Note: This chapter is taken from *Crystal Reports 8.5: The Complete Reference*. As such, it may refer to chapters and material in this book, and generally refers to Crystal Reports version 8.5, not 9.

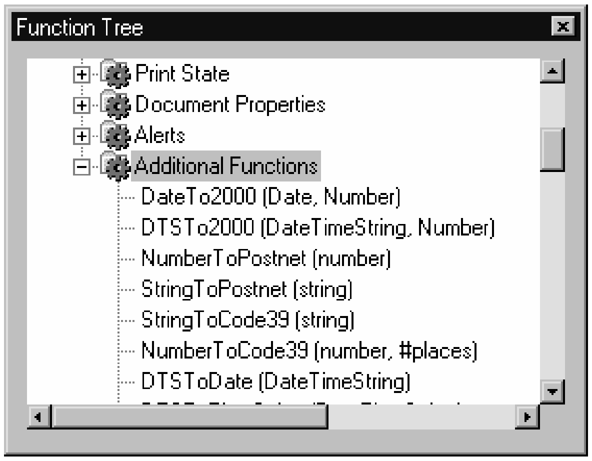
Creating User Function Libraries with Visual Basic

The more sophisticated your reports get, the more sophisticated your use of Crystal Reports formulas will become. Although the formula language's built-in functions will satisfy much of your sophisticated reporting requirements, you may eventually encounter situations in which you need extra features that aren't available in Crystal Reports 8.5. This often occurs when a large number of report designers in your company need a business-specific function, such as a way in which to determine the number of days a problem ticket has been open, excluding weekends and company holidays. In some cases, you may be able to create these specialized formulas with a great deal of formula coding. In other cases, the capabilities of the built-in formula language just won't provide the necessary flexibility (perhaps your custom function will need to look up dates in a database of company holidays).

You have several approaches for solving this potential problem. If you're using the Report Designer Component (RDC) to integrate your report with a Visual Basic program (covered in Chapter 27), you can use all the power of Visual Basic to create a value and place it in a text object in the report. But, what if you want to be able to provide a company-specific calculation to anybody using the regular Crystal Report Designer, without having to integrate the report with a VB program? With a User Function Library (UFL), you can extend the capabilities of the Crystal Reports formula language with a Windows development language, such as Visual Basic.

User Function Library Overview

In the Crystal Reports Formula Editor, the Function Tree box divides functions into logical groups. Arithmetic functions are combined in their own group, date/time functions are located together, and so forth. But, if you look at the bottom of the Function Tree box, you'll see listed in an Additional Functions category an extra collection of functions that run the gamut from Year 2000 date conversions to financial functions, to a sound-alike function (Soundex).



All functions in this Additional Functions category are considered UFLs and are provided to Crystal Reports by way of external dynamic link library (.DLL) files. By creating a Windows .DLL file using a Windows development language, you can extend the power of the Crystal Reports formula language by adding your own functions to the Additional Functions category. Each of these DLLs can expose one or more functions to the Crystal Reports formula language.

Crystal Reports installs several UFL DLLs by default. As with previous versions, when you initially install Crystal Reports 8.5, all the functions in the Additional Functions list are supplied by these .DLL files installed in the \CRYSTAL directory beneath the standard Windows installation directory (typically \WINDOWS or \WINNT). If you look in this directory, you'll recognize UFL filenames by the first few characters. All UFL filenames begin with the letters *U2*. In many (but not all) cases, the third character is the letter *L*, with the remaining characters describing the function of the UFL. All UFLs are DLLs that have a file extension of .DLL.

Note: Crystal Reports 8.5 uses 32-bit UFLs only. In previous versions of 16-bit Crystal Reports, you could also have 16-bit UFLs. The filenames for these 16-bit UFLs contained the letter *L* instead of the character *2* in the second character position.

If you wish to create your own UFLs, you have several Windows language choices. The original Crystal Reports language requirement for UFL design was the C language. However, starting with Crystal Reports 6, you have also been able to create UFLs with any language that can create automation servers that conform to Microsoft's Component Object Model (COM). These languages include Visual Basic and Delphi, among others.

