Integrating Crystal Reports with Visual FoxPro

Overview

In this session, you will learn how to seamlessly integrate Crystal Reports into your Visual FoxPro application. Crystal Reports give the developer full control over programmatically creating, modifying, printing, and exporting reports.

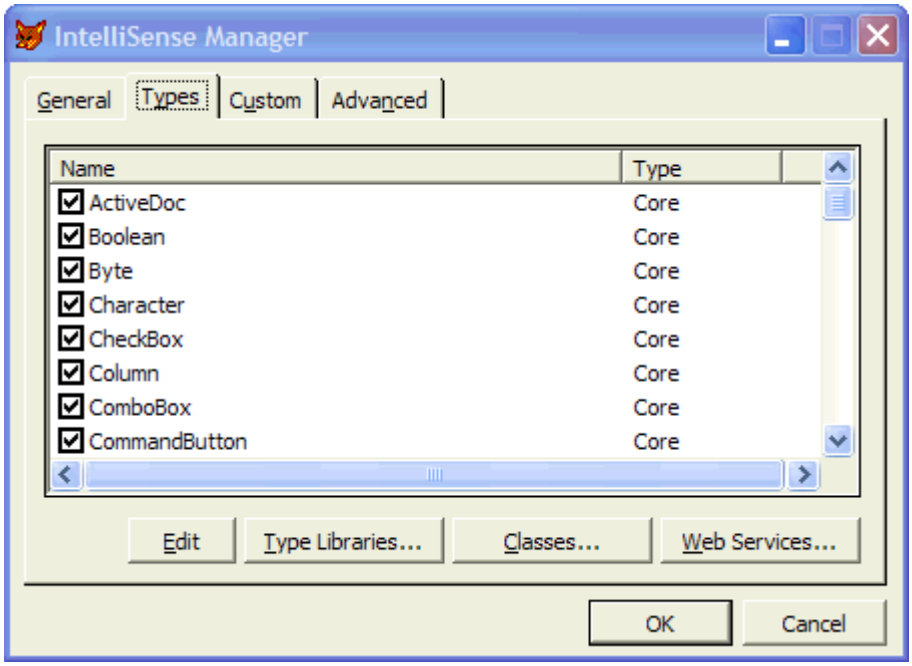
Integration Basics

For many years, Crystal Reports has provided developers with the capabilities to integrate reports into applications. The current integration method, the Report Design Control (RDC) is the method recommended by Crystal Decisions. It consists of four main part

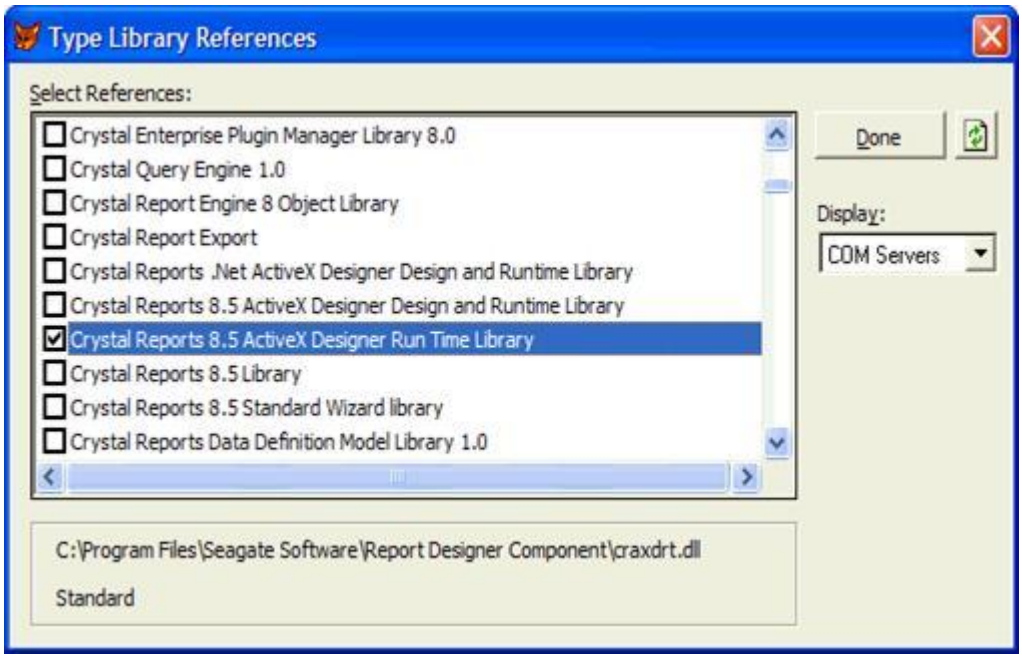
- Designer Runtime Library (CRAXDRT.DLL). The primary integration method. A COM Automation server that provides report manipulation, printing, and exporting capabilities.
- Designer Design and Runtime Library (CRAXDDRT.DLL). This component provides all the features of the CRAXDRT.DLL and supports the Distributable Report Designer.
- Report Viewer. An ActiveX control used to preview reports.
- Distributable Report Designer. An ActiveX control used to give users report design capabilities.

