal Reports Designer (the XI IDE integration) greatly simplifies the process of accessing, formatting, and integrating dynamic data within Java Server Pages (JSPs). With the Crystal Reports Designer built into the IDE, you can create or edit reports without having to leave the development environment.

Within the Java IDE, you can easily add new or existing report files to your projects as resources. This enables you to manage these reports as part of a project and deploy with a web application. Built-in wizards guide you through the process, eliminating much of the tedious work required to hand-code similar functionality.

Application configuration changes

In order to develop and deploy web applications that use the JRC, the appropriate Crystal Reports XI Release 2 JAR files must be made available to your web application.

When you are developing a web application, the JAR files listed below must be added to the project classpath. These files must also be added to the web application's WEB-INF\lib folder when deploying the web application.

- cecore.jar
- celib.jar
- CorbaIDL.jar
- ebus405.jar
- CrystalCharting.jar

- CrystalContentModels.jar
- CrystalDatabaseConnectors.jar
- CrystalExporters.jar
- CrystalExportingBase.jar
- CrystalFormulas.jar
- CrystalQueryEngine.jar
- CrystalReportEngine.jar
- CrystalReportingCommon.jar
- jrcadapter.jar
- jrcerom.jar
- rpoifs.jar
- CrystalCommon.jar
- keycodeDecoder.jar
- MetafileRenderer.jar
- rascore.jar
- rasapp.jar
- ReportViewer.jar
- ReportPrinter.jar
- serialization.jar
- Concurrent.jar
- icu4j.jar
- log4j.jar
- URIUtil.jar
- saaj.jar
- pullparser.jar
- jaxrpc.jar
- commons-logging.jar
- commons-discovery.jar
- axis.jar
- axis-ant.jar
- CRDBXMLExternal.jar
- CRDBXMLServer.jar
- CRDBJavaServerCommon.jar
- wsdl4j.jar

- xbean.jar
- xercesImpl.jar
- xml-apis.jar

The following files are required for viewing and exporting functionality:

- webreporting.jar
- webreporting-jsf.jar (provides the JSF version of the DHTML Report Page Viewer)
- crystalreportviewers115 folder

Security-related changes

A key benefit of migrating from an RDC COM-based application to a Java version is enhanced security. The Java platform provides tools for developers to implement security at the application level, including policy-based access controls, tools for creating and managing security keys and certificates, as well as Java Authentication and Authorization Service (JAAS), Java Cryptography Extension (JCE), and Java Secure Socket Extension (JSSE). You also have access to the security measures provided by Crystal Reports XI for Java, including support for single sign-on (SSO) plus database- and session-based authentication for data source access. With Crystal Reports XI's new update feature, you will automatically receive the latest hot fixes and service packs to address any security vulnerabilities.

Data access

The JRC can natively use a variety of data sources to create reports — XML data files, result sets created from a collection of plain old Java objects (POJOs), or a JDBC connection to a data source. Any JDBC data source can be used as a source for reporting, thus making a range of databases available, including MySQL, PostgreSQL, Oracle, and IBM DB2.

The wide range of database connectivity allows you to connect to almost any data to support legacy applications, providing the flexibility to choose the data source and data connectivity technology that best fits each application.

Java User Function Libraries (UFL)

You now have the ability to construct cross-platform Java User Function Libraries to be used in Crystal Reports XI and reports deployed with the JRC. The details for creating, compiling, and deploying the Java UFLs, as well as configuring Crystal Reports XI to make use of such libraries, are included in the document *Crystal Reports XI - Java User-Defined Function Libraries (UFL)*:

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

If you are migrating existing RDC applications over to the JRC, the custom COM-based UFLs have to be re-implemented based on the syntax and structure defined by the JRC SDK. For more details, refer to the JRC SDK.

Licensing changes

The RDC and JRC components in Crystal Reports XI make use of a concurrent processing license (CPL) model. Each component is set to three CPLs per process, meaning each component allows a maximum of three actions to be processed concurrently. Additional requests are queued until one of the three requests inside the engine completes.

As a general rule, if you anticipate your application regularly serving five or more concurrent users, you should consider using a server solution provided by Crystal Reports Server or BusinessObjects Enterprise. For high traffic applications, these server-based products offer more flexibility in terms of optimizing the deployment for performance.

More information on licensing is available at this location:

http://www.businessobjects.com/products/reporting/crystalreports/licensing/details.asp.