UFLs in C and Delphi

If you choose to use C to develop UFLs, the approach is dramatically different from the COM Automation Server approach with Visual Basic. There is plenty of helpful information in Developer's Help (CrystalDevHelp.chm). Look in the table of contents for "Creating User-Defined Function in C". You can also find sample UFL source code in C. Look in *Crystal Reports program directory*\Developer Files\include for UFLSAMP1.C and UFLSAMP2.C.

For information on creating UFLs with Delphi, look in the Developer's Help table of contents for "Creating User Defined Functions in Delphi 3.0".

UFLs in Visual Basic

A UFL that is installed by default with the 32-bit version of Crystal Reports is U2LCOM.DLL. This UFL doesn't actually expose any functions in the Formula Editor by itself. Instead, it acts as a gateway to COM Automation Servers that expose additional functions. By creating these automation servers in a COM-compatible language, you can expose external functions to the Crystal Reports formula language. Using Visual Basic 5 or 6, or the 32-bit version of Visual Basic 4, you can create automation servers that expose additional functions to Crystal Reports through U2LCOM.DLL.

Caution: Although you can use C to develop 16-bit UFLs for use with previous versions of Crystal Reports, Visual Basic UFLs are only recognized by the 32-bit version of Crystal Reports. As such, you must use a 32-bit Visual Basic environment to create them.

Creating a VB UFL requires a specific set of steps. You must start with an ActiveX DLL project to create the automation server. Project names must start with a certain set of characters for U2LCOM.DLL to recognize them. And, functions inside the project's class modules must be declared and created a certain way. When you compile the project into a .DLL file, Visual Basic will automatically register the automation server on your PC. Then, when you start Crystal Reports, U2LCOM.DLL will recognize the new automation server and expose its functions in the Additional Functions list inside the Formula Editor. After you create and test the UFL, you need to use the Visual Basic 6 Package and Deployment Wizard, or another similar distribution mechanism, to install and register the automation server on the end user's computer.

Note: The remainder of this chapter concentrates on developing UFLs with Visual Basic 5 and 6. The steps to create UFLs with 32-bit VB 4 are slightly different. Navigate in the Developer's Help table of contents to “Creating User-Function Libraries in Visual Basic”, then follow lower-level categories “Using Visual Basic 4.0.”

Creating the ActiveX DLL

Visual Basic refers to a COM-based automation server as an *ActiveX DLL*. When you create a new project in VB, choose this project type. A new project will be opened and a Class Module code window will immediately be displayed. Creating the functions to be exposed in the Formula Editor's Additional Functions section is now as simple as declaring public functions here in the class module.

Adding Functions to the Class Module

The CD-ROM accompanying this book includes a very simple sample UFL project that accepts a date value as its only argument and returns the next weekday after the date that is supplied. This might be used to determine when a product could next be shipped, or when a customer could next expect a return call from a sales representative. By adding a database lookup in the routine, you might also be able to include company holidays in the routine, so that Memorial Day or New Year's Day won't be returned as the next weekday.

This function is simply added to the class module as it would be to a form or a .BAS module in a regular Visual Basic .EXE project. Examine the following sample code:

```
Public Function NextWeekDay(DateArg As Date) As Date
   Do
      DateArg = DateArg + 1
   Loop Until Weekday(DateArg) <> 1 And Weekday(DateArg) <> 7
   ' Could add logic here to look up date in database to
   ' see if it falls on a company holiday, etc.
   NextWeekDay = DateArg
End Function
```

Note: Because Crystal Reports 8.5 includes Do Loop functionality, as well as improved date functions, you may be able to do this type of formula just as easily directly in the report, rather than using an external UFL. This formula is included here to show an example of how a UFL is created.

Notice that the function is declared Public. This is a requirement for the function to be exposed to Crystal Reports. You'll also notice that the arguments to the function aren't specifically designated as ByRef or ByVal. It doesn't matter which method you use to pass arguments; either type of argument, or no type at all, is fine. This function simply uses a Do loop to add one day to the date until the day doesn't fall on a Saturday or Sunday. That date is then set as the function's return value.

