FastReport 3.0

Programmer's manual

Table of Contents

FastReport components review

FastReport 包含许多元件供报表建立、修改、导出不同的格式及強化报表功能， 让我们探究每一个 FastReport 元件栏上的元件。

TfrxReport

此为最主要的报表元件，一个 TfrxReport 元件组成一份报表。在设计时期，双击此元件可打开报表设计器(Report Designer)，此元件拥有所有载入、存盘、设计及来看报表必须的属性及方法。让我们检查 TfrxReport 提供的方法：

```
procedure Clear;
```
清除报表

```
function LoadFromFile(const FileName: String; ExceptionIfNotFound:
Boolean = False): Boolean;
```
从给予的文件载入报表。假如第二个参数等于 “True”且文件不存在，将产生例外状况，假如文件载入成功，返回值为”True ” 。

```
procedure LoadFromStream(Stream: TStream);
```
从数据流(stream)载入报表。

```
procedure SaveToFile(const FileName: String);
```
保存报表至指定的文件。

```
procedure SaveToStream(Stream: TStream);
```
保存报表至数据流(stream)。

```
procedure DesignReport;
```
进入报表设计环境。报表设计环境将嵌入在你的工程文件(要执行此功能，只要在 uses 子句加入 frxDesign 单元或在工程文件中加入“TfrxDesigner” 元件)。

```
procedure ShowReport(ClearLastReport: Boolean = True);
```
开始制作报表并输出结果显示在预览窗口。例如“ClearLastReport”参数等于“False”，报表将会加入至前一个报表的后面，否则前一个建立的报表会被清除(预置值)。

```
function PrepareReport(ClearLastReport: Boolean = True): Boolean;
```
开始制作报表，但没有显示预览窗口。参数指定方式与“ShowReport” 方法(method)相同。假如报表创建成功，此函数返回“True” 。

```
procedure ShowPreparedReport;
```
显示先前使用“PrepareReport” 所建立的报表。

```
procedure Print;
```
打印报表。

```
procedure Export(Filter: TfrxCustomExportFilter);
```
使用指定的导出过滤器(export filter)导出报表内容。因为下列的方法只提供一种服务，

在大部分的情况之下，你并不须要使用它们。在增強 FastReport 的报表功能方面，他们可能是很有用的。例如，当撰写自定义的报表元件时。

function Calc(const Expr: String): Variant;
计算"Expr" 运算式并返回结果。

function GetAlias(DataSet: TfrxDataSet): String;
返回指定数据集(data set)的别名。

function GetDataset(const Alias: String): TfrxDataset;
返回指定别名(Alias)的数据集。

procedure DoNotifyEvent(Obj: TObject; const EventName: String);
执行连接至"Obj" 物件的"EventName"事件处理程序。

procedure DoParamEvent(const EventName: String; var Params: Variant);
以任意的参数类型执行 "EventName"的事件处理程序。

procedure GetDatasetAndField(const ComplexName: String; var Dataset: TfrxDataset; var Field: String);
解析"ComplexName" 复合名称(以 DataSet."Field"表示)，并返回参照的数据集及字段名称。

procedure GetDataSetList(List: TStrings; OnlyDB: Boolean = False);
从 List 参数返回报表可用的数据集列表，假如第二个参数为 True，仅返回连接到数据库的数据集。

procedure AddFunction(const FuncName: String; const Category: String = ''; const Description: String = '');
加入使用者自定函数至报表的函数列表。详细资料请参考"The functions' list extension" 一章。

TfrxReport 元件拥有下列属性：

property EngineOptions: TfrxEngineOptions;
与 FastReport 引擎相关的属性集合。

property IniFile: String;
储存 fastReport 环境变量设定的文档或注册码的名称。

property Preview: TfrxCustomPreview;
连接到"TfrxPreview" 元件，完成的报表将显示在此元件上。假如此属性空白，报表将显示于标准的预览窗口。参见"Custom preview windows creating"一章。

property PreviewOptions: TfrxPreviewOptions;
与报表预览相关的属性。

property PrintOptions: TfrxPrintOptions;

与报表打印相关的属性。

property ReportOptions: TfrxReportOptions;
定义报表相关的属性。

property ScriptLanguage: String;
报表使用的脚本语言(Script la nguage)。

property ScriptText: TStrings;
脚本语言的内容。

property AllObjects: TList readonly;
报表内所有的物件列表（包括页定义元件)。

property DataSets: TfrxReportDataSets readonly;
报表可用的数据集列表。

property Designer: TfrxCustomDesigner readonly;
连结到报表设计元件北ū 砩杓苹肪潮黄舳。

property Engine: TfrxCustomEngine readonly;
连结报表引擎。对于要使用程序码处理报表是非常有用的，它可以自定义报表处理引擎。

property Errors: TStrings readonly;
错误清单，发生在一个或其它的进程。

property FileName: String;
定义报表的文件名称；文件名会显示在设计环境的窗口标题中。

property PreviewPages: TfrxCustomPreviewPages readonly;
定义一个连结到已完成的报表页面。它可被使用在所有地方，例如打印、存盘及导出等。

property Pages[Index: Integer]: TfrxPage readonly;
报表页面列表，其中对话框类型也包括在列表中。

property PagesCount: Integer readonly;
报表的页数。

property Script: TfsScript readonly;
连结报表的“TfsScript” 元件，经由该连接，你可以为你的报表脚本语言加入变量、类型、函数以供以后调用。更多资料参见“FastScript developer’s manual” 。

property Style: TfrxStyle;
报表式样。详细资料参见“operating with styles”相关章节。

property Variables: TfrxVariables readonly;
报表变量列表。参见“operating with variables”

FastReport 引擎的相关属性集合：

```
TfrxEngineOptions = class(TPersistent)
published
property ConvertNulls: Boolean default True;
```
转换数据库字段的“Null” 值至“0”, “False” 或空字串(依字段型态而定)。

```
property DoublePass: Boolean default False;
```
使报表进行二次处理，第一次进行资料搜集(例如报表总页数)，第二次才实际进行报表处理。

```
property MaxMemSize: Integer default 10;
```
配置报表页面缓存(Cache)的最大内存使用量(Mbytes)，当“UseFileCashe” 属性等于“True” 时特别有用。假如在建立期间耗用太多内存，已建立的报表缓存页面将会被写入缓存文件，此属性并不非常的精确，它只大約的決定內存的限制。

```
property PrintIfEmpty: Boolean default True;
```
定义是否要打印空白报表（没有打印资料的报表)。

```
property TempDir: String;
```
指定保存临时文件的目录。

```
property UseFileCache: Boolean default False;
```
定义产生的预览报表是否缓存(Cache)到文件。(见“MaxMemSize” 属性)
```
end;
```