Migration considerations

The JRC provides the essential features needed to manipulate a report's data, however, it only provides a subset of all the RDC functionality. If the JRC does not offer a particular feature, an alternative is to use the Crystal Reports Server or BusinessObjects Enterprise Java SDK.

A detailed listing of each RDC feature absent in the JRC is found in the Appendix. These differences are discussed below.

Database connectivity

The JRC can access data through a JDBC driver, an XML data file, or a resultset of plain old Java objects (POJO). It is unable to access data sources using Microsoft Data Access Components (MDAC): ADO, OLE DB and OLE DB Providers, and ODBC.

NOTE	While it is possible to access ODBC data sources from Java through the JDBC-ODBC bridge driver, it is recommended that you make use of JDBC drivers where possible.

Exporting

The JRC can export a report or subreport as an Adobe Acrobat (.pdf), Crystal Reports (.rpt), rich text format (.rtf), character-separated values (.csv), or plain text (.txt) files. Other export destinations and file formats are not supported.

Report Object common properties

The JRC only supports a **Field** report object. The JRC is able to access the object's name and properties, as well as edit the report object's name, which is not possible with the RDC. Other common report object properties, including **Left**, **Top**, **Width**, **Height**, and **Border**, are not supported by the JRC.

Parameter fields

With the exception of creating a new parameter field and accessing the **HasCurrentValue** property, the JRC has a comparable set of functionalities as its RDC counterpart. Functionalities in the JRC include the ability to access and modify the **BrowseField**, **ParameterFileName**, **ParameterType**, and **ParameterValueKind** properties.

Drill-down events

RDC can listen to **Map** object drill-down events in order to trigger other actions. Although this feature is not included in the JRC, all other RDC drill-down features are supported. JRC also supports report parts, which is not supported by the RDC.

Other Report Designer Component features

The remaining RDC features listed below are not supported by the JRC (a detailed comparison of RDC and JRC features is found in the Appendix):

- BlobField object
- Box object
- Chart object
- Crosstab object
- Field object
- Line object
- Map object
- OLAP grid object
- OLE object
- Text object
- FieldHeading object
- Report options
- Summary info
- Formula field
- Group name field
- RunningTotal field

- Special field
- SQL expression field
- Summary field
- Groups
- Metadata
- Report Alerts
- Search Expert
- Events

Migration details

In this section, the finer details involved in migrating an RDC application to a J2EE web application are addressed. The JRC object model is introduced to help you gain a better understanding of the JRC architecture and its available features. Finally, code differences between RDC and JRC are compared for many important scenarios.

Java Reporting Component object model

The JRC is composed of two classes: the **ReportClientDocument** and the **CrystalReportViewer**. The reporting component deals with setting the parameters and data source for the report while the viewer is responsible for rendering the report, as well as exporting and printing.

Each component is explored further, along with their commonly used methods, in the following sections.

ReportClientDocument

The **ReportClientDocument** component is an embedded reporting solution that processes reports for viewing and exporting. It runs entirely on a web application server and does not require any additional components to run, such as BusinessObjects Enterprise. The viewers can use the reporting component to obtain a report source and then display the report associated with the report source.

The methods of the **ReportClientDocument** class methods are described below:

open: This method opens a report file specified by a URI. This is usually the first method used with a **ReportClientDocument** since you must specify which report is to be viewed by the **CrystalReportViewer**.

close: This method closes the **ReportClientDocument** and releases all resources. Invoke this method once you are finished with the report.

getReportSource: This method returns the **ReportSource** object used by the **CrystalReportViewer** to display the contents of the report.

getSubreportController: This method obtains the **SubreportController** object used to determine the names of the subreports in the report and to modify the subreports' databases.

getDatabaseController: The method returns a **DatabaseController** object used to access the tables from the report's data source.

setLocale: This method sets the locale that is used to localize various aspects of the document. This must be set before the **ReportClientDocument** is opened.

verifyDatabase: This method synchronizes the report with its data source. If the database scheme has been modified, the report is automatically updated. Fields in the report will be deleted if the corresponding fields have been deleted from the database, and if the database field type has changed, an attempt is made to map the old field type to a new type.

CrystalReportViewer

The **CrystalReportViewer** allows you to build customized report viewers for displaying reports as Dynamic HTML (DHTML). The viewers can be customized according to your needs by providing the ability to configure their display characteristics. Furthermore, the viewer allows you to programmatically export reports to a variety of formats.

