XFRX

Developer's guide

EQEUS.COM

XFRX version: 12.0
Document version: 1.57
Last update: 11:29, August 17, 2006

The latest version of this document is available at: http://www.eqeus.com.

1 Contents

2 What's new in this document

Date	XFRX version	Document modifications
August 15, 2006	12.0	**Updated chapters:** XFRX Output Target Types (page 69) Flow layout document option (page 25) Generating OpenOffice documents (ODF) (page 29)
February 1, 2006	11.3	**New chapters:** Implementing custom event hyperlinks (drilldown) in XFRX previewer (page 61) Installation / Two versions of XFRXLIB.FLL (page 12) **Updated chapters:** XFRX previewer (page 50) Properties and methods common in XFRXListener and XFRXSession classes (page 70) [new properties: PictureDPI and DefaultPictureFormat]
December 2, 2005	11.2	**New chapter:** OpenOffice Writer document option (page 29) **Updated chapter:** XFRX Output Target Types (page 69)
August 24, 2005	11.1	**New chapters:** Showing / hiding previewer toolbars (page 51) Displaying printer properties dialog (page 42) Using custom printer settings when printing (page 43)
June 2, 2005	11.0	**New chapters:** Architecture of XFRX (page 8) Drawing custom objects with XFRX#DRAW (page 44)
March 28, 2005	10.2	**New chapters:** Defining sheet names in XLS documents (page 32) Hiding the Excel sheet grid (page 32) Leaving the fields content in Excel cells (page 32) Printing XFF files, chapter Printing XFF files (page 42) **Updated chapters:**

		Handling page breaks in XLS output (page 31) Using XFF files (page 40) XFRX previewer (page 44) XFFWriter class reference (page 82)
December 29, 2004	10.1	**New chapters:** *Appending generated output to existing PDF Documents* on page 23.

3 Introduction

XFRX is a tool for transforming Visual FoxPro reports to electronic formats. It can be incorporated into Visual FoxPro applications to provide the following functionality:

- Preview reports in an advanced localizable report previewing tool with hyperlink, drilldown and search capabilities
- Convert reports output to various output formats (Currently supported output formats are: PDF, DOC, RTF, XLS, HTML, MHT, ZIP, TXT, ODT (OpenOffice Writer), XML, XFF (DBF), BMP, PNG, JPEG, GIF and multipage TIFF).
- Store generated reports in an internal input-output file format with an option to modify the generated content – add graphics, shapes, new pages, generate custom graphics objects (e.g. graphs, watermarks) and many more. The internal file format can be converted to any of the output formats without need to reprocess the report.
- You can alternatively bypass the reporting engine altogether – create any of the output formats supported by XFRX from scratch – directly for your Visual FoxPro code.

XFRX is a royalty-free product. It can be added to Visual FoxPro applications without any additional costs. Please read the License chapter for more details.

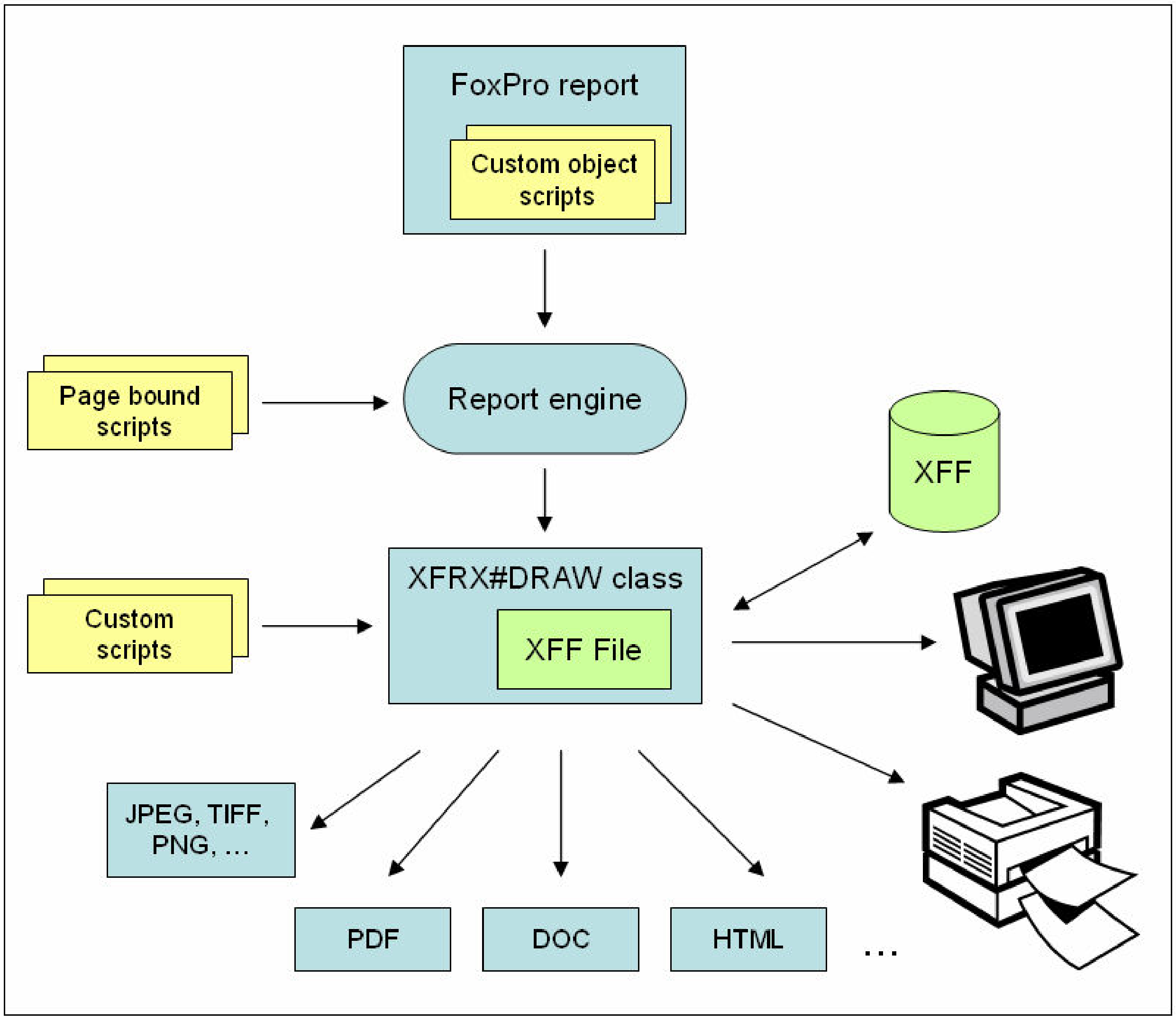
XFRX is available for Visual FoxPro 5.0, 6.0, 7.0, 8.0 and 9.0. XFRX contains its own report engine which is used in VFP 8.0 and earlier. In VFP 9.0, XFRX plugs into the new reporting architecture of the native report engine.

The differences between XFRX for VFP 5.0, 6.0, 7.0 and 8.0 and XFRX for VFP 9.0 are described in detail in this document.

Please note: To make the text easier to read, these differences are described as differences between VFP 8.0 and VFP 9.0. If VFP 8.0 is mentioned further in the text, it applies for VFP 5.0, 6.0 and 7.0 as well.

4 Architecture of XFRX

The basic architecture is described in the following schema:



FoxPro report – The FoxPro report definition, stored in .FRX, .FRT files

Report engine – The reporting engine that processes the report definition. XFRX for VFP 8.0 and earlier contains its own report engine. In VFP 9.0, XFRX plugs into the new reporting architecture and the native report engine is used.

Scripts – Methods or code snippets written in VFP. The scripts call methods of XFRX#DRAW class to "draw" to the generated document.

You can find more information about using scripts in Drawing custom objects with XFRX#DRAW chapter on page 44.

> **Custom object scripts** – scripts linked to a rectangle object on a report. When the rectangle is being processed, the script is invoked instead of rendering the

actual rectangle. The custom object scripts can alternatively be converted to pictures.

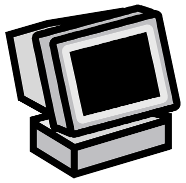
When these scripts are processed, the controlling table of the report is available, so these scripts are ideal for generating graphs or other data based graphics objects.

Page bound scripts – scripts invoked on each page, drawing either "above" of "below" the page content. These scripts are suited for watermarks.

Custom scripts – you can modify the generated reports, too. You can add text, graphics, new pages, hyperlinks, bookmarks, etc. You can even create new documents from scratch, bypassing the report engine altogether.

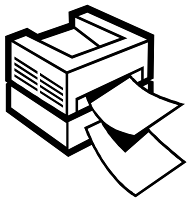
XFF File – an internal file that stores the generated reports. These files can be modified, stored to disk, previewed on screen, printed or converted to any of the target formats. You can find more information about XFF files in Using XFF files chapter on page 40.

XFRX#DRAW class – the class that wraps XFF files.



XFRX previewer – The XFF files can be previewed in an advanced report previewer that comes with XFRX. The previewer can be localized and allows for search the document and supports hyperlinks and bookmarks.

You can find more information about using the previewer in XFRX previewer chapter on page 50.



Printer – XFRX is also able to print the XFF files. Please find more information about printing in Printing XFF files chapter on page 42.

5 Installation

Evaluation version quick tip: For a quick preview of the capabilities of XFRX, unpack the whole zip archive to an empty directory and run DEMO.SCX.

XFRX consist of the following parts:

- **XFRX.APP, XFRX.FXP, XFRX.PRG**
  This is the main application. The demo version contains XFRX.APP, the commercial version is distributed as XFRX.FXP for easier compilation into target applications (Please see *Distributing XFRX with other applications* below). The source code options are distributed as XFRX.PRG.
- **XFRXLIB.FLL, HNDLIB.DLL, ZLIB.DLL**
  Support libraries. ZLIB.DLL is a freeware compression utility, please see http://www.zlib.org for more information
- **XFRXLIB directory**
  This directory containing the XFRX previewer component class library and its resource files

The installation is very easy: simply unpack the zip archive and make sure the files are accessible from VFP (they can either be placed in a local directory or the path can be set via the SET PATH command).

5.1 How XFRX handles different version of Visual FoxPro

XFRX for VFP 8 contains its own report engine that mimics the behavior of the native Visual FoxPro report engine.

XFRX for VFP 9 is implemented as an XFRXListener class, a descendant of UtilityReportListener class (please see _REPORTLISTENER.VCX in FFC for more information about the basic report listener classes provided by FFC). The XFRXListener class complies with the object-assisted reporting standards introduced in VFP 9 and perfectly fits into the new open architecture. You can use it alone or in combination with other report listeners provided with the product or by third parties.

About 95% of XFRX is written in Visual FoxPro and, to make the deployment as easy as possible, all its source code (without the visual classes for the report previewer container) is in one PRG file, both for VFP 8.0 and VFP 9.0: XFRX.PRG. This PRG file is the same for all VFP versions.

If you purchase XFRX without source codes, you will be able to download XFRX.FXP compiled in the VFP version you are using.

If you purchase the source code, before you add XFRX.PRG into your project and compile it, you need to enable (uncomment) one of the #DEFINE commands at the beginning of XFRX.H, which determines the VFP version you are using.

Example: If XFRX is used in VFP 8.0, the beginning of XFRX.H would look like this:

```
*#DEFINE VFP5
*#DEFINE VFP6
#DEFINE VFP8
*#DEFINE VFP9
```

(Please note there is not a constant for VFP 7.0. If you are using VFP 7.0, uncomment the VFP6 constant).

5.2 Differences between XFRX for VFP 8.0 and XFRX for VFP 9.0

As written in the previous paragraph, XFRX for VFP 9.0 uses the native report engine, which results in the following advantages:

1. The layout of the documents produced by XFRX for VFP 9.0 should generally be more exact as the same engine is used both for printing and for document generation.
2. XFRX for VFP 9.0 is approximately twice as fast as XFRX for VFP 8.0.
3. Variables declared as LOCAL are not visible in XFRX for VFP 8.0.
4. XFRX for VFP 8.0 has some restrictions in handling the data environment. It is able to set up the data environment, open tables, setup relations and fire the data environment methods, but the data environment instance itself is not available. This restriction does not apply to VFP 9.0.
5. XFRX for VFP 9.0 is able to display any content of general fields, including ActiveX components. XFRX for VFP 8.0 is able to extract just BMP and JPEG pictures from the general fields.
6. XFRX for VFP 8.0 does not support multicolumn layout where records are laid from left to right (rather than from top to bottom). This limitation does not apply for VFP 9.0.

5.3 Distributing XFRX with other applications

The best way to distribute XFRX is to include XFRX.FXP (or XFRX.PRG, if you purchased the source code option) into the target application's exe (add it to your project, to the section where your PRGs are). This way, you won't have to distribute this file and, more importantly, XFRX will be able to access the reports and other resource files (pictures, tables, etc.) built into the exe as well.

If you are using the XFRX previewer, add the appropriate files from the XFRXLIB directory to your project as well.

The support libraries (XFRXLIB.FLL, HNDLIB.DLL and ZLIB.DLL) cannot be included in the exe and need to be distributed along with the application. They need to be located either in the same directory where the main EXE is or in a directory defined in SET PATH command.

5.4 Two versions of XFRXLIB.FLL

There are two versions of XFRXLIB.FLL available that you can use with XFRX. The reason of this is that the "normal" XFRXLIB.FLL version requires two other DLL libraries from Microsoft to be installed on the target computers: gdiplus.dll and msvcr71.dll. If you try to use XFRX with XFRXLIB.FLL without the DLLs installed, SetParams method will return -6, "xfrxlib.fll cannot be loaded (it is missing or invalid)".

Both of the dll libraries can be downloaded from Microsoft website:

- msvcr71.dll - Microsoft Visual C++ 7.1 runtime library. If it is not installed on your pc, you can download it from http://www.eqeus.com/files/msvcr71.zip
- gdiplus.dll - This DLL is included with Windows XP and if it is not available on your pc, you can download it from http://www.microsoft.com/msdownload/platformsdk/sdkupdate/psdkredist.htm

These libraries can be distributed with your application (they are also distributed with VFP 9.0), but sometimes you may not need the gdiplus features in XFRXLIB.FLL and it may be easier to use a no-gdi+ version of XFRXLIB.FLL. This library is located in the NOGDIP subdirectory in the evaluation as well as commercial version package.

The following table indicates features not available in the no-GDI+ version if XFRXLIB:

	GDI+ version	No-GDI+ version
Exporting reports as pictures	Yes	No
Printing	Yes	No
Supporting BMP and JPG pictures in PDF	Yes	Yes
Supporting other picture formats in PDF	Yes	No
Converting report pictures to defined DPI	Yes	No
Exporting contents of general fields in VFP 9	Yes	No
Exporting BMP and JPG pictures from general fields in VFP 8	Yes	Yes
Binary comparison of images to reduce the size of PDF documents	Yes	Yes

6 Running XFRX

When XFRX is run, it returns an instance of one of three classes, depending on a parameter passed. The available parameters are:

1. "XFRX#INIT"
   Running XFRX with this parameter will return the XFRXSession class instance, which is the main class that controls the behavior of XFRX in VFP 5, 6, 7 and 8.
2. "XFRX#LISTENER"
   This option is available in Visual FoxPro 9 only and returns an instance of XFRXListener class.
3. "XFRX#DRAW"
   This option returns an instance of XFRX#DRAW class. This class is used for working with XFF files. Please see Initializing the XFRX#DRAW class instance chapter on page 40 for more information.

Important note: The evaluation version of XFRX cannot be included into VFP projects, it makes VFP crash. To avoid this please invoke XFRX via macro substitution:

```
loSession = EVALUATE("xfrx('XFRX#INIT')")
```

This way XFRX.APP does not get into the project and you will be able to compile your application without problems.

6.1 Running XFRX in VFP 5.0, 6.0, 7.0 and 8.0

Please note: To make the text easier to read, the differences between XFRX for VFP 5.0, 6.0, 7.0, 8.0 and XFRX for VFP 9.0 are further in the text described as differences between VFP 8.0 and VFP 9.0. If VFP 8.0 is mentioned, the described feature applies for VFP 5.0, 6.0 and 7.0 as well.

1. Call XFRX with "XFRX#INIT" as a parameter to obtain the XFRXSession object:

   ```
   loSession=XFRX("XFRX#INIT")
   ```

2. Call SetParams method to set the document generation parameters. (Please see page 70 for details and full parameter list).
3. If SetParams return 0 (zero), call ProcessReport method for each report to process. (Please see page 81 for details and full parameter list).
4. After all reports are processed, call Finalize method to finish the document generation process.
5. You can also call ResetPageNo() method if you need to reset the page number in between reports.

Example 1:

This code merges two reports – report1 and report2 – into a single PDF document, output.pdf

```
use demoreps\invoices order customer
local loSession, lnRetval
loSession= xfrx("XFRX#INIT")
lnRetVal = loSession.SetParams("output.pdf",,,,,,"PDF")
If lnRetVal = 0
            loSession.ProcessReport("report1")
            loSession.ProcessReport("report2")
            loSession.finalize()
Else
            ? lnRetVal
Endif
```

6.2 Running XFRX in VFP 9.0

In VFP 9.0, the standard object-assisted mode is used to run XFRX:

1. Call XFRX.APP with "XFRX#INIT" as a parameter to obtain the XFRXListener object:

   ```
   loListener=XFRX("XFRX#LISTENER")
   ```

2. Now there are two options you can choose from:
   a. Call loListener's SetParams method that has exactly the same parameters and return values as the one in XFRXSession class. (Please see page 70 for details and full parameter list).
   b. Or, you can fill in individual properties of loListener and run SetParams method without any parameters to make sure you can proceed to the next step. (Please see page 80 for full list of properties available).
3. Call the native REPORT FORM command with the new OBJECT clause for each report to process. If more reports are merged, include NOPAGEEJECT clause with each REPORT FORM call but the last one.
4. Alternatively, you can leave the NOPAGEEJECT clause in the last REPORT FORM command as well (which may be useful is some scenarios) and call loListener.finalize() to finish the document generation.