Before using the CRAXDRT.DLL, you should register the component with Visual FoxPro's Intellisense Manager. This will allow all the properties, methods, and events, to be selected using Intellsense. The following steps guide you through this registration process:

1. From the Visual FoxPro menu, select Tools | Intellisense Manager.
2. Select the "Types" tab.



3. Click “Type Libraries” to display the Type Library References dialog.



4. Select “Crystal Reports 8.5 ActiveX Designer Runtime Library” from the list. Make sure the check box next to the item is checked.
5. Click “Done” to return to the IntelliSense Manager
6. Click OK.
7. Test the IntelliSense settings. In the Command Window, type “MODIFY COMMAND Test” then press Enter
8. Type the following code into the editor window

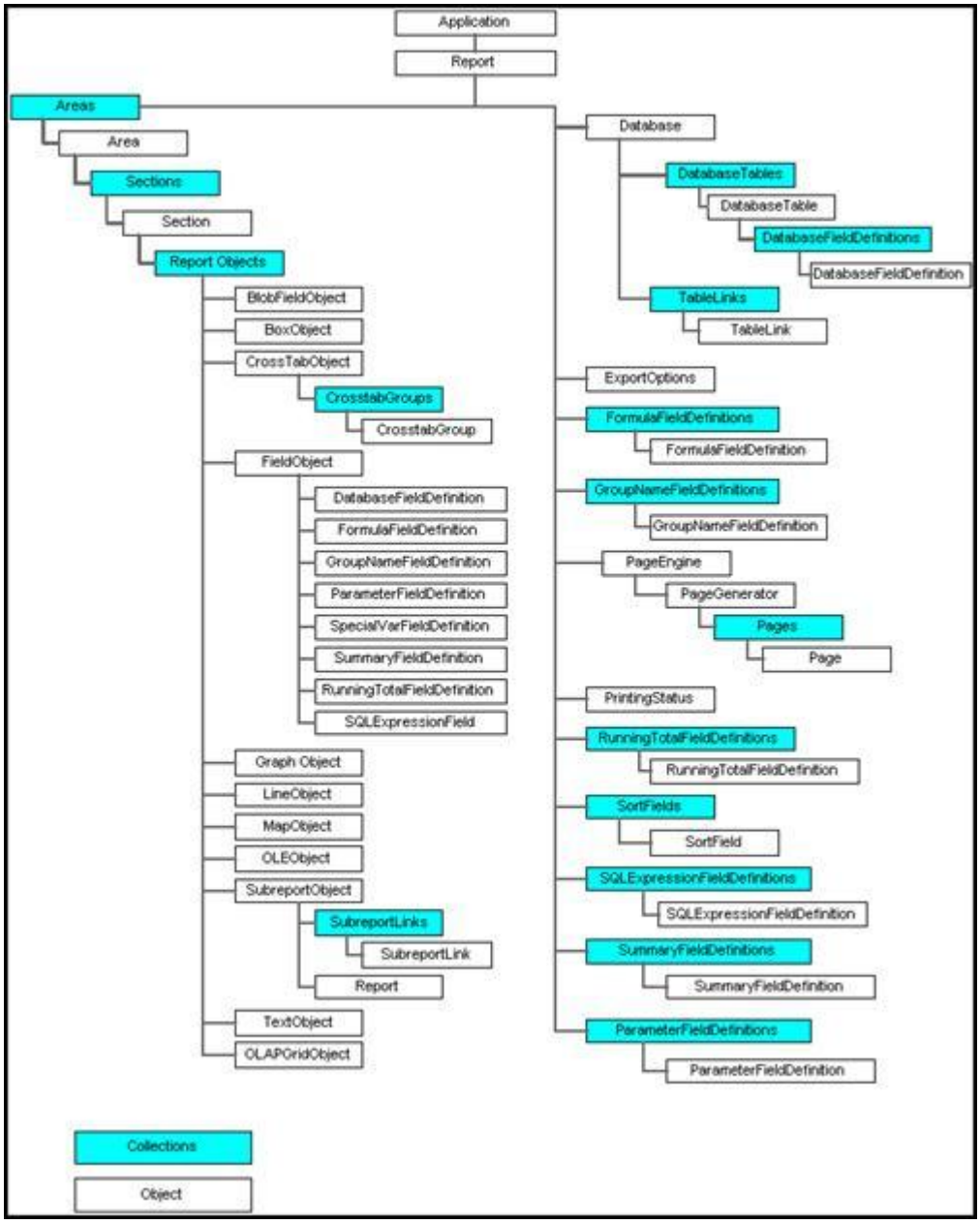
```
LOCAL oCR AS CrystalRuntime.Application
oCR.
```

