Report Objects

Doug Hennig

FoxPro developers have wanted an object-oriented report writer since VFP 3 was released. Although the classes presented in this article don't have a pretty visual designer tool, they do expose the FRX as a set of objects and provide a simple way to create reports programmatically.

I believe it was Microsoft's Robert Green who, when asked at the 1997 DevCon when we could expect an object-oriented report writer for VFP, said something like the following: "You asked for it for VFP 3 and we didn't build one. You asked for it for VFP 5 and we didn't build one. Take a hint". I don't know whether Microsoft will or won't ever create an object-oriented report writer for VFP, but I really needed the ability to create reports programmatically *now*, so I created the set of classes described in this article.

First, a little background. Since FoxPro 2.0, the FRX file has been a table. This wonderful fact means we can use the world's greatest database engine (no, not Access2000 <g>) to manipulate our reports programmatically. Why would you want to do this? Sometimes, you don't know in advance what should be in a report or how it should be formatted, and although the Report Designer is available in a runtime application, you don't want any but your most advanced users using it. Sure, you can use automation to create reports in Word or Excel, or use a third-party reporting tool like Crystal Reports, but nothing is easier to manipulate than a table, so why not create FRX files programmatically? Well, I can think of one good reason: they're ugly as sin. Although the FRX structure is documented (you can run a report that comes with VFP to print the FRX file structure), different report object types have different uses of the fields in this table structure, and that's not well documented. Also, there are a lot of fields, making for some pretty long INSERT INTO statements (I know – I've written tons of them over the years). Also, the size and position values are in 10,000th of an inch, and there are a weird bunch of "fudge" factors you have to apply to these numbers. After struggling for several hours with some really complex code that created an FRX, I finally screamed "I want report objects" (this isn't metaphorical; I really did scream this <g>). Unfortunately, there was no one around to hear me, so I created them myself.

The report object classes I created are all in SFREPOBJ.VCX, and all are based on SFCustom (in SFCTRLS.VCX, our library of base classes). Each uses SFREPOBJ.H as its include file; constants like cnBAND_HEIGHT are much easier to understand than hard-coded values like 2083.333.

I'm not going to explain the complex structure of an FRX file in this article; in addition to the documentation that comes with VFP, other FoxPro developers such as Markus Egger and Jefferey Donnici have written articles on this subject in this and other journals. I also don't have room to show much of the code in these classes (you can look at the code in the VCX yourself), but that's not really important anyway. You can consider these classes to be "black boxes"; you don't have to know how they work, just how to use them.

SFReportFile

The only class you actually instantiate directly is SFReportFile. This class represents the report file, and has properties and methods that expose the Report Designer interface programmatically. It'll instantiate objects from other classes when necessary (such as when you add a field to the report).

Table 1 shows the public properties of SFReportFile. Other than cReportFile and cUnits, the public properties simply represent the same options for the report you see in the Report Designer. Each has an Assign method that prevents values of the wrong data type or range from being stored. cUnits needs a little explanation. Sometimes, it's easier to express report units (such as the horizontal and vertical position of an object) in characters and lines. For example, if you have a 30-character field, it's easier to specify the width for it on a report as 30 characters rather than figuring out how many inches it should be. All position and size properties in all objects we'll look at should be expressed in the units defined in cUnits.

Property	Purpose
cFontName	The default font for the report (the default value is "Courier New")
cReportFile	The name of the report file to create
cUnits	The unit of measurement: "C" (characters), "I" (inches), or "M" (centimeters) (the default value is "C"; constants are defined for these values in SFREPOBJ.H)

lPrintColumns	.T. to print records in columns across the page, .F. to print top to bottom then in columns
lPrivateDataSession	.T. to use a private datasession with this report
lSummaryBand	.T. if the report has a summary band
lTitleBand	.T. if the report has a title band
lWholePage	.T. to use the whole page, .F. to use the printable page
nColumns	The number of columns in the report (the default value is 1)
nColumnSpacing	The spacing between columns
nFontSize	The default font size for the report (the default value is 10)
nLeftMargin	The left margin for the report

Table 1. Public properties of SFReportFile.