Note: Even in VFP 9.0, you can initialize the XFRXSession class via "XFRX#INIT" parameter and use the XFRX's own engine, rather than the native one.

The following examples all do exactly the same (and they also do exactly the same as the example 1 above), showing various ways of calling XFRX in VFP 9.0:

Example 2:

```
use demoreps\invoices order customer
local loSession, lnRetval
loxfrx = XFRX("XFRX#LISTENER")
lnRetval = loxfrx.SetParams("output.pdf",,,,,,"PDF")
IF lnRetval = 0
            REPORT FORM report1 OBJECT loxfrx NOPAGEEJECT
            REPORT FORM report2 OBJECT loxfrx
ELSE
            ? lnRetval
endif
```

Example 3:

```
use demoreps\invoices order customer
local loSession, lnRetval
loxfrx = XFRX("XFRX#LISTENER")
lnRetval = loxfrx.SetParams("output.pdf",,,,,,"PDF")
IF lnRetval = 0
            REPORT FORM report1 OBJECT loxfrx NOPAGEEJECT
            REPORT FORM report2 OBJECT loxfrx NOPAGEEJECT
            Loxfrx.finalize()
ELSE
            ? lnRetval
endif
```

Example 4:

```
use demoreps\invoices order customer
local loSession, lnRetval
loxfrx = XFRX("XFRX#LISTENER")
loxfrx.targetType = "PDF"
loxfrx.targetFileName = "output.pdf"
lnRetval = loxfrx.SetParams()
IF lnRetval = 0
            REPORT FORM report1 OBJECT loxfrx NOPAGEEJECT
            REPORT FORM report2 OBJECT loxfrx
ELSE
            ? lnRetval
endif
```

6.3 Using THISFORM and THIS references

Note: *This paragraph applies to VFP 8.0 only. THISFORM and THIS references are handled properly by the native report engine in VFP 9.0.*

XFRX supports using THISFORM and THIS in the expressions of the report fields. However, being normal VFP application, XFRX cannot access THISFORM and THIS objects directly. Instead, THISFORM object has to be explicitly sent to XFRX via setThisform() method, THIS needs to be sent via setThis() method. The use is very simple.

If you have THISFORM in your report, call xfrxSession.setThisform(THISFORM) before calling ProcessReport(). If you are using THIS, call xfrxSession.setThis(THIS).

6.4 Displaying progress bar in VFP 8.0

XFRX provides a simple hook so that any progress bar tool could be used for displaying the progress during the document generation process. All you have to do is to create an object which contains updateProgress() method and pass it to XFRX. During the generation process, XFRX calls updateProgress() method either after each page or after each record is processed.

This is a simple example of displaying the generation progress in a wait window:

```
loSession=xfrx("XFRX#Init")
loProgress = createobject("progress")
lnRetVal = loSession.SetParams("document",,,,,,"PDF")
if lnRetVal = 0
   loSession.setProgressObj(loProgress,2)
   loSession.ProcessReport("myReport")
   loSession.finalize()
endif

define class progress as custom
   procedure updateProgress
      lpara ta,tb, tc
      wait window nowait "Page #: "+allt(str(tb))+" Report #:
"+allt(str(ta))+" ("+allt(str(tc))+"%)"
enddef
```

The progress object is attached to XFRX with setProgressObj() method. This method takes two parameteres - the first one is the progress object, the second defines the information the updateProgress() will be getting. It can contain two values: 1 - only page number and report number will be provided in updateProgress() method, or 2 - page number, report number and percentage progress within the current report will be provided. Using the percentage progress information is more accurate and more suitable for progress bar visualization, but to provide this, XFRX has to calculate the number of records in the processed table, which, sometimes, can be very time demanding. In this cases, 1 can be used not to calculate the number of records.
The updateProgress method takes three parameteres: current report number, current page number and actual percentage progress within the current report.

6.5 Displaying progress bar in VFP 9.0

In VFP 9.0, the XFRXListener object can be chained together with another listener which would take care of the progress bar displaying. One of the possible ways to do this is to use the UpdateListener class, which is shipped with VFP 9.0, in FFC.

Example:

This sample code creates an instance of the UpdateListener class and chains it with XFRXListener:

```
loxfrx = XFRX("XFRX#LISTENER")
SET CLASSLIB TO (HOME()+"FFC\_reportlistener.vcx")
loUpdate = CREATEOBJECT("updatelistener")
loUpdate.thermFormCaption = "Report in progress ..."
loxfrx.successor = loUpdate
lnRetval = loxfrx.setparams("output.pdf",,,,,,"PDF")
IF lnRetval == 0
            REPORT FORM (lcReportName) OBJECT loxfrx NOPAGEEJECT
            loxfrx.finalize()
ELSE
            ? lnRetval
endif
```

6.6 Canceling report generation in progress

Note: *This paragraph applies both to VFP 9.0 and VFP 8.0*

The report generation process in progress can be canceled by setting the global variable gnStopXFRX to 1. For example, you can, for example, use it this way:

```
ON KEY LABEL Ctrl+C gnStopXFRX = 1
```

6.7 Printing page ranges

Note: *This paragraph applies to VFP 8 only. In VFP 9.0, please use the RANGE clause of the REPORT FORM command to achieve the same.*

To define the page range, call setPageRange() method before calling calling ProcessReport(). There are two possible ways how to call the setPageRange() method:

- setPageRange(tnFrom, tnTo)
  tnFrom and tnTo define the from-to range. If tnTo is empty, the total number of pages is used
  Example:

loSession.setPageRange(5,10)

- setPageRange(tcRange)
  tcRange is a string, which can contain page numbers and page ranges delimited by commas, the page range is defined as "from-to".
  Example:

loSession.setPageRange("1,4,10-20,25")

6.8 User-defined page size

Note: *This paragraph applies both to VFP 80. and VFP 9.0*

You can define the page size of the generated document. The user-defined page size will override the page size stored in the report. To define the user-defined page size, call setPaperSize() method with paper width and paper height as parameters:

```
      setPaperSize(nUDPaperWidth, nUDPaperHeight)
```

The unit is Inch * 10000.

See also: *HTML page size adjustment* on page 30.

6.9 Zipping the generated files

Note: *This paragraph applies both to VFP 8.0 and VFP 9.0*

The generated file can be automatically zipped. This feature is controlled by the last three parameters of SetParams() method of XFRXSession and XFRXListener classes (see the reference on page 70) or, with an equivalent behavior, by the three parameters of ZipDocument() method of XFRXListener class (reference on page 80).

Example:

With the following SetParams parameters, XFRX first creates "invoices.pdf", then creates "archive.zip" (if it doesn't exist) and adds "invoices.pdf" into the archive. Then the original "invoices.pdf" will be deleted:

```
    loSession.SetParams("invoices.pdf",,.T.,,,,"PDF","archive.zip", .t., .t.)
```

7 Interactive features

The generated documents can include hyperlinks for faster navigation, PDF documents can also include bookmarks. The hyperlinks and bookmarks are controlled via Comment field of labels and fields.

Please note: In the beta version of VFP 9 there is a bug that makes it impossible to store anything into fields' comment. Because XFRX uses the comment field to define hyperlinks and bookmarks, the "User data" field is now used for this purpose. In the final version, XFRX for VFP 9 will work both with comment and user data field.

7.1 Hyperlinks

Note: *This paragraph applies both to VFP 9 and VFP 8.*

The hyperlinks are controlled via Comment field of report labels or fields.
To create a link, you need a source field (the underlined text you will navigate from) and a target field (the place where you get when you click on a hyperlink).

- **Creating source fields**

Enter the following text into the comment of the source field:
#UR A HREF=<destination name>

The destination name is an expression, which is evaluated at the time of report generation. You can navigate to other fields in the same document, or to any URL. The destination names of other fields in the same document has to be preceeded with #.

Examples:
#UR A HREF="#top"
Navigates to the beginning of the document. "top" is a reserved word. Do not name target fields are "top".

#UR A HREF="#custlist"
Navigates to the field whose destination name is "custlist".

#UR A HREF="#"+customer.id
Navigates to the field whose destination name is eval(customer.id)

#UR A HREF="http://www.eqeus.com"
Navigates to Eqeus.com homepage

- **Creating target fields**

To add a destination name to a (target) field, add the following text into the comment:
#UR A NAME=<destination name>

Example:
#UR A NAME=customer.id
This field will be a target field for source fields with HREF="#"+customer.id

7.2 Bookmarks

Note: *This paragraph applies both to VFP 9 and VFP 8.*

Note 2: *Bookmarks are currently supported in PDF and HTML documents*

Bookmarks (document outline) serves as a "visual table of contents" to display the document structure. Users use this to interactively navigate in the document. To add a report field into the document outline simply put the following into the field's comment:

#UR OUTLINE=<outline_name>

The outline_name is an expression, which is evaluated at the time of report generation and the result is used as the bookmark item. If users click the bookmark, they will navigate to the corresponding report field.

Example:
In a report with a list of invoices grouped by customers, bookmarks containing the list of customer can be created by adding

#UR OUTLINE=invoices.customerName

into the comment of a customer name field (or any other field you want to navigate to, e.g. the first field on a page with the customer).

To enable bookmarks in the HTML output, call

```
      loSession.SetOtherParams("PRINT_BOOKMARKS",.t.)
```

before calling loSession.ProcessReport()
With bookmarks enabled, XFRX will generate three HTML pages (three files): the main page defining the page frames, the bookmark page and the page with the report output.

7.3 Document properties

Note: *This paragraph applies both to VFP 9 and VFP 8. The set... methods are implemented both in XFRXListener and XFRXSession classes.*

The following methods can be called to set various document properties. When generating a Word document, all document properties have to be set before the first report is processed. When exporting to PDF, the properties have to be set before Finalize() method is called.

- **General properties**

```
loSession.setAuthor(<author>)
loSession.setTitle(<title>)
loSession.setSubject(<subject>)
loSession.setKeywords(<keywords>)
```

- **PDF only properties**

```
loSession.setCreator(<creator>)
loSession.setProducer(<producer>)
```

- **Word only properties**

```
loSession.setComments(<comments>)
loSession.setCategory(<category>)
loSession.setManager(<manager>)
loSession.setCompany(<company>)
```

8 PDF specific features

8.1 PDF Encryption

Note: *This paragraph applies both to VFP 9 and VFP 8. The setPasswords method is implemented both in XFRXListener and XFRXSession classes.*

PDF documents can be encrypted. To set the encryption on, call setPasswords method:

```
loSession.setPasswords(tcOwnerPassword, tcUserPassword)
```

The user password can be empty. If the owner password is empty, a random string will be generated as the password.
The owner can do anything with the document. The user permissions can be set using the setPermission method:

```
loSession.setPermissions(tlPrintDocument, ;
                         tlModifyDocument, ;
                               tlCopyTextAndGraphics, ;
                               tlAddOrModifyAnnotations)
```

The default values of the permissions is .F. (false).

8.2 PDF Font Embedding

Note: *This paragraph applies both to VFP 9 and VFP 8. The method is implemented both in XFRXListener and XFRXSession classes.*

XFRX supports both whole font and font subset embedding.

To embed all characters of all used fonts, call:

```
loSession.setEmbeddingType(2)
```

before running the report.

To embed only the characters used, call:

```
loSession.setEmbeddingType(3)
```

Embedding only subset of fonts (characters used in the document) significantly reduces the size of the generated file.

To select which font to embed (e.g. when you need just to embed the barcode font, or font that is not installed on the pc the document will be sent to), add "#UR INCLUDEFONT" (without the quotation marks) to the comments of a field that uses the font (in the report). Add "#UR INCLUDEFONT SUBSET" comment to include the font subset.

8.3 Object rotation

Note: *This paragraph applies both to VFP 9 and VFP 8.*

To rotate a text or a picture in PDF output, add “#UR ROTATE” (without the quotation marks) to the comment of the report field. The text or the picture will rotate counterclockwise by the entered angle, e.g. to print vertically, add: “#UR ROTATE 90”.

8.4 Appending generated output to existing PDF Documents

Note: *This paragraph applies both to VFP 9 and VFP 8.*

From version 10.1, XFRX is able to append the generated report to an existing PDF document. It is possible to append the report at the end of the document or at an arbitrary position within the document: with either inserting the new pages or replacing the pages in the existing PDF document.

In XFRX for VFP 8 this feature is controlled by a new parameter of SetParams(...) method: tuAppend. Please see the SetParams method reference on page 70 for more information about setting this parameter.

In XFRX for VFP 9 you can interchangeably use the new parameter of SetParams(...) method or AppendToFile property of the XFRXListener class.

Notes:

1. It's not guaranteed that XFRX will be able to append to any PDF document. It works fine with PDF documents generated by XFRX and we've successfully tested PDFs from other sources, too, but the PDF specification allows for some internal structures that XFRX wouldn't be able to decode. (To be precise: XFRX doesn’t support linearized PDF documents and documents that use page tree structures).
2. Because of the way the PDF file is constructed, the size of the PDF document never shrinks, even if the number of pages in the resulting PDF document is smaller than in the original one.

9 Word specific features

9.1 Password protection

Note: *This paragraph applies both to VFP 9 and VFP 8. The setPasswords method is implemented both in XFRXListener and XFRXSession classes.*

To add passwords to Word documents, call SetPasswords() method before calling ProcessReport():

```
loSession.setPasswords(tcReadPassword, tcWritePassword,
tlRequirePassword)
```

You can omit either tcReadPassword or tcWritePassword. tlRequirePassword is optional (default value is .F.). If set to .T., Word will ask for the password even when the document is being opened first time after the generation (if tlNotOpenViewer parameter of SetParams method is set to .F.).

9.2 Word document splitting

Note: *This paragraph applies to VFP 8 only.*

When the generated documents are very long, Word application has problem with the conversion - it takes very long to convert it. To avoid this problem, XFRX can split the generated document into more smaller documents. To set this up, call SplitDocument() method before calling ProcessReport():

```
loSession.SplitDocument(tnPages)
```

tnPagesp is the number of pages each of the resulting documents would have.

10 Flow layout document option

Note: This chapter links to several sample documents at our web site. If you are reading an electronic version of this document, simply click the hyperlinks to download them. If you are reading a printed version, here are the addresses:

http://www.eqeus.com/xfrxmanual/example1.doc

http://www.eqeus.com/xfrxmanual/example2.doc

http://www.eqeus.com/xfrxmanual/example3.doc

http://www.eqeus.com/xfrxmanual/example4.doc

Word documents, RTF documents, OpenOffice Writer documents and OpenOffice Calc documents have two output options – “Absolute positioned layout” and “Flow layout”. The generated documents often look the same, but the algorithms behind these options are completely different. The absolute positioned layout always looks like the original report, but is hard to edit and bigger in size. The flow layout may not always look exactly the same, but it is a 'real' Word document - easily editable, with styles, page headers and footers, paragraphs and tab stops, which is also shorter in size and faster to open.

Each of the output options have its own parameter code that is sent at the 7th parameter (targetType) of the SetParams method:

Document type	targetType parameter
Word document with absolute layout	DOC
Word document with flow layout	FDOC
RTF document with absolute layout	RTF
RTF document with flow layout	FRTF
OpenOffice Writer document with absolute layout	ODT
OpenOffice Writer document with flow layout	FODT
OpenOffice Calc spreadsheet with absolute layout	ODS
OpenOffice Calc spreadsheet with flow layout	FODS

This chapter describes the Flow layout output option – how it works and how it differs from the Absolute positioned layout.

10.1 Running the conversion

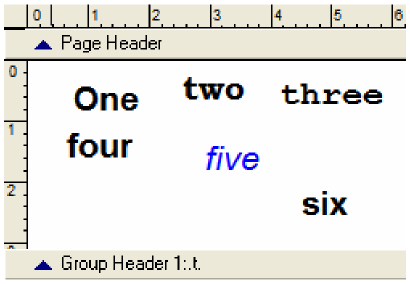
Flow layout Word option is implemented as a new target type, so simply send “FDOC” as the *tcTarget* parameter of the SetParams() method.

10.2 How it works

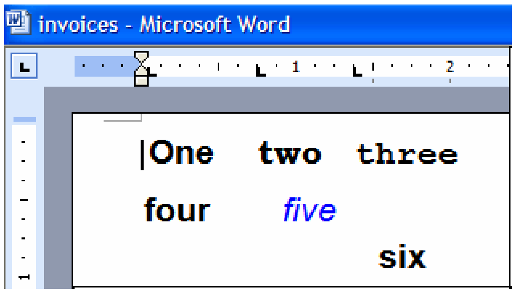
The logic is similar to the way how plain text option works. During the generation process, XFRX takes each section – one by one – and tries to split it to individual lines. Then each line is added to the output, respecting the vertical position of the line on the paper and horizontal positions of individual objects. Vertically misaligned objects are moved down to the closest baseline.