The most commonly used methods of the **CrystalReportViewer** class are described below:

setReportSource: This method sets the **ReportSource** object that the **CrystalReportViewer** renders as HTML. The report source must be set in order for the viewer to function properly.

setOwnPage: This method sets whether the viewer controls the entire HTML page. Allowing the viewer to handle the surrounding HTML content reduces the amount of code you need to add to your JSP file and allows the viewer to automatically determine certain settings:

- It enables the viewer to choose which page starting and ending tags to use based on what device is viewing the page. For example, it writes out the HTML start tag for web browsers and WML start tag for mobile devices.
- It correctly sets the content-type and charset information for the page. This ensures that pages containing international characters appears correctly.
- It automatically enables **Export** and **Print** button support in the viewer.

Set this attribute to False if the page is within a portal or if there are other viewers on the page. The printing and exporting features are disabled when **setOwnPage** is set to False since the toolbar containing the **Export** and **Print** buttons are not shown. By default, **setOwnPage** is True.

setOwnForm: This method sets whether the viewer control owns the form. If the server control owns the form, it is able to get and set values for the form. In particular, by owning the form, the viewer control is able to cache its state to the client side.

When the viewer does not own the form, it becomes the host's responsibility to restore the view state of the viewer the next time the form is rendered again. By default, **setOwnForm** is true.

setPrintMode: This method sets whether to use the PDF or ActiveX print mode when the user clicks the **Print** button. In PDF print mode, a PDF document appears, allowing the user to then print it. In ActiveX print mode, a small ActiveX control is downloaded to the client computer to execute the print job. If ActiveX print mode is selected on a system that does not support ActiveX controls, then it will default to PDF print mode.

processHttpRequest: This method generates the HTML for the report and writes the HTML directly to the **Response** object. The method is called last after the viewer display characteristics have been customized.

Code comparison

When migrating from the RDC to JRC, the code used is quite different since you are moving from Visual Basic to Java.

This section discusses the code for a property or method in the RDC followed by the equivalent code in JRC. This can be used as a reference when it comes time to migrate the sources.

Refer to the section Finding more information for the accompanying collection of sample web applications, which demonstrates commonly used API functionality in the JRC. Use these samples to learn the JRC code used in many important scenarios.

CRViewer.ViewReport

Report Designer Component

Viewing of the report is done through the Report Viewer control.

```
CRViewer1.ReportSource = Report
CRViewer1.ViewReport
```

Java Reporting Component

Viewing the report is done through the CrystalReportViewer.

```
//Create the CrystalReportViewer object
CrystalReportViewer oCrystalReportViewer =
  new CrystalReportViewer();
```

```
//Set the reportsource property of the viewer
oCrystalReportViewer.setReportSource(reportSource);

//View the report
oCrystalReportViewer.processHttpRequest
  (request, response, context, null);
```

Report.Export

Report Designer Component

Export the report without prompting the user.

```
Report.Export False
```

Java Reporting Component

Use the **ReportExportControl** to export without viewing the report first in the DHTML Viewer.

```
//Set the export options to export in the format of choice.
ExportOptions oExportOptions = new ExportOptions();
oExportOptions.setExportFormatType(ReportExportFormat.RTF);

//Set to export the report to disk
ReportExportControl oReportExportControl =
  new ReportExportControl();
oReportExportControl.setExportAsAttachment(true);

//Pass the report source to ReportExportControl
//and set ExportOptions
oReportExportControl.setReportSource(reportSource);
oReportExportControl.setExportOptions(oExportOptions);

//Export the report
oReportExportControl.processHttpRequest
  (request, response, context, null);
```

DatabaseTable.ConnectionProperties

Report Designer Component

Using DatabaseTable's **ConnectionProperties**, you can specify the information to connect to the data source.

```
'Connect the first table of the tables collection to
```

```
'the data source
Dim dbProperties As CRAXDRT.ConnectionProperties
Set dbProperties = _
  Report.Database.Tables.Item(1).ConnectionProperties
dbProperties("DSN") = "Accounting"
dbProperties("Database") = "Administration"
dbProperties("User ID") = "734"
dbProperties("Password") = "bigboard"
```

Java Reporting Component

Use **ConnectionInfo** to specify the database logon information to the data source. Only a username and password are needed since other info should already be defined in the report.

```
ConnectionInfo oConnectionInfo = new ConnectionInfo();
oConnectionInfo.UserID = "734";
oConnectionInfo.Password = "bigboard";
```