Here's some sample code that instantiates an SFReportFile object, specifies the name of report to create, indicates that the report will have a summary band, and sets the default font for the report.

```
loReport = newobject('SFReportFile', 'SFRepObj')
loReport.cReportFile  = 'CUSTREP.FRX'
loReport.lSummaryBand = .T.
loReport.cFontName    = 'Arial'
```

SFReportFile has four public methods: GetReportBand(), CreateGroupBand(), CreateVariable(), and Save(). GetReportBand() returns an object reference to the specified band so you can set its properties or add objects (such as fields and lines) to it. CreateGroupBand() creates group header and footer bands, but doesn't set any properties for these bands; you'll need to use GetReportBand() to get either the header or footer band and then set the properties and add objects. CreateVariable() creates a report variable and returns an object reference to it so you can set its properties (such as the initial value and when to reset it). Save() is used after you've defined the entire report and want to create the FRX file, so it's usually the last thing you do with the SFReportFile object.

Report Bands

SFReportFile creates an object for each band in a report by instantiating an SFReportBand object into a protected property. For example, the Init() method uses the following code to create page header, detail, and page footer band object automatically, because every report has at least these three bands:

```
with This
  .oPageHeaderBand = newobject('SFReportBand', ;
    .ClassLibrary, '', 'Page Header', This)
  .oDetailBand     = newobject('SFReportBand', ;
    .ClassLibrary, '', 'Detail', This)
  .oPageFooterBand = newobject('SFReportBand', ;
    .ClassLibrary, '', 'Page Footer', This)
endwith
```

The Init() method of SFReportBand accepts two parameters: the band type (which it puts into the protected cBandType property) and an object reference to the SFReportFile object (which it puts into oReport) so it can call back to some SFReportFile methods if necessary.

In addition to the three default bands, you can create additional bands in one of two ways. To specify that the report has a title or summary band, set the lTitleBand or lSummaryBand property to .T. To create group header and footer bands, call the CreateGroupBand() method. Groups are automatically numbered in the order they're defined; an possible enhancement would allow groups to be reordered.

The GetReportBand() method returns an object reference to the specified band. In the case of a group header or footer band, you also specify which group number you want the band for. Here's some sample code that sets the height of the page header and detail bands.

```
loPageHeader = loReport.GetReportBand('Page Header')
loPageHeader.nHeight = 8
loDetail = loReport.GetReportBand('Detail')
loDetail.nHeight = 1
```

By the way, you don't have to hard-code band names as I've done in this example. All band names are defined as constants in SFREPOBJ.H, the include file for all classes in SFREPOBJ.VCX. For example, ccBAND_PAGE_HEADER is defined as "PAGE HEADER", so you could use this code:

```
loPageHeader = loReport.GetReportBand(ccBAND_PAGE_HEADER)
```

Table 2 shows the public properties of SFReportBand; as with SFReportFile, they simply expose band options available in the Report Designer as properties. nHeight is expressed in the units defined in the cUnits property of the SFReportFile object; for example, if cUnits is "C", setting nHeight to 8 defines an 8-line high band. One nice feature: if the objects in a band extend below the defined height of the band, the band height will automatically be adjusted when you save the report (the nHeight property isn't changed, but the band's record in the FRX is).

Property	Purpose
cOnEntry	The "on entry" expression for the band
cOnExit	The "on exit" expression for the band
lConstantHeight	.T. if the band should be a constant height
lStartOnNewPage	.T. if this band should start on a new page
nHeight	The height of the band

Table 2. Public properties of SFReportBand.