报表预览的相关属性集合：

```
TfrxPreviewOptions = class(TPersistent)
published
property AllowEdit: Boolean default True;
```
允许或不允许编辑预览窗口中的报表。

```
property Buttons: TfrxPreviewButtons;
```
预览窗口中的可用按钮集合。

```
TfrxPreviewButtons = set of TfrxPreviewButton; TfrxPreviewButton
= (pbPrint, pbLoad, pbSave, pbExport, pbZoom, pbFind, pbOutline,
pbPageSetup, pbTools, pbEdit, pbNavigator);
```
此属性可用的值如下:

- pbPrint — 打印
- pbLoad — 载入文件
- pbSave — 存报表到文件
- pbExport — 导出
- pbZoom — 显示比例
- pbFind — 搜寻
- pbOutline — 选定报表边框

```
    pbPageSetup — 页面设定
    pbTools     — 工具
    pbEdit      — 编辑
    pbNavigator — 导航
```

上面的值你可以混合使用。

```
    property DoubleBuffered: Boolean default True;
```

预览窗口采用双缓存区模式。假如启用(预置值)，画面输出时屏幕不会有闪烁的情形，但处理速度会稍微下降。

```
    property Maximized: Boolean default True;
```

定义预览窗口是否最大化。

```
    property MDIChild: Boolean default False;
```

定义预览窗口是否为 MDIChild (给 MDI 介面使用)。

```
    property Modal: Boolean default True;
```

定义预览窗口是否为 Modal 模示。

```
    property OutlineVisible: Boolean default False;
```

定义是否显示报表的大纲。

```
    property OutlineWidth: Integer default 120;
```

定义报表大纲显示的宽度。

```
    property ShowCaptions: Boolean default False;
```

定义是否显示按钮的标题。当启动该属性时，你应该限制 Buttons 属性所显示按钮的个数，因为所有的按钮无法显示于同一画面。

```
    property Zoom: Extended;
```

预置的显示百分比率。

```
    property ZoomMode: TfrxZoomMode default zmDefault;
```

预置显示模式。可用的值如下:

```
  zmDefault   — 显示百分比率视"Zoom" 属性而定
  zmWholePage — 整页模示
  zmPageWidth — 页宽
  zmManyPages — 两页
end;
```

报表打印相关属性的集合:

```
TfrxPrintOptions = class(TPersistent)
published
    property Copies: Integer default 1;
```

预置的打印份数。
property Collate: Boolean default True;
不管校对份数。
property PageNumbers: String;
打印的页码。例如，"1,3,5-12,17-" 。
property Printer: String;
打印机名称。
property PrintPages: TfrxPrintPages default ppAll;
定义要打印的方式。可用的值如下：
ppAll - 全部
ppOdd - 奇数页
ppEven - 偶数页
property ShowDialog: Boolean default True;
是否显示打印窗口。
end;

报表相关属性的集合：

TfrxReportOptions = class(TPersistent)
published
property Author: String;
报表作者。
property CreateDate: TDateTime;
报表建立日期。
property Description: TStrings;
报表描述。
property Name: String;
报表名称。
property LastChange: TDateTime;
报表最后修改日期。
property Password: String;
报表密码。假如该属性为空白，当打开报表定义档时需要输入密码。
property Picture: TPicture;
报表图片。
property SilentMode: Boolean default False;
无声(Silent)方式。所有的错误信息将被保存在"TfrxReport.Errors"属性，而不会在屏幕上显示任何信息。

property VersionBuild: String;
property VersionMajor: String;
property VersionMinor: String;
property VersionRelease: String;
决定报表版本的属性。
end;

下列的事件定义于 TfrxReport 元件：2

property OnAfterPrint: TfrxAfterPrintEvent;
发生在处理完每个报表物件之后（打印后）。

property OnBeforePrint: TfrxBeforePrintEvent;
发生在处理完每个报表物件之前（打印前）。

property OnClickObject: TfrxClickObjectEvent;
当预览一份报表时，选取报表内的物件时触发该事件。
property OnGetValue: TfrxGetValueEvent;
当启动一份报表，发现未定义的变量时，该事件必须返回变量的值。

property OnManualBuild: TfrxManualBuildEvent;
当开始打印报表，假如此事件被启动，然后 FastReport 的引擎将被阻断（不处理），报表处理方法将交由程序员所写的程序处理。

property OnMouseOverObject: TfrxMouseOverObjectEvent;
当报表处于浏览窗口，且鼠标指针移到该物件上时触发此事件。

property OnUserFunction: TfrxUserFunctionEvent;
当执行报表的过程中，当调用的函数不存在，请使用“AddFunction”方法提供自定义函数。


TfrxDBDataset


TfrxUserDataset

数据存取元件。FastReport 使用这些元件读取及参考数据库的字段，这两个元件都源于 “TfrxDataSet” 并继承其大部分的功能。

TfrxUserDataSet 元件允许构建未连接到数据库的报表，而由其它来源接收数（据如：数列、文件等）。在此同时，程序员仅需提供浏览此数据集的功能，资料接收并非由此元件执行，而是用其它的方法（例如，经由 “TfrxReport.OnGetValue”事件）。

TfrxUserDataSet 元件有下列的属性：

property RecNo: Integer readonly;
目前记录编号，首笔的记录编号是“0”

property Enabled: Boolean default True;
定义此元件是否可在 designer 里面使用。

property RangeBegin: TfrxRangeBegin default rbFirst;
数据导航（navigation）的起点。下列的值可以使用：
rbFirst - 从数据的第一笔记录开始。
rbCurrent - 从当前的记录开始。

property RangeEnd: TfrxRangeEnd default reLast;
数据导航(navigation)的起点。下列的值可以使用：

reLast – 直到数据结束。
ReCurrent – 直到目前的记录。
reCount – 依"RangeEndCount" 属性而定。

property RangeEndCount: Integer;
数据集中的数据个数，此功能只在"RangeEnd"属性等于 reCount 有效。
property UserName: String;
符号名称。在报表设计环境(Designer)下，将被显示于 DataSet 的下方。

property OnCheckEOF: TfrxCheckEOFEvent;
TfrxCheckEOFEvent = procedure(Sender: TObject; var Eof: Boolean) of object; 此事件在数据集的尾端时，Eof 参数将返回 True。

property OnFirst: TNotifyEvent;
数据集移至第一笔的位置时，会触发此事件。

property OnNext: TNotifyEvent;
数据集移至下一笔的位置时，会触发此事件。

property OnPrior: TNotifyEvent;
数据集移至上一笔的位置时，会触发此事件。

TfrxDBDataSet 元件用来连接以 TDataSet, TTable 及 TQuery 为基类的数据库元件，有关数据的导航及字段的参考都是自动的，程序员不需特殊的设定。除上述属性外，该元件有下列的属性：

property CloseDataSource: Boolean default False;
报表创建完成后，关闭数据库。
property OpenDataSource: Boolean default True;
在报表创建之前打开数据库。
property FieldAliases: TStrings;
数据集字段的符号名称(别名)。
property DataSet: TDataSet; property DataSource: TDataSource;
连结至 TDataSet 或 TDataSource 类型的元件。
property OnClose: TNotifyEvent;
当关闭数据集时触发此事件。
property OnOpen: TNotifyEvent;
当打开数据集时触发此事件

