AfalinaSoft XL Report

Version 4.1

Last revised 19-Jun-02.

Table of contents

This document is a reference on XL Report by Afalina Co., Ltd. – a tool for report generation and data analysis in Delphi-applications by means of Microsoft Excel.

Programmer's Reference

XL Report source code consists of several units publishing several classes, procedures, and global variables. This reference describes only those classes and procedures that you will work with. The reference doesn't describe the contents of the protected section of classes. The classes and procedures used by XL Report inner mechanisms are also omitted.

XL Report 4 includes also XL Report G2 (the previous version) classes having ported into the XL Report 4 core for compatibility only. They are described in the "XL Report G2 Programmer's Reference".

XL Report units

XL Report includes the following units:

- **xlcUtils** – service functions;
- **xlcOPack** – XL Option Pack classes;
- **xlcClasses** – XL Report base classes;
- **xlReport** – XL Report classes;
- **xlReportG2** – TxlReportG2 ported under XL Report core;
- **xlAbout** – the **About** dialog;
- **xlReportReg** – component registering and property editors (required in design-time package creation);
- **xlStdOPack, xlPivotOPack** – XL Report standard options.
- **Excel8G2, Office8G2 and VBIDE8G2** – Excel 97 type libraries (they are reworked, so be sure to use them and not any others).

Global constants and variables

xlrOS

Unit: xlcUtils

```
var xlrOS: TxlrOS;
type
  TxlrOS = (xosUnknown, xosWin95, xosWin98, xosWin98SE,
    xosWinNT, xosWinME, xosWin2000, xosWinXP);
```

Returns OS version. Is initialized in the unit's *initialization* section.

xlrExcelVer

Unit: xlReport

```
var
  xlrExcelVer: integer = 0;
```

Returns Excel version. Contains the correct value during report generation only. Contains 0 in all other cases.

xlrLCID

Unit: xlReport

```
var
  xlrLCID: integer = LOCALE_USER_DEFAULT;
```

Returns locale identifier. Used as an *lcid* parameter in many methods of Excel interfaces. Become initialized during first report generation.

xlrDebug

Unit: xlReport

```
var
  {$IFDEF XLR_DEBUG}
    xlrDebug: boolean = true;
  {$ELSE}
    xlrDebug: boolean = false;
  {$ENDIF XLR_DEBUG}
```

Returns true if the application is compiled with {$DEFINE XLR_DEBUG} this way turning on the "visualization" mode of report generation.

Functions and procedures

A1

Unit: xlReport

```
function A1(const Row, Column: integer): string;
```

Returns a relative cell reference in the A1 reference style, given specified row and column numbers.

A1Abs

Unit: xlReport

```
function A1Abs(const Row, Column: integer): string;
```

Returns an absolute cell reference in the A1 reference style, given specified row and column numbers.

R1C1

Unit: xlReport

```
function R1C1(const Row, Column: integer): string;
```

Returns an absolute cell reference in the R1C1 reference style, given specified row and column numbers.

R1C1OfRange

Unit: xlReport

```
function R1C1OfRange(IRange: IxlRange): string;
```

Returns an absolute range reference in the R1C1 reference style. Uses IRange.AddressLocale.

TxlReport class

Unit: XLReport

The *TxlReport* class is the only class accessible in Delphi component palette. This class encapsulates all the mechanisms and algorithms of Microsoft Excel report generation. To build a report using an existing template you have to:

- Create an instance of the class, at either design-time or run-time.
- Specify the path and file name of the template workbook in the *XLSTemplate* property.
- Add all the datasets required in report generation to *DataSources* collection and define their properties.
- Call the *Report* method of this class.

Example of report building on ***tQuickStart.xls*** template with *tblCustomers* and *tblOrders* datasets:

```
procedure TfrmQuickStart.btnReport2Click(Sender: TObject);
var Report: TxlReport;
begin
  Report := TxlReport.CreateEx(Self, 'tQuickStart.xls');
  try
    Report.AddDataSet(tblCustomers);
    Report.AddDataSet(tblOrders, 'OrdersRange');
    Report.Report(false);
  finally
    Report.Free;
  end;
end;
```

Properties

ActiveSheet **published** **read/write**

Sets or retrieves the name of the worksheet to be activated after report generation.

```
property ActiveSheet: string;
```

For a report with several worksheets, you should provide this property to guarantee the activation of a certain worksheet. If this property is empty then XL Report activates the worksheet that was active in the saved template. Example:

```
Report.ActiveSheet := 'Main sheet';
```

DataExportMode **published** **read/write**

Determines the algorithm of data transfer.

```
property DataExportMode: TxlDataExportMode;
type
  TxlDataExportMode = (xdmCSV, xdmVariant, xdmDDE);
```

XL Report supports three alternate data transfer modes: *xdmDDE* (default), *xdmVariant*, *xdmCSV*.