SFReportBand just has two public methods that you'll use: AddItem() and GetReportObjects(). AddItem() is probably the method you'll use the most in any of the classes; it adds an object to a band, so you'll call it once for every object in the report. Pass AddItem() the object type you want to add: "field", "text", "image", "line", or "box" (these values are defined in SFREPOBJ.H, so you can use constants rather than hard-coded values), and it'll return a reference to the newly created object. GetReportObjects() populates the specified array with object references to the objects added to the band; the objects are ordered by vertical and horizontal position, so they appear in left-to-right and top-to-bottom order. You won't likely use this method yourself, but SFReportFile uses it to output a band and all of its objects to the FRX file.

SFReportGroup is a subclass of SFReportBand that's specific for group header and footer bands. Its public properties are shown in Table 3. The most important one is obviously cExpression, since this determines what constitutes a group.

Property	Purpose
cExpression	The group expression
lPrintOnEachPage	.T. to print the group header on each page
lResetPage	.T. to reset the page number for each page to 1
lStartInNewColumn	.T. if each group should start in a new column
nNewPageWhenLessThan	Starts a group on a new page when there is less than this much space left on the current page

Table 3. Public properties of SFReportGroup.

Here's some code that creates a group, gets a reference to the group header band and sets the group expression, height, and printing properties.

```
loReport.CreateGroupBand()
loGroup = loReport.GetReportBand('Group Header', 1)
loGroup.cExpression          = 'CUSTOMER.COUNTRY'
loGroup.nHeight              = 3
loGroup.lPrintOnEachPage     = .T.
loGroup.nNewPageWhenLessThan = 4
```

Fields and Text

The most common thing you'll add to a report band is a field (actually, an expression which could be a field name but can be any valid FoxPro expression), since the whole purpose of a report is to output data. SFReportField is the class used for fields. However, before we talk about SFReportField, let's look at its parent classes.

SFReportRecord is an ancestor class for every class in SFREPOBJ.VCX except SFReportFile. It has one property that all objects share: nObjectType, which contains the OBJTYPE value for the FRX record of the object (for example, fields have an OBJTYPE of 8). It also has a method called CreateRecord(). This method is called from the SFReportFile object when it creates a report. SFReportFile creates an object from a record in the FRX using SCATTER NAME loRecord BLANK MEMO. This cool addition to VFP creates an object with one property for each field in the record. What's so neat about this is that objects can be passed to methods very easily, and that's what happens here. SFReportFile passes a report record object to the CreateRecord() method of the SFReportRecord object, which then fills in properties of the report record object from values in its own properties. SFReportRecord is an abstract class; it isn't used directly, but is the parent class for other classes. Its CreateRecord() method simply ensures a valid report record object was passed and sets the ObjType property of this object (which will be written to the OBJTYPE field in the FRX) to its nObjectType property.

SFReportObject is a subclass of SFReportRecord that's used for report objects (here, I mean "object" in the "thingy" sense, such as a field, rather than "what you get when you instantiate a class" sense). It has the public properties shown in Table 4, which represent the minimum set of options for a report object. As with other classes, these properties simply expose options available in the Report Designer as properties.

Property	**Purpose**
cAlignment	The alignment for the object: "left", "center", or "right" (constants are defined for these values in SFREPOBJ.H)
cPrintWhen	The Print When expression
lPrintInFirstWholeBand	.T. to print in the first whole band of a new page
lPrintOnNewPage	.T. to print when the detail band overflows to a new page
lPrintRepeats	.T. to print repeated values
lRemoveLineIfBlank	.T. to remove a line if there are no objects on it
lStretch	.T. if the object can stretch
lTransparent	.T. if the object is transparent, .F. for opaque
nBackColor	The object's background color; use an RGB() value (-1 = default)
nFloat	0 if the object should float in its band, 1 if it should be positioned relative to the top of the band, or 2 if it should be relative to the bottom of the band (constants are defined for these values in SFREPOBJ.H)
nForeColor	The object's foreground color; use an RGB() value (-1 = default)
nHeight	The height of the object
nHPosition	The horizontal position for the object
nPrintOnGroupChange	The group number if this object should print on a group change
nVPosition	The vertical position for the object relative to the top of the band
nWidth	The width of the object

Table 4. Public properties of SFReportObject.