9. Notice that as soon as you press the . that IntelliSense will kick in and give you a list of PEMs from the Crystal Reports Application object.
10. Press Escape and answer Yes to the Discard Changes dialog.

Throughout this document, you'll see other LOCAL declarations for different RDC objects. These LOCAL declarations will make IntelliSense work for these different objects.

Common Reporting Functions

The most common things you'll do with the RDC are print, access data, export, and preview. Before jumping into these an overview of the RDC object model is in order.



At the top of the model is the Application object, followed by the Report object. You'll need to instantiate these before you can do anything else with the report.

The left-hand side of the object model deals with each report object. Sections, fields, and more are shown. The right hand side outlines the data access and manipulation objects and collections.

The following code shows how to instantiate the primary objects and open a report:

```
LOCAL oCR AS CRAXDRT.Application
LOCAL oRpt AS CRAXDRT.Report

oCR = CREATEOBJECT("CrystalRuntime.Application")
```

```
oRpt = oCR.OpenReport("C:\Temp\Taz.RPT")
```

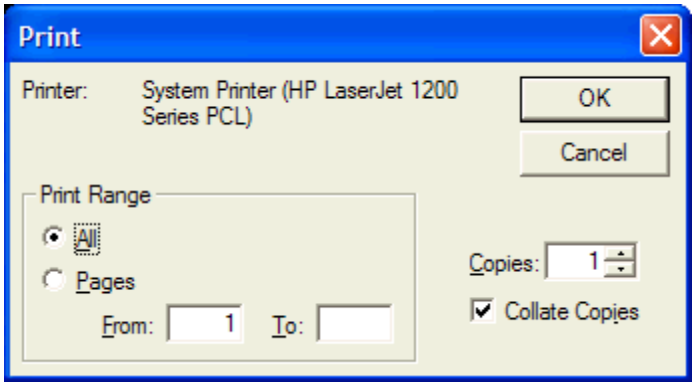
Printing a Report

Printing could be the most often performed function for a report. Here's the code to do this:

