FastReport for [x]Harbour (FRH), FastReport for Alaska Xbase++ (FRAX)

User manual

Edition 2.00

Table of Contents

Introduction

"FastReport for [x]Harbour" and "FastReport for Alaska Xbase++" are report generators for giving [x]Harbour and Xbase++ application the ability to generate reports quickly and efficiently. These generators provide all the tools that developers need to develop reports. "FastReport for [x]Harbour" and "FastReport for Alaska Xbase++" are based on excellent "FastReport 4" (with authority of the OEM-agreement with Fast Report Inc.) and fully adapted for [x]Harbour and Alaska Xbase++ correspondingly.

Further we'll name in abbreviated form "FastReport for [x]Harbour" as **FRH**, and "FastReport for Alaska Xbase ++" as **FRAX**. In descriptions of the methods and procedures etc, word "FastReport" will be used as a common term for both products - FRAX and FRH.

In spite of the fact that FRH and FRAX are compiled on Delphi, the [x]Harbour or Xbase-developer does not have necessity to have Delphi, to be able to program on Delphi, etc. All work is done in style habitual for the developer and, practically, does not differ from work with the library written on [x]Harbour or Xbase++.

"FastReport" exists in various variants and is used by developers of various programming languages, such as Microsoft Visual Studio, Microsoft Access, Microsoft FoxPro, SyBase PowerBuilder etc. It is possible to define the basic advantage of "FastReport" as a combination of breadth of opportunities, great capacities and simplicity of use. FRH and FRAX are very simple in use too, it's intuitively clear, does not require any installation, preliminary adjustment, etc. Moreover, the class frReportManager completely hides any work with external dll. The developer must not reflect on compatibility of data types, parameters, declarations, etc. Any Harbour or Xbase variables are accessible for developer in any reports; developer can call any procedures, functions, methods from reports, without restrictions. Also it is necessary to note, that FRH and FRAX do not open any DB, does not use any temporary files, pipes, etc. they work with native [x]Harbour or Xbase++ data only. That is, FRAX is an organic part of Xbase ++ applications and FRH organic part of [x]Harbour++ applications.

Let's list some characteristics of FastReport:

- Band-oriented report generator.
- No additional DLLs needed. All in one dll - FRSyst.dll
- Built-in powerful and easy-to-use designer (available in run-time).
- Full WYSIWYG
- Set of most useful components: Text, Line, Diagonal line, Picture, Shape, OLE, RichText, Chart, Barcode etc.
- Ability to export your reports to other formats (such as TXT, RTF, HTML, PDF, XLS, XML, JPG, BMP, TIFF etc).
- Dot matrix reports.
- Built-in language allows you to write code without programming.
- Various fill types, shadow.
- Script editor with syntax highlight.
- One script in the report.
- High performance.
- Strict type checking.
- Multi-language architecture allows using many script languages (Pascal, C++, Basic, Java).
- Access to any object inside your application. Standard libraries to access to base classes, controls and forms. Easy expandable library architecture.
- Debugger.

- Report preview with "Search text" and "Edit" functions.
- Text rotation 0..360 degrees.
- Paragraph gap.
- Memo object supports simple html-tags (font color, b, i, u, sub, sup).
- Styles.
- Text flow.
- URLs, Anchors
- Zooming in designer.
- In-place editing.
- Rulers, guides.
- Grid: mm, inches, pixels.
- Wizard for base type reports.
- Copy objects to Windows clipboard.
- Full Undo/Redo.
- Reports can contain dialogue forms that you can use to ask parameters before prepare a report. You can have as many dialogs as you need. FastReport uses the same designer for dialogs and have set of standard dialog controls like Button, Edit, CheckBox and other.
- Storing reports in XML format, compressed XML format (compatible with GZip) are also available.
- You can use FastReport in international applications.
- Full Bi-directional text output.
- And many other things and opportunities.

For more information look «FastReport User Manual» and other documentation.

The complete set

Standart Edition

"Fast Report for [x]Harbour" consists of three main files:

1. FRSystH.dll - the basic library that will be used by your applications. Just place this library in the directory that convenient for you. It can be a folder with your exe-file, etc.
2. FASTREPH.PRG - is the source file containing all classes and methods, procedures and functions that necessary for work with FRH. Add this file to your project.
3. FASTREPH.CH - is “include” file containing some constants that necessary for work with FRH. Add this file to your project.

"Fast Report for Alaska Xbase ++ " consists of three main files:

4. FRSyst.dll - the basic library that will be used by your applications. Just place this library in the directory that convenient for you. It can be a folder with Alaska run-time dlls, a folder with your exe-file, etc.
5. FASTREP.PRG - is the source file containing all classes and methods, procedures and functions that necessary for work with FRAX. Add this file to your project.

6. FASTREP.CH - is “include” file containing some constants that necessary for work with FRAX. Add this file to your project.

   So, to begin work it is necessary to add two files to the project. That’s all.

   The documentation of products includes:

   - FastReport for Alaska Xbase.pdf - user manual of FRH and FRAX in pdf- format (current manual).

Other documentation of FastReport in various variants it is always possible to download from http://www.fast-report.com/en/documentation/.

In addition it is possible to download language resources files. Now FastReport supports 32 languages. For monolanguage programs these files are not necessary. Default language resources are compiled in resources of FRSyst.dll (FRSystH.dll). Default language gets out at purchase, and also can be changed at any time.

About used terminology.

As a whole, in the given manual it is not used any specific terminology which would demand special explanatory. However in two cases some explanatory, probably, will be not superfluous.

The first are FastReport concepts "event" and "event handler". In terminology [x]Harbour and Xbase++ it usually name: "callback slot" and "code block assigned to slot". That is, phrases "define handler of event" and "assign code block to slot" are substantially equivalent.

And the second is a concept "property". Property is "active" instance variable. Actually, the concept "property" is intuitively clear and is used, for example, in "Xbase ++ FormDesigner" and some IDE for [x]Harbour. But the main reason is using of this concept in designer and other FastReport documentation.

Load and Unload

The central class, which completely operates FastReport, is frReportManager. Creation of an instance of this class is equivalently loading and initialization FastReport. For example:

```
FrPrn := frReportManager():new(“C:\MYPATH\FRSyst.dll”)
```

That is, method Init of a class frReportManager has one parameter:

Syntax

```
:Init([<cOptionalPath>])
```

Parameters

< cOptionalPath > - optionally, character string containing the name of the FastReport dll-file. It can include drive and path specifications.

To destroy all FastReport objects and unload FRsyst.dll method DestroyFR is used:

Syntax

```
:DestroyFR()
```

Method has no Parameters.

You can load and initialize FastReport at the beginning of work of application and unload it at quit, or do it at each report-operation. There is no great difference between these approaches.

Basic methods of frReportManager class

`:LoadFromFile(<cFileName>)`

Parameters:

<cFileName> - character string containing the name of report file. It can include drive and path specifications.

Loads a report from a file with given name. In case that file is loaded successfully returns .t., otherwise returns .f. (for example, if file is not exists).

`:ShowReport([<nNotClear>])`

Parameters:

<nNotClear> - optionally, if this parameter is equal to FR_NOTCLEARLASTREPORT (see FastRep[H].ch) then a report will be added to the previously constructed one, otherwise the previously constructed report is cleared (by default).

Starts a report and displays a result in the preview window.

`:DesignReport()`

Method has no Parameters.

Calls the report designer.

`:Clear()`

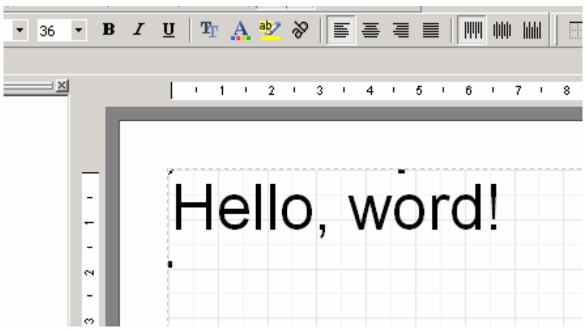
Method has no Parameters.

Clear (close) the current report.

Well, let's create the first report using these methods. On any event, for example, on some button press, call method DesignReport(). For example (for Xbase++):

```
oButton:activate := {|| FrPrn:DesignReport() }
```

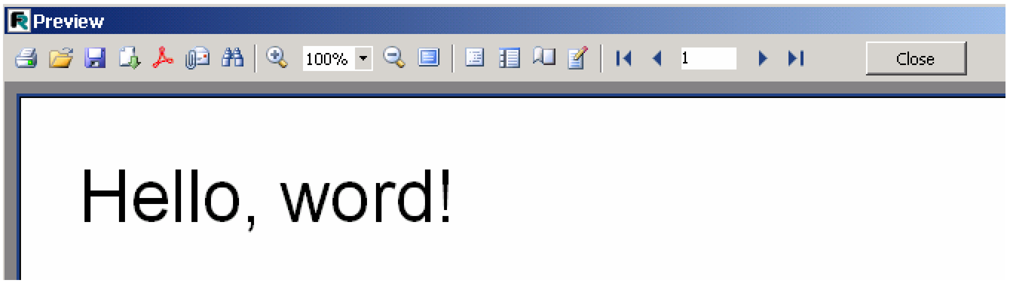
As we did not load any report the designer appears with empty report. Place text object in report area and type in memo editor «Hello, world!» For more information about work in designer see «FastReport 3 User Manual» or designer context help (fruser.chm).