FormulaFieldDefinition.Text

Report Designer Component

Specifies a new string for an existing formula.

```
'Pass the formula to the first formula in the
'FormulaFields collection
Report.FormulaFields.Item(1).Text = "{file.SALES} *.1"
'Pass the formula to the second formula in the
'FormulaFields collection
Report.FormulaFields.Item(2).Text = _
  "{file.SALES} + {file.COMMISSION}"
```

Java Reporting Component

Set the text of a **FormulaField** object.

```
FormulaField oFormulaField1 = new FormulaField();
oFormulaField1.setText("{file.SALES} *.1");
FormulaField oFormulaField2 = new FormulaField();
oFormulaField2.setText("{file.SALES} + {file.COMMISSION}");
```

ParameterFieldDefinition.AddCurrentValue

Report Designer Component

Changes the default value of the specified parameter field.

```
'Note the RDC has the ability to add multiple values to
'a single parameter.
'Add the value to the first parameter in the
'ParameterFields collection
Report.ParameterFields.Item(1).AddCurrentValue 1000
Report.ParameterFields.Item(1).AddCurrentValue 5000
Report.ParameterFields.Item(1).AddCurrentValue 10000
```

Java Reporting Component

Set multiple values to a **ParameterField** object.

```
Fields oFields = new Fields();
ParameterField oParameterField = new ParameterField();
Values oValues = new Values();
ParameterFieldDiscreteValue oParameterFieldDiscreteValue1 =
  new ParameterFieldDiscreteValue();
ParameterFieldDiscreteValue oParameterFieldDiscreteValue2 =
  new ParameterFieldDiscreteValue();
ParameterFieldDiscreteValue oParameterFieldDiscreteValue3 =
  new ParameterFieldDiscreteValue();
oParameterFieldDiscreteValue1.setValue
  (new Integer("1000"));
oParameterFieldDiscreteValue2.setValue
  (new Integer("5000"));
oParameterFieldDiscreteValue3.setValue
  (new Integer("10000"));
oValues.add(oParameterFieldDiscreteValue1);
oValues.add(oParameterFieldDiscreteValue2);
oValues.add(oParameterFieldDiscreteValue3);
oParameterField.setCurrentValues(oValues);
oFields.add(oParameterField);
```

Application.OpenReport

Report Designer Component

This is a method of the **Application** Object and is used for opening reports saved in the Crystal Reports (.rpt) format. The RDC can also open reports saved as ActiveX Designers (DSR) within Visual Basic.

Opening a report:

```
'General Declarations
'Declare an application object
Dim crxApplication as New craxdrt.Application
'Declare a Report object
Dim Report as craxdrt.Report
'In a function or Sub procedure
Set Report = _
  crxApplication.OpenReport("C:\crw\company.rpt", 1)
```

Declaring a variable as a new DSR:

```
'General Declarations
'CrystalReport1 is the name of the DSR in the Designer
'folder of the Project menu (CrystalReport1.dsr)
Dim Report as New CrystalReport1
```

Setting a **Report** object to a DSR:

```
'General Declarations
Dim Report as craxdrt.Report
Private Sub Form_Load()
  'Set the generic Report object to the DSR
  Set Report = New CrystalReport1
End Sub
```

Java Reporting Component

JRC does not support ActiveX Designers (DSR) and ActiveX viewers.

Opening a report:

```
ReportClientDocument reportDocument =

  new ReportClientDocument;

reportDocument.open(fileName);
```

SortField.Field, SortField.SortDirection

Report Designer Component

These are properties of the **SortField** object. **Field** sets the field to sort on, **SortDirection** sets the sort direction.

```
'Dim a DatabaseFieldDefinition object
Dim crxDatabaseField As craxdrt.DatabaseFieldDefinition
'Currently the sort is based on the Customer Name field and
'the application is to change it to the 'Last Year's
'Sale's' field. This field must be present on the report.
'Accessing the first table to get the 8th field
Set crxDatabaseField = _
  Report.Database.Tables.Item(1).Fields.Item(8)
'Set the field to crxDatabaseField
Report.RecordSortFields.Item(1).Field = crxDatabaseField
'Set the SortField direction
Report.RecordSortFields.Item(1).SortDirection = _
  crAscendingOrder
```

Java Reporting Component

Set the sort direction on the **ParameterField** object.

```
ParameterField oParameterField = new ParameterField();
oParameterField.setDefaultValueSortOrder
  (ParameterSortOrder.alphabeticalAscending);
```

CRViewer.EnableDrillDown

Report Designer Component

Indicates whether drill-down on summary values is allowed in the Report Viewer control.

```
CRViewer1.EnableDrillDown = True
```