Example 1:

This report definition:



will be split to three lines and the resulting document will look like this:

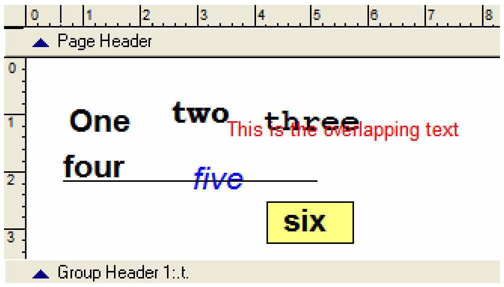


(download example1.doc)

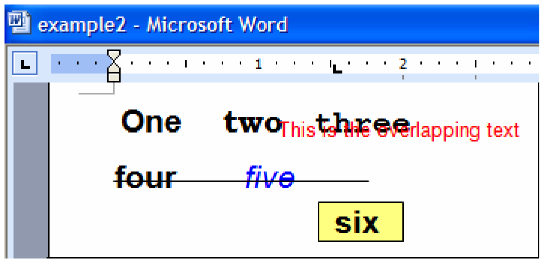
As you can see all objects are aligned to the same baseline and the horizontal position is set by a tab stop (left, right or center, depending on the field's alignment). If you add any graphics or pictures, these are added to the document and linked to the paragraph it starts at, so if you add contents above the graphics, it will move down along with the corresponding paragraph.

If XFRX cannot create distinct lines or if the text objects overlap one another, the overlapping text object is placed at the exact position as a textbox.

Example 2:



>> converts to >>



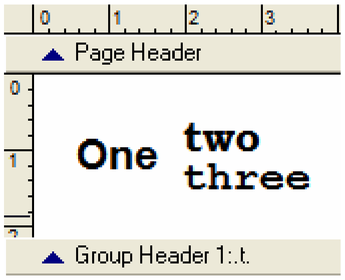
(download example2.doc)

10.3 Page headers and footers

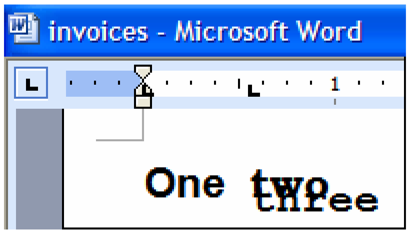
Page headers and footers defined in the report are converted to page headers and footers in the Word document, so for example, if you add a line in the middle of a page, the page footer will not move to the next page header, but stay in place.

10.4 Deficiencies

Even though most reports are converted without problems, there are certain scenarios when the output wouldn't look as expected. As stated above, all fields in one line are aligned to the same baseline. In some reports, this can cause a problem, for example:



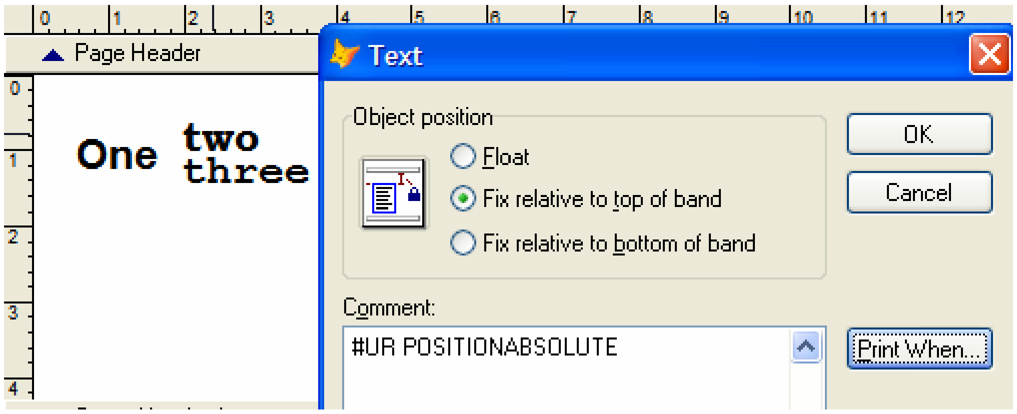
>> converts as >>



(download example3.doc)

Not only is the "two" text too below, it is also printed over the "three" one ("three" is placed too high to be on a separate line, so it is converted as a textbox). In this case, you may want to tell the engine that "two" and "three" objects should always be treated as absolute positioned textboxes.

To do this, add "#UR POSITIONABSOLUTE" as a comment of these fields:

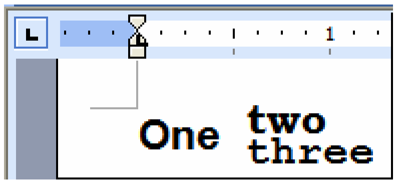


[ ] Float [✓ Fix relative to top of band] [ ] Fix relative to bottom of band

Comment:

```
#UR POSITIONABSOLUTE
```

And the resulted Word document would be generated as:



(download example4.doc)

11 Generating OpenOffice documents (ODF)

OpenOffice Writer document format is supported since XFRX version 11.2. Since 12.0, the Writer and Calc outputs are supported, both of which can be generated either using the absolute or flow layout formats (please see chapter Flow layout document option on page 25 for more information about the flow layout options).

OpenOffice is using the OASIS Open Document Format (ODF) for Office Applications, which is also supported by a variety of other office applications including StarOffice, KOffice, and IBM Workplace.

XFRX generates the file formats natively, so OpenOffice doesn't have to be installed on the computer where the document is generated.

You can find more information about the OASIS Open Document Format at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office.

More information about OpenOffice can be found at http://www.openoffice.org.

To generate the OpenOffice document, use the following codes as the 7th parameter (targetType) of the SetParams method:

Document type	targetType value
OpenOffice Writer document with absolute layout	ODT
OpenOffice Writer document with flow layout	FODT
OpenOffice Calc spreadsheet with absolute layout	ODS
OpenOffice Calc spreadsheet with flow layout	FODS

12 HTML specific features

12.1 HTML page size adjustment

Note: *This paragraph applies both to VFP 9 and VFP 8. The ShrinkHeight method is implemented both in XFRXListener and XFRXSession class.*

When generating HTML documents, XFRX makes the page a bit shorter by default (by 1.65 inches). This is to suppress the bottom of a page to wrap to a new page when printing from the Internet Explorer. Two lines are added to the printed page - at the top and bottom of the page - as header and footer. By calling ShrinkHeight() method, you can either suppress this behaviour by calling:

```
ShrinkHeight(0)
```

or set your own value by which the page will be shrinked. The unit is Inch * 10000, so to make the page shorter by 2 inches, call:

```
ShrinkHeight(20000)
```

The value sent by ShrinkHeight() method is applied to all output types, not just HTML.

See also: *User-defined page size* on page 17.

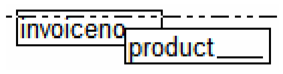
13 Excel specific features

13.1 General notes

The output to the Excel format is similar to the flow layout formats. The other formats use absolutely positioned textboxes for each report label or textbox, containing the generated text output. The XLS output, on the other hand, puts the generated text directly into the cells on the Excel sheet and sets the height of the rows and width of the columns to achieve the desired layout. Lines and rectangles, too, are added as cells' borders, rather than graphics over the sheet.

There are many advantages in this approach: the generated documents are smaller and much easier to be modified - all numeric fields can be used in calculations, it's no problem to add rows or columns, change cell attributes, etc.

There are, however, downsides, too: The fields cannot overlap, so something like this:



in the report won't convert correctly, and, as each Excel cell has a margin inside that cannot be suppressed, some fields might have to be made a little wider to accommodate the whole content.

With XFRX, the reports will probably need some tweaking, especially the complicated ones, but the result will be a normal Excel document, as if someone created it manually.

13.2 How it works

XFRX makes use of the possibility to merge more Excel cells together. Wherever a label or textbox should start or finish, XFRX creates a row and a column. To achieve the best looking results, it is a good thing to align the labels and textboxes both vertically and horizontally - result of which is a clearer document with fewer rows and columns. (Please see more about this below, in How to achieve the best results? paragraph.)

13.3 Handling page breaks

By default, XFRX does not break pages in the XLS output the same way as in other output formats – it is run in the plain mode instead, which means the output is one sheet - as long as it needs to be, with a page header on the top and a page footer at the bottom.

If more reports are processed, each report creates one sheet in the output document.

There are two options you can use to modify the way the page breaks are handled:

13.3.1 Generating sheet per page

To enable this option, call SetOtherParams method to set the “SHEET_PER_PAGE” parameter to .T.:

```
loSession.SetOtherParams("SHEET_PER_PAGE",.t.)
```

This will switch off the “plain” mode and the page breaks will correspond to other output options. Each page will be generated as a new sheet in the Excel document.

Please see more information about the SetOtherParams method in Methods common in XFRXListener and XFRXSession classes reference on page 70.

13.3.2 Generating sheet per start-each-group-on-a-new-page groups

To enable this option, call SetOtherParams method to set the “SHEET_PER_NP_GROUP” parameter to .T.:

```
loSession.SetOtherParams("SHEET_PER_NP_GROUP",.t.)
```

This option combines the plain mode and the sheet-per-page mode. The report runs in the plain mode but a new sheet is generated for each report group with “Start each group on a new page” flag set to .T.

13.4 Defining sheet names

There are two ways how to define sheet names. You can define a static text via NEXT_SHEET_NAME parameter, or you can use the NEXT_SHEET_NAME_EXPR parameter to define an expression, which will be evaluated at the bottom of each sheet and the result will be used as the new sheet name. The latter option is useful if more sheets are generated during one report run.

Please see more information about SetOtherParams method in Methods common in XFRXListener and XFRXSession classes reference on page 70.

13.5 Hiding the Excel sheet grid

The background grid is visible by default. To hide it, please set DISPLAY_GRID_LINES parameter to false:

```
loSession.SetOtherParams("DISPLAY_GRID_LINES",.F.)
```

13.6 Leaving the fields content in Excel cells

By default, the content of non-stretchable fields is cut according to the size of the field, but you can optionally leave the full content of the field in the Excel cell. To enable this option, set the LEAVE_FULL_FIELD_CONTENT parameter to .T.:

```
loSession.SetOtherParams("LEAVE_FULL_FIELD_CONTENT",.T.)
```

13.7 How to invoke the XLS output

It is pretty much the same as with the other targets:

```
	local loSession, lnRetval
```

```
loSession= xfrx("XFRX#INIT")
loSession.initLog()
lnRetVal = loSession.SetParams("output.xls",,,,,,"XLS")
If lnRetVal = 0
      loSession.SetOtherParams("NEXT_SHEET_NAME","first") && the name of
the sheet, optional
      loSession.ProcessReport("report1")
      loSession.SetOtherParams("NEXT_SHEET_NAME","second")
      loSession.ProcessReport("report2")
      loSession.finalize()
ENDIF
```

This example creates a two sheet document. As you can see, SetOtherParams() method can be used to define the sheet names. If it is not called, the default names are "sheet1", "sheet2", etc.

13.8 XLS cells adjustment

When XLS document is generated, the vertical and horizontal coordinates of objects are adjusted - if the difference between two coordinates is smaller than a certain value, the coordinates are 'aligned'. This approach significantly reduces the number of rows and columns in the generated document.

It is possible to define this minimal difference. The greater the number is, the lesser number of rows/columns is generated, but if the number is too big, fields might get overlapped and could be left out.

Call SetOtherParams method with "HORIZONAL_ADJUSTMENT" or "VERTICAL_ADJUSTMENT" to define the minimal horizontal and/or vertical difference.

Example:

```
loSession.SetOtherParams("HORIZONTAL_ADJUSTMENT",1000) && default value = 76
loSession.SetOtherParams("VERTICAL_ADJUSTMENT",1000) && default value = 180
```

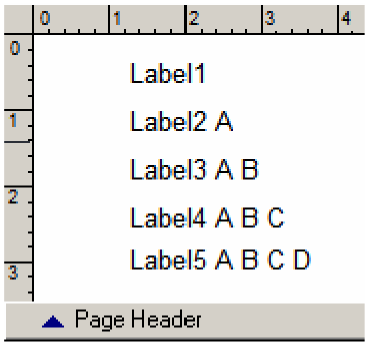
13.9 How to achieve the best results

1. **Align the fields.**
   Have a look at the following document: (http://www.eqeus.com/xls1.xls)
   Columns B and C are almost invisible (if you make them wider, you can see that customer names start at column B, "Customer List" starts at column C and "Customer" starts at column D - which is something that we don't notice in normal report but have better result in the XLS output if the fields are aligned), row 4 is very narrow, and between each customer, there's added a very thin row, too.

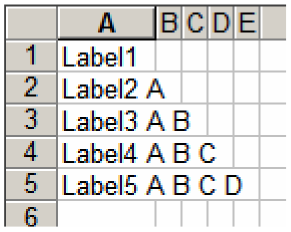
   "Fixing" this is very simple - we aligned the "Customer", "Customer List" labels and the customer textbox, moved the line below the header a little bit higher so it lands on the cells below the "Customer" and "Total" captions. We also aligned the textboxes vertically.
   The resulting document looks much better: (http://www.eqeus.com/xls2.xls)

2. **Problem with label width**
The size of a label cannot be modified in the report designer - it always takes the size of the text entered. However, as we mentioned before, the Excel cells have little margins inside, so if we create a cell as wide as the label and put the text into it, the whole text wouldn't fit in - the last character or two disappear!
XFRX takes care of this and makes the cell a bit wider, but this can bring another problem - if there is another label or a text field near the right edge of the label, increasing the width can result in overlapping the other label or the text field, result of which would be that one of the two labels disapper (there can be only one thing inside the cell). So please be careful about this and make sure there is enough space between the labels.

3. **Variable labels widths**
As the width of the label depends on its content, we cannot align both right and left edges of more labels and sometimes it might be better to replace labels with textboxes. For example, if there are many labels in a column, all left aligned like this:

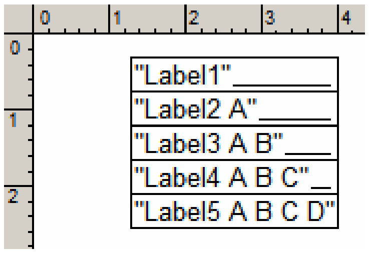


When creating the Excel document, XFRX will create a column for the right edge of each label:

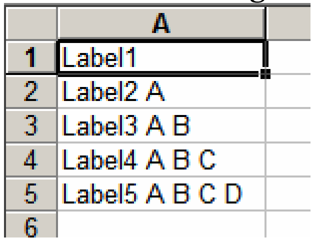


	A	B	C	D	E
1	Label1				
2	Label2 A				
3	Label3 A B				
4	Label4 A B C				
5	Label5 A B C D				
6					

However, if the labels are converted to textboxes, we can align them:



And the result might look better:



	A
1	Label1
2	Label2 A
3	Label3 A B
4	Label4 A B C
5	Label5 A B C D
6	

It is actually quite easy to convert all labels to textboxes, just open the report in FoxPro and replace the object type:

```
USE report.frx
REPLACE objtype WITH 8 ALL FOR objtype = 5
USE
```

13.10 Numeric field picture format in Excel

The format definition of numeric cells in Excel is different from the format syntax used in Foxpro. XFRX is now able to convert the simple format definitions and allows for user-defined implicit and/or explicit Excel-type format definitions.
In Visual Foxpro, the numeric field is converted to its text representation based on an explicit format definition (format field in the report expression definition) or field's decimal places and SET DECIMAL setting.

When a numeric field is transformed to an Excel cell, the following algorithm is used:

1. **If there is an explicit XLS format defined, use it.**

To define an explicit XLS format for a numeric field, add:
#UR XLSF= (expression)
to the field's comment.

Example 1:
#UR XLSF="General"
The "General" formatting - no special formatting, the number of decimal places is determined by the field's value

Example 2:
#UR XLSF="Standard"
The "Standard" formatting - two decimal places, thousand and decimal separators will be used according to the Excel defaults

Example 3:
#UR XLSF="#0.00"
Two decimal places, no leading zeros

Example 4:
#UR XLSF="#0.00;[red]#0.00"
Two decimal places, display negative numbers in red

2. **If the field contains a format definition, try to look up the format in a conversion table.**

A conversion table can be populated programmatically when XFRX is executed so that the formats that are often used and cannot by converted automatically by XFRX would not require an explicit definition in each field in the report.

To add an entry to the conversion table, use addXLSFormatConversion method of XFRXSession class.

Example 5:
loSession.addXLSFormatConversion("@L 999999.99","000000.00")

3. **If the field contains a format definition and it is not listed in the conversion table, try to convert it.**

XFRX is able to convert simple format definitions containing the following characters: '9', '#', ',', '.' and ' '. For example, 999,999.99 is converted to ###,###.00.

If the format cannot be converted, use the implicit XLS numeric format, if available.

To define the implicit XLS numeric format, call setDefaultXLSFormat method of the XFRXSession object with the implicit format as a parameter.

Example 6:
loSession.setDefaultXLSFormat("General")

4. **If the implicit XLS numeric format is not defined, XFRX creates a format definition to display the same number of decimals as in VFP report output.**

If you want to specify that a certain numeric field should be converted as a text cell in the excel sheet, define "TEXT" as its explicit XLS numeric format:
#UR XLSF='TEXT'
For a live example of formatting the Excel cells, please have a look at the "XLS formatted numeric cells sample" report in the demo application.

14 Converting reports to plain text

14.1 Limitations

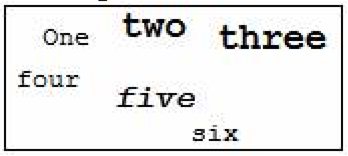
The limitations are defined by the nature of the plain text output:

- Font size and style are not converted (the text alignment is supported)
- Graphics on the report is ignored
- Rectangles and vertical lines are ignored, horizontal lines can be optionally converted as a series of dashes, for more information please see the Horizontal lines section below
- Overlapping objects are removed from the output
- XFRX needs to be able to split the objects to distinct lines, otherwise some objects would be removed form the output

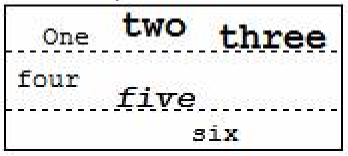
14.2 How it works

During the generation process, XFRX takes each section – one by one – and tries to split into to individual lines. Then each line is added to the output, respecting the vertical position of the line on the paper and horizontal positions of individual objects. Vertically misaligned objects are moved down to the closest baseline, for example:

This report definition:



Will be split to three lines:

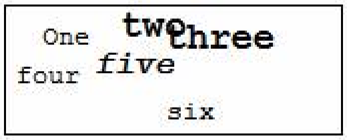


And the result would look like this:

```
 One two  three
four five
        six
```

If XFRX will not be able to create distinct lines or if objects would overlap one another, XFRX will try removing some of the objects.