Save report in file, for example "c:\my reports\1.fr3", and close designer. On some event, write code:

```
FrPrn:LoadFromFile("c:\my reports\1.fr3")
FrPrn:ShowReport()
```

When this event occurs you'll see result in the preview window:



Hello, word!

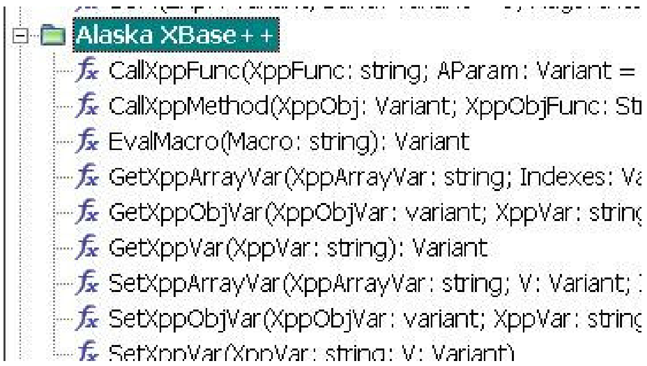
OK. Now we can create and show reports.

Work with data.

Let's consider which way our report can get data. In general, it is possible to divide data that we can receive at construction of reports into three logic groups:

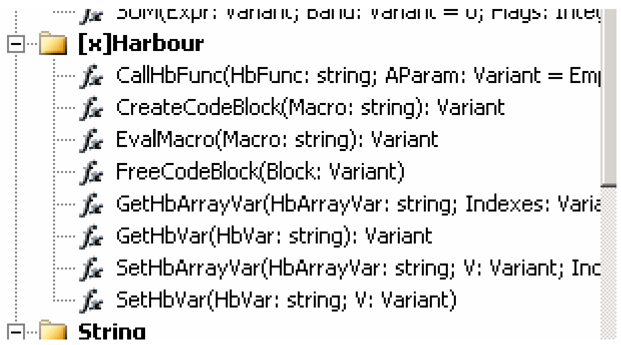
1. The first group is values of various variables of any data type, and also calculated values, which turn out as a result of execution of procedures, functions, etc.
2. The second group is that we name "WorkArea", in other terms can have names – "DataSet", "RecordSet", etc. Usually it is an essence concerning to a DB.
3. And the third group. It's everything that is not WorkArea, but can be presented as WorkArea. In common, everything that we can order as a two-dimensional array with additions of such concepts as "name of field ", "current cursor position ", etc.

Let's consider ways by which we can get data of the first group. There are some ways to transfer data from[x]Harbour or Xbase++ to FastReport Engine. The first way to get values of variables, to call procedures and functions etc, is to take advantage of built-in FastReport functions. Come to designer and pay attention to right toolbar, page "Functions". We see the list of procedures and functions among which there are functions of a category "Alaska Xbase ++":



Alaska XBase++
CallXppFunc(XppFunc: string; AParam: Variant =
CallXppMethod(XppObj: Variant; XppObjFunc: St
EvalMacro(Macro: string): Variant
GetXppArrayVar(XppArrayVar: string; Indexes: Va
GetXppObjVar(XppObjVar: variant; XppVar: strin
GetXppVar(XppVar: string): Variant
SetXppArrayVar(XppArrayVar: string; V: Variant;
SetXppObjVar(XppObjVar: variant; XppVar: strin
SetXppVar(XppVar: string; V: Variant)

Or for [x]Harbour:



SUM(Expr: Variant; Band: Variant = 0; Flags: Integ
[x]Harbour
CallHbFunc(HbFunc: string; AParam: Variant = Em
CreateCodeBlock(Macro: string): Variant
EvalMacro(Macro: string): Variant
FreeCodeBlock(Block: Variant)
GetHbArrayVar(HbArrayVar: string; Indexes: Varia
GetHbVar(HbVar: string): Variant
SetHbArrayVar(HbArrayVar: string; V: Variant; Inc
SetHbVar(HbVar: string; V: Variant)
String

Let's consider these built-in functions in detail. The declaration of functions is presented in Pascal syntax. There is nothing terrible, for example, Microsoft Office uses VBasic. In OLE-technology we use VBasic too, though we program in other language. Small explanatory about data type "variant" - it is the same as variables in Xbase ++. That is, "variant" can be any data type.

In the designer, you can easily drag any function in any text object of the report. Square brackets are used in FastReports text objects for allocation of expression.

Built-in functions

*Firstly we consider functions for **FRAX**:*

```
function GetXppVar(XppVar: string): Variant
```

Parameters: <XppVar> – name of Xbase++ variable.

Returns value of any Xbase++ variable. Objects and arrays more than three-dimensional are not supported. For work with these types look corresponding functions. Example:

```
[GetXppVar('MyVar')]
```

```
function GetXppArrayVar(XppArrayVar: string; Indexes: Variant): Variant
```

Parameters: <XppArrayVar> – name of Xbase++ array variable. <Indexes> - array of integers that describe the array index.

Returns value of an element of Xbase++ array of any dimension. Example:

```
[GetXppArrayVar('MyArray', [1,6])]
```

```
function GetXppObjVar(XppObjVar: variant; XppVar: string): Variant
```

Parameters: <XppObjVar> – string name of Xbase++ object variable or array of strings that describe the object (nested objects). <XppVar> - name of Xbase++ instance variable.

Returns value of Xbase++ instance variable. Example:

```
[GetXppObjVar('MyObject', 'MyObjVar')]
[GetXppObjVar(['MyObject', 'MySubObject'], 'MyObjVar')]
```

```
function CallXppFunc(XppFunc: string; AParam: Variant=EmptyVar): Variant
```

Parameters: <XppFunc> - the character string identifying the name of the Xbase++ function. <AParam> - optionally, array of parameters that will be passed to the Xbase++ function.

Call an Xbase++ function with a variable parameter list and return result value. Examples:

```
[CallXppFunc('SubStr', ['HGFHGFHGFHG', 2, 5])]
[CallXppFunc('MyFunction', [NIL, 10])]
[CallXppFunc('DbSkip')]
```

```
function CallXppMethod(XppObj: Variant; XppObjFunc: String; AParam: Variant = EmptyVar): Variant
```

Parameters: <XppObj> – string name of Xbase++ object variable or array of strings that describe the object (nested objects). <XppObjFunc> - The character string identifying the name of the method. <AParam> - optionally, array of parameters that will be passed to the Xbase++ method.

Call an Xbase++ method with a variable parameter list and return result value. Examples:

```
[CallXppMethod('MyDialog', 'setTitle', ['bla-bla-bla'])]
```

```
function EvalMacro(Macro: string): Variant
```

Parameters: <Macro> - character string which contains a macro expression. Any expression which the Xbase++ macro operator(&) can execute is allowed.

Execute of a macro expression in a string. Example:

```
[EvalMacro('SetAppWindow():hide()')]
```

---

*And functions for **FRH**:*

```
function GetHbVar(HbVar: string): Variant
```

Parameters: <HbVar> – name of [x]Harbour++ variable.

Returns value of any [x]Harbour variable. Objects and arrays are not supported. Example:

```
[GetHbVar('MyVar')]
```

```
function GetHbArrayVar(HbArrayVar: string; Indexes: Variant): Variant
```

Parameters: <HbArrayVar> – name of [x]Harbour array variable. <Indexes> - array element index.

Returns value of an element of [x]Harbour array. Example:

```
[GetHbArrayVar('MyArray', [6])]
```

```
function CallHbFunc(HbFunc: string; AParam: Variant=EmptyVar): Variant
```

Parameters: <HbFunc> - the character string identifying the name of the [x]Harbour function. <AParam> - optionally, array of parameters that will be passed to the [x]Harbour function.

Call an [x]Harbour function with a variable parameter list and return result value. Examples:

```
[CallHbFunc('SubStr', ['HGFHGFHGFHG', 2, 5])]
[CallHbFunc('MyFunction', [NIL, 10])]
[CallHbFunc('DbSkip')]
```

```
function EvalMacro(Macro: string): Variant
```

Parameters: <Macro> - character string which contains a macro expression. Any expression which the [x]Harbour macro operator(&) can execute is allowed.

Execute of a macro expression in a string. Example:

```
[EvalMacro('SomeVar := Directory(SomeDir)')]
```

```
function CreateCodeBlock(Macro: string): Variant
```

<Macro> - character string which contains a macro expression for codeblock. Create codeblock. Created codeblock must be freed. Example:

```
var
  CodeBlock: Integer;
procedure ReportOnStartReport(Sender: TfrxComponent);
begin
  CodeBlock := CreateCodeBlock('{||ITEMS->PartNo}');
end;
```

```
procedure FreeCodeBlock(Block: Variant)
```

<Block> - variable that contain codeblock.