TfrxDesigner

TfrxDesigner 元件是报表设计器，当使用此元件，你的工程文件就可以使用报表设计器，此元件它只包含一些报表设计器的设定，当加入"frxDesign" 单元到 uses 清单中，就表明与报表设计器连接上了。

此元件包含下列的属性：

```
    property CloseQuery: Boolean default True;
```
定义结束设计报表是否询问储存报表之用。
```
    property OpenDir: String;
```
打开报表的预置数据目录。

```
    property SaveDir: String;
```
储存报表的预置数据目录。

```
    property Restrictions: TfrxDesignerRestrictions;
```
报表设计环境下，限制不同的报表操作标识(flag)，此标识包含单一或混合数据值：

```
    drDontInsertObject        - 禁止插入物件
    drDontDeletePage          - 禁止删除页面
    drDontCreatePage          - 禁止建立新的页面
    drDontChangePageOptions   - 禁止修改页面属性
    drDontCreateReport        - 禁止建立新报表
    drDontLoadReport          - 禁止载入报表
    drDontSaveReport          - 禁止储存报表
    drDontPreviewReport       - 禁止预览报表
    drDontEditVariables       - 禁止编辑变量
    drDontChangeReportOptions - 禁止修改报表属性
```

```
    property OnLoadReport: TfrxLoadReportEvent;
TfrxLoadReportEvent = function(Report: TfrxReport): Boolean of
object;
```
此事件发生在载入报表之时。利用此事件，你可以从数据库载入报表。

```
    property OnSaveReport: TfrxSaveReportEvent;
TfrxSaveReportEvent = function(Report: TfrxReport;
SaveAs:Boolean): Boolean of object;
```
此事件发生在储存报表之时。利用此事件，你可以将报表储存于数据库中。
```
    property OnShow: TNotifyEvent;
```
此事件发生在启动报表设计环境时。

TfrxPreview

此元件专供建立自定义报表合预览窗口使用。

```
procedure AddPage;
```
加入空白页面到报表末端。

```
procedure DeletePage;
```
删除当前页。

```
procedure Print;
```

打印报表。

```
procedure LoadFromFile;
```
显示文件载入窗口。

```
procedure LoadFromFile(FileName: String);
```
载入指定的文件。

```
procedure SaveToFile;
```
显示文件储存窗口。

```
procedure SaveToFile(FileName: String);
```
储存文件到指定的文件名称。

```
procedure Edit;
```
载入当前页至设计模式供编辑使用。

```
procedure Export(Filter: TfrxCustomExportFilter);
```
使用指定的导出过滤器导出报表。

```
procedure First;
```
第一页。

```
procedure Next;
```
下一页。

```
procedure Prior;
```
上一页。

```
procedure Last;
```
最后一页。
```
procedure PageSetupDlg;
```
显示页面设定窗口。

```
procedure Find;
```
显示文字搜寻窗口。

```
procedure FindNext;
```
继续找下一个。

```
procedure Cancel;
```
取消创建报表。

```
procedure Clear;
```
清除报表。

你可以使用下列属性：

```
property PageCount: Integer readonly;
```

报表页数。

property PageNo: Integer;
目前报表页码(起始值为 1)。要移至其它页面，指定此属性的值即可。

property Tool: TfrxPreviewTool;
选取工具。

property Zoom: Extended;
显示比例，"1" 代表 100% 。

property ZoomMode: TfrxZoomMode;
显示模式，可以的显示模式如下：
zmDefault - 预置值，显示比例根据"Zoom" 属性而定
zmWholePage - 整页模式
zmPageWidth - 与页面宽度相同
zmManyPages - 一屏显示多页

property OutlineVisible: Boolean;
是否显示报表大纲(树状结构)。

property OnPageChanged: TfrxPageChangedEvent;
目前页面要改变时，此事件将被触发。

TfrxBarcodeObject
TfrxOLEObject
TfrxChartObject
TfrxRichObject
TfrxCrossObject
TfrxCheckBoxObject
TfrxGradientObject

可在报表内部使用的物件，这些元件自己没做任何事情，它们会自动将元件的单元加入 uses 清单，加入你打算打开一份报表，此功能会自动被加入报表，未包括此物件至工程文件的话，打开报表时将会发生错误。

TfrxDialogContols
附加项(add-in) 物件的集合，可用于报表内的对话窗口，它包含下列元件：button,edit box, list box 等。此元件自己不会执行任何事，加入此元件"frxDCtrl"将会自动加入"Uses"清单。

TfrxBDEComponents
BDE 数据库元件，数据库界面采用 BDE(Borland Database Engine)时，工程文件必须加入此元件。

TfrxADOComponents

ADO 数据库元件，数据库连接采用 ADO(Advance Data Object)时，工程文件必须加入此元件。

TfrxIBXComponents

Interbase 数据库元件，数据库采用 IBX 连接 Interbase 后台数据库时，工程文件必须加入此元件。

上述数据库元件，可被使用在报表的对话窗口上，它包括下列的元件：“Database”，“Table” 及“Query” 。这些元件本身并不做任何事；它们只会将元件隶属的单元自动加入 uses 清单中。

Working with TfrxReport component

Loading and saving a report 存取报表

报表定义表格与工程文件的表格储存在同一个文件(.DFM)，在大部分的情况下，并不需要额外的操作步骤，因此载入报表便相当简单。假如你要将报表储存在文件或数据库的 BLOB 字段，你必须使用“TfrxReport”提供的方法来载入及储存报表。

```
function LoadFromFile(const FileName: String;
ExceptionIfNotFound: Boolean = False): Boolean;
```

载入指定的报表，假如第二个参数的值等于”True ” 且指定的文件不存在，然后它会产生一个例外。假如文件载入成功，它返回“True”。

```
procedure LoadFromStream(Stream: TStream);
```

从数据流(stream)载入报表。

```
procedure SaveToFile(const FileName: String);
```

储存报表至指定的文件名。

```
procedure SaveToStream(Stream: TStream);
```

储存报表至数据流(stream) 。

报表预置的后缀名称为“FR3”。

范例：

```
frxReport1.LoadFromFile('c:\1.fr3');
frxReport1.SaveToFile('c:\2.fr3');
```

Designing a report 设计报表

通过“TfrxReport.DesignReport”方法调用报表设计器(report designer)，要具有设计报表的功能，你必须在工程文件中加入“TfrxDesigner”

元件，或在 uses 加入“frxDesgn”单元。
范例：
```
frxReport1.DesignReport;
```

Running a report 执行报表

应用下列两个“TfrxReport”元件的方法启动报表：

```
procedure ShowReport(ClearLastReport: Boolean = True);
```
启动报表并显示结果在浏览窗口。假如“ClearLastReport”参数等于“False”，然后报表将会清前一份报表的末端，否则前一份报表的内容将会被清除ど柚。

```
function PrepareReport(ClearLastReport: Boolean = True):
Boolean;
```
启动报表，但不开启预览窗口，参数用法同“ShowReport”方法，假如报表建立成功，它返回“True”。
在大部分的情况下，采用第一种方法比较方便，当报表建立的过程中，它会立刻显示预览窗口。当我们要把报表加入上一份报表的后面时，“ClearLastReport”参数是非常方便的技巧在批次报表打印时特别有效)。
范例：
```
frxReport1.ShowReport;
```