For example:



will be converted as:

```
                One two
              Four
                      six
```

14.3 Horizontal lines

Horizontal lines can optionally be converted to a series of dashes. By default, this option is switched on. To suppress horizontal lines, call the setOtherParams method of XFRXSession object with two parameters: "PLAIN_SHOW_LINES" and .F., before calling processReport() method.

Example:

```
local loSession, lnRetval
loSession= xfrx("XFRX#INIT")
lnRetVal = loSession.SetParams("output.txt",,,,,,"PLAIN")
If lnRetVal = 0
      loSession.SetOtherParams("PLAIN_SHOW_LINES", .F.) && do not print
horizontal lines
      loSession.ProcessReport("report.frx")
      loSession.finalize()
ENDIF
```

14.4 Characters per inch setting

To convert the object absolute positions to character positions, XFRX calculates the number of characters per line and the number of lines per page using horizontal and vertical character-per-inch values. By default, these values are 10 characters per horizontal inch and 5.2 characters per vertical inch, which results in approximately 80 characters x 55 lines on a letter size paper.

You may want to change the character density (e.g. when using a condensed font on a dot matrix printer). To do this, send PLAIN_CPI_HORIZONTAL or PLAIN_CPI_VERTICAL as the first parameter of the SetOtherParams() method and the actual value as the second parameter.

Example:

```
local loSession, lnRetval
loSession= xfrx("XFRX#INIT")
lnRetVal = loSession.SetParams("output.txt",,,,,,"PLAIN")
If lnRetVal = 0
      * changing the character density to 12x6 per sq. inch
      loSession.SetOtherParams("PLAIN_CPI_HORIZONTAL",12)
      loSession.SetOtherParams("PLAIN_CPI_VERTICAL",8)
      loSession.ProcessReport("report.frx")
      loSession.finalize()
ENDIF
```

15 Using XFF files

XFRX is able to save the generated report to a binary file (XFF file). This file can be any time later used to:

- Transform the stored report output to any of the target formats supported by XFRX (PDF, Word, HTML, Excel, TXT, etc.)
- Print the stored report output
- Preview the stored report output in the XFRX previewer

You can also modify the content of existing XFF files or create new XFF files from scratch, bypassing the reporting engine altogether.

15.1 Converting reports to XFF files

To create a XFF file, send "XFF" as the target parameter and the name of the file as the output name parameter of the SetParams().

Example: *Creating a XFF file*

VFP 8 approach:

```
LOCAL loSession
loSession= xfrx("XFRX#INIT")
IF loSession.SetParams("output.xff",,,,,,"XFF") = 0
    loSession.ProcessReport("report")
    loSession.finalize()
ENDIF
```

VFP 9 approach:

```
LOCAL loObj
loObj = xfrx("XFRX#LISTENER")
if loObj.SetParams("output.xff",,,,,,"XFF") = 0
     REPORT FORM myreport object loObj
endif
```

By default, .XFF extension (XFrx File) is added to the output file. The XFF file is internally stored as a normal DBF file, and because Memo fields are used for some of its columns, another file with the same name and .FPT extension is created.

15.2 Initializing the XFRX#DRAW class instance

(Please see the XFRX#DRAW class reference on page 82).

XFRX methods do not work with the XFF file directly, but always via the XFRX#DRAW class, which represents a wrapper around the XFF file. To create an instance of this class, call XFRX with “XFRX#DRAW” parameter. Then call *openDocument(tcXFFFileName)* to attach the XFRX#DRAW object to an existing XFF file:

```
LOCAL loXFF
loXFF = xfrx("XFRX#DRAW")
IF loXFF.openDocument("output.xff")
      …
ELSE
      ? "XFF file cannot be open"
ENDIF
```

You can also create an empty XFF file by calling *CreateDocument(tcXFFFileName)* method.

15.3 Creating temporary XFF files

Sometimes you may want to create the XFF file just in memory, use it in your application (e.g. for report previewing and printing) and release it afterwards.

To do this, leave the XFF file name empty. In this case a temporary XFF file will be created and you will be able to access its XFRX#DRAW class instance. The way how to access the instance differs in VFP 8.0 and VFP 9.0.

VFP 8.0 approach

In VFP 8.0, the instance is returned by the Finalize() method of the XFRXSession class:

```
LOCAL loSession, loXFF
loSession= xfrx("XFRX#INIT")
IF loSession.SetParams(,,,,,,"XFF") = 0
    loSession.ProcessReport("report")
    loXFF = loSession.finalize()
ENDIF
*
* now the loXFF instance can be used as if the XFF file was opened
* with the openDocument method call
*
```

VFP 9.0 approach

In VFP 9.0, the instance is stored in oxfDocument property of the XFRXListener class:

```
LOCAL loObj, loXFF
loObj = xfrx("XFRX#LISTENER")

loObj.targetType = "XFF"
loObj.targetFileName = "" && output to a temporary file

REPORT FORM report OBJECT loObj

loXFF = loObj.oxfDocument
```

15.4 Converting XFF files to other output formats

To process the stored report, you need to initialize a XFRX#DRAW object, link it to the stored file and send it as a parameter of TransformReport method of XFRXListener class (in VFP 9.0) or XFRXSession class (in VFP 8.0) instance.

Example: *Transforming a stored XFF file to a PDF document.*

VFP 8 approach:

```
LOCAL loSession, loXFF
loSession= xfrx("XFRX#INIT")
loXFF = xfrx("XFRX#DRAW")
IF loReport.openDocument("output.xff")
    lnRetVal = loSession.SetParams("output.pdf",,,,,,"PDF")
    IF lnRetVal = 0
        loSession.TransformReport(loXFF)
    endif
ENDIF
```

VFP 9 approach:

```
LOCAL loObj, loXFF
loObj = xfrx("XFRX#LISTENER")
loXFF = xfrx("XFRX#DRAW")
IF loXFF.openDocument("output.xff")
    lnRetVal = loObj.SetParams("output.pdf",,,,,,"PDF")
    IF lnRetVal = 0
        loObj.transformReport(loXFF)
    ENDIF
ENDIF
```

15.5 Printing XFF files

To send the content of a XFF file to a printer, call printDocument method of the XFRX#DRAW class instance. The name of the printer, job name and page range can be sent as parameters.

Please see the full parameter list and further details in the XFRX#DRAW class reference.

Example: *Printing an XFF file*

```
loXFF.printDocument(getprinter(),"xfrx - invoice", "1,2,4-10")
```

15.5.1 Displaying printer properties dialog

A printer properties dialog for a given printer can be invoked via *_xfPrinterProperties* function, returning the printer properties structure as a string.

This string can be saved as a user preference and sent to XFRX as the 5th parameter of PrintDocument method when printing. This functionality is similar to SYS(1037) introduced in VFP 9.0, with two differences/improvements:

- The page setup and printer selection dialogs are skipped, which saves two clicks for the users and preempts confusions in case the printer has already been selected. (Very often, there is a printer selection box in the "main" form and a button to invoke printer properties).

- This implementation works in earlier versions of Visual FoxPro, too (from VFP 5.0)

The *_xfPrinterProperties* method takes 3 parameters:

tcPrinterName - the printer name

tcTag2 - the DEVMODE structure to use as a default (if not specified, the default printer settings will be used)

tlShowProperties - if set to .T., the printer properties dialog box will show up. If OK button is clicked, the DEVMODE structure with selected printer settings will be returned. If Cancel button is clicked, an empty string will be returned.
If this parameter is set to .F., the dialog will not be displayed and the default printer setting will be returned.

Example:

```
SET PROCEDURE TO xfrx ADDITIVE && this is required as the function is
implemented inside XFRX.FXP
lcPrinter = GETPRINTER() && select a printer
lcTag2 = _xfPrinterProperties(lcPrinter, "", .F.) && do not show the dialog,
return the default settings
lcTag2 = _xfPrinterProperties(lcPrinter, lcTag2, .T.) && show the dialog box
now
IF EMPTY(lcTag2)
      && CANCEL button was clicked
ELSE
      && OK button was clicked
ENDIF
```

15.5.2 Using custom printer settings when printing

The printer settings structure can be sent to *PrintDocument* method as the fifth parameter. If this parameter is empty, the default printer settings are used.

The printer settings structure can be retrieved by *_xfPrinterProperties* procedure (see the previous chapter), or, if the printer settings are saved with the report, it is stored in *Tag2* field in the first record of the FRX file.

Example:

```
loXFF.printDocument(lcPrinter, "job name", 1, 3, lcTag2)
```

15.6 Previewing XFF files

Please see chapter Previewing XFF files in the next chapter (XFRX previewer chapter on page 50) for more information about previewing XFF files in the advanced report previewer that comes with XFRX.

16 Drawing custom objects with XFRX#DRAW

The XFRX#DRAW class described in the previous chapter is also able to modify the content of XFF files: to add custom graphics, text, new pages, hyperlinks and bookmarks, both during the report generation process as well as after the report is generated.

16.1 Drawing custom objects to existing XFF files

To add custom graphics objects to XFF files, instantiate the XFRX#DRAW object as described in previous chapter and use the properties and methods described in chapter 16.3 XFRX#DRAW functions below. All changes are immediately written to the XFF file and once the XFRX#DRAW class reference is released, the changes are written to the disk (or other media the XFF file is stored on).

16.2 Drawing custom objects during report generation process

Alternatively, you can create scripts, which can be invoked on the fly, during the report generation process. These scripts can either be implemented as methods of a class, or, for rectangle-bound scripts, can be entered into a comment field of a rectangle object on the report.

Note: In XFRX for VFP 9.0, the XFF file is always internally used when a report is processed. However, in XFRX for VFP 8.0, you can generate the target documents "directly", in which case, however, the XFF scripts would not be evaluated.

16.2.1 Page-bound scripts

The page-bound scripts are implemented as methods of an arbitrary class – you can use a custom class that is instantiated before XFRX is executed, implement the scripts as methods of the current form or whatever else is convenient in your application environment. Each script method accepts one parameter: the XFRX#DRAW object reference. This reference links to the document that is being generated.

Example:

```
DEFINE CLASS myXFRXScripts as Custom

PROCEDURE drawBlueRectangle
LPARAMETERS oXFD
      oXFD.setColor(0,0,255,-1,-1,-1)
      oXFD.drawRectangle(100,200,50,50)
ENDPROC

ENDDEFINE
```

Before XFRX is called, the scripts are registered with *RegisterScript()* in the XFRX session class instance so that XFRX new about these scripts and invoked them as required. You can execute the scripts on each page, odd or even pages or arbitrary pages. Please see Methods common in XFRXListener and XFRXSession classes chapter for detail description of this method and its parameters.

Example:

```
*
*Draw blue rectangle on each page
*
loScripts = createobject("myXFRXScripts")
loXFRXSession.registerScript(loScripts,"drawBlueRectangle",0,"ALL",0)
```

16.2.2 Rectangle-bound scripts

The rectangle-bound scripts are bound to a specific rectangle on the report and are executed *instead* of drawing this rectangle to the output.

There are two ways how the scripts can be implemented: named scripts, which are implemented as a method of a class, or scripts written directly to the comment field of the rectangle in the report definition.

16.2.2.1 Named scripts

If the script is implemented as a method of a class, it needs to be registered similarly as page-bound scripts, but only the first two parameters are required – the object reference and the script name:

```
loScripts = createobject("myXFRXScripts")
loXFRXSession.registerScript(loScripts,"drawBlueRectangle")
```

To link the script with a rectangle, add "#UR SCRIPT NAME <script name>" (without the quotation marks) as a comment of the rectangle.

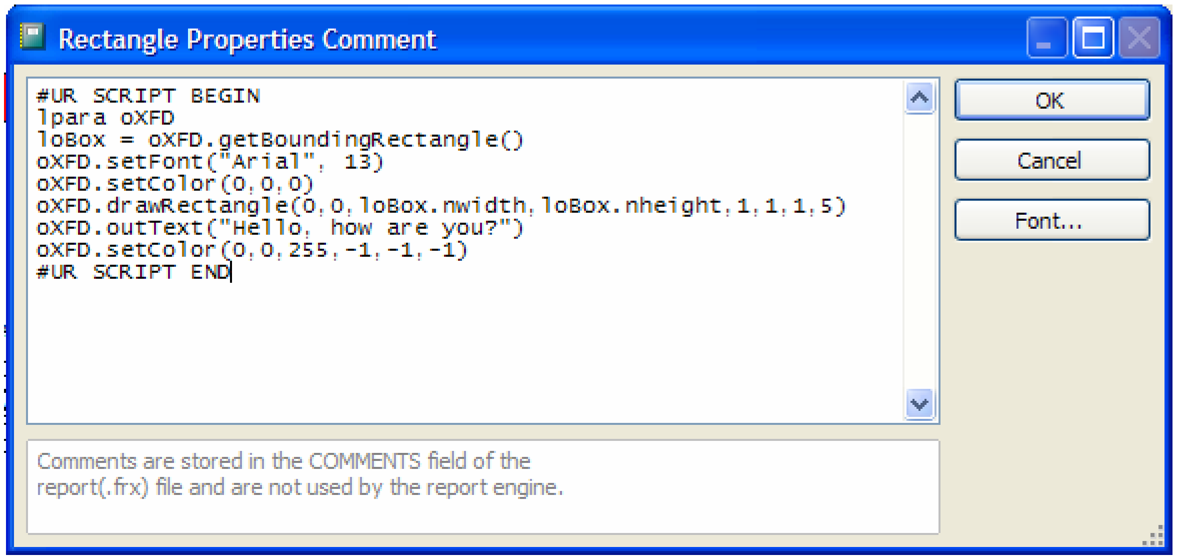
16.2.2.2 Scripts written in the rectangle comment field

Alternatively, the script can be entered into the comment field of the rectangle. This technique is available for VFP 8.0 and higher, but its usage is limited for VFP 8.0 because the comment field is small and only a limited amount of text can be entered into it.

The script is entered between #UR SCRIPT BEGIN and #UR SCRIPT END lines.

These scripts are neither named not registered. The content between these lines is supposed to accept the XFRX#DRAW object reference parameter and is evaluated via EXECSCRIPT( ) function.

Example:



```
#UR SCRIPT BEGIN
lpara oXFD
loBox = oXFD.getBoundingRectangle()
oXFD.setFont("Arial", 13)
oXFD.setColor(0,0,0)
oXFD.drawRectangle(0,0,loBox.nwidth,loBox.nheight,1,1,1,5)
oXFD.outText("Hello, how are you?")
oXFD.setColor(0,0,255,-1,-1,-1)
#UR SCRIPT END
```

Comments are stored in the COMMENTS field of the report(.frx) file and are not used by the report engine.

When inside of the rectangle-bound script, the coordinates origin is shifted: 0,0 represents the upper left hand corner of the rectangle.

Note: The position and size of the bounding rectangle can be retrieved with *GetBoundingRectangle()* method.

16.2.2.3 Converting rectangle-bound objects to pictures

The content of the rectangle-bound scripts can be alternatively rendered as a picture, which is then placed to the output. To do this, add PICTURE clause to the #UR SCRIPT BEGIN or #UR SCRIPT NAME <script name> line in the comment field.

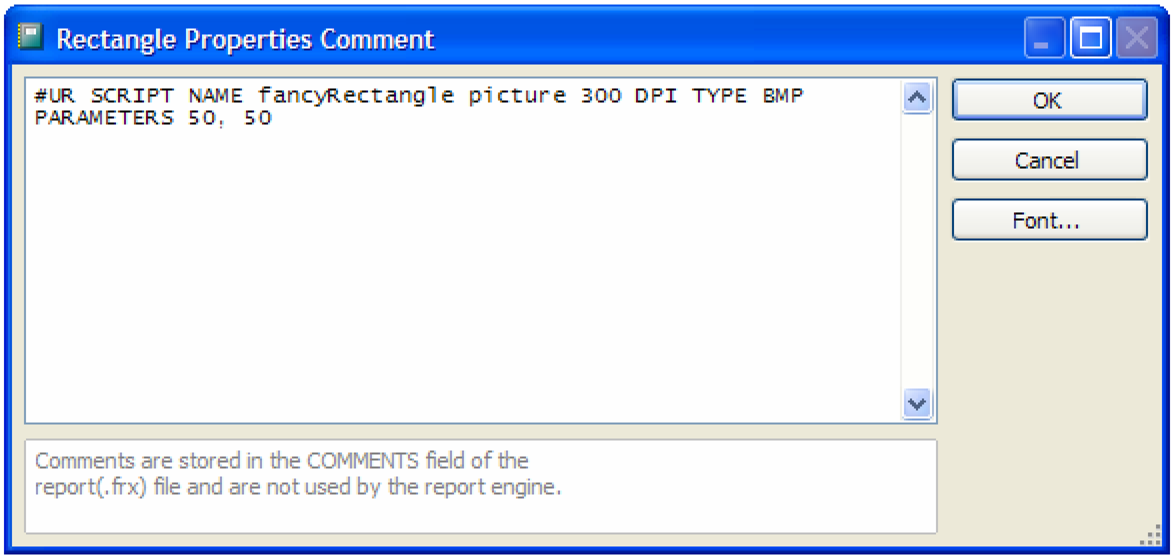
The PICTURE can optionally be followed by a DPI value, representing the DPI (dots per inch) of the picture created. The default DPI is 96, which is suitable for previewing on screen, but you may want to increase the DPI value to increase the picture quality.