Method	*Algorithm*	*Comments*
xdmVariant	Variant array with direct assignment of values to ranges or cells	The slowest method because of *OLEVariant* usage. Excel recognizes data types accurately enough and works without restrictions.
xdmCSV	Passing of CSV-buffer via clipboard	Medium speed. Enough of the problems in data type recognition but there is no choice sometimes.

xdmDDE	DDE-calls and the Fast DDE Table Format	The quickest method. The only format that uses native data types. **Strings with more then 255 symbols are trimmed**. All numbers are transferred as real (Excel works with real numbers only). Be careful, is used by **default**!

Example:

```
Report.DataExportMode := xdmVariant;
```

DataSources published read/write

TCollection descendant that contains data sources participating in report generation.

```
property DataSources: TxlDataSources;
```

Each item (of *TxlDataSource* type) determines a certain dataset, its properties, and flags. Use standard methods to get access to the items, to add or delete them. To add items, we recommend you to use *AddDataSet* method of *TxlRepor* class. Example:

```
Report.DataSources[i].Range := 'OrdersRange';
```

See also descriptions of *TxlDataSource* and *TxlDataSources* classes below.

Debug public read/write

Turns on (off) the report debug mode.

```
property Debug: boolean; default false;
```

If you set this property to True (default – False), XL Report will generate a report visualizing the changes to the report and allowing access to the hidden worksheet as well as VBA module in the report.

MacroAfter published read/write

Sets or retrieves the full name of the VBA procedure to be called after report generation.

```
property MacroAfter: string;
```

Full name includes module name, dot and procedure name. The procedure must be declared public. Example:

```
Report.MacroAfter := 'Module1.BuildChart';
```

In order to pass parameters to the procedure see the *OnMacro* event.

MacroBefore published read/write

Sets or retrieves full name of the VBA procedure to be invoked before report generation.

```
property MacroBefore: string;
```

The full name includes module name, dot and procedure name. The procedure must be declared public. Example:

```
Report.MacroBefore := 'Module1.DoBuildRange';
```

In order to pass parameters to the procedure see *OnMacro* event.

MultisheetAlias published read/write

Retrieves or sets the alias of the main dataset of a multiple-sheet report.

```
property MultisheetAlias: string;
```

XL Report allows creating multiple-sheet reports where each record of the main dataset specified in this property causes creation of a separate worksheet named after the value of the designated field (see MultisheetField property).

MultisheetField **published** **read/write**

Retrieves or sets the field name being used as a source of sheet names in a multiple-sheet report.

```
property MultisheetField: string;
```

XL Report allows creating multiple-sheet reports where each record of the main dataset specified in the MultisheetAlias property causes creation of a separate worksheet named after the value of the field specified in this property.

Options **published** **read/write**

Contains report generation options.

```
property Options: TxlReportOptionsSet read FOptions write SetOptions
  default [xroDisplayAlerts, xroAutoOpen];
type
  TxlReportOptions = (xroOptimizeLaunch, xroNewInstance, xroDisplayAlerts,
     xroAddToMRU, xroAutoSave, xroUseTemp, xroAutoOpen, xroAutoClose,
     xroHideExcel, xroSaveClipboard, xroRefreshParams);
  TxlReportOptionsSet = set of TxlReportOptions;
```

xroAddToMRU - adding the report to the MRU list (default False)

Turn this option on in order to add a report to the list of most recently used workbooks (MRU). Can be useful if you plan to work with the report after the application closes.

xroAutoClose - automatically closing datasets (default False)

Setting this option to True will automatically close all datasets engaged in report generation immediately after their use. **Attention**! Changing this flag leads to the same change of the *xrgoAutoClose* flag for every item in the *DataSources* collection.

xroAutoOpen - automatically opening datasets (default True)

Setting this option to True will automatically open all closed datasets engaged in report generation. If this option is False and a dataset is not open – an exception is raised. **Attention**! Changing this flag leads to the same change of the *xrgoAutoOpen* flag for every item in the *DataSources* collection.

xroAutoSave - saving a report workbook automatically (default False)

By default, XL Report generates a report without saving it. The user may pay attention to the warning message appearing before closing a report workbook. Turn this option on in order to save a report and prevent this message appear. The xroUseTemp option and TempPath property affects report saving.