Previewing a report 预览报表

在报表预览窗口显示报表有两种方式：不管是调用“TfrxReport.ShowReport”方法(前面已提及)或使用“TfrxReport.ShowPreparedReport”方法。在第二种状况，报表创建的过程不会执行，但是报表的结果会显示于屏幕。这个意思是说，你应该使用“PrepareReport”方法创建报表或载入先前已经建立的报表。
范例：
```
  if  frxReport1.PrepareReport then
      frxReport1.ShowPreparedReport;
```
在这个案例中，报表创建完成在先，然后显示报表于浏览窗口。创建大型的报表可能耗费许多时间，那也是为什么使用“ShowReport”方法会比“PrepareReport/ShowPreparedReport” 来的好的原因，我们指定“TfrxReport.PreviewOptions” 属性，指定预览的参数。

Printing a report 打印报表

在大部份的情况，你会从预览窗口打印报表。要手动打印报表，你应该使用“TfrxReport.Print”方法，
例如：
```
    frxReport1.Print;
```
在此同时，你可以设定打印对话窗口的参数。你可以从“TfrxReport.PrintOptions”属性指定打印的预置值及取消显示打印窗口。

Loading and saving a finished report 存取已完成的报表

它可以从预览窗口执行，这也可以手动的用“TfrxReport.PreviewPages”方法执行：

```
function LoadFromFile(const FileName: String;
ExceptionIfNotFound: Boolean = False): Boolean;
procedure SaveToFile(const FileName: String);
procedure LoadFromStream(Stream: TStream);
procedure SaveToStream(Stream: TStream);
```

指定参数与 TfrxReport 对应的方法类似，报表文件后缀名必须是”FP3” 。

例：

```
frxReport1.PreviewPages.LoadFromFile('c:\1.fp3');
frxReport1.ShowPreparedReport;
```

注：当已完成报表载入后，预览报表必须通过 “ShowPreparedReport”方法来执行。

Exporting a report 导出报表

它可从预览窗口执行，此项功能也可以用“TfrxReport.Export”方法来执行，在此方法的参数中，你必须指定要使用的导出过滤元件：

如：

```
frxReport1.Export(frxHTMLExport1);
```

导出过滤元件必须是可用的（你必须将它放入工程文件的表单上）且设定正确。

Creating a custom preview window 建立自定义预览窗口

FastReport 显示报表于标准的预览窗口。假如为了某种理由它无法满足你，你可以使用自定义的预览窗口。为此，FastReport 的“TfrxPreview” 元件于是诞生了，要显示报表，这个元件必须连接到“TfrxReport.Preview”属性。

Building a composite report (batch printing) 建立复合报表(批量打引)

在某些情况下，我们必须一次打印数份报表，或封装及实现多份报表于同一个预览窗口。要执行这项工作，在 FastReport 中有多个工具，允许建立一份新的报表，置于另一份已存在的报表末端，«TfrxReport.PrepareReport»方法有此«ClearLastReport»????, ????«True»，此参数定义是否清除前一份已建立的报表。下列的程序码示范如何从两个报表定义文件，批次建立一份报表：

```
frxReport1.LoadFromFile('1.fr3');
frxReport1.PrepareReport;
frxReport1.LoadFromFile('2.fr3');
frxReport1.PrepareReport(False);
frxReport1.ShowPreparedReport;
```

我们载入第一个报表并在后台处理报表(不显示)，然后我们载入第二份报表到同一

«TfrxReport»物件，并置参数 «ClearLastReport» 的值为假(False)建立报表。此功能允许第二份报表的结果接在第一份报表的后面。最后，我们将两份报表显示在同一个预览窗口中。

Numbering of pages in a composite report 复合报表中的页数

你可以使用 «Page», «Page#», «TotalPages» 及 «TotalPages#» 系统变量显示页码或总页数。在复合式报表，这些变量代表的意义如下：

Page -目前报表的页码
Page# -批次报表的页码
TotalPages -目前报表的总页数(报表必须设定为 two-pass)
TotalPages# -批次报表的总页数

Combination of pages in a composite report 复合报表的合并页

如上所述，当打印时，报表设计的属性 «PrintOnPreviousPage»可以利用前一页的可用空间打印下一页的内容。在复合报表中，它允许你在前一份报表最后一页的可用空间上建立一份新的报表，要执行此功能，必须在每一份连续报表的第一个设计页面启动属性«PrintOnPreviousPage»。

Interactive reports 交互式报表

在交互式报表中，我们可以在预览窗口定义任何报表物件对鼠标按下的反应。例如，使用者选择数据列，然后做一份新的报表，显示选取列的明细数据。任何报表都可以成为交互式报表，要执行此操作，你只需建立 TfrxReport.OnClickObject 事件处理程序。

下面是此事件处理的范例：

```
   procedure TForm1.frxReport1ClickObject(Page: TfrxPage;
View: TfrxView; Button: TMouseButton; Shift: TShiftState;
  var Modified: Boolean);
   begin
   if View.Name = 'Memo1' then
    ShowMessage('Memo1 contents:' + #13#10 +
    TfrxMemoView(View).Text);
       if View.Name = 'Memo2' then
         begin

             TfrxMemoView(View).Text := InputBox('Edit', 'Edit
             Memo2 text:', TfrxMemoView(View).Text);
             Modified := True;
        end;
  end;
```

在 «OnClickObject» 事件处理程序中，你可以执行下列工作：

-修改物件或页面的内容，但前提是«Modified»属性必须被指定。
-调用 «TfrxReport.PrepareReport» 方法重新建立报表。

在此范例中，点选 «Memo1» 物件将显示此物件的内容，当点选«Memo2,»将显示 Dialog 窗口，物件的内容可于此窗口内被修改。设定«Modified» 标识为«True» 允許保留及显示修改后的内容。

同样的方法，它也可被定义为一个单击，有不同的反映。例如，执行一个新的报表。下列的注释是必要的。在中，在一个预览窗口仅显示一个报告，由一个元件组成（不像 FastReport 2.x 版）。这就是运行一个报告，其它 TfrxReport 物件，必须删除的原因。

要给使用者按下物件一个提示，我们可以在鼠标指针移至物件上方时变更鼠标指针显示。要达到此目的，请在报表设计环境下，选取想要的物件，并设定不同于预置的 Cursor 属性即可。

可单击(clickable)物件有许多详细的定义规则。在简单的报表中，可以随意定义目录(contents)中物件的名称。可是，在复杂报表的报表中却不行。 例如，创建逐条的报表在有序的数据中。一个用户单击目录为“12”的«Memo1»物件。在该物件上数据将怎样排序？这就是你该明确知道主键值排列顺序的原因。
FastReport 可分配一个包含任何数据(我们的例子中是主键值的数据)的字符串到任何报表的物件。此字符串储存在 «TagStr» 属性中。