Frees codeblock. Example:

```
procedure ReportOnStopReport(Sender: TfrxComponent);
begin
  FreeCodeBlock(CodeBlock);
end;
```

Thus, in any text object of report (and in scripts, but see below about it) we can get any data with direct call of build-in FastReport functions.

OnGetValue – event

The second way to get data of the first group is to define code block as handler of FastReport event - "OnGetValue". The class frReportManager has a universal method to define handler of any FastReport event:

```
:SetEventHandler(<cObjectName>, <cPropName>, [<bEventHandler>])
```

Parameters: <cObjectName> - character string containing the name of base FastReport object. Now it's possible to have one of two values – "Report" or "Designer". <cPropName> - name of event. Full list of supported events see below. <bEventHandler> - code block that will execute when event occur, default - NIL.

Define code block - <bEventHandler> that will execute when FastReport event occurs. Example:

```
SetEventHandler("Report", "OnProgress" ,{|x,y|MyShowProgress(x, y)})
```

For more information about SetEventHandler method see below. Now consider "OnGetValue"-event only:

```
OnGetValue(VarName) -> VarValue
```

Thus, handler should have one parameter - a name (identifier) of "variable" and should return value of this variable. Example:

```
FUNCTION GetValue(VarName)
     DO CASE
          CASE VarName == “MyVar1”
               RETURN 56
          CASE VarName == “MyVar2”
               RETURN Substr(&MyVar3, 1, 3)
     OTHERWISE
          RETURN NIL
     ENDCASE
RETURN NIL
```

PAY ATTENTION! If parameter VarName is "another's" variable, which value should be defined by another way, handler should return NIL.

Now define event handler:

```
FrPrn:SetEventHandler("Report", "OnGetValue", ;
                              {| VarName | GetValue(VarName)})
```

That’s all. In any text objects of report we can write:

```
[MyVar1]
```

And in preview we will see value – 56.

AddFunction and AddVariable

These ways of access to data are "combination" of first two. These ways are a little bit more convenient because functions and variables appear in the designer visually, on pages "Data Tree". Besides, the name of variables can contain spaces, that is, this name can be "well readable" pseudonym for "worse readable" real variables or expressions. AddFunction and AddVariable are methods of frReportManager class.

```
:AddFunction(<cFunction>, [<cCategory>], [<cDescription>])
```

Parameters: < cFunction> - function declaration in Pascal syntax. It means only that the type of all parameters should be specified, and also the type of returned result should be specified. As a whole, it can be neglected, specifying everywhere type of data - Variant. For example:

```
“function MyFunc(Aparam: string; Bparam: Boolean): Double”
“function MyFunc(Aparam: Variant; Bparam: Variant): Variant”
```

These declarations are equivalent. There is only one nuance - control over types of transferred values. Such control will be carried out in the first case, and will not be in the second. Further the table of conformity of types of data [x]Harbour and Xbase++ and FastReport:

[x]Harbour and Xbase++	FastReport (FastScript)
C – character string	String
D – date	TDate, TDateTime
L – logic	Boolean
N – numeric (X, 0)	Byte, Word, Integer, Longint, Cardinal
N – numeric (X, Y)	Real, Single, Double, Extended, Currency
A – array	Array

But, as it has been already told above, there is no any necessity to use exact data types. The data type Variant can have values of all listed types.

< cCategory> - the character string identifying the name of category. The category is easier way of ordering the list of functions. If such category does not exist, it will be created.

<cDescription> - some comment for function. It is visualized at the bottom of the list of functions.

Adds a user function to the list of the functions available in the report. ***OnUserFunction-event handler must be assigned.***

Example for use AddFunction method and OnUserFunction event.

```
OnUserFunction(cMethodName, aParams) -> xResult
```

Handler should have two parameters - <cMethodName> - the character string identifying the name of procedure or function. < aParams > - array of parameters for procedure or function.

```
FUNCTION CallUserFunction(FName, FParams)
LOCAL RES
  IF (FName == "XBASESTR")
    RES := Str(FParams[1], FParams[2], FParams[3])
  ENDIF
RETURN RES
```

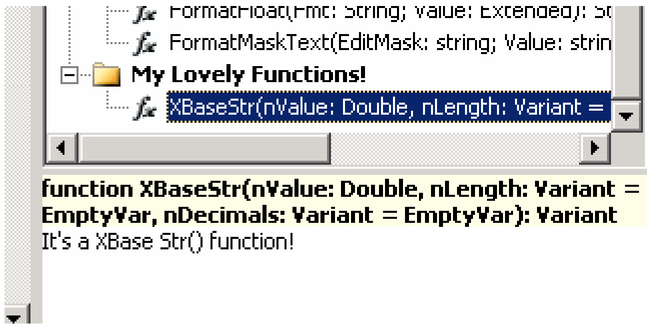
Now define event handler:

```
FrPrn:SetEventHandler("Report", "OnUserFunction", {|FName,;
                      FParams| CallUserFunction(FName, FParams)})
```

And add function XbaseStr():

```
FrPrn:AddFunction("function XbaseStr(nValue: Double,;
  nLength: Variant=EmptyVar, nDecimals: Variant = EmptyVar):Variant",;
            "My Lovely Functions!", "It's a Xbase++ Str() function!")
```

Now we can use function XbaseStr().


FormatFloat(Fmt: String; Value: Extended): St
FormatMaskText(EditMask: string; Value: strin
My Lovely Functions!
XBaseStr(nValue: Double, nLength: Variant =

function XBaseStr(nValue: Double, nLength: Variant = EmptyVar, nDecimals: Variant = EmptyVar): Variant
It's a XBase Str() function!

It is necessary to pay attention to syntax of optional parameters. Optional parameters have value EmptyVar. One or a group of optional parameters must be at the right of the list of parameters strictly.

After addition function exists up to unloading FastReport. Unlike functions, variables (the user variables) exist in the report. It means, that variables are a part of the report and are saved/loaded in the report. Generally, these variables are not “real” variables as can have simple value, but also can be executed expression.

IMPORTANT: When accessing a report variable its value is calculated if it is of string type. That means that the variable which value is "GetXppVar(SomeVar)" will return a value of SomeVar - variable, but not the "GetXppVar(SomeVar)"-string. You should be careful when assigning string-type values to report variables. Simple string value must be in Pascal string quotes - '. Examples for xValue:

```
"GetXppVar(SomeVar)"    -> Value of SomeVar
"'GetXppVar(SomeVar)'" -> string - GetXppVar(SomeVar)
```

Thus, it is very convenient way "to hide" nuances of realization from the user if you give the user an opportunity to use the designer. Usually, variables are entered in the designer, menu – «Report-> Variables». But if there is a necessity, it can be done by code, using methods frReportManager:

```
:AddVariable(cCategory, cName, xValue)
```

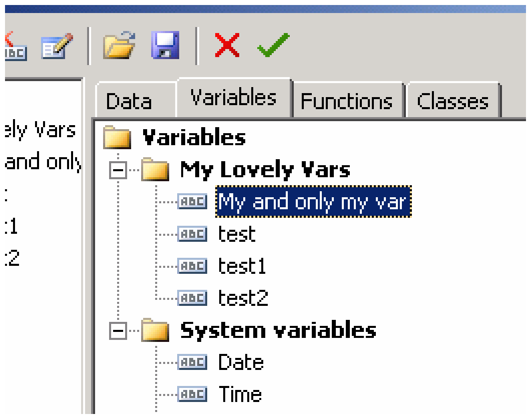
Parameters: < cCategory> - the character string identifying the name of category. The category is easier way of ordering of the list of variables. If such category does not exist, it will be created. <cName> - the character string identifying the name of variable. <xValue> - current value if variable.

Adds a user variable to the list of the variables available in the report.

Example:

```
FrPrn:AddVariable("My Lovely Vars", "My and only my var", 10)
```

After this call we can see in designer:



In any text objects of report we can write - [My and only my var]. And in preview we will see current value – 10.

It is possible to add, to change values, to delete variables both in a code, and in dialogues of the designer.

```
:SetVariable(<sName>, <xValue>)
```

Parameters: <cName> - the character string identifying the name of variable. <xValue> - current value if variable.

Modify a user variables current value.

```
:GetVariable(<cName>)
```

Parameters: <cName> - the character string identifying the name of variable.

Return a user variables value.

```
:DeletVariable(<cName>)
```

Parameters: <cName> - the character string identifying the name of variable.

Delete a user variable

```
:DeleteCategory(cCategory)
```

Parameters: <cName> - the character string identifying the name of category.

Delete a category with ALL its variables.

WorkAreas

So, work with DB-data. Methods of frReportManager:

```
:SetWorkArea(<cFrAlias>, <nWorkArea>, [<lOem>], [<aRangeParams>])
```

Add work area to FastReport data list or modify parameters of work area.