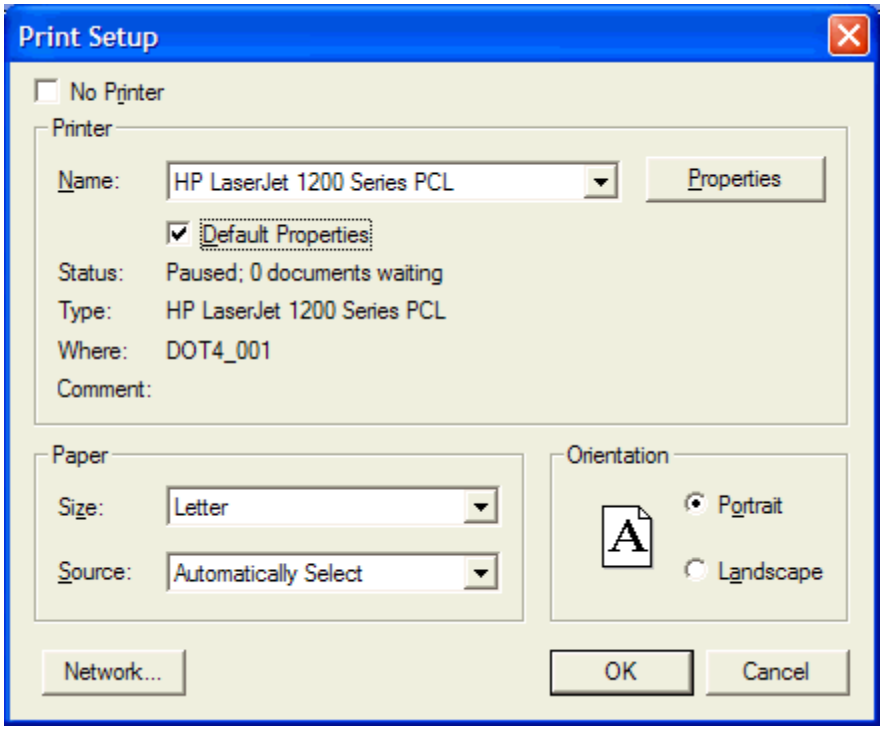
```
LOCAL oCR AS CRAXDRT.Application
LOCAL oRpt AS CRAXDRT.Report

oCR = CREATEOBJECT("CrystalRuntime.Application")
oRpt = oCR.OpenReport("C:\Temp\Taz.RPT")
IF oRpt.HasSavedData
oRpt.DiscardSavedData()
ENDIF
oRpt.PrintOut()
```

It's a good idea to discard any data that may have been saved with the report. The PrintOut method will display a print dialog to the user. If you don't want the dialog to show, pass False as the first parameter.



Two other methods of interest are PrinterSetup( ) and SelectPrinter( ). PrinterSetup( ) lets you programmatically change printers. SelectPrinter( ) displays the Print Setup dialog.



Accessing Data

It's almost certain that the data location will change between design and runtime of the report. Using the Database object, DatabaseTables collection, and DatabaseTable object, you can change the report location.

The exact steps for accessing data depend on the type of data and the access method, either direct, via ODBC, or through OLE DB. In general, you need to create the Database object, then use the DatabaseTables collection to drill down to each DatabaseTable object. Note that collections in Crystal Reports are 1-based. There will be one DatabaseTable object for each table in the report. The following code shows each of these methods.

Direct Data Access

Direct data can be used directly from Crystal Reports without going through ODBC or OLEDB. Fox 2x and Access are examples of direct data. In the case of direct data, the location and name of the table may change at runtime.

```
LOCAL oCR AS CRAXDRT.Application
LOCAL oRpt AS CRAXDRT.Report
LOCAL oDB AS CRAXDRT.Database
LOCAL ocDBT AS CRAXDRT.DatabaseTables
LOCAL oDBT AS CRAXDRT.DatabaseTable

oCR = CREATEOBJECT("CrystalRuntime.Application")
oRpt = oCR.OpenReport("C:\EFox\Direct1.RPT")
```

```
* Create the Database object
oDB = oRpt.Database()

* Get a references to the DatabaseTables collection
ocDBT = oDB.Tables()

* Get a reference to the DatabaseTable object for table 1
oDBT = ocDBT.Item(1)

* Set the location
oDBT.Location = "C:\EFox\Customer.DBF"

oRpt.PrintOut()
```

ODBC Data

Once you've setup the DSN, connecting to ODBC data is pretty simple.

```
LOCAL oCR AS CRAXDRT.Application
LOCAL oRpt AS CRAXDRT.Report
LOCAL oDB AS CRAXDRT.Database
LOCAL ocDBT AS CRAXDRT.DatabaseTables
LOCAL oDBT AS CRAXDRT.DatabaseTable

oCR = CREATEOBJECT("CrystalRuntime.Application")
oRpt = oCR.OpenReport("C:\EFox\ODBC1.RPT")

* Create the Database object
oDB = oRpt.Database()

* Get a references to the DatabaseTables collection
ocDBT = oDB.Tables()

* Get a reference to the DatabaseTable object for table 1
oDBT = ocDBT.Item(1)

* Set the location
* This one works for a DSN
oDBT.SetLogOnInfo("TazODBCRuntime")

IF oRPt.HasSavedData
  oRPT.DiscardSavedData()
ENDIF

oRpt.PrintOut()
```