让我们以 FastReport 的范例(FastReportDemo.exe 内的【Simple list】)来说明，这是一家公司的客户明细，打印的内容包括【client’s name】，【address】 【contact person】 等字段，数据来源是 DBDEMOS 演示数据库的“Customer.db”数据表，该数据表有一个主键值【CustNo】字段，它并未输出在报表。我们的工作是决定点下的物件是哪一笔记录，然后取得该记录的主键值。要执行该工作，必须在 Master Data 区域所有物件的«TagStr» 属性

在报表建立期间， «TagStr» 属性的内容会以相同的方法被重新计算，当字符物件的内容被计算，所有用到此变量的值将会被取代。
假如主鍵值(Primary Key) 是复合字段(它包含多个字段)， «TagStr» 属性的内容可以是下列的写法：
[Table1."Field1"];[Table1."Field2"]
建立报表后， «TagStr» 属性的值包含'1000;1' ， 这样就不难取得键值。

Access report objects from a code 存取报表物件程序代码

FastReport 的物件，例如报表页面(page)、数据带(band)、备注(memo)物件，并非直接可由你的程序代码来存取的，这个意思是你不能直接使用物件的名称来存取，例如，从你的表格(Form)上面存取按钮(Button)，要存取某个物件，必须透过«TfrxReport.FindObject» ; 方法的协助而取得：

```
var
     Memo1 :  TfrxMemoView;
     Memo1 := frxReport1.FindObject('Memo1') as TfrxMemoView;
```

执行上述程序代码之后，我们可以取得物件的属性或执行物件的方法，另外，你也可以使用属性«TfrxReport.Pages» 取得报表页面的值：

```
Var
Page1: TfrxReportPage;
begin Page1 := frxReport1.Pages[0] as TfrxReportPage;
 :
end;
```

Creating a report form from a code 使用程序代码建立报表

你必须使用报表设计器建立大部分的报表，这是一个定律，不过，在某些案例(例如，当报表的格式尚未建立)，我们必须透过程序代码的协助手动建立报表。

手动建立报表，应依序执行下列的步骤：

-清除报表元件的内容
-加入报表的数据来源
-加入报表的版面(页面)
-加入页面的报表区域
-设定区域的属性，且将它连接至数据
-加入物件至每一个区域
-设定物件的属性，且将它连接至数据

让我们来看建立简单的清单【list】 报表，假设我们有下列的元件：TfrxReport 及 frxDBDataSet1: TfrxDBDataSet （ 最后一个元件透过 DBDEMOS 连接【Customer.db】 数据表）。我们的报表包含一页，里面有«Report Title» 及 «Master Data» 区域，在 «Report Title» Page: 21 区域上面有一个物件显示 "Hello FastReport!" ，且 «Master Data» 包含一个物件，上面有一个物件连接至 "CustNo" 字段。

```
Var
  Page: TfrxReportPage;
  Band: TfrxBand;
  DataBand: TfrxMasterData;
  Memo: TfrxMemoView;

{ 清除报表元件的内容 }
frxReport1.Clear;

{ 加入数据集至报表}
frxReport1.DataSets.Add(frxDBDataSet1);

{ 加入页面 }
Page := TfrxReportPage.Create(frxReport1);
```

```
{ 建立唯一的页面名称 }
Page.CreateUniqueName;

{ 设定预置的字段、纸张大小 }
Page.SetDefaults;

{ 修改纸张的方向}
Page.Orientation := poLandscape;

{ 加入报表标题区域}
Band := TfrxReportTitle.Create(Page);
Band.CreateUniqueName;
{ it is sufficient to set the «Top» coordinate and height for
a band }

{ 坐标的单位采用象素(pixels) }
Band.Top := 0;
Band.Height := 20;

{加入物件至报表标题区域}
Memo := TfrxMemoView.Create(Band);
Memo.CreateUniqueName;
Memo.Text := 'Hello FastReport!';
Memo.Height := 20;

{ 物件自动调整大小与区域同宽 }
Memo.Align := baWidth;

{ 加入主数据区域 }
DataBand := TfrxMasterData.Create(Page);
DataBand.CreateUniqueName;
DataBand.DataSet := frxDBDataSet1;

{ Top 坐标必须大于前一个加入区域的top + height}
DataBand.Top := 100;
DataBand.Height := 20;

{ 加入物件至主数据区域}
Memo := TfrxMemoView.Create(DataBand);

Memo.CreateUniqueName;
{ 连接至数据 }
Memo.DataSet := frxDBDataSet1;
Memo.DataField := 'CustNo';
Memo.SetBounds(0, 0, 100, 20);

{ 调整文字至物件的右边界 }
Memo.HAlign := haRight;

{ 显示报表 }
```

```
frxReport1.ShowReport;
```

让我们解释某些细节。

所有被报表所使用的数据来源，都必须被加入数据来源清单。在我们的案例中，这个动作由程序代码«frxReport1.DataSets.Add(frxDBDataSet1)» 生成，否则，报表无法工作。

并不需要调用 Page.SetDefaults，因为在此案例报表页面的格式为 A4，边界为 0mm， SetDefaults 将導致页面边界设为 10mm，至其他的设定则由印表機的预置值所取代。

当加入区域至报表页面，你应该确认它并未与其他的区域重叠，要执行此工作，你可以设定«Top» 及 «Height» 坐标，而不需要修改«Left» 及 «Width» 坐标，因为区域与所在的页面的宽度一定相同。有一点必须特别说明的，页面上区域位置出现的次序是非常的重要，因为输出的顺序是以报表设计时的次序为准。

物件的坐标和大小是以象素(Pixels)计算，因为所有物件的«Left,» «Top,» «Width,» 及 «Height» 属性均拥有 «Extended» 类型，你可以指出非整数的值，下列的常数被定义用来转换象素(Pixel)至厘米(cm)或英吋(inch)：

```
fr01cm = 3.77953;
fr1cm = 37.7953; fr01in = 9.6; fr1in = 96;
```

例如，数据带(Band)的高度等于 5mm 可以设定如下：

```
Band.Height := fr01cm * 5; Band.Height := fr1cm * 0.5;
```

Creating a dialogue form from a code 建立对话窗口程序代码

如同我们所知的，报表可以包含对话窗口。下列的范例示范如何建立含有«OK» 按钮的对话窗口。

```
{ 为了在报表内使用对话窗口，Delphi 程序必须引用 frxCtrl 单元 }
uses frxDCtrl;
var
 P age: TfrxDialogPage;
 Button: TfrxButtonControl;

{ 新增页面 }
Page := TfrxDialogPage.Create(frxReport1);

{ 建立唯一的页面名称 }
Page.CreateUniqueName;

{ 设定大小 }
Page.Width := 200;
Page.Height := 200;
```

```
{ 设定窗口显示位置 }
Page.Position := poScreenCenter;

{ 加入按钮(button) }
Button := TfrxButtonControl.Create(Page);
Button.CreateUniqueName;
Button.Caption := 'OK';
Button.ModalResult := mrOk;
Button.SetBounds(60, 140, 75, 25);

{ 显示报表 }
frxReport1.ShowReport;
```