The CreateRecord() method first uses DODEFAULT() to get the behavior of SFReportRecord, then it has some data conversion to do. For example, object colors are stored in the PENRED, PENGREEN, and PENBLUE fields in the FRX record, but we want to have a single nForeColor property that we set (for example, to red using RGB(255, 0, 0)) like we do with VFP controls. Other properties are similar; for example, the value in nFloat updates the FLOAT, TOP, and BOTTOM fields in the FRX.

So, we're finally back to SFReportField, the subclass of SFReportObject that holds information about fields in a report. This class adds the properties shown in Table 5 to those of SFReportObject. As you can see, you have the same control over the properties of an object in a report as you do in the Report Designer.

Property	**Purpose**
cDataType	The data type of the expression: "N" for numeric, "D" for date, and "C" for everything else (only required if you'll edit the report in the Report Designer later)
cExpression	The expression to display
cFontName	The font to use (if blank, which it is by default, SFReportFile.cFontName is used)
cPicture	The picture (format and inputmask) for the field
cTotalType	The total type: "N" for none, "C" for count, "S" for sum, "A" for average, "L" for lowest, "H" for highest, "D" for standard deviation, and "V" for variance (constants are defined for these values in SFREPOBJ.H)
lFontBold	.T. if the object should be bolded
lFontItalic	.T. if the object should be in italics
lFontUnderline	.T. if the object should be underlined

lResetOnPage	.T. to reset the variable at the end of each page; .F. to reset at the end of the report
nFontSize	The font size to use (if 0, which it is by default, SFReportFile.nFontSize is used)
nResetOnGroup	The group number to reset the value on

Table 5. Public properties of SFReportField.

As with SFReportObject, SFReportField's CreateRecord() method uses DODEFAULT() to get the behavior of SFReportRecord and SFReportObject, then it does some data conversion similar to what SFReportObject does (for example, lFontBold, lFontItalic, and lFontUnderline are combined into a single FONTSTYLE value).

SFReportText is a subclass of SFReportField, since it has the same properties but only slightly different behavior. It automatically adds quotes around the expression since text objects always contain literal strings rather than expressions. It also sets the PICTURE field in the FRX to match the alignment of the data (because that's how alignment is handled for text objects), and sizes the object appropriately for the size of the text (in other words, it acts like setting the AutoSize property of a Label control to .T.).

Here's some code that adds text and field objects to the detail band and sets their properties. This code uses characters as the units, so values are in characters or lines.

```
loObject = loDetail.AddItem('Text')
loObject.cExpression = 'Country:'
loObject.nVPosition  = 1
loObject.lFontBold   = .T.
loObject = loDetail.AddItem('Field')
loObject.cExpression = 'CUSTOMER.COUNTRY'
loObject.nWidth      = fsize('COUNTRY', 'CUSTOMER')
loObject.nVPosition  = 1
loObject.nHPosition  = 10
loObject.lFontBold   = .T.
```

Lines, Boxes, and Images

SFReportShape is a subclass of SFReportObject that defines the properties for lines and boxes (it isn't used directly but is subclassed). nPenPattern is the pen pattern for the object: 0 = none, 1 = dotted, 2 = dashed, 3 = dash-dot, 4 = dash-dot-dot, and 8 = normal. nPenSize is the pen size for the line: 0, 1, 2, 4, or 6.

SFReportLine is a subclass of SFReportShape that's used for line objects. It adds one property, lVertical, that should be set to .T. to create a vertical line or .F. (the default) for a horizontal one. Its CreateRecord() method sets the height for a horizontal line or the width for a vertical one to the appropriate value based on the pen size.