You can also specify the picture type by adding TYPE clause followed by one of the following values: BMP, JPG, PNG and TIF. The default picture type is JPG.

16.2.2.4 Sending parameters to named scripts

The named scripts can optionally accept further parameters. The parameters are specified via PARAMETERS clause followed by the parameters, which are evaluated at the time of report processing.

Example:



```
#UR SCRIPT NAME fancyRectangle picture 300 DPI TYPE BMP
PARAMETERS 50, 50
```

Comments are stored in the COMMENTS field of the report(.frx) file and are not used by the report engine.

```
DEFINE CLASS myXFRXScripts as Custom
       PROCEDURE fancyRectangle
              lpara oXFD, boxwidth, boxheight
              *
              * This script draws little boxes around the bounding rectangle.
              * The height and width of the rectangles can be sent as parameters.
              * The default width and height is 5 points. (1 point = 1/72 inch).
              *
              IF EMPTY(boxwidth)
                     boxwidth = 5
              endif
              IF EMPTY(boxheight)
                     boxheight = boxwidth
              endif
              oXFD.setUnit("pt")
              loBox = oXFD.getBoundingRectangle()
              oXFD.setColor(0,0,0)
              LOCAL i
              FOR i = 0 TO INT(loBox.nwidth/BOXWIDTH)-1

       oXFD.setColor(0,0,0,INT(RAND()*256),INT(RAND()*256),INT(RAND()*256))
                     oXFD.drawRectangle(i*BOXWIDTH, 0, BOXWIDTH, boxheight,1,1)

       oXFD.setColor(0,0,0,INT(RAND()*256),INT(RAND()*256),INT(RAND()*256))
                     oXFD.drawRectangle(i*BOXWIDTH, ;
                            INT(loBox.nheight/boxheight-1)*boxheight, BOXWIDTH,
                     boxheight,1,1)
              endfor
              FOR i = 1 TO INT(loBox.nHeight/boxheight)-2

       oXFD.setColor(0,0,0,INT(RAND()*256),INT(RAND()*256),INT(RAND()*256))
                     oXFD.drawRectangle(0, i*boxheight, BOXWIDTH, boxheight,1,1)

       oXFD.setColor(0,0,0,INT(RAND()*256),INT(RAND()*256),INT(RAND()*256))
                     oXFD.drawRectangle(int(loBox.nwidth/BOXWIDTH-1)*BOXWIDTH, ;
                            i*boxheight, BOXWIDTH, boxheight,1,1)
              endfor
       ENDPROC
ENDDEFINE
```

16.3 XFRX#DRAW functions

This chapter gives a brief list of properties and methods implemented in the XFRX#DRAW class. For detailed description if the properties, methods and their parameters please see the XFRX#DRAW class reference on page 82).

16.3.1 XFF document navigation

Whenever you draw anything to a XFF document, the drawing is always added on top of the "current page". The current page is stored in a readonly property *CurrentPage*. The total number of pages is stored in *PageCount* property. To navigate on the pages, use *GoTop(), GoBottom()* and *GoToPage(nPageNo)* methods.

16.3.2 Adding pages

AddPage() method creates a new page at the end of the document. You can either specify the size of the new page or use the same size as the current page.

16.3.3 Coordinates units

The default unit is Point (1/72 inch). You can use *setUnit(cUnit)* method to switch to a different unit. The available units are: centimeters (cm), inches (in), points (pt) and pixels (px).

16.3.4 Drawing functions

Use *SetColor()* method to set colors for subsequent drawing method calls, the default colors are black for pen and white for background.

Drawing text

Use *setFont()* method to set the font name, size and attributes. *DrawText()* draws a text either at a given position, at a position defined by *SetPos()* method or at end of an output of the previous *DrawText()* method call – "current position". After each *DrawText()* method call, the "current position" advances depending on the height and width of the text. *GetXPos()* and *GetYPos()* return the current position. *DrawTextBox()* draws a text into a given bounding rectangle – word wraps long lines and optionally expands the height of the bounding box to accommodate the whole text.

To maintain backward compatibility with PDF Library, *OutText()* is a synonym of *DrawText()*.

Drawing basic shapes

drawLine() draws a point to point line, *drawRectangle()* draws a rectangle, rounded rectangle or an ellipse.

Drawing pictures

DrawPicture() method draws a picture within a given bounding rectangle.

16.3.5 Tooltips, hyperlinks and bookmarks

Each text drawn by *DrawText()* or *DrawTextBox()* can have a tooltip, can serve as a target for a bookmark and can be a hyperlink or a target or a hyperlink. Please see *tcLinkName, tcLinkRef, tcBookmark* and *tcTooltip* parameters of *DrawText()* and *DrawTextBox()* methods in XFRX#DRAW class reference on page 82).

17 XFRX previewer

XFRX ships with its own report previewer. The previewer allows for:

- Directing the report generation to the previewer or to preview an existing XFF file
- Searching the document
- Exporting the document to any of the supporting output formats
- Printing the document
- Navigating via hyperlinks and bookmarks
- Invoking FoxBase code on hyperlink click (custom events)

The previewer is implemented as a container object, so it can be very easily added inside any VFP form.

Please note: XFF files don't have to been stored as physical files. You can also create temporary XFF files that will be automatically released when you close the previewer. (Please see Creating temporary XFF files on page 41 for more information).

17.1 Setting up the XFRX previewer

(Note: this section has been added to reflect changes in XFRX ver. 11.3. If you are using previous XFRX versions, please see Using XFCont class paragraph below in this chapter).

There are several ways how you can setup the XFRX previewer. You can either use the already prepared standalone previewing component or start with the basic classes of the previewer component to add the previewing capabilities to your own forms.

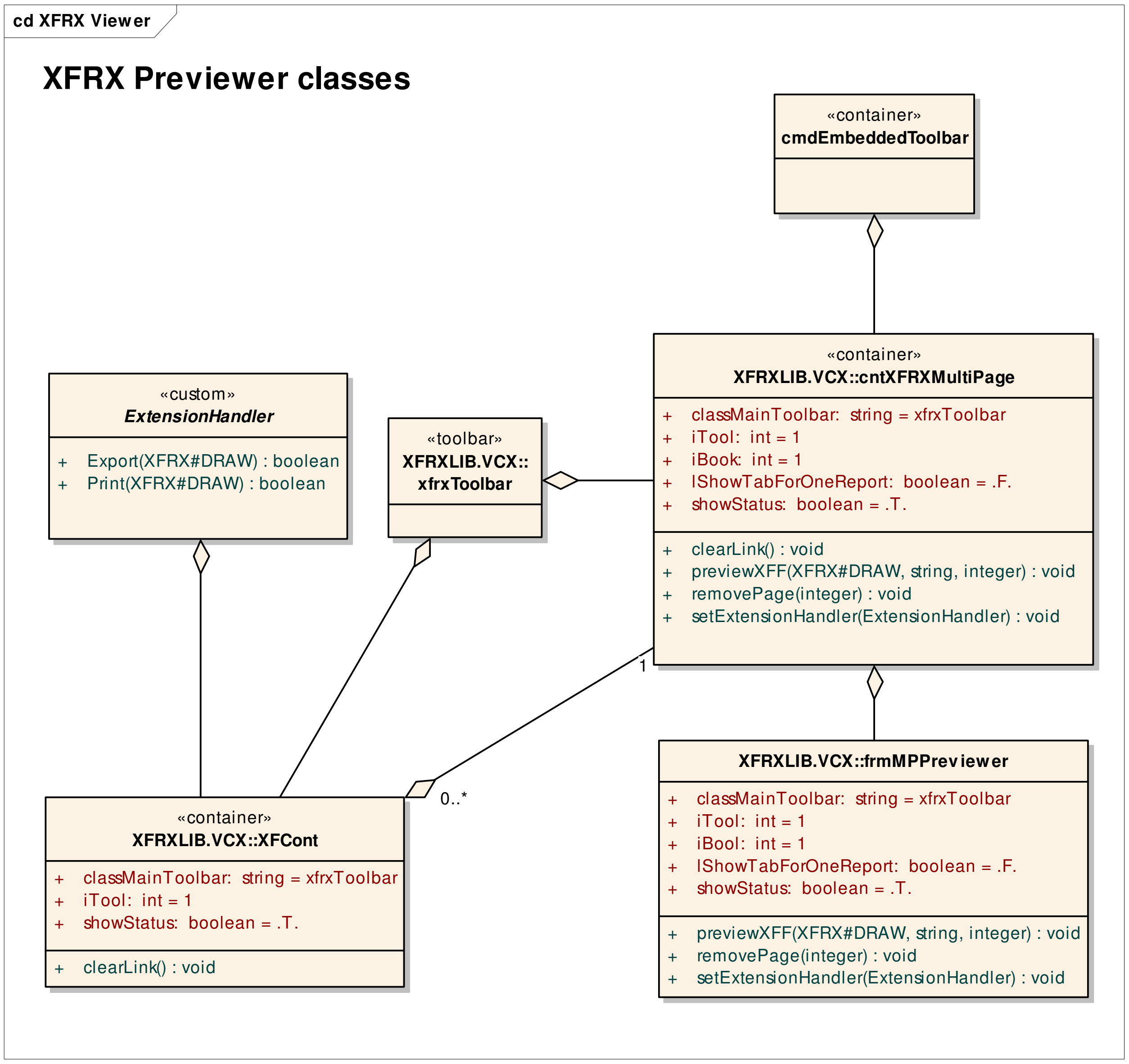
The following subchapters describe using three classes implemented in XFRXLIB.VCX which you can use as a starting point to implement the XFRX previewer in your application:

	frmMPPreviewer	cntXFRXMultiPage	XFCont
Implemented as a form (the easiest setup)	Yes	No	No
Implemented as a container (can be embedded into existing forms)	No	Yes	Yes
supports multi-tab interface	Yes	Yes	No
supports embedded toolbar	Yes	Yes	No

- **XFCont** is the basic XFRX previewer class. It is implemented as a container object and supports searching, hyperlinks, bookmarks, printing and exporting.
- **cntXFRXMultiPage** class extends the XFCont class functionality with multi-tab interface – it can display multiple previewer tabs in a page frame class. It also supports a (fake) embedded toolbar, which is a toolbar-like control at the top of

the container, which is very useful if the previewer form is not inside the main VFP screen.

- **frmMPPreviewer** is a form class that implements **cntXFRXMultiPage** and serves as a "proxy" to the cntXFRXMultiPage class features. If all you need is a full screen report previewer, this one is the easiest to setup.


cd XFRX Viewer
XFRX Previewer classes
«container»
cmdEmbeddedToolbar
«container»
XFRXLIB.VCX::cntXFRXMultiPage
+ classMainToolbar: string = xfrxToolbar
+ iTool: int = 1
+ iBook: int = 1
+ lShowTabForOneReport: boolean = .F.
+ showStatus: boolean = .T.
+ clearLink() : void
+ previewXFF(XFRX#DRAW, string, integer) : void
+ removePage(integer) : void
+ setExtensionHandler(ExtensionHandler) : void
«custom»
ExtensionHandler
+ Export(XFRX#DRAW) : boolean
+ Print(XFRX#DRAW) : boolean
«toolbar»
XFRXLIB.VCX::
xfrxToolbar
1
0..*
«container»
XFRXLIB.VCX::XFCont
+ classMainToolbar: string = xfrxToolbar
+ iTool: int = 1
+ showStatus: boolean = .T.
+ clearLink() : void
XFRXLIB.VCX::frmMPPreviewer
+ classMainToolbar: string = xfrxToolbar
+ iTool: int = 1
+ iBool: int = 1
+ lShowTabForOneReport: boolean = .F.
+ showStatus: boolean = .T.
+ previewXFF(XFRX#DRAW, string, integer) : void
+ removePage(integer) : void
+ setExtensionHandler(ExtensionHandler) : void

17.1.1 Common methods and properties

Even though you can use different classes to provide the previewing capabilities in your application, all classes share most of the properties and methods you can use to control and customize them.

17.1.1.1 Controlling multi-tab preview features

Note: The properties and methods described in this paragraph are implemented in cntXFRXMultiPage and frmMPPreviewer classes. The multi-tab features are not implemented in XFCont class.

Property / Method	Usage
lShowTabForOneReport (property)	Possible values: • .F. (default) – the tab with page captions is hidden if only one report is previewed. It will became automatically visible when the second report gets previewed. • .T. – the tab with page captions is always visible.
PreviewXFF (method)	Syntax: PreviewXFF(toXFF, tcCaption, tnPageNo) Call this method to preview the XFF file in the previewer. You can call this method multiple times, which would either add new reports as new pages or replace existing pages, depending on parameters. Parameters: *toXFF* The reference of a XFF file that is going to be previewed *tcCaption (optional)* The caption of the page with this report if previewing multiple reports. *tnPageNo(optional)* The page on which the XFF file is going to be previewed. If this parameter is empty or greater than the number of existing pages, a new page is added. If a page with this number already exists, it will be replaced.
RemovePage	Syntax:

(method)	RemovePage(tnPageNo) Removes a page from the previewer. Parameters: *tnPageNo (optional)* The number of the page to remove. If the parameter is empty or the number is greater than the number of existing pages, the last one is removed.

17.1.1.2 Registering an extension handler

The extension handler allows for extending the functionality of the basic classes without need of creating subclasses. All you need is to create a custom class, which implements certain methods. The custom class is registered with the XFRX previewer class (using SetExtensionHandler method) and the previewer calls the extension methods on appropriate events.

Currently supported event methods are:

Method	Usage
Export	This method is called when the Export button is clicked in the toolbar. Parameters: toXFF – the handler of the XFF file that is being previewed
Print	This method is called when the Print button is clicked in the toolbar. Parameters: toXFF – the handler of the XFF file that is being previewed

Example:

```
SET PATH TO xfrxlib
SET CLASSLIB TO xfrxlib

LOCAL loPreview, loSession, loExtensionHandler

loExtensionHandler = CREATEOBJECT("myExtensionHandler")

loPreview = CREATEOBJECT("frmMPPreviewer")
loSession=EVALUATE([xfrx("XFRX#LISTENER")])

*
* create a memory XFF file
*
lnRetVal = loSession.SetParams(,,,,,,"XFF")
If lnRetVal = 0
      SELECT * ;
            FROM customers JOIN orders ON customers.customerid =
orders.customerid ;
            JOIN orderdetails ON orders.orderid = orderdetails.orderid ;
            order by customers.companyname, customers.customerID,
orders.orderID ;
            INTO CURSOR custlist

      REPORT FORM custlist OBJECT loSession
ENDIF

*
* assign the extension handler
*
loPreview.setExtensionHandler(loExtensionHandler)
*
* preview the report
*
loPreview.previewXFF(loSession.oxfdocument)
loPreview.show(1)

DEFINE CLASS myExtensionHandler as Custom
      PROCEDURE Print
            LPARAMETERS toXFF
            RETURN .t. && continue with the default behavior
      ENDPROC

      PROCEDURE Export
            LPARAMETERS toXFF
            *
            * now you can process the XFF file
            *
            RETURN .F. && override the default behavior
      ENDPROC
ENDDEFINE
```

17.1.1.3 Showing / hiding bookmarks

The bookmark panel behavior is controlled by the iBook property:

Property	Usage
iBook	Controls if the bookmark should be displayed in the previewer. The allowed values are: • -1 = Disable bookmarks • 0 = Enable bookmarks but hide them (users need to click on the bookmark button to see them) • 1 = Enable bookmarks, always showing them (default) • 2 = Enable bookmarks, but show or hide them automatically based on whether or not there are bookmarks defined in the report to preview (default)

17.1.1.4 Showing / hiding previewer toolbars

There are two toolbars in the previewer control – one is placed at the bottom of the control, the other one is available as a floating toolbar panel, which by default docks at the top of the main window, or, alternatively, as an "embedded toolbar". All three classes implement *iTool* and *ShowStatus* properties that control visibility of the toolbars:

Property	Usage
classMainToolbar	The class name of the toolbar. The default value is "xfrxToolbar". You can create a child of the default xfrxToolbar class (defined in xfrxlib.vcx) and use this property to instruct the previewer to use your class rather than the default one. This property works for the "real" toolbar – if iTool property is 0 or 1. if iTool is set to 2, cmdEmbeddedToolbar container class is used as the "embedded" toolbar.
iTool	Controls the visibility of the toolbar at the top. The allowed values are: • -1 = Disable toolbar • 0 = The toolbar will be enabled but hidden. It can be invoked via right click shortcut menu

	• 1 = The toolbar will be enabled and visible. (default) • 2 = The toolbar will be displayed as embedded to the form (available only in cntXFRXMultiPage and frmMPPreviewer classes.
ShowStatus	Controls the visibility of the toolbar at the status bar of the previewer container. The allowed values are: • .T. = show the toolbar (default) • .F. = hide the toolbar

17.1.2 Using frmMPPreviewer class

This class serves as a standalone (fullscreen) previewer window. All you need to do to make it work is to pass it a reference to the XFF file to preview, using the previewXFF method, as described in Controlling multi-tab preview features paragraph above. (If you are not sure how to create a XFF file reference, please read chapters 15.2: Initializing the XFRX#DRAW class instance and 15.3: Creating temporary XFF files). Here is a very simple “Hello World” example:

```
SET CLASSLIB TO XFRXLIB.VCX ADDITIVE loPreview =
CREATEOBJECT("frmMPPreviewer") && create the previewer object
toViewer.previewXFF(loXFF)                && loXFF is XFF file reference
loPreview.show(1)                         && show the form as modal
```

17.1.3 Using cntXFRXMultiPage class

1. Add an instance of cntXFRXMultiPage class (of XFRXLIB.VCX) into your form. Adjust the size of the cntXFRXMultiPage class object container as you need. The previewer, including the status bar, icons and scroll bars will be displayed in the container. If you resize the container in runtime, its content is automatically adjusted.
2. When the form is about to be released, the cntXFRXMultiPage class instance needs to do some cleanup. To enable that please call the class instance’s `clearLink()` method from form’s destroy method:

```
procedure Destroy
    thisform.cntXFRX.clearLink()
endproc
```

3. When running the form, make sure the XFRXLIB.VCX class library is referenced in SET CLASSLIB command:

```
SET CLASSLIB TO XFRXLIB.VCX ADDITIVE
```

4. Call the PreviewXFF method to preview a XFF file, as described in Controlling multi-tab preview features paragraph above.

