crystal decisions™

Crystal Reports

Using the ActiveX component through the Foundation Classes of Microsoft Visual C++ 6

Overview

The Crystal Reports ActiveX Component was actually intended to be used in Visual Basic and most of the installed-base uses it this way. However, because this component adheres to the ActiveX architecture, it can be used in any programming language that supports ActiveX. This of course includes Microsoft Visual C++. Visual C++ used to be reserved for high-end software engineering, but Visual C++ 6 along with the Microsoft Foundation Classes brings a nice interface that's usability is attracting corporate developers as well. This document will explain—at the corporate developer level—how to use the Crystal Reports ActiveX Component in VC++. This document will assume that you are using 32-bit Crystal Reports version 6 or 7, but will work in almost any 32-bit version of Crystal Reports.

Contents

The ActiveX Component in VB vs. VC++

The Crystal Reports ActiveX Component was written in Visual C++. Because of this a different layer (or API) is exposed when using this component through

Visual C++. The ActiveX component is built as a wrapper class for the Crystal Reports Print Engine (CRPE32.DLL). The first layer is the one exposed when used through Visual C++ and includes more functions. The next layer on top of this has fewer functions, and this is the layer exposed when the component is used through Visual Basic (a "higher-level" layer). For the most part, the interface is the same through either language, but you will find syntax differences. The main one difference is that through Visual Basic you set properties and call methods (functions) of the component at runtime, whereas in Visual C++, there are no runtime properties, everything is set through functions. This document will not attempt to expose each of these differences, but rather how to use this component through VC++ alone.

Where Do I Want My Control?

In an MFC-based Visual C++ application, the ActiveX component must reside in a container that is capable of holding it. Most of the time, this is a CDialog-derived class, but can also theoretically be held in any CWnd-derived class such as CView. For the purposes of this document, we will assume the component will be held inside a CDialog-derived class. This means that either your application is a dialog-based application, or that you are using document/view architecture but derive at least one of your views from CFormView.

Creating the Application

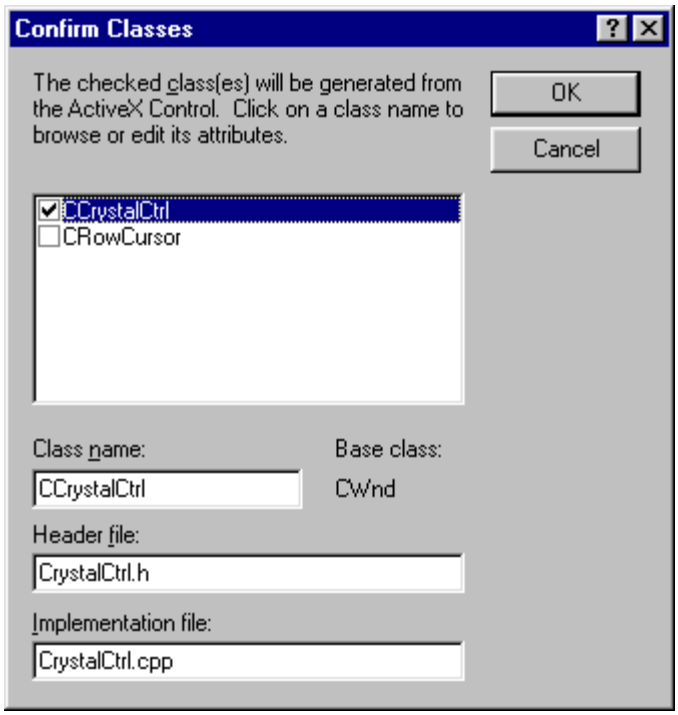
From the Microsoft Visual C++ 6 environment, select File | New. From the list of available project types, choose MFC AppWizard (exe). For the project name type Crw. Click on the Ok button to start the AppWizard. You could choose any of the three types of applications outlined here, but for the purpose of this document, choose Single Document and then click Next 5 times to advance to the Step 6 of 6 window of the AppWizard. We'll leave the rest of the defaults for the AppWizard as is for simplicity reasons. From the list of classes, select CCrwView. From the drop-down menu below, change it's base class to CFormView and then click Finish. If the New Project Information dialog pops up, click Ok to close it. You may now build this application and execute it to test it. From the Build menu, select Execute Crw.exe. A dialog will pop up asking you if you want to build the file, click Yes to this. Once this is finished building, the application will execute and you can see the skeleton app. Exit the Crw application when finished and return to the Visual C++ environment.

Adding In the Crystal Reports ActiveX Component

The first window that you should see is the dialog template resource for the CFormView class that we created. Notice that this is what was shown in the main window when you executed the Crw.exe application. Click on the existing static control on the dialog template that says TODO: Place form controls on this dialog and press the Delete key to get rid of it. Now right-click in the middle of the dialog template and select Insert ActiveX Control from the pop-up menu. This will give you a list of all ProgIDs (programming identifiers) that are registered on the system. You might have to scroll down to find Crystal Report Control. Note: make sure you are selecting Crystal Report Control and not Crystal Report Smart Viewer. Click Ok and you should be back to the dialog template window and you will see a Crystal Report object in the top left-hand corner of the dialog template. Note: the button-like object you see will not be displayed at runtime.