If the TempPath property is set, the report will be saved in the path specified in this property. If not, the report will be saved in the current folder of the application.

If xroUseTemp = True, the saved report will be marked as a temporary file of XL Report. When the application closes, such files are deleted. If xroUseTemp = False, the saved report isn't deleted under the assumption that they will be used after the application closes.

xroDisplayAlerts - suppressing Excel warnings (default True)

This option is a complete analog of the Excel.Application.DisplayAlerts property. If false, suppresses Excel prompts and alert messages, for instance, while attempting to write over existing file.

xroHideExcel - creating a report in background (default False)

Set this option to True if you need to create a report in background mode (an invisible report). This works if (and only if) the report is built in the Excel instance created by XL Report. You should set the xroAutoSave to True and define the path in the TempPath property. The report is closed immediately after generation.

xroNewInstance - using a new or existing instance of Excel (default False)

By default, XL Report creates a report using an existing instance of Excel. If there are no such instances, XL Report creates a new instance. You may need to turn this option on in order to prevent interference with the Excel instance launched by the user or by your application.

xroOptimizeLaunch - optimizing the namber of Excel launches (default *True*)

If xroOptimizeLaunch is on an instance of Excel.Application is created during first report generation and is kept connected with till the application closes. You can break the connection and release the Excel instance by invoking the TxlReport.ReleaseExcelApplication class method. This approach allows saving the time in case of multiple Excel launches. If you have a few reports in your application, set this option to False. Be sure to set it to False if you use XL Report in a DLL or in a run-time package.

xroRefreshParams – refreshing the Params collection (default False)

XL Report allows transferring into the report the data defined in the Params collection. This option controls the automatic refreshing of the collection. If it is set to True, XL Report, starting report generation process, will empty the collection, open the report template, find all the formulas on the XLRParams_paramname pattern, and fill the Params with the parameters found. TxlReport.Params is much like TQuery.Params. The difference is: TQuery.Params is refreshed when the SQL property changes, while TxlReport.Params is refreshed before the report generation and if the xroRefreshParams option is on.

xroSaveClipboard – preserving the clipboard contents (default False)

XL Report algorithms as well as Excel methods use the Clipboard extensively. By default, the clipboard content isn't preserved while generating a report. Set this option to True if you need to preserve it. . **Attention**! Some Excel methods of all versions contain several regrettable bugs causing incorrect cell formatting in a report. If you found such a behavior, turn off this option.

xroUseTemp - marking a report as a temporary (default False)

If xroUseTemp = True, the saved report will be marked as a temporary file of XL Report. See the xroAutoSave option.

Params	published	read/write

Sets or retrieves the path and file name of the template workbook.

```
property Params: TxlReportParams;
```

XL Report allows transferring individual values to a report. It keeps them as TxlReportParam instances in the Params collection. You use standard methods to change the contents of this collection or to get access to its items. TxlReportParam exposes only two properties – Name (String) and Value (Variant). The formula on the XLRParams_ParamName pattern in the template will be replaced with the value of the ParamName parameter in the report. The contents of the Params collection can be influenced by the state of the xroRefreshParams option.

ParamByName	published	read only

Retrieves a pointer to the report parameter by its name.

```
property ParamByName[Name: string]: TxlReportParam;
```

XL Report allows transferring individual values to a report. It keeps them as TxlReportParam instances in the Params collection. You use standard methods to change the contents of this collection or to get access to its items. TxlReportParam exposes only two properties – Name (String) and Value (Variant). The formula on the XLRParams_ParamName pattern in the template will be replaced with the value of the ParamName parameter in the report. The contents of the Params collection can be influenced by the state of the xroRefreshParams option.

Preview	published	read/write

If set to True, show the report in preview.

```
property Preview: boolean; default false;
```

TempPath	published	read/write

Sets or retrieves the name of the folder to keep XL Report temporary report files.

```
property TempPath: string;
```

XL Report allows placing temporary report files at a separate folder. You specify the folder in this property. If the folder doesn't exist, an exception is raised. Temporary report files are created only if the xroUseTemp option is on. They have the xlrtmp extension. When the application finishes, all temporary files will be deleted.