You can create as many functions as you need inside the same class module. Just make sure that the functions are named with unique names that won't conflict with other Crystal Reports built-in functions. For example, if you create a UFL function named ToText, it will conflict with the existing Crystal Reports function of that name. Also, there are several reserved words and function names that you cannot use for your function names. In particular, U2LCOM.DLL makes special use of these function names:

UFInitialize
UFTerminate
UFStartJob
UFEndJob

And, COM itself uses the following reserved words, so you can't use any of these for your own function names:

QueryInterface
AddRef
Release
GetTypeInfoCount
GetTypeInfo
GetIDsOfNames
Invoke

Tip: Function-name prefixing determines exactly how your UFL function names will appear in the Formula Editor. This is discussed in detail later in the chapter.

Your functions can accept as many arguments as necessary, and the arguments can be of any standard Crystal Reports data type. The actual names you give the arguments inside your VB function will be displayed in the Formula Editor as arguments for the Crystal Reports function. This requires you to think carefully about what argument names you want to use in your function. Although you'll notice straightforward argument names like *str* and *date* inside built-in Crystal Reports functions, you're limited to argument names that don't conflict with VB reserved words. With the sample application, an argument named *date* would be consistent with other Crystal Reports functions, but this conflicts with a VB reserved word and thus can't be used as a function argument. You'll notice that *DateArg* is used instead, and it will appear in the Formula Editor as the argument for this function.

When you return the function's value, Crystal Reports automatically converts several VB data types into correct types for Crystal Reports formulas. In particular, functions that return VB's various numeric data types, regardless of precision (integer, long, and so on), will be converted to simple numbers in Crystal Reports. Date, string, and Boolean VB data types will be converted to the corresponding data types in Crystal Reports. You can also pass an array *to* Visual Basic from Crystal Reports. Visual Basic will recognize the array data type and allow you to manipulate the various array elements inside your function. However, the function *cannot* return an array to Crystal Reports—you can only return a single value.

Tip: Testing your ActiveX DLL functions is made somewhat more complicated by the ActiveX DLL integrated development environment (IDE). Microsoft recommends creating a project group and actually calling the automation server functions from another project that's loaded in the VB IDE at the same time. You can also choose to create a regular .EXE project and a simple form to test your UFL functions. You can then create the ActiveX DLL project and just copy your already tested functions from the old project to the new one.Another method of debugging involves “running” your ActiveX project, and then starting a second instance of Visual Basic. You’ll find that the functions exposed by the ActiveX project in the first VB instance will appear in the second instance’s References list. It’s even possible to run the ActiveX project and then start Crystal Reports. You’ll see the function exposed in the Formula Editor Additional Functions category.

Special-Purpose Functions

As mentioned previously, several reserved words can't be used for your function names. Four of those names are reserved for special functions that your UFL can use for special UFL processing during your report, as discussed in the following sections.

UFInitialize

This function is called just after the DLL is loaded into memory, before any functions are actually performed:

```
Public Function UFInitialize () As Long
```

One-time initialization of variables, opening of files, or other initialization code can be placed here. Return a value of zero (0) to indicate that the function completed properly. If any errors or problems occur, return any nonzero value to indicate to Crystal Reports that initialization failed.

UFTerminate

This function is called just before the DLL is unloaded from memory, after any other functions are already completed:

```
Public Function UFTerminate () As Long
```

Use this function to deallocate any variables or execute any other cleanup code that you'd like to perform. Return a value of zero to indicate that the function completed properly. If any errors or problems occur, return any nonzero value to indicate to Crystal Reports that UFTerminate failed.

UFStartJob

This is similar to UFInitialize, except that it occurs once for every different report that may be processing:

```
Public Sub UFStartJob (job As Long)
```

You can determine the report that is being started by looking at the job argument that is passed to the function. Use this function for any per-job initialization. Notice that this is actually a subroutine, not a function. You cannot set any return code if errors occur in the routine.