Modifying report page's properties 修改报表页面属性

有时我们必须从程序代码修改报表页面的设定(例如，修改页面对齐方式或大小)。TfrxReportPage 类别包括下列定义报表页面大小的属性：

```
property Orientation: TPrinterOrientation default poPortrait;
property PaperWidth: Extended;
property PaperHeight: Extended;
property PaperSize: Integer;
```

属性 «PaperSize» 设定纸张格式。纸张格式必须定义在 Windows.pas 中的标准值之一(例如，DMPAPER_A4)??????????,FastReport 会自动填入 «PaperWidth» 及«PaperHeight» ?属性值(纸张大小以 mm 计算)。设定报表格式为 DMPAPER_USER (或 256) 值，将代表自定义纸张大小。在这个案例中，属性 «PaperWidth» 及 «PaperHeight» 的值必须自行指定。

下列的范例示范如何修改报表第一页的参数 (假设我们已经产生报表)：

```
var
 Page: TfrxReportPage;
{ 报表的第一页 }
Page := TfrxReportPage(frxReport1.Pages[0]);

{ 修改页面大小 }
Page.PaperSize := DMPAPER_A2;

{ 修改纸张方向 }
Page.Orientation := poLandscape;
```

Report construction with the help of a code

FastReport 引擎通常负责报表的创建,某些时候它必须建立一份 FastReport 引擎无法处理的非标准格式报表，在这种状况下，我们可以使用 «TfrxReport.OnManualBuild»事件撰写程序代码来建立报表，FastReport 引擎交由此事件来处理报表，在此的同时，报表建立自动变成下列的方式：

FastReport 引擎：

-报表的准备工作(script, data sources initialization, bands' tree

forming)
-所有的报表运算(aggregate functions, event handlers)
-跳页/栏的格式(automatic showing a page/column header/footer, report title/summary)
-其他例行性的工作

处理程序(Handler):
-以特定的次序打印 band 的內容

«OnManualBuild» 事件处理程序的实体是发出打印特定区域的命令给 FastReport 引擎，此引擎本身将执行下列工作：建立新的页面，执行报表的脚本(Scripts)等等。

此引擎为 «TfrxCustomEngine» 类别，连接此类别的是 «TfrxReport.Engine» 属性。
procedure NewColumn;
建立新栏，假如目前是最后一栏，它将自动建立新页。
procedure NewPage;
建立新页。

procedure ShowBand(Band: TfrxBand); overload;
显示数据带(Band)。

procedure ShowBand(Band: TfrxBandClass); overload;
显示指定类型的数据带(Band)。

function FreeSpace: Extended;
返回页面的可打印空间(以象素表示)，在下一个区域出现之后，此值会递减。

property CurColumn: Integer;
返回/设定目前的字段编号

property CurX: Extended;
返回/设定目前 X 坐标的位置。

property CurY: Extended;
返回/设定目前 X 坐标的位置。当下一个 Band 输出之后，此值将会變小。

property DoublePass: Boolean;
定义报表是否进行两次处理。

property FinalPass: Boolean;
判断目前是否为最后一次处理。

property FooterHeight: Extended;
返回页尾的高度。

```
property HeaderHeight: Extended;
```
返回页首的高度。

```
property PageHeight: Extended;
```
返回可打印页面的高度。

```
property PageWidth: Extended;
```
返回可打印页面的宽度。

```
property TotalPages: Integer;
```
返回製作完成报表的总页数(仅在二次处理报表的第二次处理时使用)。

让我们来看一个简单的处理程序，一个报表有两个位连接数据的«Master Data»区域，此处理程序以交錯的次序显示区域，每个区域显示六次，在六个区域之后，出现一小段的分隔(Gap)。

```
 var
i: Integer;
Band1, Band2: TfrxMasterData;

{ find required bands }
Band1 := frxReport1.FindObject('MasterData1') as TfrxMasterData;
Band2 := frxReport1.FindObject('MasterData2') as TfrxMasterData;

for i := 1 to 6 do begin
 { lead/deduce bands one after another }
 frxReport1.Engine.ShowBand(Band1);
 frxReport1.Engine.ShowBand(Band2);
 { make a small gap }

if i = 3 then
    frxReport1.Engine.CurY := frxReport1.Engine.CurY + 10;
end;
```

下个范例显示两个紧邻的組区域。

```
var
  i, j: Integer;
 Band1, Band2: TfrxMasterData;
 SaveY: Extended;
Band1 := frxReport1.FindObject('MasterData1') as TfrxMasterData;
Band2 := frxReport1.FindObject('MasterData2') as TfrxMasterData;

SaveY := frxReport1.Engine.CurY;
for j := 1 to 2 do
begin

for i := 1 to 6 do
begin
 frxReport1.Engine.ShowBand(Band1);
 frxReport1.Engine.ShowBand(Band2);
 if i = 3 then
 frxReport1.Engine.CurY := frxReport1.Engine.CurY + 10;
end;
```

```
frxReport1.Engine.CurY := SaveY;
 frxReport1.Engine.CurX := frxReport1.Engine.CurX + 200; end;
```

Printing an array 打印数列

主要的范例程序代码位于 «FastReport Demos \PrintArray» 目录，让我们为您解释数个细节。

要打印数列，我们使用含有一个 «Master Data» 区域的报表，这些区域将被显示多次，次数与数列的元素一样。要达到此功能，放置«TfrxUserDataSet» 元件在 Form 上面，并设定它的属性：

RangeEnd := reCount

RangeEndCount := 数列的元素个数

做完上述设定之后，我们连接数据区域至 «TfrxUserDataSet» 元件。要展现数列的元素，请放置一个内容为[element]的文字物件至«Master Data» 区域， «element» 变量的值必须由«TfrxReport.OnGetValue» 事件提供。

Printing a TstringList 打印 TStringList

主要范例的程序代码位于«FastReport Demos \PrintStringList» 目录，此方法与 PrintArray 的方法相同。

Printing a file 打印文件

主要范例的程序代码位于«FastReport Demos\PrintFile» 目录，让我们解释它的细节。

为了打印，你应该使用一个拥有 «Master Data» 区域的报表，此区域仅会被打印一次(要执行此功能，它将会连接至含有一笔数据的 DataSource，从 DataSource 清单中选取”SingleRow”， 允许区域的缩放«Stretch» 及分割«AllowSplit»。这意味著，区域会视区域上所有物件所需的空间大小而自动缩放，假如区域未发现足够空间的話，有一部份的内容将会印在下一页。

文件内容可透过物件(内含[file]变量)的 «Text» 属性展现，此变量就如同前一个范例，由«TfrxReport.OnGetValue» 事件提供变量的值，缩放可由物件的«Stretch» 项目或«StretchMode» 属性 = smActualHeight 来设定。