Java Reporting Component

Sets whether to enable drill down in the CrystalReportViewer.

```
oCrystalReportViewer.setEnableDrillDown(true);
```

CRViewer.EnableToolbar

Report Designer Component

Specifies whether the toolbar is to appear in the Report Viewer control.

```
CRViewer1.EnableToolbar = True
```

Java Reporting Component

Sets whether to display the toolbar in the CrystalReportViewer.

```
oCrystalReportViewer.setDisplayToolbar(true);
```

CRViewer.Height

Report Designer Component

Sets the height of the Report Viewer control within the parent form.

```
'To set the height to match the parent form
'Place within the Form Resize event.
CRViewer1.Height = ScaleHeight
```

Java Reporting Component

Sets the height of the CrystalReportViewer.

```
oCrystalReportViewer.setHeight(ScaleHeight);
```

CRViewer.Left

Report Designer Component

Sets the left side of the Report Viewer control within the parent form.

```
'To set the control to the left edge of the parent form
'Place within the Form Resize event.
CRViewer1.Left = 0
```

Java Reporting Component

Sets the position for the left side of the CrystalReportViewer.

```
oCrystalReportViewer.setLeft(0);
```

CRViewer.EnableExportButton

Report Designer Component

Specifies whether an **Export** button is available in the Report Viewer control.

```
CRViewer1.EnableExportButton = True
```

Java Reporting Component

Sets whether to display the **Export** button in the CrystalReportViewer.

```
oCrystalReportViewer.setHasExportButton(true);
```

CRViewer.EnableGroupTree

Report Designer Component

Specifies whether a group tree appears in the Report Viewer control.

```
CRViewer1.EnableGroupTree = True
```

Java Reporting Component

Sets whether to display the group tree in the CrystalReportViewer.

```
oCrystalReportViewer.setDisplayGroupTree(true);
```

CRViewer.EnableNavigationControls

Report Designer Component

Specifies whether the navigation controls appear in the Report Viewer control.

```
CRViewer1.EnableNavigationControls = True
```

Java Reporting Component

Sets whether to display the page navigation buttons in the CrystalReportViewer.

```
oCrystalReportViewer.setHasPageNavigationButtons(true);
```

CRViewer.EnablePrintButton

Report Designer Component

Specifies whether a print button appears in the Report Viewer control.

```
CRViewer1.EnablePrintButton = True
```

Java Reporting Component

Sets whether to display the print button in the CrystalReportViewer.

```
oCrystalReportViewer.setHasPrintButton(true);
```

CRViewer.EnableRefreshButton

Report Designer Component

Specifies whether a refresh button appears in the Report Viewer control.

```
CRViewer1.EnableRefreshButton = True
```

Java Reporting Component

Sets whether to display the **Refresh** button in the CrystalReportViewer.

```
oCrystalReportViewer.setHasRefreshButton(true);
```

CRViewer.EnableSearchControl

Report Designer Component

Specifies whether a Search control appears in the Report Viewer control.

```
CRViewer1.EnableSearchControl = True
```

Java Reporting Component

Sets whether to include or exclude the **Search** button and associated text box when rendering the report in the CrystalReportViewer.

```
oCrystalReportViewer.setHasSearchButton(true);
```

CRViewer.Top

Report Designer Component

This is a property of the Report Viewer control and sets the topside of the control within the parent form.

```
'To set the control to the left edge of the parent form
'Place within the Form Resize event
CRViewer1.Top = 0
```

Java Reporting Component

Sets the position for the top of the CrystalReportViewer.

```
oCrystalReportViewer.setTop(0);
```

CRViewer.Width

Report Designer Component

This is a property of the Report Viewer control and sets the width of the control within the parent form.

```
'To set the width to match the parent form
'Place within the Form Resize event.
CRViewer1.Width = width
```

Java Reporting Component

Sets the width of the CrystalReportViewer.

```
oCrystalReportViewer.setWidth(width);
```

Application.LogonServer, DataBase.LogonServer

Report Designer Component

Log on to the specified server. This is a method of the **Application** and **Database** Objects.