UFEndJob

This function is called when the current job finishes (when all report pages are finished processing, but before a new UFStartJob and before UFTerminate):

```
Public Sub UFEndJob (job As Long)
```

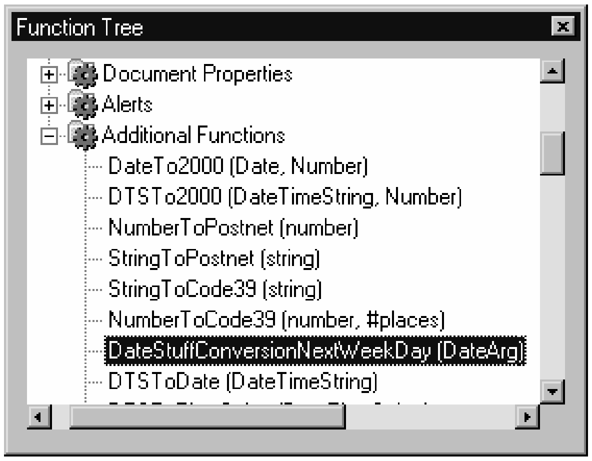
The actual job that is ending is passed with the job argument. Place any per-job cleanup code here. Notice that this is actually a subroutine, not a function. You cannot set any return code if errors occur in the routine.

Naming and Saving the DLL

The combination of your function names, class module name, and ActiveX DLL project name all determine how (and if) your functions appear in the Formula Editor. In particular, you *must* start your project name with the characters "CRUFL". The choice of the remaining characters is up to you, allowing you to give the project a meaningful name. The CRUFL prefix indicates to U2LCOM.DLL that functions in this automation server will be exposed to Crystal Reports. If you don't prefix your project name with these characters, the functions won't appear in the Formula Editor. Name your project as you would any other VB project. Select the project in the Project Explorer window and give the project a name in the Properties window.

After you properly name the project, you need to actually create the automation server .DLL file. Choose File | Make CRUFL*xxx*.DLL from the VB pull-down menus. You'll be prompted to choose a location for the actual .DLL file. To be consistent with other UFL files, you can choose to save it in the \CRYSTAL directory below your regular Windows directory. This is not absolutely required, however, because the actual file location is not significant to U2LCOM.DLL. Instead, the DLL is *registered* by VB when the .DLL file is created. The Registry setting contains the actual location of the .DLL file, so it can be located in any drive or directory you choose.

After you create and register the DLL, start Crystal Reports. Open an existing report or create a new report. Open the Formula Editor and look at the functions that appear in the Additional Functions category. You'll see that your functions and their arguments now appear in the Formula Editor.



DateTo2000 (Date, Number)
DTSTo2000 (DateTimeString, Number)
NumberToPostnet (number)
StringToPostnet (string)
StringToCode39 (string)
NumberToCode39 (number, #places)
DateStuffConversionNextWeekDay (DateArg)
DTSToDate (DateTimeString)

Error Handling

Crystal Reports automatically handles certain error conditions that may occur. For example, if you declare a function argument as numeric in your VB UFL, but pass a string in the Formula Editor, Crystal Reports automatically catches the error. You don't have to provide any special code to trap these kinds of errors.

Even though you design error-handling logic into your UFL functions, you may want to return function-specific error text to Crystal Reports from within your function. Declare and set the UFErrorText public string variable to accomplish this. Examine the following sample code:

```
Public UFErrorText As String
...
On Error GoTo NextWeekDay_Error
...
NextWeekDay_Error:
UFErrorText = Err.Description
```

Here, UFErrorText is declared in the general declarations section of the class module and is set if any error occurs inside the code. Whenever you set the value of this variable inside a function, Crystal Reports will stop the report, display the formula calling the function in the Formula Editor, and display the contents of UFErrorText in a dialog box. Because setting the value of this variable is what triggers the Crystal Reports error, use this only for error handling. Attempting to use UFErrorText to store other values or attempting to read its contents may cause your report formulas to stop prematurely. Also, U2LCOM.DLL resets the value of the variable during report operations, so you can't reliably read its contents inside your code.