Parameters: <cFrAlias> - the character string identifying the name of work area for FasrReport. Further references to this work area in FastReport are done by this name. <nWorkArea> - number (real Harbour or Xbase number) of WorkArea. <lOem> - If true, then data in oem code page (default - .f.). <aRangeParams> - Range parameters. Array consists of 3 elements:

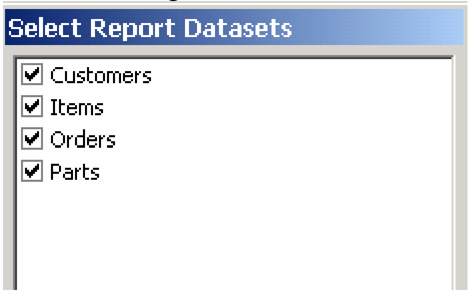
- 1 element - FR_RB_FIRST(0) or FR_RB_CURRENT(1)
- 2 element - FR_RE_LAST(0) or FR_RE_CURRENT(1) or FR_RE_COUNT(2)
- 3 element - A number of records in the WorkArea, if the second element is FR_RE_COUNT else is ignored.

Default - {FR_RB_FIRST, FR_RE_LAST, 0}

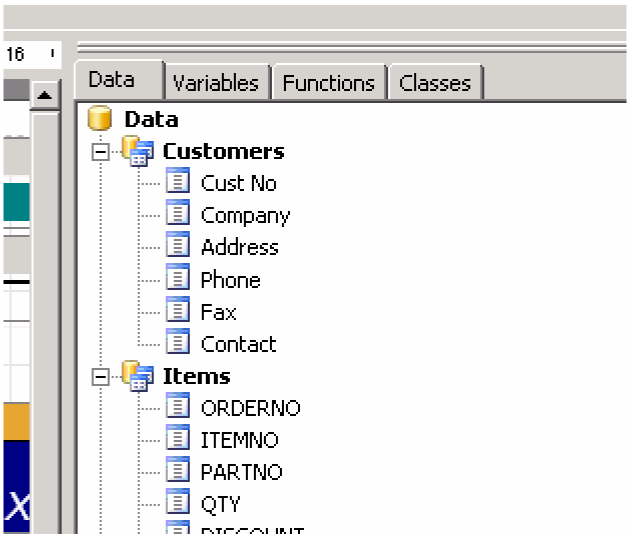
Examples:

```
FrPrn:SetWorkArea("Orders", 2)
FrPrn:SetWorkArea(Alias(), Select(), .t.,{FR_RB_FIRST,FR_RE_COUNT,10})
```

After such calls in designer we can select/unselect work areas for current report (menu – «Report- Data»):



After select we can use work area:



For information about report creation look «FastReport User Manual» and other documentation.

`:SetFieldAliases(<cFrAlias>, <cFieldAliases>)`

Set symbolic names of work area fields and/or hide some fields.

Parameters: <cFrAlias> - the character string identifying the name of work area for FastReport. <cFieldAliases> - the character string constructed as follows:

RealField1=Alias1;RealField2=Alias2; RealField3; RealField4;………………...

String consists of parts, which are divided by a semicolon. Each part can consist of a real name of field, or of pair "real name of filed=alias". If the pair with alias is present then field will be visible in FastReport as this alias, if the real name only, then as this name. If field is not present, it will not be visible in FastReport. For example:

```
:SetFieldAliases("Cust","CUSTNO=Cust No;Company;AD1=Address;Phone")
```

Only four fields will be visible, CUSTNO and AD1 will be visible as 'Cust No' and 'Address'

`:RemoveDataSet(cFrAlias)`

Remove work area or UserDataSet (see below) from Fast Report data list.

Parameters: <cFrAlias> - the character string identifying the FastReport-name of work area.

`:ClearDataSets()`

Remove All WorkAreas or UserDataSets (look below) from Fast Report data list.

So, now by means of these methods we can obtain data from a DB. It is already enough to create reports on the basis of separate work area, but still it is not enough to create more complex reports on the basis of many work areas.

Complex reports on the basis of many work areas.

ATTENTION! Read the following information carefully. The understanding of principles of FastReport work with Xbase++ or [x]Harbour work areas will allow you to create reports of any complexity.

Usually, complex reports are contained data from several working areas, thus these working areas have any dependences with each other. In Xbase ++ the most widespread way of creation of such dependences is use of command SET RELATION (DbSetRelation). Any such dependence can be described as the relation between two work areas at which change of a position of the cursor in one work area leads to change of a position of the cursor in the second work area (or/and to change of a set of records in the second table). For example:

```
USE Items NEW
USE Parts NEW
SET INDEX TO Parts
SELECT Items
DbSetRelation("Parts", {||PartNo})
```

Add these two work areas to the current report:

```
FrPrn:SetWorkArea("Items", Select("Items"))
FrPrn:SetWorkArea("Parts", Select("Parts"))
```

NOW IT IS NOT ENOUGH FOR WORK. Let's explain why it’s so .

SetWorkArea-method creates object in FastReport that operates work area. This object calls such functions as DbGoTop(), DbSkip(), Eof(), etc. Besides, for the best performance, this object has a small cache in which current record of work area is stored. We have added two objects in FastReport, but now these objects know nothing about each other. Let's explain it more carefully:

Let's assume, that the first object which operates the first work area, has called function DbSkip(). The real cursor position of the second work area has changed. However, the second object knows nothing about it. FastReport has requested current record of the second object. The second object has returned record from its cache. So:

If we have some work areas that have relations - we should inform FastReport about it.

For informing FastReport that two work areas have dependence use method of frReportManager – SetResyncPair:

```
:SetResyncPair(<cParentAlias>, <cChildlAlias>)
```

Parameters: <cParentAlias> - the character string identifying the FastReport-name of parent work area. <cChildlAlias> - the character string identifying the FastReport-name of child work area.

Establishes a rule - after changing of parent work area cursor position (in FastReport object), object of child work area must resynchronize its cursor position with real Xbase++ data. Example:

```
FrPrn:SetResyncPair("Items", "Parts")
```

It is necessary to note, that parent can have as much children as necessary, and child can have as much parents as necessary. Only circular references are not allowed. It means, that the parent cannot be a child of its child directly or indirectly.

Now we can add a code that we have started to write above. So, completely this fragment should look approximately so:

```
USE Items NEW
USE Parts NEW
SET INDEX TO Parts
SELECT Items
DbSetRelation("Parts", {||PartNo})
FrPrn:SetWorkArea("Items", Select("Items"))
FrPrn:SetWorkArea("Parts", Select("Parts"))
FrPrn:SetResyncPair("Items", "Parts")
FrPrn:DesignReport() // for example
FrPrn:ClearResyncPair("Items", "Parts")
FrPrn:ClearDataSets()
..........................
```

Consider other methods of frReportManager, which concern to work areas.

```
:ClearResyncPair(<cParentAlias>, <cChildlAlias>)
```

Parameters: <cParentAlias> - the character string identifying the FastReport-name of parent work area. <cChildlAlias> - the character string identifying the FastReport-name of child work area.

Cancel the rule, which has been established by a SetResyncPair-call

---

```
:Resync(<cFrAlias>)
```

Parameters: <cFrAlias> - the character string identifying the FastReport-name of work area.

Resynchronize FastReport work area object cursor position with real Xbase++ data. This method should be used if during construction of the report when there was call Xbase ++ functions in which the current position of the cursor has been changed. For example:

In FastReport script (look below):

```
procedure SomeBandOnAfterPrint(Sender: TfrxComponent);
begin
  CallXppFunc('MyFunction');
end;
```

In Xbase++:

```
FUNCTION MyFunction()
   SELECT Items
   DbSkip(5)
   FrPrn:Resync("Items")
RETURN NIL
```

```
:SetMasterDetail(<cMasterAlias>, <cDetailAlias>, <bScopeValue>)
```

Creating master-detail relationships between two work areas. It means that each time the current record in the master work area changes (in FastReport), the new values in master fields are used to set scope in detail work area. Detail work area must have active index.

Parameters: <cMasterAlias> - the character string identifying the FastReport-name of master work area.

<cDetailAlias> - the character string identifying the FastReport-name of detail work area.

<bScopeValue> - code block, must return value that will be passed to DbSetScope()-function.

For example:

```
FrPrn:SetMasterDetail ("Customers", "Orders",;
                                   {||Customers->CustNo})
```

That is, at change of a current cursor position in the table "Customers", function DbSetScope() for the table "Orders" will be called. The parameter for DbSetScope() (OrdScope()) is value that return <bScopeValue>-code block.

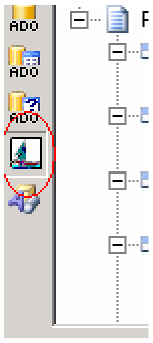
Master can have as much as necessary details, but detail can have only one master. Circular references are not allowed.

```
:ClearMasterDetail(<cDetailAlias>)
```

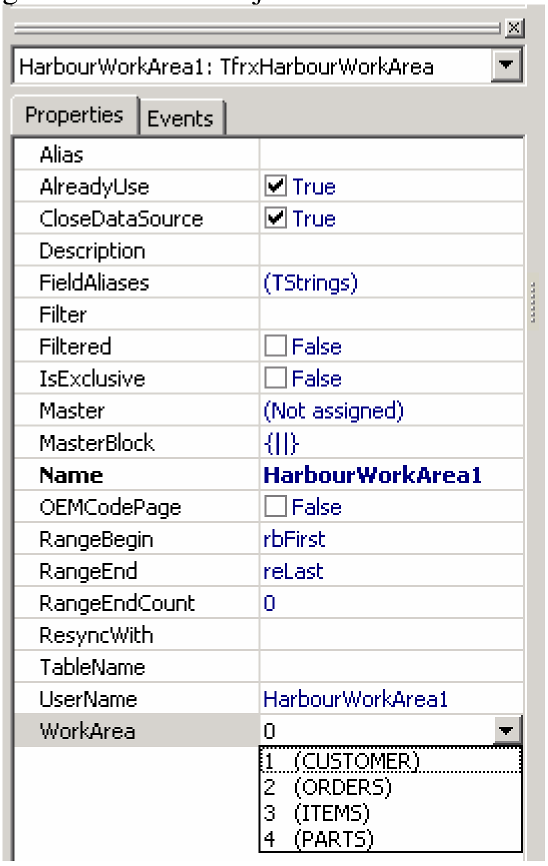
Parameters: <cDetailAlias> - the character string identifying the FastReport-name of detail work area.