17.1.4 Using XFCont class

1. Add an instance of XFCONT class (of XFRXLIB.VCX) into your form. Adjust the size of the XFCONT class container as you need. The previewer, including the status bar, icons and scroll bars will be displayed in the container. If you resize the container in runtime, its content is automatically adjusted.
2. When the form is about to be released, the XFCONT class instance needs to do some cleanup. To enable that please call the XFCont class instance's clearLink() method from form's destroy method:

```
procedure Destroy
    thisform.cntXFRX.clearLink()
endproc
```

3. When running the form, make sure the XFRXLIB.VCX class library is referenced in SET CLASSLIB command:

```
SET CLASSLIB TO XFRXLIB.VCX ADDITIVE
```

17.1.4.1 VFP 8.0 approach

In VFP 8.0, there are two options how to preview report in the previewer. The options differ in the calling mechanism, although internally they both work exactly the same.

17.1.4.1.1 Generating the "CNT" output type

The approach is using the standard syntax of calling XFRX, without explicitly using the XFRX#DRAW class (although it is used in the background). You cannot open an existing XFF file with this approach – you always need to run a report that will be previewed.

1. Do not send anything as the output file name
2. Use "CNT" as the output type parameter in the SetParams method call
3. Call Reset method of the previewer container class
4. Call SetOtherParams method of the XFRXSession class instance with the reference of XFCont instance as a parameter

Example:

```
local loSession, lnRetval
loSession= xfrx("XFRX#INIT")
lnRetVal = loSession.SetParams(,,,,,,"CNT")
If lnRetVal = 0
      *
      * we are assuming the XFCont container is
      * available as thisform.cntXFRX
      *
      thisform.cntXFRX.reset()
      loSession.setotherparams(thisform.cntXFRX)
      loSession.ProcessReport("report")
      loSession.finalize()
ENDIF
```

17.1.4.1.2 Using the XFRX#DRAW class

The advantage of this method is that you can preview an existing XFF file – the XFF file is opened and "converted" to "CNT" output type via the TransformReport method:

Example:

```
      LOCAL loSession
      loSession= xfrx("XFRX#INIT")
      loReport = xfrx("XFRX#DRAW")
      IF thisform.oReport.openDocument("file.xff")
            thisform.cntXFRX.reset()
            lnRetVal = loSession.SetParams(,,,,,,"CNT")
            IF lnRetVal = 0
                  loSession.setOtherParams(thisform.cntXFRX)
                  loSession.TransformReport(thisform.oReport)
            endif
      ENDIF
```

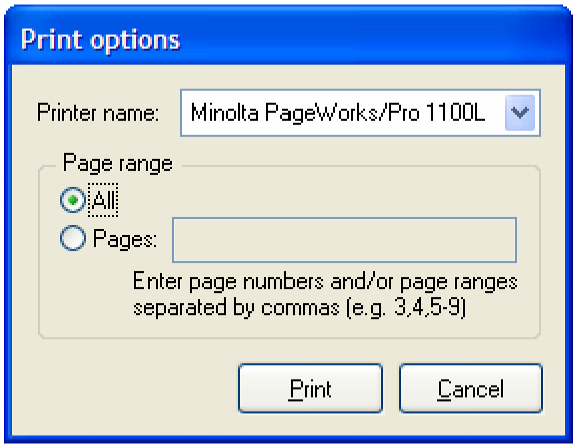
17.1.4.2 VFP 9.0 Approach

In VFP 9.0 the logic is the same, just the syntax is a little bit different – the reference to the XFCont object is copied to XFRXPreviewer property of the XFRXListener class instance and then previewReport method with the XFRX#DRAW reference as parameter is called:

```
      LOCAL loSession
      loSession= xfrx("XFRX#LISTENER")
      thisform.oReport = xfrx("XFRX#DRAW")
      IF thisform.oReport.openDocument("file.xff")
            thisform.cntXFRX.reset()
            loSession.XFRXPreviewer = thisform.cntXFRX
            loSession.previewReport(thisform.oReport)
      ENDIF
```

17.2 Printing from the previewer

By default, when users click the “Print” button in the previewer toolbar (🖨), the Print method of the XFCont class is called. This method displays a dialog box with a printer selection and allows for entering a page range:



Printer name: Minolta PageWorks/Pro 1100L

Page range: [✓ All] Pages:

Enter page numbers and/or page ranges separated by commas (e.g. 3,4,5-9)

When a XFF file is previewed, the corresponding XFRX#DRAW reference is accessible via oXFRXWriter property of the previewer class instance.

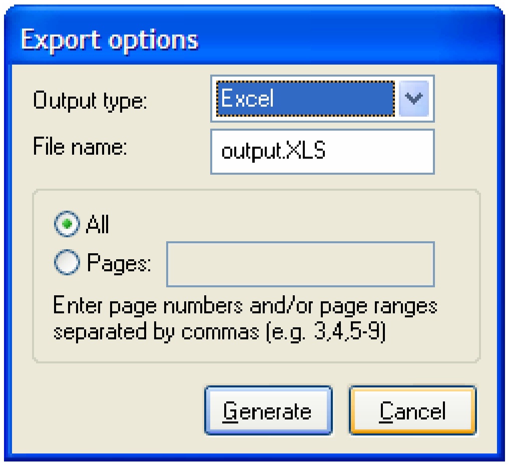
If you would like to change the way the previewer reacts to the print event, you can either setup an extension handler (please see Registering an extension handler paragraph above in this chapter for more information) or you can override the Print method, create your own printing dialog and then run XFRX#DRAW’s printDocument method. The source code of XFCont class is available so you can use the current Print method as a template.

The printer job name generated by the Print method is stored in cJobName property of the XFCont class and can be changed both in design time and runtime. (The default value is “XFRX”).

See also: Printing XFF files chapter on page 42 for more information about printing the content of XFF files.

17.3 Exporting reports from the previewer

Similarly to printing, when users click the “Export” button in the previewer toolbar (⇨), the Export method of the XFCont class is called. This method displays a dialog box with output type and a file name selection and converts the XFF file that is being previewed to the output document:


Export options

Output type: Excel

File name: output.XLS

[✓ All]

Pages:

Enter page numbers and/or page ranges separated by commas (e.g. 3,4,5-9)

If you would like to create your own custom exporting solution, you can either setup an extension handler (please see Registering an extension handler paragraph above in this chapter for more information) or you can create a child of the XFCont class and override the Export method.

17.4 Previewer localization

Each localization is stored in two files:

- XFRXLIB_<localization_code>.dbf - strings translation
- XFRXLIB_<localization_code>.vcx - find and page selection dialogs localization

To create a new localization, please follow these steps:

1. Choose a code for your localization (e.g. DE for German)
2. Create a copy of default localization DFB (localization\default\XFRXLIB_XXX.DBF (+CDX, +FPT)). Change the suffix according to your localization code. The default localization DBF contains English strings that need to be translated.
3. Create XFRXLIB_<localization_code>.vcx visual class library using localization\default\locClass.prg. Send the localization code as a parameter. Example: Copy locClass to the XFRXLIB directory and call:

   ```
   do locClass with "DE"
   ```

   which will create the XFRXLIB_DE class library. Translate the two dialog classes in the class library.
4. Copy the localization files to XFRXLIB directory.

To activate the localization in your code, call setLanguage method of XFCont class. Send the localization code as a parameter.

Example:

```
this.cntXFRX.setLanguage("DE")
```

18 Implementing custom event hyperlinks (drilldown) in the XFRX previewer

XFRX recognizes two types of hyperlinks:

- "normal" hyperlinks (printed in blue), which navigate to other places in the report or to an external web address (using hyperlinks is described in detail in "Interactive features / Hyperlinks" chapter on page 19) and
- "custom event" hyperlinks (printed in green), where XFRX allows for assigning a custom VFP code that will be called when users click the hyperlink.

This "custom event" hyperlink feature can be used for invoking application specific actions (information forms, custom processes, etc.) or for implementing drill-down functionality – invoking detailed report where the field user clicked on is taken as a parameter for the report (for example, clicking a customer name in the report listing all customers can run a report with detail information about this specific customer). The new report can be directed to a new page of a multipage previewer, which could provide a comfortable environment for "drilling down" specific information – with the ability to go back to the original report without closing the current one, side by side report comparison, exporting / printing selected reports, etc.

18.1 Setting up custom event hyperlinks

The custom event hyperlinks are defined the same way as normal hyperlinks: you add #UR A HREF= to the label or field comment, followed by "vfpev" as a "protocol", followed by a FoxPro expression to evaluate.

Example: *Hello world*

```
#UR A HREF="vfpev:\\messagebox('Hello world')"
```

If you add this to a field on your report a "Hello world" messagebox pops up whenever you click on that field.

Please be aware that the text following after HREF= is evaluated during the report execution, at the time the field is about to be rendered. The result is then stored with that particular field and evaluated again when users click on the field. For example, if you need to call your function with a customer ID as a parameter (for example stored in customers.cusID as an integer value), you use something like this:

```
#UR A HREF="vfpev:\\myfunction("+trans(customers.cusID)+")"
```

During the report generation the customers.cusID expression is evaluated and the results (e.g. myfunction(1), myfunction(2), etc.) are stored with the individual fields. Myfunction function with the stored parameter is then called when users click on the field.

18.2 Accessing the calling previewer

In case the custom event should result in running a new report, the routine that processes the new report may want to previewer the resulting output in the original previewer, in which the click event occurred. In that case, you can use **thisviewer** variable as the previewer reference. For example, the following hyperlink definition would be sending the previewer reference as the second parameter of runreport method:

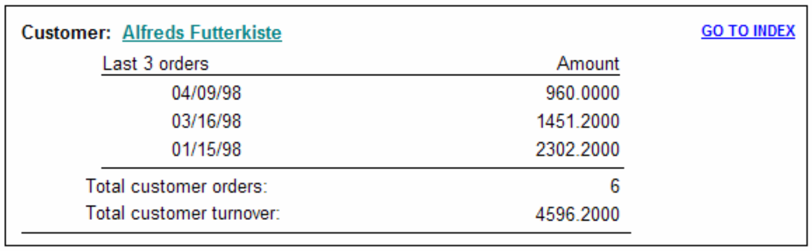
```
#UR A HREF="vfpev:\\runreport('"+allt(customerid_a)+"',thisviewer)"
```

18.3 Drilldown solution example

(The following example is available in the DrilldownSample subdirectory of the evaluation as well as commercial package).

The sample solution consists of three reports:

- A customer index page followed by a brief list of customers. For each customer there is a list of last three orders, total number of orders and the total turnover:

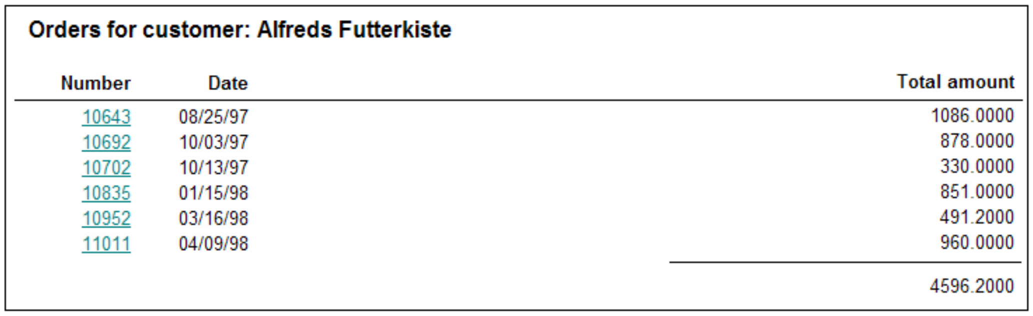


Customer: Alfreds Futterkiste

GO TO INDEX

Last 3 orders	Amount
04/09/98	960.0000
03/16/98	1451.2000
01/15/98	2302.2000
Total customer orders:	6
Total customer turnover:	4596.2000

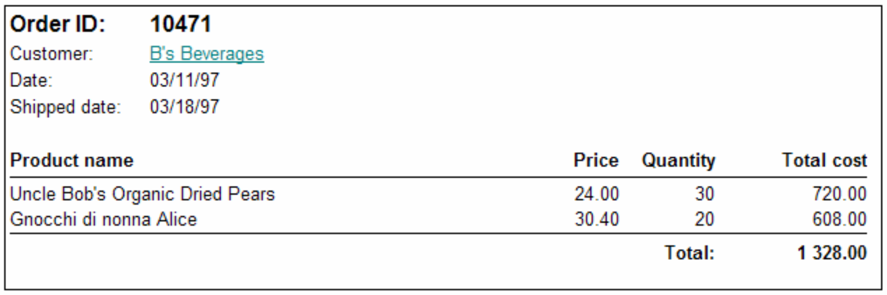
- When you click on the customer name, a new report with customer details is generated, listing all customer's orders:



Orders for customer: Alfreds Futterkiste

Number	Date	Total amount
10643	08/25/97	1086.0000
10692	10/03/97	878.0000
10702	10/13/97	330.0000
10835	01/15/98	851.0000
10952	03/16/98	491.2000
11011	04/09/98	960.0000
		4596.2000

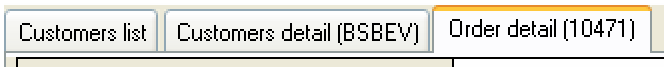
- By clicking on an order number, an order detail report is generated:



Order ID: **10471**
Customer: B's Beverages
Date: 03/11/97
Shipped date: 03/18/97

Product name	Price	Quantity	Total cost
Uncle Bob's Organic Dried Pears	24.00	30	720.00
Gnocchi di nonna Alice	30.40	20	608.00
		Total:	**1 328.00**

Whenever a new report is generated, it is added as a new page to the previewer container:



Here is the full source code:

```
Lparameters tnReportType, tuPar1, toViewer
Set Path To src; xfrxlib; libs; drilldownsample

Local loSession, lnRetval, lcPageCaption, loPreview

If Empty(tnReportType)
  *
  * no report type was sent - we need to initialize the previewer
  * and run the 1st report
  *
  Set Classlib To xfrxlib
  loPreview = Createobject("frmMPPreviewer")
  tnReportType = 1
  toViewer = loPreview
Endif

*
* initialize the XFRX listener
*
loSession=Evaluate([xfrx("XFRX#LISTENER")])
lnRetval = loSession.SetParams(,,,,,,"XFF")
If lnRetval = 0
  *
  * now let's see which report we want to run, select the data
  * and run the report
  *
  Do Case
  Case tnReportType = 1
    Select companyname From customers Order By companyname Into Cursor
custindex
    Report Form custindex Object loSession Nopageeject
    Select * ;
      FROM customers Join orders On customers.customerid = orders.customerid ;
      JOIN orderdetails On orders.orderid = orderdetails.orderid ;
      order By customers.companyname, customers.customerid, orderDate Desc,
orders.orderid ;
      INTO Cursor custlist
    Report Form custlist Object loSession
    lcPageCaption = "Customers list"

  Case tnReportType = 2
    Select * ;
      FROM customers Join orders On customers.customerid = orders.customerid ;
      JOIN orderdetails On orders.orderid = orderdetails.orderid ;
      order By customers.companyname, customers.customerid, orders.orderid ;
      WHERE customers.customerid = tuPar1 ;
      INTO Cursor custlist
    Report Form custDet Object loSession
    lcPageCaption = "Customers detail ("+Alltrim(tuPar1)+")"

  Case tnReportType = 3
    Select * ;
      FROM customers Join orders On customers.customerid = orders.customerid ;
      JOIN orderdetails On orders.orderid = orderdetails.orderid ;
      JOIN products On products.productid = orderdetails.productid ;
      order By customers.companyname, customers.customerid, orders.orderid ;
      WHERE orders.orderid = tuPar1 ;
      INTO Cursor custlist
    Report Form OrdDet Object loSession
    lcPageCaption = "Order detail ("+Alltrim(Str(tuPar1))+")"
  Endcase
Endif

*
* now preview the report
*
toViewer.previewXFF(loSession.oxfdocument, lcPageCaption)
*
* show the preview if not yet visible
*
If loPreview.Visible = .F.
  *
  * preview in modal Windows
  *
  loPreview.Show(1)
Endif
```

As you can see, the code is first called without parameters, which automatically runs the first report (customer list). Then the same code is then called (recursively – as the previewer is in modal mode) from the previewer when users click on the custom event with parameters controlling which report should be run, which parameters should be used for the select statement and what should be the caption of the corresponding page in the previewer.

The report fields' comments are defined as follows:

Report	Field	Comment
Customer list	Customer name	`#UR A HREF="vfpev:\\runreport(2,'"+allt(customerid_a)+"',this viewer)"`
Customer detail	Order number	`#UR A HREF="vfpev:\\runreport(3,"+allt(str(orderid_a))+",this viewer)"`

19 Converting reports to pictures

XFRX is able to export individual report pages as BMP, PNG, GIF and JPEG pictures, multiple pages can be exported to TIFF image format.

The picture generation process is divided into two steps:

1. the report is generate as XFF cursor
2. savePicture method of the XFRX#DRAW class is called to generate the picture (Please see XFRX#DRAW class reference on page 82 for details and a complete list of parameters)

The advantage of this approach is that once the XFF file is generated, it is very easy to get the page count, generate individual pictures in a cycle, etc.