Function-Name Prefixing

If you look back at the preceding illustration of the Function Tree, you'll notice that the function name added to Crystal Reports is quite long, and it differs from the actual name you gave the function in your VB code. By default, U2LCOM.DLL uses *function-name prefixing* to actually combine the project name (without the CRUFL characters), class module name, and the function name to create the function that appears in the Formula Editor. For example, the sample UFL application on the companion CD-ROM is named as follows:

- Project name: CRUFLDateStuff
- Class module name: Conversion
- Function name: NextWeekDay (with the DateArg argument)

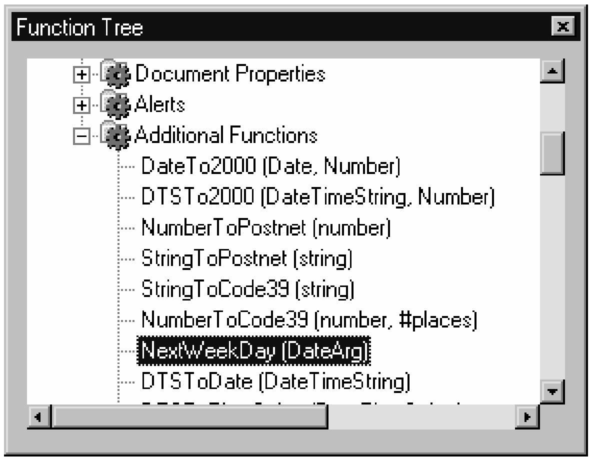
The resulting function appears as DateStuffConversionNextWeekDay(DateArg) in the Formula Editor.

Function-name prefixing is designed to minimize the chance of duplicate function names appearing in the Formula Editor. If you inadvertently give a UFL function the same name as a preexisting Crystal Reports function (or another function created by a different UFL), errors or unpredictable formula behavior can result. By automatically adding the project and class module names to your function names, the chances of duplicate function names is reduced. However, as you can see, report designers are also presented with rather lengthy function names.

If you wish, you can turn off function-name prefixing, so only the function names themselves will be shown in the Formula Editor, without the project or class module names appended to them. Just be extra careful to make sure you don't use function names that already exist in the Formula Editor. To turn off function-name prefixing, declare the public Boolean variable UFPrefixFunctions in the general declaration section of the class module. Then, set the variable to false in the class's Initialize function. Here's some sample code:

```
Public UFPrefixFunctions As Boolean
...
Private Sub Class_Initialize()
   UFPrefixFunctions = False
End Sub
```

By setting the UFPrefixFunctions variable to false and then creating (or re-creating) the DLL and restarting Crystal Reports, function names will appear in the Formula Editor without prefixing.



Caution: Don't confuse the class's Initialize event with the UFInitialize function that's reserved for use by Crystal Reports. The class's Initialize event is accessed by choosing Class in the object drop-down list in the Code window and then choosing Initialize in the procedure drop-down list.

Distributing the UFL

Although Visual Basic automatically registers your automation server DLL when you choose File | Make CRUFL*xxx*.DLL from the pull-down menus, you need to distribute the DLL to any other report designer who will want to use the functions of the UFL. You also may want to distribute the UFL as part of a larger VB project that integrates reports, using one of the integration methods discussed earlier.

To simply distribute the UFL to other report designers who will be using the Crystal Reports designer, it's probably best to use the Package and Deployment Wizard with VB 6, the Application Setup Wizard with VB 5, or another distribution tool. This creates a setup program that will install your UFL on the target system and register the automation server on the designer's computer. If you simply wish to pass the .DLL file to a designer, they need to manually register the .DLL file so that U2LCOM.DLL will recognize it. VB provides the REGSVR32.EXE program to register an automation server from the command line.

Although U2LCOM is typically installed automatically with 32-bit Crystal Reports, make sure that the target system has it installed in the \CRYSTAL directory below the Windows program directory. Without this UFL file, no functions exposed by the automation server will appear in the Formula Editor.

If you want to include your UFL in reports that are integrated with a VB program, you'll want to make sure the .DLL file is distributed and registered when the whole VB application is installed by your end users. This can be accomplished simply by adding the UFL Automation Server you created earlier to your new project by using Project | References from the VB pull-down menus. The name of your UFL will appear in the list of available references. Check the box next to the automation server name to add it to the project. When you later distribute the Visual Basic project, the distribution tool will automatically include the .DLL file and register it on the target computer.