XLSTemplate	published	read/write

Retrieves the collection of report parameters.

```
property XLSTemplate: string;
```

If you omit the path, XL Report will seek the template workbook in the start folder of the application (at run-time). At design-time, XL Report seeks the workbook in the project folder. Relative paths are allowed. In such a case, the path is considered as relative to the start folder at run-time or the project folder at design-time.

Excel Type Library interfaces

The following is a list of Excel Type Library interfaces that you can use with *TxlReport* instance. The interfaces are available in event handlers of *TxlReport* **only**. You can use these interfaces to carry out some additional actions during report generation.

INames	public	read only

Points to the *Workbook.Names* interface.

```
property INames: IxlNames;
```

```
type
  IxlNames = Excel8G2.Names;
```

This interface lets you access *Names* collection of a report workbook. Example:

```
var IName: IxlName;
begin
…
  IName:=Report.INames.Item('Range01', EmptyParam, EmptyParam, xlrLCID);
…
end;
```

ITempSheet public read only

Points to the hidden worksheet (XLRPT_Tempsheet) in the report.

```
property ITempSheet: IxlWorksheet;
type
  IxlWorksheet = Excel8G2._Worksheet;
```

This interface lets you access the hidden worksheet inserted by XL Report during report generation. Example:

```
  Report.ITmpSheet.Range('qryMaster_Date').Value := StartDate;
```

IWorkbook public read only

Points to the *Workbook* interface.

```
property IWorkbook: IxlWorkbook;
type
  IxlWorkbook = Excel8G2._Workbook;
```

This property lets you access the report workbook. Example:

```
Report.IWorkbook.Worksheets.Add(EmptyParam, EmptyParam, 1,EmptyParam, xlrLCID);
```

IWorkbooks public read only

Points to the *Application.Workbooks* collection interface.

```
property IWorkbooks: IxlWorkbooks;
type
  IxlWorkbooks = Excel8G2.Workbooks;
```

This property lets you access the collection of all open workbooks in Excel. Example:

```
NewWorkbook := Report.IWorkbooks.Add(EmptyParam, xlrLCID);
```

IWorksheets public read only

Points to the *Application.Worksheets* collection interface.

```
property IWorksheets: IxlWorksheets
type
  IxlWorkbooks = Excel8G2.Workbooks;
```

This property lets you access the collection of all worksheets of the report workbook. Example:

```
INewSheet := Report.IWorksheets.Add(EmptyParam, Report.ITempSheet,
  EmptyParam, EmptyParam, xlrLCID);
```

IModule public read only

Points to the service module interface.

```
property IModule: IxlVBComponent;
type
  IxlVBComponent = OLEVariant;
```

This property lets you access the service VBA module of the report workbook. You can use it in order to place some VBA code into the module.

IWorksheetFunction public read only

Points to *Application.WorksheetFunction* interface.

```
property IWorksheetFunction: IxlWorksheetFunction;
type
  IxlWorksheetFunction = Excel8G2.WorksheetFunction;
```

This property lets you access an interface of Excel worksheet functions. You can use this property in order to fulfill some additional calculations by using Excel tools.

IXLSApp public read only

Points to *Excel.Application* interface.

```
property IXLSApp: IxlApplication;
type
  IxlApplication = Excel8G2._Application;
```

This property lets you access the *Excel.Application* interface.

Methods

AddDataSet public

Adds a dataset to the *DataSources* collection.

```
function AddDataSet(ADataSet: TDataSet; ARange: string = '';
  AMacroBefore: string = ''; AMacroAfter: string = '';
  AOptions: TxlRangeOptionsSet = [xrgoAutoOpen, xrgoPreserveRowHeight];
  AOnMacro: TxlOnMacro = nil;
  AOnBeforeDataTransfer: TxlDataTransferHandleEvent = nil;
  AOnAfterDataTransfer: TxlDataTransferHandleEvent = nil): TxlDataSource;
```

This method adds a new item to the DataSources collection. The item refers to the ADataSet dataset whose data should be placed in the ARange range. The other method parameters initialize corresponding properties of the item

CreateEx public virtual

Creates an instance of *TxlReport* class.

```
constructor CreateEx(AOwner: TComponent; AXLSTemplate: string;
  AActiveSheet: string = '';
  AMacroBefore: string = ''; AMacroAfter: string = '';
  AOptions: TxlReportOptionsSet = [xroDisplayAlerts, xroAutoOpen];
  ATempPath: string = '';
  AOnBeforeBuild: TxlReportHandleEvent = nil;
  AOnAfterBuild: TxlReportHandleEvent = nil); virtual;
```

A replacement for standard constructor approach.