If this sounds complicated, please have a look at the example, it is actually quite simple:

```
loSession= xfrx("XFRX#INIT")
*
* nothing is sent as the file name, so only a memory cursor is created
*
lnRetVal =loSession.SetParams(,,,,,,"XFF")
If lnRetVal = 0
        loSession.ProcessReport("invoices")
        local loXFF
        *
        * the finalize method returns a XFRX#DRAW object reference,
        * which will be used to save the pictures
        *
        loXFF = loSession.finalize()
        LOCAL lnI, lnJpegQuality
        lnJpegQuality = 80
        *
        * loXFF.pagecount contains the number of pages of the
        * report that
        * was just generated
        *
        * we are now going to save all pages one by one as
        * separate jpeg pictures
        *
        FOR lnI = 1 TO loXFF.pagecount
                loXFF.savePicture("page"+ALLTRIM(STR(lnI))+".jpg", ;
                               "jpg",lnI,lnI,24,lnJpegQuality)
        ENDFOR
        MESSAGEBOX("Pictures saved.")
Endif
```

20 Chaining listeners in VFP 9.0

The instances of the XFRXListener class can be chained together as well as with other report listeners so that multiple output formats could be generated by one REPORT command. The following example creates XFRXListener objects for HTML and PDF documents and chains them together with FFC's UpdateListener to display a progress bar during the report execution. This approach ensures that the HTML and PDF documents will contain exactly the same output (if there were two successive report runs, the data could have been modified).

```
LOCAL loObj, loObj2, loObj3
loObj = xfrx("XFRX#INIT")
loObj.targetType = "HTML"
loObj.targetFileName = "invoices.htm"

loObj2 = xfrx("XFRX#INIT")
loObj2.targetType = "PDF"
loObj2.targetFileName = "invoices.pdf"
loObj.successor = loObj2

loObj3 = NEWOBJECT("updatelistener", "_reportlistener.vcx")
loObj3.thermFormCaption = "Report in progress ..."
loObj2.successor = loObj3

REPORT FORM myReport object loObj
```

21 Running XFRX in a web-based environment

XFRX can be easily incorporated into applications compiled as COM DLLs to provide reporting features in the web based environment.

We have prepared a sample application of this technique. It is available online at http://www.eqeus.net/x.net. Should you be interested in the source code of this demo, please send us an email to support@eqeus.com and we will send it to you.

22 Reference

22.1 XFRX Output Target Types

XFRX currently supports the following target types:

Document type	**targetType value**
PDF document	PDF
Word document with absolute layout	DOC
Word document with flow layout	FDOC
RTF document with absolute layout	RTF
RTF document with flow layout	FRTF
OpenOffice Writer document with absolute layout	ODT
OpenOffice Writer document with flow layout	FODT
OpenOffice Calc spreadsheet with absolute layout	ODS
OpenOffice Calc spreadsheet with flow layout	FODS
Excel document	XLS
Plain text document	PLAIN
HTML document	HTML
HTML document with included graphics	MHT
XML document	XML
XFF document (internal binary document)	XFF
XFRX previewer	CNT

Remarks:

- "DOC" and "FDOC" output types require MS Word (ver. 2000 or higher) do be installed. "RTF" and "FRTF" do not require any third party product to be installed and can be opened in MS Word 97 and higher.
- "FRTF" document can also be opened with certain limitations in Open Office (http://www.openoffice.org) and PC Suite (http://www.software602.com/products/pcs). Open Office does not support any graphics in RTF, so lines, rectangles and pictures will not be displayed.
  PC Suite does not correctly display lines and rectangles, pictures, however, are displayed correctly.
- Remarks: "XLS" output type requires MS Excel (ver. 2000 or higher) to be installed
- Stored "XML" and "XFF" documents can be loaded back and previewed or transformed to other output types

22.2 Properties and methods common in XFRXListener and XFRXSession classes

Properties

PictureDPI	DPI (dots-per-inch) resolution to which all pictures are recomputed. You can use this property to reduce the output document size. **This property must be sent before calling the SetParams method.** Default value: 0 (do not recompute, keep pictures as they are)
DefaultPictureFormat	Picture format of activeX components converted to pictures. Allowed values: “bmp”, “jpg” Default value: “jpg”

Main methods

ErrorMessage()	Syntax: `lnRetVal = loXFRXobj.ErrorMessage(lnErrorNo)` This method returns a error message corresponding to the value returned from *SetParams()* method.
SetParams()	Call this method to set the document generation parameters. Syntax: `lnRetVal = loXFRXobj.SetParams(<tcOutputName>, <tcTempDirectory>, <tlNotOpenViewer>, <tcCodePage>, <tlSilent>, <tlNewSession>, <tcTarget>, <tcArchive>, <tlAdditive>, <tlDeleteFileAfter>, <tuAppend>)` Return values: 0 ... everything was ok, you can start processing reports -1 ... cannot load Word or Excel application -2 ... the Word or Excel application version must be 2000 or higher -3 ... cannot create or open the output file -4 ... unknown target -5 ... hndlib.dll cannot be loaded (it is missing or an old version is used) -6 ... xfrxlib.fll cannot be loaded (it is missing or invalid). Please see Frequently asked questions if you are getting this error. -7 ... zlib.dll cannot be loaded -8 ... An old xfrxlib.fll is used

	-10 ... The existing document is either corrupted or in an unsupported format Note: You can retrieve an English message for the code returned with *ErrirMessage()* method. Parameters: *tcOutputName* the name of the document to create. *tcTempDirectory* directory where temporary files will be created. If blank, the temporary files will be created in the current directory [optional] *tlNotOpenViewer* if set to .T., documents won't be opened after the generation [optional] *tcCodePage* is a codepage of the generated document [optional]. If you don't specify this parameter, cpcurrent() is used. *tlSilent* if set to .T., no messages will be printed [optional]. This option is useful if your application is not in English and/or you would like to handle the Processing... message and error messages in your code. *tlNewSession* by default, word document will be open in the current Word session, if exists. If this paramenter is set to .T., the document will always be open in a new Word session. This option is not used for PDF targets. [optional] *tcTarget* The output type to be generated. One of the values listed at ***XFRX Output Target Types*** (above)**.** *tcArchive* The name of the zip archive to be created. If not empty, the generated file will be added to the archive after generation (optional) *tlAdditive* If set to .T. and the archive already exists, the file will be added. (optional) Please note: you can add several files to the archive, but existing files with the same name will not be overwritten - the new files will always be added. *tlDeleteFileAfter* If set to .T., the generated file is be deleted after it is copied to the archive. (optional)

tuAppend
Specifies whether the generated document will be appended to an existing file. This parameter can be either logical, numeric or a string, with the following meaning:

Type	Value	Meaning
Logical	.F.	An existing document will be overwritten
	.T.	The generated report will be appended at the end of the existing document
Numeric	0	An existing document will be overwritten
	<page No.>	The generated report will be inserted to the existing document at the given page number.
String	“R<page No.>”	The page will be replaced with the generated report.
	“R<from>:<to>”	The page range will be replaced with the generated report.

Examples:

Parameter	Meaning
`.T.`	The generated report will be appended to the end of the existing document.
`1`	The generated report will be inserted at the beginning of the existing document.
`5`	The generated report will be inserted to the existing document, between pages 4 and 5.
`“R5”`	The generated report will be inserted between pages 4 and 6, replacing page 5.
`“R4:8”`	The generated report will be inserting between pages 3 and 9, replacing pages 4,5,6,7 and 8.
`.F.` or `0`	The existing document will be overwritten.

This parameter applies to PDF documents only.

(optional, default value = .F.)

SetOtherParams(...)

This method is used to add various output type specific parameters. The first parameter is a string value representing the parameter to be set, the second parameter is the values to be set.

Example:

To set the next sheet name in the XLS document to “Customer 1”, call:

```
.SetOtherParams("NEXT_SHEET_NAME", "Customer 1")
```

HTML specific parameters

Name	Type	Note

PRINT_BOOKMARKS	Logical, Numeric	You can send two parameters with this option. The first parameter is logical and controls if the bookmark should be displayed (.T.) or not (.F. = default value). The second parameter specifies what kind of bookmarks should be displayed: • Sending 0 will show both page number bookmarks and the ones defined in the report (default) • Sending 1 will show only the bookmarks defined in the report • Sending 2 will show only the page number bookmarks

Plain text specific parameters

Name	Type	Note
PLAIN_CPI_HORIZONTAL	Numeric	Horizontal characters per inch value
PLAIN_CPI_VERTICAL	Numeric	Vertical characters per inch value
PLAIN_SHOW_LINES	Logical	Default value = .T. This parameter allows for suppressing horizontal lines from the generated file

Please find more information about plain text parameter in Converting reports to plain text chapter on page 37.

XLS specific parameters

Name	Type	Note
DISPLAY_GRID_LINES	Logical	Default value = .T. If this parameter is set to .F., the grid lines in the XLS output will not be visible.
LEAVE_FULL_FIELD_CONTENT	Logical	Default value = .F. By default, the content

			of non-stretchable fields is cut according to the size of the field. This option enables to leave the full content of the field in the Excel cell.
	NEXT_SHEET_NAME	String	The name of the next sheet to be inserted by next report run.
	NEXT_SHEET_NAME_EXPR	String	This expression, if defined, is evaluated at the bottom of each sheet and the result is used as the sheet name. This parameter takes precedence over the NEXT_SHEET_NAME parameter.
	SHEET_PER_PAGE	Logical	Default value = .F. By default, the XLS document is generated in “Plain” mode – one long sheet is generated, with the page header at the top and the footer at the bottom. If this parameter is set to .T., then each report page is generated as a new sheet.
	SHEET_PER_NP_GROUP	Logical	Default value = .F. This option combines the plain mode and the sheet-per-page mode. If this parameter is set to .T. (and SHEET_PER_PAGE is .F.), then a new sheet is generated for each report group with “Start each group on a new page” flag set to .T.
	HORIZONTAL_ADJUSTMENT VERTICAL_ADJUSTMENT	Numeric	The maximal coordinates difference considered as zero.

	Please see XLS cells adjustment chapter on page 33 for more information.
Finalize()	This method finishes the document generation and displays the generated document (unless the document preview was explicitly suppressed). The *Finalize()* method does not have to be called in VFP 9.0, if the last REPORT FORM command doesn't contain the NOPAGEEJECT clause.

Methods for registering page bound XFF scripts

RegisterScript()	Syntax: `RegisterScript(toScriptObject, tcScriptMethod, tnZOrder, tcPageScope, tnNumberingType)` Parameters: *toScriptObject* The PDFLScripts object reference *tcMethodName* The name of the method containing the script *tnZIndex* 0 ... print below (before) the report page 1 ... print above (after) the report page *tcScope* Defines which pages to invoke the scripts on. Can contain "ALL", "ODD", "EVEN" and numbers delimited with commas. Hyphens can be used to define the from-to scope. Examples: "ODD", "1,5-19", "ODD,4,8" *tnPageNumberingType* Defines how XFRX determines the page number: 0 ... absolute - the page number in the document, 1 ... relative - the page number in the current report, 2 ... the value of _PAGENO
Unregister AllScripts ()	Syntax: `UnregisterAllScripts()`

	If multiple reports are merged together and you would like to apply scripts on a certain report but not on subsequent ones, call this method to remove all scripts registrations. This method does not need to be called at the end of the reports processing.

Page size adjustment methods

setPaperSize()	Use this method to define a user-define page size. Syntax: `setPaperSize(nUDPaperWidth, nUDPaperHeight)` Please see *User-defined page size* paragraph on page 17 (above) for more information.
ShrinkHeight()	Use this method to shorten the page size. Please see *HTML page size adjustment* on page 30 for more information.

Methods to define document encryption

setPasswords()	Sets passwords for PDF and Word document protection.
	
	PDF syntax:
	`setPasswords(tcOwnerPassword, tcUserPassword)`
	
	Parameters:
	
	The user password (tcUserPassword) can be empty. If the owner password (tcOwnerPassword) is empty, a random string will be generated as the password.
	The owner can do anything with the document. The user permissions can be set using the setPermissions() methods.
	
	Word syntax:
	`setPasswords(tcReadPassword, tcWritePassword, tlRequirePassword)`
	
	You can omit either tcReadPassword or tcWritePassword. tlRequirePassword is optional (default value is .F.). If set to .T., Word will ask for the password even when the document is being opened first time after the generation (if DoNotOpenViewer property is set to .F.).

setPermissions()	Sets the user permissions in the PDF document. Syntax: setPermissions(tlPrintDocument, tlModifyDocument, tlCopyTextAndGraphics, tlAddOrModifyAnnotations)

Methods to set document properties:

setAuthor()	Sets the document author property. Syntax: set(tcvalue) Parameters: tcValue (char) is a value that will be used in the document properties. Remarks: This value will be used for PDF, Word and Excel documents.
setCategory()	Sets the document "Category" property. Syntax: set(tcvalue) Parameters: tcValue (char) is a value that will be used in the document properties. Remarks: This value will be used for Word and Excel documents only.
setComments()	Sets the document "Comments" property. Syntax: set(tcvalue) Parameters: tcValue (char) is a value that will be used in the document properties. Remarks: This value will be used for Word and Excel documents only.
setCompany()	Sets the document "Company" property. Syntax: set(tcvalue) Parameters: tcValue (char) is a value that will be used in the document properties. Remarks: This value will be used for Word and Excel documents only.
setCreator()	Sets the document "creator" property.

	Syntax: set(tcvalue) Parameters: tcValue (char) is a value that will be used in the document properties. Remarks: This value will be used for PDF documents only.
setKeywords()	Sets the document “keywords” property. Syntax: set(tcvalue) Parameters: tcValue (char) is a value that will be used in the document properties. Remarks: This value will be used for PDF, Word and Excel documents.
setManager()	Sets the document “Manager” property. Syntax: set(tcvalue) Parameters: tcValue (char) is a value that will be used in the document properties. Remarks: This value will be used for Word and Excel documents only.
setProducer()	Sets the document “producer” property. Syntax: set(tcvalue) Parameters: tcValue (char) is a value that will be used in the document properties. Remarks: This value will be used for PDF documents only.
setSubject()	Sets the document “Subject” property Syntax: set(tcvalue) Parameters: tcValue (char) is a value that will be used in the document properties. Remarks: This value will be used for PDF, Word and Excel documents.
setTitle()	Sets the document title property. Syntax: set(tcvalue)

	Parameters: tcValue (char) is a value that will be used in the document properties. Remarks: This value will be used for PDF, Word and Excel documents.

22.3 XFRXListener class

Properties:

AppendToFile	Specifies whether the generated document will be appended to an existing file. This parameter can be either logical, numeric or a string, with the following meaning: Type / Value / Meaning Logical / .F. / An existing document will be overwritten .T. / The generated report will be appended at the end of the existing document Numeric / 0 / An existing document will be overwritten <page No.> / The generated report will be inserted to the existing document at the given page number. String / "R<page No.>" / The page will be replaced with the generated report. "R<from>:<to>" / The page range will be replaced with the generated report. Examples: Parameter / Meaning `.T.` / The generated report will be appended to the end of the existing document. `1` / The generated report will be inserted at the beginning of the existing document. `5` / The generated report will be inserted to the existing document, between pages 4 and 5. `"R5"` / The generated report will be inserted between pages 4 and 6, replacing page 5. `"R4:8"` / The generated report will be inserting between pages 3 and 9, replacing pages 4,5,6,7 and 8. `.F.` or `0` / The existing document will be overwritten. This property applies to PDF documents only.
CodePage	The code page of the output document.
DoNotOpenViewer	If set to .T., documents won't be opened after the generation. Default value = .F.
NewViewerSession	By default, Word and HTML documents will be open in the current instance of the application, if exists. If this parameter is set to .T., the

	document will always be open in a new Word / Web browser instance. Default value = .F.
QuietMode	This is a property of the ReportListener class. If set to .T., no message will be printed during the report generation process.
targetFileName	The name of the file to be generated.
targetType	The output type to be generated. One of the values listed at ***XFRX Output Target Types*** (above)**.**

Methods:

Note: The common methods are described at *Properties and methods common in XFRXListener and XFRXSession classes* (above).

zipDocument()	Specifies that the generated document should be added to a zip archive. Syntax: `zipDocument(ArchiveName, AddToArchive, DeleteSourceDocument)` Return values: none Parameters: *ArchiveName (char)* is the file name of the zip archive *AddToArchive (logical)* specifies, if the file should be added to existing archive (.T.) or if a new archive should be created (.F.) *DeleteSourceDocument (logical)* if set to .T., the source document will be deleted after adding to the archive. **Note:** You can send these three parameters to XFRX either via this method or as the last three parameters of the SetParams method. This method is available for your convenience of you prefer to fill in XFRXListener properties and call SetParams without parameters.

22.4 XFRXSession class

Note: The common methods are described at *Properties and methods common in XFRXListener and XFRXSession classes* (above).

ProcessReport()	Processes the report. Syntax: `ProcessReport(<tcReportName>, <tcForClause>, <tlSummary>, <tcScopeClause>, <tcWhileClause>, <tlPlain>)` Parameters: *tcReportName* the name of the report file *tcForClause* the FOR clause [optional] *tlSummary* The summary clause switch. If set to .T., the report is processed as if SUMMARY clause was used in the REPORT FORM command [optional] *tcScopeClause* The string corresponding to the report's scope clause (such as NEXT 4, REST, ALL, RECORD nRecno). Default value is "ALL" [optional] *tcWhileClause* The while clause [optional] *tlPlain* Plain format switch. Send .T. as this parameter to switch the HTML output to PLAIN format. The plain format doesn't generate any page breaks to the document - one long page will be generated. The default value is .F.