Remove master-detail relationships between two work areas.

Built-in object for work with WorkAreas

We have considered definition of work areas from Xbase++ ([x]Harbour) a code. Besides you can define work areas used in the report directly in the designer. For this purpose the built-in FastReport-objects are used – TfrxAlaskaWorkArea for Alaska Xbase and TfxhHarbourWorkArea for [x]Harbour. Presence of this object allows building many reports "only by mouse". For creation of object choose on the panel of objects (Tab «Data) choose corresponding dataset: 

FastReport-designer will create object

and will display its properties in the «object inspector». 

HarbourWorkArea1: TfrxHarbourWorkArea

Properties | Events

Alias	
AlreadyUse	✓ True
CloseDataSource	✓ True
Description	
FieldAliases	(TStrings)
Filter	
Filtered	False
IsExclusive	False
Master	(Not assigned)
MasterBlock	{||}
Name	**HarbourWorkArea1**
OEMCodePage	False
RangeBegin	rbFirst
RangeEnd	reLast
RangeEndCount	0
ResyncWith	
TableName	
UserName	HarbourWorkArea1
WorkArea	0

1 (CUSTOMER)
2 (ORDERS)
3 (ITEMS)
4 (PARTS)

Let’s consider these properties.

Alias – alias of work area. In combobox-editor there are list of aliases of all opened work areas at this moment.

AlreadyUse – if true, it means that work area is opened now, else FastReport will call DbUseArea().

ClosedDataSourse – this property is not used now.

Description – object’s description.

FieldAliases – List of aliases of fields of work area. Has the same sense that :SetFieldAliases-method.

Filter - The filtering condition.

Filtered - Determines whether the work area should filter the records using the condition in the Filter property

IsExclusive – when AlreadyUse=false this value is used for DbUseArea(). Ignored if AlreadyUse is True.

Master, MasterBlock – the same sense as first and third parameters of :SetMasterDetail-method.

Name – name of object.

OEMCodePage – Determines whether the work area uses OEM or ANSI code pages. Ignored if AlreadyUse is True.

RangeBegin, RangeEnd, RangeEndCount - the same sense as fourth parameter of :SetWorkArea-method.

ResyncWith - the same sense as in :SetResyncPair()-method.

TableName - Ignored if AlreadyUse is True. The character string containing name of the file for DbUseArea().

UserName - object's user name (for designer data-tree and so on).

WorkArea – numeric number of work area. When AlreadyUse=false 0-value is used as flag for open in a free work area (first parameter of DbUseArea()). In combobox-editor there are numeric list of all opened work areas at this moment.

It's possible to write script event handlers for BeforeOpen, AfterOpen, BeforeClose and AfterClose event. For this purpose double click on value of corresponding event in the inspector of objects.

UserDataSet

There is wide spectrum of data which are not a DB, but work with which can become much more convenient if to present these data as DB-data. Example of such data - arrays, files, etc. Also, working with DB-data, in some complex cases, the full control over data processing is necessary. For these cases FastReport offers special mechanism - UserDataSet.

UserDataSet is a virtual data set which formation is made by the programmer. For creation new UserDataSet the method of frReportManager - SetUserDataSet() is used:

```
      :SetUserDataSet(<sFrAlias>, <sFields>, <bGoTop>, <bSkipPlus1>,
<bSkipMinus1>, <bCheckEOF>, <bGetValue>)
```

Parameters: < sFrAlias> - the character string identifying the name of dataset for FastReport.

<sFields> - the character string containing one or more field names in dataset. Separate field names with semicolons.

<bGoTop> - This event's handler code block must move the cursor to the beginning of the dataset.

<bSkipPlus1> - This event's handler code block must move the cursor to the next record.

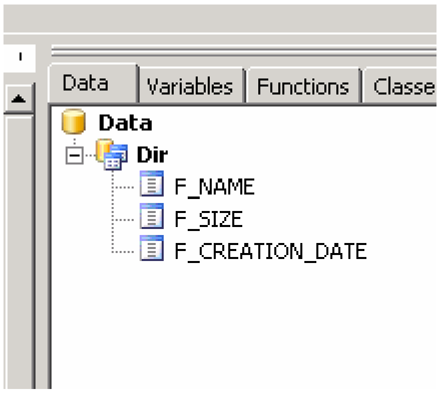
<bSkipMinus1> - This event's handler code block must move the cursor to the previous record.

<bCheckEOF> - This event's handler code block must return the .t. if the end of the dataset is reached.

<bGetValue> - This event's handler code block must return current field value. Has one parameter - a name of a field.

Let's consider use of UserDataSet. Let us suppose, we wish to print array, which was returned with function Directory().

```
PRIVATE aDir, I := 1, DirName := "C:\*.*"
aDir := Directory(DirName)
FrPrn:SetUserDataSet("Dir", "F_NAME;F_SIZE;F_CREATION_DATE",;
                               {||I := 1}, ;
                               {||I := I + 1}, ;
                               {||I := I - 1}, ;
                               {||I > Len(aDir)}, ;
{| cField | xx:=IF(cField="F_NAME",1, IF(cField="F_SIZE",2,9)),;
 aDir[I, xx]})
```

After such calls in designer we can see: 

So that, for FastReport there is no difference between real work area and virtual userdataset. Just for navigation on work area FastReport uses "usual" calls: DbGoTop(), DbSkip(), Eof(), etc. But for navigation on userdataset FastReport executes code blocks that had been defined by the developer. As a result, we can generate any possible data set with a minimum quantity of the written code.

For remove UserDataSet from FastReport data list use RemoveDataSet and ClearDataSets methods.

Other methods of frReportManager class

Report Processing

```
:PrepareReport(<nNotClear>)
```

Parameters: < nNotClear> - the same as in ShowReport()

Starts a report without a preview window. See example of ShowPreparedReport().

```
:ShowPreparedReport()
```

Displays the report, which was previously built via the "PrepareReport" call. Example:

```
FrPrn:LoadFromFile(RepDir + "1.fr3")
FrPrn:PrepareReport()
FrPrn:LoadFromFile(RepDir + "4.fr3")
FrPrn:PrepareReport(FR_NOTCLEARLASTREPORT)
```

```
FrPrn:ShowPreparedReport()
```

```
:LoadFromBlob(<nWorkArea>, <cFieldName>)
```

Loads a report from a memo or blob field. <nWorkArea> - number (real Harbour or Xbase number) of WorkArea. <cFieldName>) – name of memo or blob field.

```
:SaveToBlob(<nWorkArea>, <cFieldName>)
```

Saves a report to a memo or blob field. <nWorkArea> - number (real Harbour or Xbase number) of WorkArea. <cFieldName>) – name of memo or blob field.

```
:LoadFromResource(<xValue>)
```

Loads a report from a resource with given name or number. Returns .t. if the file is loaded successfully. <xValue> - name or number of resource (resource must be RCDATA type).

```
:SaveToFile(<cFileName>)
```

Parameters: <cFileName> - character string containing the name of report file. It can include drive and path specifications.

Saves a report to a file with given name.

```
:Print([<lDialogIsChild>])
```

Parameters: <lDialogIsChild> - if true then PrintDialog will be the child of current foreground window (default –False).

Prints a report. See frPrintOptions class for more information.

```
:DoExport(<cExportObjectName>)
```

Exports a report using the specified export filter object.

Parameters: <cExportObjectName> - character string containing the name of filter object. Possible value one of:

```
     "PDFExport", "HTMLExport", "RTFExport", "CSVExport", "XLSExport",
"DotMatrixExport", "BMPExport", "JPEGExport", "TXTExport",
"TIFFExport", "GIFExport", "SimpleTextExport", "MailExport",
"XMLExport", "ODSExport", "ODTExport"
```

The detailed descriptions of these objects see below.

```
:LoadStyleSheet(<cFileName>)
```

Parameters: <cFileName> - character string containing the name of style sheet file. It can include drive and path specifications.

Loads style sheet to current report.

```
:GetErrors()
```

Returns list of errors in one character string with chr(13)+chr(10) delimiter. It’s important in silent mode.

Visual options

`:SetIcon(<xValue>)`

Parameters: <xValue> - the numeric resource ID or string NAME of the icon.