Right-click on this control and select ClassWizard from the ensuing pop-up menu. Move to the Member Variables tab. We haven't renamed the identifier of the component yet, so it is currently the default, which is IDC_CRYSTALREPORT1. Select this Control ID and click on Add Variable. You will now get the following dialog:

Click Ok to this. This means that even though we've inserted an instance of the control onto the form, we haven't actually added the Crystal Report Component class to the project. Visual C++ will now do this for you. The next window you see shows you the classes associated with the component. We don't need the CRowCursor object, so uncheck that. The dialog should look like this:



Click Ok to continue. We are now prompted for the name of the member variable that we wish to add. Visual C++ has already filled in m_ for us as this is the standard naming convention for member variables. Change it to m_Report and click Ok. Click Ok again to close the ClassWizard. Now we have a control added to our dialog template and a variable with which to refer to it in code.

Manipulating the Component

There are two ways to manipulate this component: at design-time or runtime. For simplicity reasons, let's set the filename of the Crystal Report that we are going to display at design-time. Rick-click on the Crystal Report object on the

dialog template and select Crystal Properties CrystalReport Object. From the ensuing pop-up properties dialog, change to the Control tab. Set the ReportFileName by either typing in the full path of an existing RPT file or by clicking on the "…" button and browsing to an RPT file.

For the first part of this example, let's assume that this report is using a Crystal Reports native driver and requires no logon. In this case, all we have to do is tell the Crystal Reports component to go ahead and print the report. Let's assume we want to do this on the click of a button (you could also do this on the File | Print menu option as well if you wanted to, but for this example, we'll just add a button). You should have a floating Controls toolbar on your screen somewhere. If not, turn it on by right clicking on the toolbars at the top and checking off the Controls option from the pop-up menu. On the Controls toolbar, click on the button object, and then click once more in an empty area on the dialog template. Now you have a button.

Click once on the button and then type Print Report and press enter. Now double-click on the button. We are prompted for the name of a function. This is the function that will be called when we click on the button at runtime. Change the function name to OnPrintReport and click Ok. We are taken directly to the code for this function. All we have to do is call one function here to have the report display. Delete the line // TODO: Add your control notification handler code here and add code so your function looks like this:

```
void CCrwView::OnPrintReport()
{
	m_Report.SetAction(1);
}
```

Now from the Build menu, select Execute Crw.exe. It will prompt to re-build, click Yes to this. When the build is finished the Crw application will be run. You will see the button we created on the window. Click on it and the report window will pop up displaying your report. You can now close the pop-up window and then click the X to close the Crw application or close the main Crw application first and it will close the report window for you.

This is the most basic of applications, but is intended to be a starting point from which you can add more functionality. The rest of this report will briefly outline some other basic features of the Crystal Reports ActiveX Component.

Passing a Parameter

If you return to the dialog template and right-click on the Crystal Reports object and select Crystal Properties CrystalReport Object, you can change the ReportFileName from the Control tab. Change this to a report with a parameter, ensuring that this report does not having Save Data with Report turned on. Run your application again and you will see that you are now prompted for the value of that parameter before the report is displayed. If you wanted to pass that parameter at runtime and not see that prompting dialog, you would add the following line to your function so the function looks like this:

```
void CCrwView::OnPrintReport()
{
```

```
        m_Report.SetParameterFields(0,"ParameterName;Value;Tr
ue");
        m_Report.SetAction(1);
}
```

You would replace ParameterName with the name of your parameter and replace Value with the value you want to pass (probably not generated until runtime). The True tells the component not to prompt for the parameter value.

Passing a Formula

Passing formulas work much the same way with just a different syntax as shown below:

```
void CCrwView::OnPrintReport()
{
        m_Report.SetFormulas(0,"FormulaName=Value");
        m_Report.SetAction(1);
}
```

Note that if that value is a string, you need to pass it in with single quotes like this:

```
m_Report.SetFormulas(0,"FormulaName='Value'");
```

If the value is a date, you need to pass it in like this:

```
m_Report.SetFormulas(0,"FormulaName=Date(yyyy,mm,dd)");
```

Changing the Database to Report Off

If you use a native driver, you can change the database that you are reporting off of at runtime, as long as the database that you are changing to has the same table-structure. If the table-structure might be a little different (such as having an extra field in the database), ensure that you have Verify On Every Print turned on for the report from the Report Designer. The syntax for changing database files is as follows:

```
m_Report.SetDataFiles(0,"c:\\directory\\new database.mdb");
```

Connecting Using ODBC

The last function that this document will explain is the connection string. If you are using ODBC, and accessing a database such as SQL Server, Oracle, or Informix, you will need to logon to the database server before your report can query it. The Crystal Reports RPT file will store the ODBC DSN that you used when you created the report, the username, and default database. All you need to pass here if you aren't changing any of these things at runtime is the password. The syntax for this is as follows:

```
m_Report.SetConnect("PWD=xxx");
```

If you wanted to change the DSN, you would just include it in the connection string along with the new user id.

```
m_Report.SetConnect("DSN=newDSN;UID=xxx;PWD=xxx");
```

Unlike using our native driver, to change database you would not call the SetDataFiles function. With ODBC, you would change the connection string. The SetDataFiles function when used with ODBC will change *table* names, not database names. See below:

```
m_Report.SetConnect("UID=xxx;PWD=xxx;DBQ=<CRWDC>DBQ=databas
e_name");

m_Report.SetDataFiles(0,"database.owner.tablename");
```

Notice that when changing tables above, the syntax was database.owner.tablename. This is the syntax for SQL Server. If you were using a different database such as Oracle, the syntax would be userid.tablename. You can refer to the Crystal Reports developer's help for more information on the syntax for other databases.

Contacting Crystal Decisions for Technical Support

We recommend that you refer to the product documentation and that you visit our Technical Support web site for more resources.

Self-serve Support:

http://support.crystaldecisions.com/

Email Support:

http://support.crystaldecisions.com/support/answers.asp

Telephone Support:

http://www.crystaldecisions.com/contact/support.asp