XML Data

Crystal Reports uses the CRXML ODBC driver to connect to XML data. You'll need to setup a DSN for the data access. The following code shows how to access the XML file at runtime.

```
LOCAL oCR AS CRAXDRT.Application
LOCAL oRpt AS CRAXDRT.Report
```

```
LOCAL oDB AS CRAXDRT.Database
LOCAL ocDBT AS CRAXDRT.DatabaseTables
LOCAL oDBT AS CRAXDRT.DatabaseTable

oCR = CREATEOBJECT("CrystalRuntime.Application")
oRpt = oCR.OpenReport("C:\EFox\XML1.RPT")

* Create the Database object
oDB = oRpt.Database()

* Get a references to the DatabaseTables collection
ocDBT = oDB.Tables()
* Get a reference to the DatabaseTable object for table 1
oDBT = ocDBT.Item(1)



* This one works for a DSN
oDBT.SetLogOnInfo("TazXMLRuntime")
IF oRPt.HasSavedData
   oRPT.DiscardSavedData()
ENDIF

oRpt.PrintOut()
```

ADO Data

Using ADO data is a bit different than the previous data access methods. Because ADO is an object held in memory, there is no physical file for Crystal Reports to use. In this case, create the ADO RecordSet and pass it to Crystal Reports.

```


LOCAL oCR AS CRAXDRT.Application
LOCAL oRpt AS CRAXDRT.Report
LOCAL oDB AS CRAXDRT.Database
LOCAL ocDBT AS CRAXDRT.DatabaseTables
LOCAL oDBT AS CRAXDRT.DatabaseTable
LOCAL oConn AS ADODB.Connection
LOCAL oRS AS ADODB.Recordset

* Handle the ADO stuff
oConn = CREATEOBJECT("ADODB.Connection")
oConn.ConnectionString = "Provider=VFPOLEDB.1;Data
Source=C:\eFox\Data\tastrade.dbc;Password=''"
oConn.Open()
oRS = CREATEOBJECT("ADODB.RecordSet")
oRS.Open("Select * FROM Customer", oConn)

oCR = CREATEOBJECT("CrystalRuntime.Application")

oRpt = oCR.OpenReport("C:\EFox\ADO1.RPT")
```

```
* Create the Database object
oDB = oRpt.Database()

* Get a references to the DatabaseTables collection
ocDBT = oDB.Tables()

* Get a reference to the DatabaseTable object for table 1
oDBT = ocDBT.Item(1)

* Pass the Record Set to Crystal Reports
oDBT.SetDataSource(oRS)

IF oRPt.HasSavedData
   oRPT.DiscardSavedData()
ENDIF

oRpt.PrintOut()
```

Parameter Fields

Parameter fields are used to pass information from your application to the RDC. One of the most common uses is for providing query or sort information to the report.

```


LOCAL oCR AS CRAXDRT.Application
LOCAL oRpt AS CRAXDRT.Report
LOCAL oDB AS CRAXDRT.Database
LOCAL ocDBT AS CRAXDRT.DatabaseTables
LOCAL oDBT AS CRAXDRT.DatabaseTable
LOCAL ocParm AS CRAXDRT.ParameterFieldDefinitions
LOCAL oParm AS CRAXDRT.ParameterFieldDefinition
oCR = CREATEOBJECT("CrystalRuntime.Application")

oRpt = oCR.OpenReport("C:\EFox\Parms.RPT")

* Create the Database object
oDB = oRpt.Database()

* Get a references to the DatabaseTables collection
ocDBT = oDB.Tables()

* Get a reference to the DatabaseTable object for table 1
oDBT = ocDBT.Item(1)

* This one works for a DSN
oDBT.SetLogOnInfo("TazODBCRuntime")

IF oRPt.HasSavedData
   oRPT.DiscardSavedData()
ENDIF
```

```
* Get the Special Message Parameter
ocParm = oRpt.ParameterFields()
oParm = ocParm.Item(1)
oParm.SetCurrentValue("This is the runtime special message")

oRpt.PrintOut()
```