Application object:

```
CrxApplication.LogonServer "pdsodbc.dll", "Accounting", _
  "Administration", "bobg", "bigboard"
```

Database object:

```
'Log onto the data source
Report.DataBase.LogonServer "pdsodbc.dll", "Accounting", _
  "Administration", "bobg", "bigboard"
```

Java Reporting Component

Set the database logon information. The database will be logged on to when the report is viewed.

```
ConnectionInfos oConnectionInfos = new ConnectionInfos();
IConnectionInfo oIConnectionInfo = new ConnectionInfo();
oIConnectionInfo.UserID = "bobg";
oIConnectionInfo.Password = "bigboard";
oConnectionInfos.add(oIConnectionInfo);
oCrystalReportViewer.setDatabaseLogonInfos
  (oConnectionInfos);
```

CRViewer.ShowFirstPage

Report Designer Component

Displays the first page of the report in the Report Viewer control.

```
CRViewer1.ShowFirstPage
```

Java Reporting Component

Displays the first page of the report in the CrystalReportViewer.

```
oCrystalReportViewer.showFirstPage();
```

CRViewer.ShowLastPage

Report Designer Component

Displays the last page of the report in the Report Viewer control.

```
CRViewer1.ShowLastPage
```

Java Reporting Component

Displays the last page of the report in the CrystalReportViewer.

```
oCrystalReportViewer.showLastPage();
```

CRViewer.ShowNextPage

Report Designer Component

Displays the next page of the report in the Report Viewer control.

```
CRViewer1.ShowNextPage
```

Java Reporting Component

Displays the next page of the report in the CrystalReportViewer.

```
oCrystalReportViewer.showNextPage();
```

CRViewer.ShowPreviousPage

Report Designer Component

Displays the previous page of the report in the Report Viewer control.

```
CRViewer1.ShowPreviousPage
```

Java Reporting Component

Displays the previous page of the report in the CrystalReportViewer.

```
oCrystalReportViewer.showPreviousPage();
```

CRViewer.ShowNthPage

Report Designer Component

Displays the nth page of the report in the Report Viewer control.

```
CRViewer1.ShowNthPage 3
```

Java Reporting Component

Displays the specified page of the report in the CrystalReportViewer.

```
oCrystalReportViewer.showNthPage(3);
```

CRViewer.Zoom

Report Designer Component

Sets the magnification factor for the report in the Report Viewer control.

```
CRViewer1.Zoom 150
```

Java Reporting Component

Sets the zoom factor for displaying the report in the CrystalReportViewer.

```
oCrystalReportViewer.setZoomFactor(150);
```

Reset Report

Report Designer Component

To reset the report, set the report object to "Nothing", then open the report again and set the new properties.

```
'Set the Report object to Nothing
Report = Nothing
```

Java Reporting Component

Set the ReportDocument object to null.

```
oReportDocument = null;
```

Thick-client Java Reporting Component applications

The JRC is designed to be used in a J2EE web application, but it is also possible to use it in a thick-client application. This is done by using the ReportViewerBean Viewer in a Java Swing application instead of the CrystalReportViewer in JSP.

Below are the steps to use the Java Swing viewer to work with the JRC:

1. Create a WEB-INF\lib folder in the root of your Java *.class file folder tree. For example:

```
com\
   foobar\
WEB-INF\
   lib\
```

2. Copy the following JAR files from …\Program Files\Common Files\Business Objects\3.0\java\lib, ...\Program Files\Common Files\Business Objects\3.0\java\lib\external, and ...\Program Files\Common Files\Business Objects\3.0\crystalreportviewers115\JavaViewer folders to the WEB-INF\lib folder:

   - ReportViewer.jar
   - jrcerom.jar
   - Concurrent.jar
   - CrystalCharting.jar
   - CrystalCommon.jar
   - CrystalContentModels.jar
   - CrystalExporters.jar
   - CrystalExportingBase.jar
   - CrystalFormulas.jar
   - CrystalQueryEngine.jar
   - CrystalReportEngine.jar
   - CrystalReportingCommon.jar
   - icu4j.jar
   - keycodeDecoder.jar
   - log4j.jar
   - MetafileRenderer.jar
   - rasapp.jar
   - rascore.jar
   - rpoifs.jar
   - Serialization.jar
   - URIUtil.jar
   - xercesImpl.jar
   - xml-apis.jar

3. Add the above JAR files to your classpath and ensure that ReportViewer.jar and jrcerom.jar are the first two JARs listed.

NOTE	If you are using Eclipse, you may have to quit Eclipse, edit your classpath file to move the JARs to the beginning of the list, and then start Eclipse. Otherwise, you will receive a "java.lang.IncompatibleClassChangeError: Implementing class" exception at run time.