If an icon is specified, it is displayed at the left end of FastReport windows. The icon and its numeric ID or string NAME must be defined as a resource.

`:SetTitle(<cTitle>)`

Parameters: < cTitle> - the character string containing title of FastReport taskbar-window

Set title of FastReport taskbar-window

```
      :SetVisualActions(<nAction>, [<lTaskListToFalse>],
[<bBeforeAction>], [<bAfterAction>])
```

Define actions, which will execute before and after such methods as ShowReport, DesignReport etc. The method allows to adjust change of active windows (Xbase++ and FastReport), as it is necessary.

Parameters: <nAction> - The following values are available:

	"Before" action	"After" action
FR_ACT_NONE (0)	No action	No action
FR_ACT_HIDE (1)	SetAppWindow():hide()	SetAppWindow():show()
FR_ACT_DISABLE (2)	SetAppWindow():disable()	SetAppWindow():enable()
FR_ACT_USER (3)	Eval(bBeforeAction)	Eval(bAfterAction)

Default - FR_ACT_NONE

<lTaskListToFalse>	SetAppWindow():taskList := .f.	SetAppWindow():taskList := .t.

Default – False

<bBeforeAction> - if nAction = FR_ACT_USER code block that will execute before visual methods, else ignored.

<bAfterAction> - if nAction = FR_ACT_USER code block that will execute after visual methods, else ignored.

Parameters FR_ACT_HIDE, FR_ACT_DISABLE and lTaskListToFalse work in Xbase++ only.

`:LoadLangRes(<cLangFile>)`

Parameters: <cFileName> - character string containing the name of FastReport language file. It can include drive and path specifications.

Load language resource file

Manual Build reports

Set of methods those provide "Manual-build"-reports for fast migration of "old-style" reports.

```
:StartManualBuild(<bManualBlock>, <nOrientation>, <nPaperSize>,
<nUnits>)
```

Start Manual-build report. <bManualBlock> - codeblock in which is done formation of the report. <nOrientation> - orientation of paper, can be:

```
FR_PORTRAIT(0)
FR_LANDSCAPE(1)
```

<nPaperSize> - size of paper. It is the same that PaperSize property of TfrxPage (look at designer).

<nUnits> - Units of measure of coordinates. Can be:

```
FR_CM(0)
FR_MM(1)
FR_INCHES(2)
FR_PIXELS(3)
```

Example:

```
FrPrn:StartManualBuild({||MyManualReport()}, FR_LANDSCAPE, ,
FR_PIXELS)
```

```
:MemoAt(<cStr>, <nLeft>, <nTop>, <nWidth>, <nHeight>)->ObjectName
```

Place new Memo object with text at given coordinates. <cStr> - text of memo object. <nLeft>, <nTop>, <nWidth>, <nHeight> - coordinates of memo object. Return name of new object, it can be used at SetManualObjProperty()-method.

Example:

```
tmp_Name := FrPrn:MemoAt("Some memo with bottom frame ...", 30,
30, 300, 50)
```

```
:LineAt(<nLeft>, <nTop>, <nWidth>, <nHeight>)->ObjectName
```

Place new Line object at given coordinates. <nLeft>, <nTop>, <nWidth>, <nHeight> - coordinates of line object. Return name of new object, it can be used at SetManualObjProperty()-method.

Example:

```
FrPrn:LineAt(30, 200, 100, 100)
```

```
:PictureAt(<cFileName>, <nLeft>, <nTop>, <nWidth>, <nHeight>)-
>ObjectName
```

Place new Picture object at given coordinates. <cFileName>- file with graphics. <nLeft>, <nTop>, <nWidth>, <nHeight> - coordinates of line object. Return name of new object, it can be used at SetManualObjProperty()-method.

Example:

```
FrPrn:PictureAt(DataDir + "logo.bmp" , 30, 400, 300, 300)
```

```
:NewPage()
```

Start new page in "Manual-build"-report

```
:SetDefaultFontProperty(<cPropName>, <xValue>)
```

Set any property of font that used for new memo objects. <cPropName> - name of font property. <xValue> - value for property.

Example:

```
FrPrn:SetDefaultFontProperty("Name", "Times New Roman")
FrPrn:SetDefaultFontProperty("Size", 16)
```

```
:SetManualObjProperty(<cObjName>, <cPropName>, <xValue>)
```

Set any property of new objects that was created by :MemoAt, :LineAt, :PictureAt methods. <cObjName> - name of object that was returned «At»-methods. <cPropName> - name of property. <xValue> - new value.

Example:

```
FrPrn:SetManualObjProperty(tmp_Name+".Frame", "Typ",
"[ftBottom]")
```

Look at demo program for full «ManualBuild» example.

Utils

```
SendMail(<cServer>, <nPort>, <cUserField>, <cPasswordField>,;
    <cFromField>, <cToField>, <cSubjectField>, <cCompanyField>,;
    <cTextField>, <cFileNames>)
```

Send a mail. Return: if Ok - empty string, else error text. <cTextField> and <cFileNames> are Chr(13) + Chr(10) delimited strings.

Example:

```
cTextField := "Hello" + Chr(13) + Chr(10) + "How are you?"
cFileNames := "C:\MyAttach1.bin"+ Chr(13) + Chr(10) +
"C:\MyAttach2.txt"

S := FrPrn:SendMail("someserver.com", 25, "MyUserName",
"MyPassword", "from@xxx.ru", "to@xxx.es", "MySubject", "MyCompany",
cTextField, cFileNames)

   IF S <> ""
      MsgBox(S)
   ELSE
      MsgBox("Mail is sent")
   ENDIF
```

Print options

The set of the methods, which are responsible for adjustment of printing, is allocated in a separate class - frPrintOptions. The instance of this class is created at initialization of frReportManager. The name of instance-variable is PrintOptions.

```
:PrintOptions:SetCopies([<nCopies>]) // ->OldValue
```

Parameters: <nCopies> - a number of the printable copies by default.

Set or get a number of the printable copies (default - 1)

```
:PrintOptions:SetCollate([<lCollate>]) // ->OldValue
```

Parameters: <lCollate> - whether to collate the copies. Default - True

Set or get copies collation.

```
:PrintOptions:SetPageNumbers([<cParam>]) // ->OldValue
```

Parameters: <cParam> - character string containing page numbers, which are to be printed. For example: "1,3,5-12,17-".

Set or get page numbers, which are to be printed. Default – empty string.

```
:PrintOptions:SetPrinter([<cPrinter>]) // ->OldValue
```

Parameters: <cParam> - character string containing printer name.

Get or set printer name. Default - default printer in OS.

```
:PrintOptions:SetPrintPages([<nPages>]) // ->OldValue
```

Defines the pages to be printed.

Parameters: <nPages> - The following values are available:

- FR_PP_ALL - all pages
- FR_PP_ODD - odd pages
- FR_PP_EVEN - even pages

Default - FR_PP_ALL

```
:PrintOptions:SetShowDialog([<lShow>]) // ->OldValue
```

Whether to display a print dialog. Default – True.

```
:PrintOptions:SetReverse([<lReverse>]) // ->OldValue
```

Get or set reports reverse. Default – False.

```
PrintOptions:ClearOptions()
```

Set all print options to default values.

Preview Options

The set of the methods, which are responsible for adjustment of preview window, is allocated in a separate class - frPreviewOptions. The instance of this class is created at initialization of frReportManager. The name of instance-variable is PreviewOptions.

```
:PreviewOptions:SetAllowEdit(<lAllow>)
```

Enables or disables a finished report editing. Default – True

---

```
:PreviewOptions:SetButtons(<nButtons>)
```

A set of buttons, which will be available in the preview window.

Parameters: < nButtons> - the arithmetic sum of following possible values:

FR_PB_PRINT	1
FR_PB_LOAD	2
FR_PB_SAVE	4
FR_PB_EXPORT	8
FR_PB_ZOOM	16
FR_PB_FIND	32
FR_PB_OUTLINE	64
FR_PB_PAGESETUP	128
FR_PB_TOOLS	256
FR_PB_EDIT	512
FR_PB_NAVIGATOR	1024

Default – all buttons.

```
:PreviewOptions:SetDoubleBuffered(<lDouble>)
```

Set double-buffer mode for the preview window. If enabled (by default), the preview window will not flicker during repainting, but the process speed would be reduced.

```
:PreviewOptions:SetMaximized(<lMax>)
```

Defines whether the preview window is maximized.

```
:PreviewOptions:SetBounds(<nLeft>, <nTop>, <nWidth>, <nHeight>)
```

Defines position and size of the preview window. System of coordinates is standard for Windows.

```
:PreviewOptions:SetOutlineVisible(<lVis>)
```

Defines whether the panel with the report outline is visible.

```
:PreviewOptions:SetOutlineExpand(<lExpand>)
```

Defines whether the tree of the report outline is expanded.

```
:PreviewOptions:SetOutlineWidth(<nWidth>)
```

Defines width of the panel with the report outline (in pixels).

```
:PreviewOptions:SetShowCaptions(<lShow>)
```

Defines whether it is necessary to display button captions. When enabling this property, you should limit the number of the displayed buttons with SetButtons() function, since all the buttons would not find room on the screen.

```
:PreviewOptions:SetZoom([<nZoom>]) //->OldValue
```

Set default zooming, for example - 1.4, Default – 1.0.

```
:PreviewOptions:SetZoomMode([<nZMode>]) //->OldValue
```

Set default zooming mode.