Exporting Reports

Exporting a report is done in two steps. In step one, you set the report options. Then, in step two, you actually do the export. The following code will export a report to a Microsoft Excel file:

```
LOCAL oCR AS CRAXDRT.Application
LOCAL oRpt AS CRAXDRT.Report
LOCAL oExp AS CRAXDRT.ExportOptions
oCR = CREATEOBJECT("CrystalRuntime.Application")
oRpt = oCR.OpenReport("C:\Temp\Taz.RPT")
oExp = oRpt.ExportOptions()
oExp.DestinationType = 1  && crEDTDiskFile
oExp.FormatType = 27  && crEFTExcel70
oExp.DiskFileName = "C:\Temp\Taz.XLS"
oRpt.Export(.F.)
```

The False parameter on the Export method tells the RDC to not prompt the user for Export options.

Previewing Reports

The last major thing that you'll do with the RDC is previewing reports. This is an ActiveX control that you can drop on a VFP form. The Report Viewer allows you to control which options the user has available at runtime. For example, you can set the Visible property of the Export button to prohibit exporting the report.

Select the Crystal Report Viewer Control from the Controls tab of the VFP Options dialog.

The following form code shows you how to preview a report. In this example, the Report Viewer control has been dropped on the form and named oleCRViewer.

```
DEFINE CLASS form1 AS form
  Caption = "Report Preview"
  oCrystal = .F.
  oReport = .F.
  PROCEDURE Init
    LPARAMETERS tcReport

    LOCAL cr AS crViewer.crViewer
```

```
    WITH This
      .WindowState = 2
      .oCrystal = CREATEOBJECT("CrystalRuntime.Application")
      .oReport = .oCrystal.OpenReport(tcReport)

      WITH .oleCRViewer
        .EnableExportButton = .T.
        .EnableProgressControl = .T.
        .ReportSource = ThisForm.oReport
        .EnableAnimationCtrl = .F.
        .ViewReport()
      ENDWITH
      .Visible = .T.
    ENDWITH
  ENDPROC

  PROCEDURE Resize
    WITH This.oleCRViewer
      .Top = 1
      .Left = 1
      .Height = This.Height - 2
      .Width = This.Width - 2
    ENDWITH
  ENDPROC

  PROCEDURE Error
    LPARAMETERS nError, cMethod, nLine

    IF nError != 1440
      DODEFAULT()
    ENDIF
  ENDPROC

  PROCEDURE Destroy
    WITH This
      .oReport = NULL
      .oCrystal = NULL
    ENDWITH
  ENDPROC

  PROCEDURE Activate
    WITH This.olecrViewer
      .Top = 1
      .Left = 1
      .Height = ThisForm.Height - 2
      .Width = ThisForm.Width - 2
    ENDWITH
  ENDPROC
ENDDEFINE
```

COM Integration

There are two ways to integrate COM into a Crystal application. The first is through UserFunctionLibraries. The second is via event binding.

User Function Libraries

User Function Libraries (UFLs) are COM components that are available under Additional Functions in the Functions listing in the Crystal Reports Formula Editor. The component is a standard COM component.

You need to follow two rules when building the component. First, the project name must begin with CRUFL. When Crystal Reports launches, it scans the registry for any components using this naming convention.

Second, you must define parameter and return types. If you fail to do this, Crystal Reports will not recognize the component. This requires that you use VFP 7.0 to create UFLs as previous versions do not support this.

Note: There are some changes to UFLs with Crystal Reports 9. First, you must base your OLEPUBLIC object on the Session class. No other class will work. Second, all functions in the class must have data types of parameters and return values declared. If you don't want the function to show up in Crystal Reports, define it as PRIVATE.