4. Copy the CRConfig.xml file from your ...\Program Files\Common Files\Business Objects\3.0\java folder to the root of your Java *.class folder tree. For example:

```
CRConfig.xml
com\
   foobar\
WEB-INF\
   lib\
```

5. Edit the CRConfig.xml file's <reportlocation> tag to specify the relative path to the folder containing the .rpt files. The path is relative to the WEB-INF\lib folder. For example, if you specify ".", then the .rpt file is loaded from the WEB-INF\lib folder.

```
<?xml version="1.0" encoding="utf-8"?>
<CrystalReportEngine-configuration>
    <reportlocation>.</reportlocation>
```

6. Add code similar to the following to create the ReportViewerBean, open the report, set the report on the viewer, and start the viewer:

```
import java.awt.*;
import javax.swing.*;
import com.crystaldecisions.ReportViewer.*;

// Use this for talking to JRC in-process
import com.crystaldecisions.reports.sdk.*;
import
com.crystaldecisions.sdk.occa.report.reportsource.*;

public class HelloWorldSwing
{
  private static void createAndShowGUI()
  {
    try
    {
      //Make sure we have nice window decorations.
      JFrame.setDefaultLookAndFeelDecorated(false);

      //Create and set up the window.
      JFrame frame = new JFrame("HelloWorldSwing");
      frame.setTitle("Testing 1, 2, 3");

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      ReportViewerBean viewer = new ReportViewerBean();
      viewer.init(new String[0], null, null, null);

      ReportClientDocument rpt =
        new ReportClientDocument();
      rpt.open("MyReport.rpt", 0);

      IReportSource rptSource = rpt.getReportSource();
      viewer.setReportSource(rptSource);

      frame.getContentPane().add
        (viewer, BorderLayout.CENTER);
      frame.setSize(700, 500);
      frame.setVisible(true);

      viewer.start();
    }
    catch (Exception exception)
    {
      System.out.println(exception.toString());
    }
  }

  public static void main(String[] args)
  {
    javax.swing.SwingUtilities.invokeLater(new
Runnable()
    {
      public void run()
      {
        createAndShowGUI();
      }
    });
  }
}
```

Appendix: Feature comparison

Feature Comparison of RDC and JRC		
	RDC	**JRC**
Parameters		
Main report	Y	Y
Subreports	Y	Y
Database		
Log on / off	Y	Y
Subreport db logon	Y	Y
Set Datasource Location	Y	Y
Dictionary / Query	Y	
Field Mapping	Y	
Verify database	Y	Y
Show SQL Query	Y	
Select Distinct Records only	Y	
Perform Grouping on Server	Y	
Runtime Dataset support (ADO)	Y	
Runtime Dataset support (XML)		Y
Runtime Dataset support (Java Resultset)		Y
Data Types		
ODBC	Y	
ADO	Y	
DAO	Y	
Oracle	Y	
DB2	Y	
ADO.NET	Y	
Informix	Y	
Sybase	Y	

Feature Comparison of RDC and JRC		
	RDC	**JRC**
Access	Y	
Excel	Y	
Field Definition	Y	
COM object	Y	
Java Bean	Y	
Crystal Data	Y	
IIS Log data	Y	
Web Log data	Y	
Lotus Notes data	Y	
File System data	Y	
Btrieve data	Y	
BDE data	Y	
Xbase data	Y	
ACT data	Y	
Exchange data	Y	
Outlook data	Y	
NT Event Log data	Y	
JDBC		Y
XML		Y
POJO		Y
Exporting from subreports		
Destinations:		
Application	Y	
Disk file	Y	Y
Exchange folder	Y	
Lotus domino	Y	
Lotus Domino mail	Y	
MS mail (MAPI)	Y	

Feature Comparison of RDC and JRC		
	RDC	**JRC**
HTTP Response		Y
Stream		Y
Formats:		
PDF (Acrobat)	Y	Y
Crystal Reports	Y	Y
HTML	Y	
Excel	Y	
Word	Y	
Record style	Y	
RTF	Y	Y
CSV	Y	Y
Text	Y	Y
Tab separated text	Y	
XML	Y	
Report Definition	Y	
ODBC	Y	
Editable RTF	Y	Y
Printing from subreports and drill-downs		
Printer setup	Y	Y (in XI Release 2)
Orientation	Y	
Page ranges, copies	Y	Y (in XI Release 2)
Collate copies	Y	Y (in XI Release 2)
Report Object Common Properties		
Name	R	RW
Kind	R	R
Left	RW	