Printing a TStringGrid 打印栅格

最初的范例程序代码位于 «FastReport Demos \PrintStringGrid» 目录，让我们解释某些细节。

«TStringGrid» 元件以多列与行的界面显示数据表的内容，这意味者报表的缩放不只是在高度，连宽度也是一样。要打印这个元件的内容，使用 «Cross-tab» 物件(? «TfrxCrossObject» 元件加入工程文件才可以使用)。此物件的责任是在未知数据列数及栏数的情况下打印数据表，此物件有两个版本：

«TfrxCrossView» ???打印使用者提供的数据，而 «TfrxDBCrossView» ??? 打印来自数据库的数据。

让我们使用 TfrxCrossView ，此物件必须事先设定好，要执行此功能，让我们进入报表设计环境并环境物件调用物件编辑器，重复的设定列与栏的标题及数据表字段数目。在我们的案例中，所有的值都必须是«1»。在我们的范例，列与栏的标题及列与栏的和总值都必须不能修改。必须在 StringGrid 的 «TfrxReport.OnBeforePrint» 事件填入物件的值，而值须以 «TfrxCrossView.AddValue» 方法填入，它的参数填入后：组成一列、一栏及项目值的索引(因为一个物件包含许多值在一个 cell 字段)。

Printing TTable and TQuery

最初的范例出现在 «FastReport's Demos\PrintTable» 目录，工作的原理是与 TStringGrid 相同的。在这个案例中，列的索引是有次序的编号，栏的索引则是数据表字段名称的索引编号，cell 的值则是数据表的字段内容，Cell 项目在«Cross-tab» ?????? ?不能编辑的，这一点是非常的重要，且数据表的标题也是一样。

Working with a list of variables

使用者可以在报表中指定一个或数个变量，变量的值或运算式参考到变量时将会被自动计算，可以被指定到每个变量。变量经由数据树(Data Tree)窗口插入至报表，而经常使用的复杂运算式以变量的别名取代会是非常的方便。
当使用报表变量时，我们必须引用 "frxVariables" 单元，变量经由 "TfrxVariable" 类别来展现。

```
TfrxVariable = class(TCollectionItem)
published
  property Name: String;
```

变量名称

```
  property Value: Variant;
```

变量值

```
end;
```

变量清单可由"TfrxVariables"类别来表示，它包含所有存取此变量清单的方法。

```
TfrxVariables = class(TCollection)
public
function Add: TfrxVariable;
```

加入变量至变量清单的末端

```
function Insert(Index: Integer): TfrxVariable;
```

加入变量至指定的清单位置

```
function IndexOf(const Name: String): Integer;
```
返回变量名称的索引

```
procedure AddVariable(const ACategory, AName: String; const
AValue: Variant);
```
加入变量至指定的类别

```
procedure DeleteCategory(const Name: String);
```
删除类别及此类别下的所有变量。

```
procedure DeleteVariable(const Name: String);
```
删除一个变量

```
procedure GetCategoriesList(List: TStrings; ClearList: Boolean
= True);
```
返回类别清单

```
procedure GetVariablesList(const Category: String; List:
TStrings);
```
返回指定类别中的所有变量清单

```
property Items[Index: Integer]: TfrxVariable readonly;
```
变量清单

```
property Variables[Index: String]: Variant; default;
```
变量值
```
end;
```

假如变量清单很长，经由变量类别的区分是非常方便的。例如，当有下列的变量清单：
Customer name
Account number
in total
total vat
也可以用下列方式表示：
Properties
Customer name
Account number
Totals
In total
total vat

有下列的限制：

-至少必须建立一个变量类别
-类别是第一层，变量是第二层
-不可有巢狀的类别
-变量的名称必须是唯一

Creating a list of variables 建立变量清单

报表变量储存在“TfrxReport.Variables”属性中，要手动建立变量清单，必须执行下列的步骤：

-清除变量清单
-建立类别
-建立变量
-重复步骤 2, 3 建立其他的类别

Clearing a list of variables 清除所有变量

透过 “TfrxVariables.Clear” 方法的协助，我们可以清除所有的变量：

```
frxReport1.Variables.Clear;
```

Adding a category 新增变量分类

至少必须建立一个变量分类。类别和变量存放在同一个清单(List)，类别与变量最大的不同在于类别名称的第一个字元是空白。所有的变量紧接在类别之后，并被视为属于此一类别。

新增类别的方式有两种方法：

```
frxReport1.Variables[' ' + 'My Category 1'] := Null;
```

或

```
var
  Category: TfrxVariable;
  Category := frxReport1.Variables.Add;Category.Name := ' ' +
'My category 1';
```

Adding a variable 新增变量

只有在变量类别已经新增之后，才可以新增变量。所有的变量清单的位置紧接在类别之后，并被视为属于此一类别，然变量的名称不只在类别是唯一的，且所有的变量名称都不能相同。

有多种方法可加入变量至变量清单：

```
frxReport1.Variables['My Variable 1'] := 10;
```

此方法可新增变量或修改已存在变量的值。

```
var
    Variable: TfrxVariable;
    Variable := frxReport1.Variables.Add;
    Variable.Name := 'My Variable 1';
    Variable.Value := 10;
```

两种新增变量的方法可加入变量至清单的末端，假如变量要加入至指定的位置，请使用 “Insert” 方法：

```
var
    Variable: TfrxVariable;
    Variable := frxReport1.Variables.Insert(1);
    Variable.Name := 'My Variable 1';
    Variable.Value := 10;
```

假如变量要加至指定的分类，请使用“AddVariable”方法：

```
frxReport1.Variables.AddVariable('My Category 1', 'My Variable 2', 10);
```

Deleting a variable 删除变量

```
frxReport1.Variables.DeleteVariable('My Variable 2');
```

Deleting a category 删除变量类别

删除变量类别及其下所有的变量，使用下列程序代码：

```
frxReport1.Variables.DeleteCategory('My Category 1');
```

Modifying the variable’s value 修改变量的值

有两种方法可修改变量的值：

```
frxReport1.Variables['My Variable 2'] := 10;
```

或

```
Var
    Index: Integer;
    Variable: TfrxVariable;

    { search for the variable }
    Index := frxReport1.Variables.IndexOf('My Variable 2');
    { if it is found, change a value }
    if Index <> -1 then
    begin
        Variable := frxReport1.Variables.Items[Index];
        Variable.Value := 10;
    end;
```

Working with styles

类型是一个元素，它具有名称、属性及决定设计属性，例如颜色(color)、字型(font)及外框(frame)。类型决定报表物件应该如何显示，这个物件像是TfrxMemoView有Style属性，此处必须设定为类型的名称，当套用一个值到此属性，此类型的设计属性将套用到此物件。

类型的集合包含数个类型，它参考到同一份报表。“TfrxReport”元件提供“Styles”属性，它参考到物件的“TfrxStyles”类型，类型的集合也必须具有名称，类型的集合代表整份报表的设计(展现方式)。