Event Binding

The RDC has a few events available that you can hook into. The following code shows and example of how to bind to some of these events.

```
LOCAL oCR AS CRAXDRT.Application
LOCAL oRpt AS CRAXDRT.Report
LOCAL oDB AS CRAXDRT.Database
LOCAL ocDBT AS CRAXDRT.DatabaseTables
LOCAL oDBT AS CRAXDRT.DatabaseTable
oCR = CREATEOBJECT("CrystalRuntime.Application")
oRpt = oCR.OpenReport("C:\EFox\ODBC1.RPT")

* Setup the event binding
oEvents = NEWOBJECT("CrystalEvents")
oEvents.oRpt = oRpt
EVENTHANDLER(oRpt, oEvents)

* Create the Database object
oDB = oRpt.Database()

* Get a references to the DatabaseTables collection
ocDBT = oDB.Tables()

* Get a reference to the DatabaseTable object for table 1
oDBT = ocDBT.Item(1)
```

```
* Set the location

* This one works for a DSN
oDBT.SetLogOnInfo("TazODBCRuntime")

IF oRPt.HasSavedData
  oRPT.DiscardSavedData()
ENDIF

oRpt.PrintOut()
RETURN

DEFINE CLASS CrystalEvents AS session OLEPUBLIC

  IMPLEMENTS IReportEvent IN "CrystalRuntime.Application"

  oRpt = NULL

  PROCEDURE IReportEvent_NoData(pCancel AS LOGICAL) AS VOID ;
    HELPSTRING "Fires this event when there is no data"

    * add user code here
  ENDPROC

  PROCEDURE IReportEvent_BeforeFormatPage(PageNumber AS Number) ;
    AS VOID ;
    HELPSTRING "Fires this event before formatting a page"

   * add user code here
  ENDPROC

  PROCEDURE IReportEvent_AfterFormatPage(PageNumber AS Number) ;
    AS VOID ;
    HELPSTRING "Fires this event after formatting a page"

         STRTOFILE("AfterFormatPage", "C:\temp\cr.txt")
  ENDPROC

  PROCEDURE IReportEvent_FieldMapping(reportFieldArray AS VARIANT, ;
    databaseFieldArray AS VARIANT, useDefault AS LOGICAL) AS VOID ;
    HELPSTRING "Fires this event if database is changed while "
    + "verifing database"

    * add user code here
  ENDPROC
ENDDEFINE
```

Creating Reports

The RDC gives you complete control over creating your reports. You can do this either programmatically or use the runtime designer and let users create their own reports. With both of these options, additional licensing is required.

Coded Reports

With coded reports, you either create a report from scratch and build it in code or you can modify an existing report. Each object on the report is accessible using the RDC object model. You create a report using the New( ) method of the Application object. You can save it using the SaveAs( ) method of the Report object.

The Runtime Designer

The Runtime Designer is an embeddable control that gives report design capabilities to end users. It does not provide preview capabilities, so it is commonly placed on a page frame with one page used for the designer and the second for previewing the report. Drop the Embeddable Crystal Reports 8.5 Designer Control onto the form or page. Also note that you need to use the CRAXDDRT.DLL automation server instead of the CRAXDRT.DLL.

```
DEFINE CLASS form1 AS form
  Top = 0
  Left = 0
  Height = 479
  Width = 563
  DoCreate = .T.
  Caption = "Report Designer"
  Name = "Form1"
  ocrystal = NULL
  oreport = NULL
  ADD OBJECT pgfcrystal AS pageframe WITH ;
    ErasePage = .T., ;
    PageCount = 2, ;
    TabStyle = 1, ;
    Top = 12, ;
    Left = 12, ;
    Width = 541, ;
    Height = 461, ;
    Name = "pgfCrystal", ;
    Page1.Caption = "Design", ;
    Page1.Name = "Page1", ;
    Page2.Caption = "Preview", ;
    Page2.Name = "Page2"

  ADD OBJECT form1.pgfcrystal.page1.oledesign AS olecontrol WITH ;
    Top = 8, ;
    Left = 11, ;
    Height = 421, ;
    Width = 517, ;
    Name = "oleDesign"

  ADD OBJECT form1.pgfcrystal.page2.olepreview AS olecontrol WITH ;
    Top = 8, ;
    Left = 11, ;
    Height = 421, ;
    Width = 517, ;
```

```
  Name = "olePreview"

PROCEDURE Init
  LPARAMETERS tcReport

  WITH This
    .WindowState = 2
    .oCrystal = CREATEOBJECT("CrystalDesignRuntime.Application")
    IF EMPTY(tcReport)
      .oReport = .oCrystal.NewReport()
    ELSE
      .oReport = .oCrystal.OpenReport(tcReport)
    ENDIF

    WITH THis.pgfCrystal.Page1.oleDesign
      .DisplayToolbar = .T.
      .DisplayFieldView = .T.
      .ReportObject = ThisForm.oReport
    ENDWITH

    WITH .pgfCrystal.Page2.olePreview
      .EnableExportButton = .T.
      .EnableProgressControl = .T.
      .EnableAnimationCtrl = .F.
      .ReportSource = ThisForm.oReport
    ENDWITH
    .Visible = .T.
  ENDWITH
ENDPROC

PROCEDURE Resize
  WITH This
    .pgfCrystal.Top = .Top + 5
    .pgfCrystal.Left = .Left + 5
    .pgfCrystal.Width = .Width - 10
    .pgfCrystal.Height = .Height - 10
  ENDWITH

  WITH This.pgfCrystal
    .Page1.oleDesign.Top = .Top + 3
    .Page1.oleDesign.Left = .Left + 3
    .Page1.oleDesign.Width = .Width - 18
    .Page1.oleDesign.Height = .Height - 18
  ENDWITH

  WITH This.pgfCrystal
    .Page2.olePreview.Top = .Top + 3
    .Page2.olePreview.Left = .Left + 3
    .Page2.olePreview.Width = .Width - 18
    .Page2.olePreview.Height = .Height - 18
  ENDWITH
ENDPROC

PROCEDURE pgfcrystal.Page1.Activate
  This.Refresh()
```

```
  ENDPROC

  PROCEDURE pgfcrystal.Page2.Activate
    This.olePreview.ViewReport()
  ENDPROC
ENDDEFINE
```