The following code adds a heavy blue line on line 4 of the page header band:

```
loObject = loPageHeader.AddItem('Line')
loObject.nWidth     = lnWidth
loObject.nVPosition = 4
loObject.nHPosition = 0
loObject.nPenSize   = 6
loObject.nForeColor = rgb(0, 0, 255)
```

Boxes use SFReportBox, which is also a subclass of SFReportShape. It adds nCurvature (the curvature of the box corners; the default is 0, meaning no curvature) and lStretchToTallest (.T. to stretch the object relative to the tallest object in the band) properties.

SFReportImage, a subclass of SFReportObject, is used for images. The name of the image file or General field that's the source of the image goes in the cImageSource property and lImageFile should be set to .T. if an image file is used rather than a General field. nStretch defines how to scale the image: 0 = clip, 1 = isometric, and 2 = stretch (the same values used by the Stretch property of an Image control). Its CreateRecord() method automatically puts quotes around the image source if a file is used, and sets the appropriate field in the FRX record if the cAlignment property is set to "Center" (only applicable for General fields).

Report Variables

Report variables are defined using the CreateVariable() method of the SFReportFile object. This method returns an object reference to the SFReportVariable object it created so you can set properties of the variable. Table 6 lists the public properties for variables.

Property	**Purpose**
cInitialValue	The initial value
cName	The variable name
cTotalType	The total type: "N" for none, "C" for count, "S" for sum, "A" for average, "L" for lowest, "H" for highest, "D" for standard deviation, and "V" for variance (constants are defined for these values in SFREPOBJ.H)
cValue	The value to store
lReleaseAtEnd	.T. to release the variable at the end of the report
lResetOnPage	.T. to reset the variable at the end of each page; .F. to reset at the end of the report
nResetOnGroup	The group number to reset the variable on

Table 6. Public properties of SFReportVariable.

The following code creates a report variable called lnCount and specifies that it should start at 0 and increment by 1 for each record printed in the report. This variable is then printed in the summary band of the report, showing the number of records printed.

```
loVariable = loReport.CreateVariable()
loVariable.cName         = 'lnCount'
loVariable.cValue        = 1
loVariable.cInitialValue = 0
loVariable.cTotalType    = 'Sum'
loSummary = loReport.GetReportBand('Summary')
loObject  = loSummary.AddItem('Field')
loObject.cExpression = 'ltrim(str(lnCount)) + ' + ;
  '" record" + iif(lnCount = 1, "", "s") + " printed"'
loObject.nWidth      = 21
loObject.nVPosition  = 2
loObject.nHPosition  = 0
loObject.lFontBold   = .T.
```

Putting it all Together

CUSTREP.PRG and EMPREP.PRG are sample programs that create reports for the CUSTOMER and EMPLOYEE tables in the VFP TESTDATA database. CUSTREP creates (and previews) CUSTREP.FRX, which shows customers grouped by country, with the maximum order amount subtotaled by country and totaled at the end of the report. EMPREP creates EMPREP.FRX, which shows the name and photo of each employee. These reports aren't intended to be realistic; they just show off various features of the report classes described in this article, including printing images and lines, setting font sizes and object colors, positioning objects in different bands, use of report variables and group bands, etc.

Conclusion

Although you might write a fair bit of code to create an FRX using the report object classes I presented in this article, the code is simple: create some objects and set their properties. It sure beats writing 50 INSERT INTO statement with 75 fields to fill for each. Please report (pun intended) to me any suggestions you have for improvements. Have fun!

Doug Hennig is a partner with Stonefield Systems Group Inc. in Regina, Saskatchewan, Canada. He is the author of Stonefield's add-on tools for FoxPro developers, including Stonefield Database Toolkit and Stonefield Query. He is also the author of "The Visual FoxPro Data Dictionary" in Pinnacle Publishing's "The Pros Talk Visual FoxPro" series. Doug has spoken at the 1997 and 1998 Microsoft FoxPro Developers Conferences (DevCon) as well as user groups and regional conferences all over North America. He is a Microsoft Most Valuable Professional (MVP). He can be reached at dhennig@stonefield.com.