类型库包括数个类型的集合，透过类型库的协助，我们可以方便的管理类型的

集合。
TfrxStyleItem 代表类型。
```
TfrxStyleItem = class(TCollectionItem)
public
property Name: String;

Style name.
property Color: TColor;
Background color.
property Font: TFont;
Font.
property Frame: TfrxFrame;
Frame.
end;
```

TfrxStyles 类别代表类型的集合，它是由读取(reading) 、储存(saving)、新增(adding)、删除(deleting) 及搜寻等操作的方法(method)所組成。类型集合预置的后缀名为".FS3" 。

```
TfrxStyles = class(TCollection)
public
constructor Create(AReport: TfrxReport);
```

建立类型集。
```
function Add: TfrxStyleItem;
```
加入新类型。

```
function Find(const Name: String): TfrxStyleItem;
```
返回指定名称的类型。

```
procedure Apply;
```
套用报表类型。

```
procedure GetList(List: TStrings);
```
返回类型名称清单。

```
procedure LoadFromFile(const FileName: String);
procedure LoadFromStream(Stream: TStream);
```
读取类型集。

```
procedure SaveToFile(const FileName: String);
procedure SaveToStream(Stream: TStream);
```
储存类型集。

```
property Items[Index: Integer]: TfrxStyleItem; default;
```
类型清单。

```
property Name: String;
```
类型集名称。
```
end;
```

最后，“TfrxStyleSheet” 类别代表类型库，它提供读取/储存和新增、删除及类型集的搜寻等方法。

```
TfrxStyleSheet = class(TObject)
public
constructor Create;
建立类型库

procedure Clear;
清除类型库

procedure Delete(Index: Integer);
删除指定索引编号的类型

procedure GetList(List: TStrings);
返回类型集合名称清单.

procedure LoadFromFile(const FileName: String);
procedure LoadFromStream(Stream: TStream);
载入类型库

procedure SaveToFile(const FileName: String);
procedure SaveToStream(Stream: TStream);
储存类型库

function Add: TfrxStyles;
加入新的类型至类型库

function Count: Integer;
返回类型库的类型集个数

function Find(const Name: String): TfrxStyles;
返回集合名称的类型集

function IndexOf(const Name: String): Integer;
返回类型集名称的索引编号

property Items[Index: Integer]: TfrxStyles; default;
类型集清单
end;
```

Creation of style sets 建立类型集

下列的程序代码示范如何建立类型集，并增加两个类型至类型集。在这些操作完成之后，这些类型将套用至报表。

```
Var
  Sttyle: TfrxStyleItem;
  Styles: TfrxStyles;
```

```
  Styles := TfrxStyles.Create(nil);

{ 第一个类型}
Style := Styles.Add;
Style.Name := 'Style1';
Style.Font.Name := 'Courier New';

{ 第二个类型}
Style := Styles.Add;
Style.Name := 'Style2';
Style.Font.Name := 'Times New Roman';
Style.Frame.Typ := [ftLeft, ftRight];

{ 套用类型至报表}
frxReport1.Styles := Styles;
```

你可以使用普通的方法建立及使用类型集:

```
Var
  Style: TfrxStyleItem;
  Styles: TfrxStyles;

  Styles := frxReport1.Styles;
  Styles.Clear;

{ 第一个类型 }
Style := Styles.Add;
Style.Name := 'Style1';
Style.Font.Name := 'Courier New';

{ 第二个类型 }
Style := Styles.Add;
Style.Name := 'Style2';
Style.Font.Name := 'Times New Roman';
Style.Frame.Typ := [ftLeft, ftRight];

{ 套用类型至报表 }
frxReport1.Styles.Apply;
```

Modifying/adding/deleting a style

修改指定名称的类型:

```
var
  Style: TfrxStyleItem;
  Styles: TfrxStyles;

Styles := frxReport1.Styles;

{ search for a style}
Style := Styles.Find('Style1');

{ modify the font size }
```

```
Style.Font.Size := 12;
```

加入类型至报表类型集：

```
Var
  Style : TfrxStyleItem;
  Styles: TfrxStyles;

  Styles := frxReport1.Styles;

{ 新增类型 }
Style := Styles.Add;
Style.Name := 'Style3';
```

删除指定名称的类型：

```
var
  Style: TfrxStyleItem;
  Styles: TfrxStyles;

Styles := frxReport1.Styles;

{ 删除类型 }
Style := Styles.Find('Style3');
Style.Free;
```

修改完成之后，你应该调用套用(“Apply”) 方法：

```
{ 套用修改后的结果 }
frxReport1.Styles.Apply;
```

Saving/restoring a set 存取模板

```
frxReport1.Styles.SaveToFile('c:\1.fs3');
frxReport1.Styles.LoadFromFile('c:\1.fs3');
```

Clear report styles

执行的方法有两种：

```
frxReport1.Styles.Clear;
```

或

```
frxReport1.Styles := nil;
```

Styles library creation

下列的范例示范如何建立类型库及新增两个类型至类型库。

```
Var
  Styles: TfrxStyles;
  StyleSheet: TfrxStyleSheet;
```

```
StyleSheet := TfrxStyleSheet.Create;

{ the first set }

Styles := StyleSheet.Add;
Styles.Name := 'Styles1';

{ here one can add styles to the Styles set}

{ the second set }
Styles := StyleSheet.Add;
Styles.Name := 'Styles2';

{ here one can add styles to the Styles set}
```

Displaying a list of style sets, and application of a selected style

类型库为经常被使用在显示特定的输出类型，这些类型可透过“ComboBox”或“ListBox.” 由使用者选取，并套用至报表。

显示类型清单：

```
StyleSheet.GetList(ComboBox1.Items);
```

套用类型至报表：

```
frxReport1.Styles := StyleSheet.Items[ComboBox1.ItemIndex];
```

或

```
frxReport1.Styles := StyleSheet.Find[ComboBox1.Text];
```

Modification/adding/deleting of a styles set

修改指定名称的类型集：

```
Var
  Styles: TfrxStyles;
  StyleSheet: TfrxStyleSheet;

  { search for the required set }

  Styles := StyleSheet.Find('Styles2');

  { modify a style with the Style1 name from the found set }

  with Styles.Find('Style1') do
  Font.Name := 'Arial Black';
```

新增类型至类型集：

```
Var
    Styles: TfrxStyles;
    StyleSheet: TfrxStyleSheet;

    { the third set }
```

```
    Styles := StyleSheet.Add;
    Styles.Name := 'Styles3';
```

从类型集中删除指定类型：

```
  ivar: Integer;
  StyleSheet: TfrxStyleSheet;

{ search for the third set }

i := StyleSheet.IndexOf('Styles3');

{ if find, delete }

if i <> -1 then
   StyleSheet.Delete(i);
```

Saving and loading a styles library(储存及载入类型库)

类型库预置的后缀名为”FSS” 。

```
Var
StyleSheet: TfrxStyleSheet;
StyleSheet.SaveToFile('c:\1.fss');
StyleSheet.LoadFromFile('c:\1.fss');
```