Parameters: <nZMode> - the following values are available:

FR_ZM_DEFAULT	0
FR_ZM_WHOLEPAGE	1
FR_ZM_PAGEWIDTH	2
FR_ZM_MANYPAGES	3

Default - FR_ZM_DEFAULT

```
:SetPictureCacheInFile(<lCache>)
```

Cache pictures or not, default - true. <lCache> - Boolean value.

Current Report Options

The set of the methods, which are responsible for adjustment of a current report, is allocated in a separate class - frReportOptions. The instance of this class is created at initialization of frReportManager. The name of instance-variable is ReportOptions.

```
:ReportOptions:SetAuthor([<cAuthor>]) // ->OldValue
```

Set or get report author.

```
:ReportOptions:SetCompressed([<lCompressed>]) // ->OldValue
```

Set or get report zip-compressed.

```
:ReportOptions:SetCreateDate([<cDateTime>]) // ->OldValue
```

Set or get report creation date.

Parameters: <cDateTime> - this parameter must use the current locale's date/time format. In the US, this is commonly MM/DD/YY HH:MM:SS format; in Russia this is DD.MM.YYYY HH:MM:SS. Specifying AM or PM as part of the time is optional, as are the seconds. Use 24-hour time (7:45 PM is entered as 19:45, for example) if AM or PM is not specified.

```
:ReportOptions:SetLastChange([<cDateTime>]) // ->OldValue
```

Set or get last change date. Parameter is the same as in :ReportOptions:SetCreateDate().

```
:ReportOptions:SetDescription([<cDescription>]) // ->OldValue
```

Set or get report description.

```
:ReportOptions:SetInitString([<cInitString>]) // ->OldValue
```

Set or get report init string for dot-matrix reports

```
:ReportOptions:SetName([<cName>]) // ->OldValue
```

Set or get report name.

```
:ReportOptions:SetPassword([<sPassword>]) // ->OldValue
```

Set or get report password. If password is not blank, a password is required when opening a report.

```
:ReportOptions:SetVersion([<aVersion>]) // ->OldValue
```

Set or get report version.

Parameters: <aVersion> - array with 4 character string element: {<cVersionMajor>, <cVersionMinor>, <cVersionRelease>, <cVersionBuild>}

Engine Options

The set of the methods, which are responsible for adjustment of a FastReport engine, is allocated in a separate class - frEngineOptions. The instance of this class is created at initialization of frReportManager. The name of instance-variable is EngineOptions.

```
:EngineOptions:SetConvertNulls(<lConvert>)
```

Set rule - converts the "Null" value of the DB field into "0", "False," or empty string, depending on the field type, or not converts. Default – True.

```
:EngineOptions:SetDoublePass(<lDouble>)
```

Makes a report a two-pass one. Default – False.

```
:EngineOptions:SetPrintIfEmpty(<lPrint>)
```

Defines, whether it is necessary to print a blank report (one which containing no data lines).

```
:EngineOptions:SetSilentMode(<lSilent>)
```

"Silent" mode. Thus all messages about errors are stored in the Errors property (see GetErrors() method), without displaying any messages on the screen. Default – False.

```
:EngineOptions:SetNewSilentMode(<nMode>)
```

More powerful variant of SetSilentMode(). The following values are available:

• FR_SIM_MESSAGEBOXES	(0)
• FR_SIM_RETHROW	(1)
• FR_SIM_SILENT	(2)

```
:EngineOptions:SetMaxMemSize(<nSize>)
```

The maximum size of memory in Mbytes, allocated to the report pages' cache. It becomes useful in cases when the "UseFileCashe" property is equal to True (see SetUseFileCache() method). If a report begins to occupy more memory during construction, caching of the constructed report pages into a temporary file is performed. This property is inexact and allows only approximate determination of the memory limit. Default – 10.

```
:EngineOptions:SetTempDir(<cDir>) //->OldValue
```

Specifies a path to the directory for storing temporary files.

```
:EngineOptions:SetUseFileCache(<lUse>)
```

Defines, whether it is necessary to use report pages caching into the file.

Interaction with FastScript

For interaction with FastScript- subsystem three methods exist in frReportManager. Instead of report variables, script variables are not visible in the report variables list and are not calculated.

```
:GetScriptVar(<cName>)
```

Parameters: <cName> - character string containing the name of script variable.

Get script variable value. For example:

In FastScript:

```
MyVar := 10;
CallXppFunc(‘MyFunction’);
```

In Xbase++

```
FUNCTION MyFunction()
  IF FrPrn:GetScriptVar("MyVar")=10 …
```

```
:SetScriptVar(<cName>, <xValue>)
```

Set script variable value.

Parameters: <cName> - character string containing the name variable. <xValue> - any value. If such variable does not exist, it is created.

```
:Calc(<cExpression>)
```

Parameters: <cExpression> - any valid FastScript expression.

Calculates any FastScript expression and returns the result.

Universal methods of frReportManager

FastReport is a product with greater rates of development. The quantity of opportunities, new objects, etc. will increase. The quantity of various adjustments, options for various objects, is great. To create a method of frReportManager for each option is not meaningful. Instead of it universal methods, which can work with any object of FastReport, are realized.

Set event handlers

```
:SetEventHandler(<cObjectName>, <cPropName>, [<bEventHandler>])
```

Parameters: <cObjectName> - character string containing the name of FastReport object. <cPropName> - name of event. <bEventHandler> - code block that will execute when event occur, default - NIL.

Define code block - <bEventHandler> that will executes when FastReport event occur.

List of supported events of "Report" object:

`OnAfterPrint(<cObjName>)`

When starting a report. It occurs after handling each object. <cObjName> - name of report object.

`OnAfterPrintReport()`

It occurs after report printing.

`OnBeforeConnect(<cConnectName>)`

When external data source (ADO, IBX, DBX etc.) is used, it occurs before connect to external data source. <cConnectName> - name of connection object.

`OnBeforePrint(<cObjName>)`

When starting a report. It occurs before handling each object. <cObjName> - name of report object.

`OnBeginDoc()`

It occurs before report building.

`OnEndDoc()`

It occurs after report building.

`OnClickObject(<cObjName>, <nButton>)`

When previewing a report in the preview window. Occurs when clicking the object. <cObjName> - name of report object. <nButton> - mouse button:

• Left	0
• Right	1
• Middle	2

`OnGetValue(<cVarName>) -> VarValue`

When starting a report. Occurs when an unknown variable is met. An event handler must return the value of this variable (see topic above).

`OnPreview()`

It occurs before preview window show.

`OnPrintPage(<nCopyNo>)`

It occurs before page of report printing. <nCopyNo> - number of copy.

`OnPrintReport()`

It occurs before report printing.

`OnProgress(<nProgressType>, <nProgress>)`

Event handler is called for indicating current task progress (report building, printing, exporting). This handler can use your own indicator. <nProgressType> - type of progress:

• Building	0
• Exporting	1
• Printing	2

<nProgress> - current page number.

`OnProgressStop(<nProgressType>, <nProgress>)`

It occurs when progress is stopped. Parameters are the same as in OnProgress().

`OnProgressStart(<nProgressType>, <nProgress>)`

It occurs when progress is started. Parameters are the same as in OnProgress().

`OnUserFunction(<cMethodName>, <aParams>) -> xResult`

Occurs when calling a function, added with the help of the AddFunction() method (look topic above).

List of supported events of " Designer" object:

`OnInsertObject()`

It occurs when new object is inserted.

`OnLoadReport() ->lResult`

The event occurs when loading a report. With the help of this event, you can manage report's loading. Event handler must load a report and return True if operation was successful.

`OnSaveReport(<lSaveAs>) ->lResult`

The event occurs when saving a report. With the help of this event, you can manage process of report's saving. Event handler must save a report and return True if operation was successful. <lSaveAs> - parameter is True in case the user selects "Save As..." menu item.

`OnShow()`

The event occurs when the designer starts.

`OnShowStartupScreen()`

The event occurs when the designer shows.

Get/Set Properties

`:GetProperty(<cObjectName>, <cPropName>) - xValue`

Universal function that return value of any FastReport-object property.

Parameters: <cObjectName> - character string containing the name of FastReport object. The full list of objects look below. For the enclosed objects the divider "." is used. For example:

```
"Designer.DefaultFont"
```

<cPropName> - character string containing the name of property. The full list of properties look below.

```
:SetProperty(<cObjectName>, <cPropName>, <xValue>)
```

Universal function that set value of any FastReport-object property.

Parameters: <cObjectName>, <cPropName> - the same as in GetProperty(). <xValue> - any value.