Distribution

Many people are confused about distributing their reports. There are three issues to be aware of, RPT files, Crystal Reports runtime libraries, and support DLLs. Be aware that the exact file and location of these files varies depending on the version of Windows used. Some DLLs also need to be registered. Crystal Reports ships with the file Runtime.HLP that lists all the needed files, locations, and registration information.

First, you must distribute the RPT files. Crystal Reports requires each file to be available. You can distribute these separately or compile them into your EXE, then copy them out as needed.

Second, you must know which Runtime libraries to distribute. Generally, you'll need the CRAXDRT.DLL. Look at the Runtime.HLP to find a listing of needed files.

Finally, you need to deal with support DLLs. There are different DLLs needed depending on features used, export options, graphing etc. For example, ODBC data access uses different DLLs than OLEDB/ADO access. Again, refer to Runtime.HLP for a list of the exact files you need.

Licensing

Crystal Decisions provides a number of licensing options, depending on your distribution needs. The licensing plans described here were in affect at the time this document was written in March, 2002. You should check with Crystal Decisions to verify that licensing terms have not changed. The simplest of these licensing programs is the Developer Edition Licensing. This is the standard licensing scheme and gives you royalty free distribution of canned reports provided the reports run on the desktop.

The Report Creation API License allows you to create reports at runtime, either through code or the embeddable designer. Your reports must still run on the desktop. The cost for this license is $199 per user.

If you are using web-based reporting, the reports are run on a server, or the reports are distributed via some kind of scheduling mechanism, you'll need to investigate a concurrent licensing plan. The costs for these plans are based on the number of concurrent users and the type of distribution you are using. Contact Crystal Decisions for specific licensing options and costs.

Licensing information is available on your Crystal Reports CD in the License.HLP file, from the web at www.crystaldecisions.com/crystalreports/licensing, or from a Crystal Decisions sales rep at 1-800-877-2340.

Resources

Crystal Reports ships with several files that are aimed at the developer. The first is CrystalDevHelp.CHM. This file is a good place to start when you first begin working with Crystal Reports.

In addition, several PDF files are located in the Docs folder on the distribution CD. These files are not installed with the product. I've copied the entire directory to my harddrive so it is always accessible.

Finally, there are whitepapers, FAQS, KB articles, and more on the web at www.crystaldecisions.com.

Conclusion

Crystal Reports provides a robust and complete reporting environment for any data source through the RDC. It is easy to provide canned reports or to create them from scratch either programmatically or via a runtime designer. Keep in mind that the use of some components requires additional licensing from Crystal Decisions.