22.5 XFRX#DRAW class

Properties:

PageCount	The total number of pages in the XFF document (*numeric)*

Methods:

AddPage()	Adds a new page at the end of the document. Syntax:

	`XFRXDraw::AddPage([tcPageSize])`
	`Or`
	`XFRXDraw::AddPage(tnPageWidth, tnPageHeight)`
	
	Parameters:
	
	tcPageSize
	The page size of the new page. Available values: “A4”, “letter”
	
	tnPageWidth, tnPageHeight
	The width and height of the new page. The unit can be defined by *SetUnit*() method. The default unit it point (1/72 in).
	
	If no page size is specified, the current page size is used.

CreateDocument()	Creates a new XFF file. Syntax: `XFRXDraw::CreateDocument([tcFileName])` Return values .T. … the file was successfully created .F. … file couldn’t have been created () An existing file is overwritten. Parameters: *tcFileName* The file name if the document to be created. If this parameter is empty, the XFF file is created in memory.

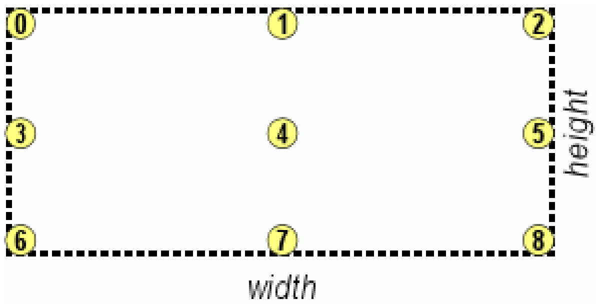
DrawLine()	Draws a line on the current page.
	
	Syntax:
	`XFRXDraw::DrawLine(tnXPos, tnYPos, tnXToPos, tnYToPos`
	`[, tnLineWidth[, tnPenPattern]])`
	
	Parameters:
	
	tnXPos, tnYPos
	The X,Y coordinates of the start point of the line.
	
	tnXToPos, tnYToPos
	The X,Y coordinates of the end point of the line.
	
	tnLineWidth
	The line width in points. 0 represents hairline. Optional, default value = 1.
	
	tnPenPattern
	The border line pattern:

	0 ... no border 1 ... dotted 2 ... dashed 3 ... dash-dot 4 ... dash-dot-dot 8 ... solid line Optional, default value = 8.
DrawPicture()	Draws a picture on the current page into a given bounding rectangle. Syntax: `XFRXDraw::DrawPicture(tnXPos, tnYPos, tnWidth, tnHeight, tcFilename[, tnAdjType])` Parameters: *tnXPos, tnYPos* The X,Y coordinates of the upper left hand corner of the bounding rectangle. *tnWidth* The width of the rectangle *tnHeight* The height of the rectangle *tcFileName* The picture file name *tnAdjType* 0 ... clip picture 1 ... stretch picture, retain shape 2 ... stretch picture, fill frame
DrawRectangle()	Draws a rectangle on the current page. Syntax: `XFRXDraw::DrawRectangle(tnXPos, tnYPos, tnWidth, tnHeight [, tnLineWidth[, tnFill[, tnPenPattern, tnRoundFactor]]])` Parameters: *tnXPos, tnYPos* The X,Y coordinates of the upper left hand corner. *tnWidth* The width of the rectangle *tnHeight*

	The height of the rectangle *tnLineWidth* The line width in points. 0 represents hairline. Optional, default value = 1. *tnFill* 0 … not filled, 1 … filled. Optional, default value = 0 *tnPenPattern* The border line pattern: 0 … no border 1 … dotted 2 … dashed 3 … dash-dot 4 … dash-dot-dot 8 … solid line Optional, default value = 8. *tnRoundFactor* The radius of the rectangle edges curvature. 0 makes the rectangle right-angled (no rounding), 99 creates an ellipse. Optional, default value = 0. Note: the border and fill color needs to be defined via *SetColor()* method before calling *DrawRectangle()* method.

DrawText	Draws a text, using a font defined by *SetFont* method.
	
	Syntax:
	`XFRXDraw::DrawText(tnLeft, tnTop, tcText[, tnRotate[, tcLinkName[, tcLinkRef[, tcBookmark[, tcTooltip]]]]])`
	`Or`
	`XFRXDraw::DrawText(tcText[, tnRotate[, tcLinkName[, tcLinkRef[, tcBookmark[, tcTooltip]]]]])`
	
	Parameters:
	tnLeft, tnTop
	The coordinates where the text will be drawn. If the second syntax is used (without the coordinates), the text will be drawn at the current position: which is either a position defined by *SetPos()* method or a the end of an output if the previous *DrawText()* method call.
	
	tcText
	The text to be drawn
	
	tnRotate
	The text rotation angle in degrees. The label will be rotated counterclockwise. Optional, default value = 0.
	
	tcLinkName
	If this parameter is filled, the text will be a named target of a hyperlink.

	(see *tcLinkRef* parameter). *tcLinkRef* If this parameter is filled, the text will be a hyperlink. It will navigate to a target in the local document with *tcLinkRef* name. To navigate to a http URL, include "http://" at the beginning of the parameter. *tcBookmark* I not empty, the parameter contains a name of a bookmark that will be created and will navigate to this text. *tcTooltip* The tooltip displayed when hovering mouse over this text.
DrawTextBox()	Word wraps the give text and print it into a given bounding rectangle. If the height of the rectangle is zero, then the rectangle stretches to accommodate to full text and the height of the rectangle is returned. Syntax: `XFRXDraw::DrawTextBox(tnLeft, tnTop, tnWidth, tnHeight, tcText[, tnAlign[, tnVAlign[, tnRotate[,tnRotationPoint[, tcLinkName[, tcLinkRef[, tcBookmark[, tcTooltip]]]]]]]])` Return value: The height of the bounding rectangle is returned. Parameters: *tnLeft, tnTop* The X and Y coordinates of the upper left hand corner of the bounding rectangle. *tnWidth, tnHeight* The width and the height of the bounding rectangle. If *tnHeight* is 0, the rectangle stretches to accommodate to full text and the height of the rectangle is returned. *tnAlign* The horizontal alignment of the text inside of the rectangle. The allowed values are: 0 ... "left" 1 ..."center" 2 ... "right". Optional, the default value is 0 (left). *tnVAlign* The vertical alignment of the text inside of the rectangle. The allowed values are: 0 ... "top" 1 ..."center"

	2 ... "bottom". Optional, the default value is 0 (top). *tnRotate* The text rotation angle in degrees. The label will be rotated counterclockwise. Optional, default value = 0. *tnRotationPoint* The point within the rectangle around which the text is rotated. The allowed values are 0..8, optional, default value = 0 (left, top). Please see the picture on the right for the point positions.  0 1 2 3 4 5 6 7 8 height width *tcLinkName* If this parameter is filled, the text will be a named target of a hyperlink. (see *tcLinkRef* parameter). *tcLinkRef* If this parameter is filled, the text will be a hyperlink. It will navigate to a target with *tcLinkRef* name. *tcBookmark* I not empty, the parameter contains a name of a bookmark that will be created and will navigate to this text. *tcTooltip* The tooltip displayed when hovering mouse over this text.
GetBounding Rectangle()	Returns the bounding rectangle coordinates in the rectangle-bound script. Syntax: `XFRXDraw::GetBoundingRectangle()` Return value: *NULL* - the bounding rectangle is not available *Object* – an object with the following properties: nLeft, nTop – the coordinates of the upper left hand corner of the rectangle nWidth, nHeight – the width and height of the rectangle The unit of the coordinates can be set by *SetUnit()* method. The default unit is Point (1/72 in).
GetPageWidth()	

GetPageHeight()	
GetXPos()	Returns the horizontal coordinate of the current position. Syntax: `XFRXDraw::GetXPos()`
GetYPos	Returns the vertical coordinate of the current position. Syntax: `XFRXDraw::GetYPos()`
OpenDocument()	Opens an existing XFF file. Syntax: `XFRXDraw::OpenDocument(tcFileName)` Return values .T. … the file was successfully opened .F. … file couldn't have been opened (file doesn't exist or is locked)
PrintDocument()	Prints the XFF file to the specified printer. Syntax: `XFRXDraw::PrintDocument(tcPrinterName, tcJobname, tnFrom [, tnTo[, tcDEVMODE]])` Return values -100 … cannot initialize printer 0 … no errors Parameters: *tcPrinterName* Then name of the printer to which the document will be sent *tcJobName* The printer job name *tnFrom* If numeric, the first page to be printed. *tnTo* The last page to be printed.

	Alternatively, *tnFrom* can contain a string value, containing the pages numbers separated by commas and ranges separated by dashes, e.g.: "1,2,5-6,8,20-30"
SavePicture()	SavePicture method saves the report page(s) as a picture. Syntax: `XFRXDraw::SavePicture(tcFilename, tcType, tnFrom [, tnTo [, tnBpp [, tnJPEGQuality [, tnThumbnailWidth [, tnThumbnailHeight] ] ] ] ])` Return values 0 ... no errors was encountered -1 ... unknown image format -2 ... page out of range Parameters: *tcFilename* The name of the file to be generated. *tcType* The format of the picture to be saved. Currently supported options are: BMP, GIF, JPEG, PNG, TIFF. *tnFrom* The page number to be saved. If TIFF format is being saved, the parameter specifies the first page to be saved. *tnTo* If TIFF format is being saved, the parameter specifies the last page to be saved. Ignored otherwise. *tnBpp* Bits per pixel. Currently supported values are 2, 16 or 24. *tnJPEGQuality* Specifies the JPEG compression quality. The range is from 1 - the lowest quality to 100 - the best quality. The default value is 0 - default quality. Ignored for other output types. *tnThumbnailWidth* The width of the output picture in pixels. *tnThumbnailHeight* The height of the output picture in pixels. If both tnThumbnailWidth and tnThumbnailHeight are omitted, the

	original page size is used. If only one of the values is sent, the other one is calculated accordingly.
SetColor()	Sets the foreground and background drawing colors. Syntax: `XFRXDraw::SetColor(tfr, tfg, tfb[, tbr, tbg, tbb])` Parameters: *tfr, tfg, tfb* Red, green and blue components of the foreground color *tbr, tbg, tbb* Red, green and blue components of the background color (optional)
SetFont()	Sets font name and attributes for subsequent DrawText() or DrawTextBox() calls. Syntax: `XFRXDraw::SetFont(tcFontName, tnSize[, tlBold, tlItalic, tlUnderline])` Parameters: *tcFontName* The font name *tnSize* The font size in points *tlBold, tlItalics, tlUnderline* The font attributes. These parameters are optional, the default value is .F.
SetPos()	Sets the position where the text will be drawn by the subsequent *DrawText()* method call. Syntax: `XFRXDraw::SetPos(tnXPos, tnYPos)` Parameters: *tnXPos, tnYPos* The X and Y (horizontal and vertical) coordinates of the position. The unit of the coordinates is defined by *SetUnit()* method, the default unit is Point (pt).

SetUnit()	Sets the drawing units. Syntax: `XFRXDraw::SetUnit(tcUnit)` Parameters: *tcUnit* The unit code. The allowed values are: “in” – inches, “cm” – centimeters, “pt” – points and “px” – pixels. Note: The default unit is point.

22.6 XFCont class

XFRX Preview container class. This section describes properties that you can modify in your custom implementation of the previewer.

iBook	Controls if the bookmark should be displayed in the previewer. The allowed values are: • -1 = Disable bookmarks • 0 = Enable bookmarks but hide them (users need to click on the bookmark button to see them) • 1 = Enable bookmarks, always showing them • 2 = Enable bookmarks, but show or hide them automatically based on whether or not there are bookmarks defined in the report to preview (default)
iTool	Controls the visibility of the toolbar at the top. The allowed values are: • -1 = Disable toolbar • 0 = The toolbar will be enabled but hidden. It can be invoked via right click shortcut menu • 1 = The toolbar will be enabled and visible. (default)
ShowStatus	Controls the visibility of the toolbar at the status bar of the previewer container. The allowed values are: • .T. = show the toolbar (default) • .F. = hide the toolbar

23 XFRX License

Disclaimer: The software and information provided by Eqeus.com are provided "as is", without any warranty whatsoever. You use the software on your own risk. Eqeus.com disclaims all warranties and conditions, either express or implied, including, without limitation, warranties of merchantability and fitness for a particular purpose. In no event will Eqeus.com be liable for any damages, including, without limitation, direct, indirect, special, incidental or consequential damages of any kind, even if Eqeus.com has been advised of the possibility of such damages. You agree not to obtain or use the software in any state or country that does not allow the exclusion or limitation of liability for incidental or consequential damages.

The software license allows for distributing XFRX and/or PDF Library with software applications to end users to provide an option to transfer reports into the target output formats XFRX supports.

The license purchase entitles you to receive all new versions of XFRX and/or PDF Library for free for one year from the purchase date.

The permission is subject to the following restrictions:

1. XFRX and PDF Library source codes must NOT be distributed.
2. XFRX and PDF Library or any part of it must NOT be distributed as a software tool or part of a software tool provided to application developers to be incorporated into their products, included, but not limited, software frameworks and report generation tools.

24 Frequently asked questions

1. Installation & setup

Q1.1: ***I am evaluating XFRX. It works fine in development, but when I try to compile the demo in my project, the VFP quits or crashes. Is that because it is a demo version?***

A1.1: This is an issue in the demo version of XFRX - it cannot be included into VFP projects, it makes VFP crash. The workaround is to invoke XFRX via macro substitution:

```
loSession = EVALUATE[XFRX('XFRX#INIT')]
```

This way XFRX.APP does not get into the project and you will be able to compile without problems.
The commercial version is not affected by this problem.

Q1.2: ***XFRX works fine on some computers but on others I am getting this error:" Library xfrxlib.fll is invalid" or "cannot load xfrxlib.fll".***

A1.2: The xfrxlib.fll library requires two DLLs from Microsoft to be installed (gdiplus.dll and msvcr71.dll) - most probably one of these is missing on the machine. In order to save reports as pictures, XFRX requires two additional libraries to be installed:

- msvcr71.dll - Microsoft Visual C++ 7.1 runtime library. If it is not installed on your pc, you can download it from http://www.eqeus.com/files/msvcr71.zip
- gdiplus.dll - This DLL is included with Windows XP and if it is not available on your pc, you can download it from http://www.microsoft.com/msdownload/platformsdk/sdkupdate/psdkredist.htm

If you are not planning to export reports as pictures, you can download a special version of XFRXLIB.FLL (http://www.eqeus.com/files/xfrxlib-nogdiplus.zip), which doesn't require the two dlls mentioned above.

Q1.3: Do I have to distribute my reports (FRT and FRX) files with my application or is XFRX able to access the reports built in into the exe?

A1.3: XFRX is able to access your built-in reports. To do this, add XFRX.FXP into your project and compile it into your exe as well.

Q1.4: XFRX doesn't seem to be creating a private data session for my reports even though their private data session flag is set on. We are using VFP 6.

A1.4: Please make sure you have installed the latest VFP 6 service pack. The session class is not available until SP3.

2. Problems with XFRX output

Q2.1: The paper size is wrong, landscape reports are generated as portrait.

A2.1: Make sure you don't delete the information in the EXPR field of the first record of the page. XFRX doesn't use printer settings stored in the FRX file and uses PAPERSIZE and ORIENTATION values from the EXPR field of the first record. If they are not there, the default value is used, which can result in wrong orientation or wrong paper size.

Q2.2: In my reports, I am using graphs (word documents, bar codes) that are embedded in a general field. When I tried to print of these reports with XFRX it does not print the embedded general field.

A2.2: XFRX for VFP 8.0 is able to print BMP and JPEG pictures stored in general fields, but ActiveX components are not supported.

This restriction does not apply to XFRX for VFP 9.0, which is able to process any content of general fields, including ActiveX components.

Q2.3: I'm trying to include some GIF images on the reports but the images don't appear in the PDF and RTF output.

A2.3: GIF images are supported in other output types, but PDF and RTF support only BMP and JPEG image formats.

Q2.4: I am finding that the Excel output feature is creating a lot of extra boxes and not making columns like I thought.

A2.4: This is how the XLS output works - it creates a column wherever an object starts or finishes. You may achieve better results if you try aligning the objects. Please have a look at the *Excel specific features* chapter in the Developer's guide for some ideas how you can make the output look better.

Q2.5: My report has the variable which is replaced with ".F." in the output document.

A2.5: If the variable is declared as LOCAL, XFRX for VFP 8.0 won't be able to see it (the variable is out of scope in XFRX), changing it to PRIVATE or PUBLIC fixes this problem.

3. VFP 9 related questions

Q3.1: We will eventually go to Version 9 of Visual Foxpro. But right now we are using VFP8. Should I order your product for Version 8 or Version 9? If I ordered V8 what will it cost to upgrade to V9?

A3.1: We do not differentiate individual VFP versions - once you purchase XFRX, you can use it in VFP 5, 6, 7, 8 and 9 without any further costs.

Q3.2: I am currently using XFRX in VFP 8.0 and would like to upgrade to VFP 9.0. Do you need to modify any code to start using XFRX for VFP 9.0?

A3.2: Yes, you need to modify your code a bit if you want to start using the new features in VFP 9.0. If you do not modify how XFRX is called in your application, it will still work fine after you upgrade to VFP 9.0 and you don't need to change anything if you are satisfied with the results. If you, however, want to use the VFP 9.0 native report engine in cooperation with XFRX, you need to modify your code as described in the Developer's guide, *Running XFRX* chapter.

Q3.3: What are the main differences between XFRX for VFP 8.0 and XFRX for VFP 9.0?

A3.3: Please see the Developer's guide,

Differences between XFRX for VFP 8.0 and XFRX for VFP 9.0 chapter.