Feature Comparison of RDC and JRC		
	RDC	**JRC**
Top	RW	
Width	RW	
Height	RW	
Border	RW	
HasDropShadow	RW	
EnableCanGrow	RW	
CloseAtPageBreak	RW	
EnableKeepTogether	RW	
EnableSuppress	RW	
HorizontalAlignment	RW	
CssClass	RW	
ConditionFormula	RW	
HyperlinkType	RW	
HyperlinkText	RW	
LinkedURI	RW	
Parameter Field		
Create	Y	
Properties:		
BrowseField		RW
CurrentValues	RW	RW
DefaultValues	RW	RW
DefaultValueSortMethod	RW	RW
DefaultValueSortOrder	RW	RW
DiscreteOrRangeKind	RW	RW
EditMask	RW	RW
EnableAllowEditingDefaultValue	RW	RW
EnableAllowMultipleValue	RW	RW
EnableNullValue	RW	RW

Feature Comparison of RDC and JRC		
	RDC	**JRC**
HasCurrentValue	RW	
MaximumValue	RW	RW
MinimumValue	RW	RW
ParameterFieldName	R	RW
ParameterType	R	RW
ParameterValueKind	R	RW
PromptText	RW	
ReportName	R	RW
EnableShowDescriptionOnly	RW	
DisallowEditing	RW	
NeedsCurrentValue	RW	
PlaceInGroup	RW	
GroupNumber	RW	
EnableExclusiveGroup	RW	
PicklistSortMethod	RW	
EnableSortBasedOnDesc	RW	
Drill Down		
Group	Y	Y
Chart	Y	Y
Map	Y	Y
Embedded subreport	Y	Y
On-demand subreport	Y	Y
Groups, charts, maps in subreport	Y	Y
Report part		Y
Subreport Object		
Insert new	Y	
Format Object Common Properties	Y	

Feature Comparison of RDC and JRC		
	RDC	**JRC**
Properties:		
SubreportName	RW	
EnableOnDemand	RW	
SubreportLinks	RW	
ReimportSubreport	RW	
OpenSubreport	Y	Y
Add/Modify/Remove child objects in subreport	Y	
BlobField Object	**Y**	**Not supported**
Box Object	**Y**	**Not supported**
Chart Object	**Y**	**Not supported**
Crosstab Object	**Y**	**Not supported**
Field Object	**Y**	**Not supported**
Line Object	**Y**	**Not supported**
Map Object	**Y**	**Not supported**
OLAP Grid Object	**Y**	**Not supported**
OLE Object	**Y**	**Not supported**
Text Object	**Y**	**Not supported**
FieldHeading Object	**Y**	**Not supported**
Report Options	**Y**	**Not supported**
Summary Info	**Y**	**Not supported**
Formula Field	**Y**	**Not supported**
Group Name Field	**Y**	**Not supported**
RunningTotal Field	**Y**	**Not supported**
Special Field	**Y**	**Not supported**
SQLExpression Field	**Y**	**Not supported**
Summary Field	**Y**	**Not supported**
Custom function	**Y**	**Not supported**
Groups	**Y**	**Not supported**

Feature Comparison of RDC and JRC		
	RDC	JRC
Metadata	Y	Not supported
Repository	Y	Not supported
OLAP	Y	Not supported
Templates	Y	Not supported
Report Alerts	Y	Not supported
Report Parts	Y	Not supported
Creating Report	Y	Not supported
Search Expert	Y	Not supported
Events	Y	Not supported

Functional Comparison of RDC to JRC		
	RDC	**JRC**
Viewing		
Report Page Viewer		Y
Report Parts Viewer		Y
Java thick client Viewer		Y (in XI Release 2)
ActiveX Viewer	Y	
Report Modification		
Insert and Delete report objects	Y	Y
Modify report objects	Y	Y
Set Selection Formula	Y	Y
Set Parameter Fields	Y	Y
Set Formula Fields	Y	Y
Set Database Connection	Y	Y
Set Table Location	Y	Y
Set / Change Grouping options	Y	Y
Print Options	Y	Y
Set Sorting options	Y	Y

Finding more information

Accompanying sample applications

http://support.businessobjects.com/communityCS/FilesAndUpdates/crxi_r2_migrating_rdc_to_jrc_samples.zip.asp

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/jrcxir2_desktop_samples.zip.asp

Deploying JRC in web and desktop applications

http://support.businessobjects.com/communityCS/TechnicalPapers/cr_xi_r2_jrc_deployment.pdf

CRConfig.xml configuration

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

www.businessobjects.com