About compatibility of data types:

If property is string property then <xValue> is character string. For example:

```
FrPrn:SetProperty("Report", "ScriptLanguage", "C++Script")
```

If property is any of numeric types property then <xValue> is numeric. For example:

```
FrPrn:SetProperty("Designer", DefaultLeftMargin", 15)
```

If property is Boolean (logic) type property then <xValue> is logic. For example:

```
FrPrn:SetProperty("PDFExport", "ShowDialog", .f.)
```

If property is enumerated type property then <xValue> is character string with enumerated type representation, For example:

```
FrPrn:SetProperty("RTFExport", "HeaderFooterMode", "hfText")
```

If property is set type property then <xValue> is character string with set type representation - set is written in the brackets, elements are divided by comma. For example:

```
FrPrn:SetProperty("Designer", "Restrictions",;
      "[drDontCreateReport, drDontLoadReport, drDontSaveReport]")
```

If property is TStrings property then <xValue> is character string with the individual strings delimited by carriage returns and line feeds. Access to such property is done through sub-property Text. For example:

```
FrPrn:SetProperty("MailExport.Lines", "Text",
     "Hello!" + Chr(13)+ Chr(10) + "Look attachment!")
```

Object – "Report"

```
property Version: String
```

Read only. Return version of FastReport.

```
property DotMatrixReport: Boolean
```

Defines whether the report is dot-matrix. When setting this property to True, the report may contain dot-matrix pages and objects. Don't set this property directly. Use the "File|New..." menu item to create dot-matrix reports.

```
property IniFile: String
```

A file name or a name of the registry key for storing the FastReport environment settings. By default – "\Software\Fast Reports for Xbase (for [x]Harbour)".

`property OldStyleProgress: Boolean`

Show progress like in 2.xx version (in separate window). Default – False.

`property ScriptLanguage: String`

Script language used in a report. The following values are valid: PascalScript, C++Script, BasicScript, JScript.

`property ScriptText: TStrings`

Script text.

`property ShowProgress: Boolean`

Defines whether show progress or not.

`property OnStartReport: String`

Name of the script procedure which will be called when report starts.

`property OnStopReport: String`

Name of the script procedure which will be called when report finishes.

`property  OnRunDialogs: String;`

Name of the script procedure which will be called when dialog window shows.

Object – "Designer"

`property CloseQuery: Boolean`

Defines, whether it is necessary to ask about report saving on the designer closing.

`property DefaultScriptLanguage: String`

Default script language used in a designer. The following values are valid: PascalScript, C++Script, BasicScript, JScript.

`property DefaultFont: TFont`

Default font used in a designer for text objects. This property type has following sub-properties:

- `property Charset: Byte`

Identify the character set of the font. Each typeface (specified by the Name property) supports one or more character sets. Check the information supplied by the font vendor to determine what values of Charset are valid.

- `property Color: TColor (TColor = -$7FFFFFFF-1..$7FFFFFFF)`

Specify the color of the text characters. The low three bytes represent RGB color intensities for blue, green, and red, respectively. The value $00FF0000 represents full-intensity, pure blue, $0000FF00 is pure green, and $000000FF is pure red. $00000000 is black and $00FFFFFF is white.

- `property Height: Integer;`

Specify the height of the font in pixels.

- `property Name: String;`

Specify the typeface of the font.

- `property Pitch: enumerated - (fpDefault, fpVariable, fpFixed)`

Specifies whether the characters in the font all have the same width.

- `property Size: Integer`

Specifies the height of the font in points.

- `property Style: Set - [fsBold, fsItalic, fsUnderline, fsStrikeOut]`

Add special characteristics to characters that use the font.

`property DefaultLeftMargin: Extended`

Default left margin.

`property DefaultRightMargin: Extended`

Default right margin

`property DefaultTopMargin: Extended`

Default top margin

`property DefaultBottomMargin: Extended`

Default bottom margin

`property DefaultPaperSize: Integer`

Default paper size.

`property DefaultOrientation: enumerated - (poPortrait, poLandscape)`

Default orientation

`property OpenDir: String`

The name of a folder, from which the file is opened by default.

`property SaveDir: String`

The name of a folder, where the file is saved to by default.

```
     property Restrictions: Set - (drDontInsertObject,
drDontDeletePage, drDontCreatePage, drDontChangePageOptions,
drDontCreateReport, drDontLoadReport, drDontSaveReport,
drDontPreviewReport, drDontEditVariables, drDontChangeReportOptions,
drDontEditReportData)
```

A set of flags, which forbid different operations in the designer.

Export Objects

Purpose of export objects properties is obviously from the name of properties. All export objects have properties:

```
property ShowDialog: Boolean
property FileName: String
property ExportNotPrintable: Boolean
property UseFileCache: Boolean
property DefaultPath: String
property ShowProgress: Boolean
property OverwritePrompt: Boolean
```

Object – "PDFExport"

```
property Compressed: Boolean
property EmbeddedFonts: Boolean
property OpenAfterExport: Boolean
property PrintOptimized: Boolean
property Outline: Boolean
property Author: String
property Subject: String
property Background: Boolean
property Creator: String
property HTMLTags: Boolean
```

Object – "HTMLExport"

```
property OpenAfterExport: Boolean
property FixedWidth: Boolean
property ExportPictures: Boolean
property PicsInSameFolder: Boolean
property ExportStyles: Boolean
property Navigator: Boolean
property Multipage: Boolean
property MozillaFrames: Boolean
property UseJpeg: Boolean
property UseGif: Boolean
property AbsLinks: Boolean
property Background: Boolean
property Centered: Boolean
property EmptyLines: Boolean
```

Object – "RTFExport"

```
property ExportPageBreaks: Boolean
property ExportPictures: Boolean
property OpenAfterExport: Boolean
property Wysiwyg: Boolean
property Creator: String
property SuppressPageHeadersFooters : Boolean
property HeaderFooterMode: enumerated -(hfText, hfPrint, hfNone)
```

Object – "CSVExport"

```
property Separator: String
property OEMCodepage: Boolean
property OpenAfterExport: Boolean
```

Object – "XLSExport"

```
property ExportStyles: Boolean
property ExportPictures: Boolean
property MergeCells: Boolean
property OpenExcelAfterExport: Boolean
property Wysiwyg: Boolean
property AsText: Boolean
property Background: Boolean
property FastExport: Boolean
property PageBreaks: Boolean
property EmptyLines: Boolean
property SuppressPageHeadersFooters: Boolean
```

Objects – "BMPExport, TIFFExport, JPEGExport, GIFExport"

```
property CropImages: Boolean
property Monochrome: Boolean
```

in JPEGExport:

```
     property JPEGQuality: Integer
```

Object – "MailExport"

```
property Address: String
property Subject: String
property Lines: TStrings
property ShowExportDialog: Boolean
property FromMail: String
property FromName: String
property FromCompany: String
property Signature: TStrings
property SmtpHost: String
property SmtpPort: Integer
property Login: String
property Password: String
property UseIniFile: Boolean
property LogFile: String
```

Object – "XMLExport"

```
property ExportStyles: Boolean
property ExportPageBreaks: Boolean
property OpenExcelAfterExport: Boolean
property ShowProgress: Boolean
property Wysiwyg: Boolean
property Background: Boolean
property Creator: String
property EmptyLines: Boolean
property SuppressPageHeadersFooters: Boolean
```

Object – "DotMatrixExport"

```
property CustomFrameSet: String
property EscModel: Integer
property GraphicFrames: Boolean
property InitString: String
property OEMConvert: Boolean
property PageBreaks: Boolean
property SaveToFile: Boolean
property UseIniSettings: Boolean
```

Event:

```
OnTranslate(cString) -> cTranslatedString
```

Occurs for translate text. An event handler must return the translated value.

Object – "SimpleTextExport"

```
property PageBreaks: Boolean
property Frames: Boolean
property EmptyLines: Boolean
property OEMCodepage: Boolean
property OpenAfterExport: Boolean
```

Object – "TXTExport"

```
property ScaleWidth: Extended
property ScaleHeight: Extended
property Borders: Boolean
property Pseudogrpahic: Boolean
property PageBreaks: Boolean
property OEMCodepage: Boolean
property EmptyLines: Boolean
property LeadSpaces: Boolean
property PrintAfter: Boolean
property PrinterDialog: Boolean
property UseSavedProps: Boolean
property InitString: String
property CustomFrameSet: String
```

Object – "ODSExportt"

```
property ExportStyles: Boolean
property ExportPageBreaks: Boolean
property OpenAfterExport: Boolean
property ShowProgress: Boolean
property Wysiwyg: Boolean
property Background: Boolean
property Creator: String
property EmptyLines: Boolean
property SuppressPageHeadersFooters;
```

Object – "ODTExportt"

```
property ExportStyles: Boolean
property ExportPageBreaks: Boolean
property OpenAfterExport: Boolean
property ShowProgress: Boolean
property Wysiwyg: Boolean
property Background: Boolean
property Creator: string
property EmptyLines: Boolean
property SuppressPageHeadersFooters: Boolean
```

In summary

"FastReport for Alaska Xbase ++" and "FastReport for [x]Harbour" have a plenty of various options, properties, etc. But, by and large, the "lion's part" of these options will be necessary only by development of not ordinary, non-standard reports. In other cases all these options will remain options by default, and you will use not more than ten basic methods of frReportManager class. Developers hope that "FastReport for Alaska Xbase ++" and "FastReport for [x]Harbour" will help you in your work.