ConnectToExcelApplication public class method

Allows to use any given instance of *Excel.Application*.

```
class procedure ConnectToExcelApplication(OLEObject: OLEVariant);
```

Can be useful if Excel is used in an application not only for reporting. If xroOptimizeLaunch is set to True, all used interfaces will be freed before the connection is made.

Edit public

Use this method to open a template.

```
procedure Edit;
```

Template file must be supplied in *XLSTemplate* property. If this property is empty, Excel is launched with a new workbook. All event handlers will not be activated.

MergeReports public class method

Creates several reports and merges them in one workbook.

```
class procedure MergeReports(Reports: array of TxlReport;
  SheetPrefixes: array of string); override;
```

The first parameter of the method is an array of TxlReport – reports that will be merged in a resulting workbook. The process of generation of every report is isolated from the others because each report is created in a separate workbook. This allows to be not bothered with event handlers and/or macros created for some/all reports. After processing the current report, the ***OnlyValues*** option will be applied to its visible worksheets and every name (the Name object) will be deleted from them (in order to prevent probable name conflicts). All the reports will be added to the first instance of *TxlReport* in the *Reports* array. Every worksheet will be added with its own name prefixed with the prefix taken from the *SheetPrefixes* parameter. You should ensure that both of the arrays have equal number of elements.

ReleaseExcelApplication public class method

Frees all Excel interfaces.

```
class procedure ReleaseExcelApplication;
```

If *roOptimizeLaunch* flag is *True*, then once loaded Excel will be unloaded only after the application finishes. You can use this class method if you want to unload Excel.

Report public virtual

Creates a report based on the template specified in the *XLSTemplate* property.

```
procedure Report;
procedure Report(const APreview: boolean);
procedure Report(Workbook, ExcelApp: OLEVariant;
  const NewWorkbookName: string = '');
```

The second variant of the Report method is left for compatibility only and isn't recommended for future use. If *APreview* is true then the report is opened in preview.

The third variant can be used in several ways. If you pas the path and file name of a workbook in the NewWorkbookName parameter, the report will be saved to this workbook. For example,

```
NewWorkbookName := ExtractFilePath(ParamStr(0)) + 'NewWorkbook.xls';
xlReport1.Report(EmptyParam, EmptyParam, NewWorkbookName);
```

If you pas the path and file name of a workbook in the Workbook parameter, the report will be saved to this workbook. For example,

```
ExistingWorkbookName := ExtractFilePath(ParamStr(0)) + 'QReport.xls';
xlReport1.Report(ExistingWorkbookName, EmptyParam);
```

You can also pass the interface of an open workbook in the Workbook parameter. This helps in case of multiple Excel instances. Example:

```
var
  IXLApp, ITargetWorkbook: OLEVariant;

IXLApp := CreateOLEObject('Excel.Application');
ITargetWorkbook := IXLApp.Workbooks.Open(
  ExtractFilePath(ParamStr(0)) + 'Book1.xls');
try
  xlReport1.Report(ITargetWorkbook, EmptyParam);
except
  IXLApp.Visible := true;
end;
```

You can also specify a concrete Excel instance in this method via the ExcelApp parameter. Example:

```
var
  IXLApp: OLEvariant;

IXLApp := CreateOLEObject('Excel.Application');
try
  xlReport1.Report(EmptyParam, IXLApp);
except
  IXLApp.Visible := true;
end;
```

And finally, you can add a report to an existing workbook and save the resulting workbook under a new name. See below.

```
ExistingWorkbookName := ExtractFilePath(ParamStr(0)) + 'QReport.xls';
NewWorkbookName := ExtractFilePath(ParamStr(0)) + 'QReport_June.xls';
xlReport1.Report(ExistingWorkbookName, EmptyParam, NewWorkbookName);
```

Note. An exception arises, if the file specified in the Workbook parameter is missed. Likewise, an exception arises, when XL Report attempts to write to the existing file specified in the NewWorkbookName parameter. You should take care of both situations yourself.

ReportTo public virtual

Creates a report and inserts it into an existing workbook.

```
procedure ReportTo(const WorkbookName: string; const NewWorkbookName: string = '');
```

Use this method in order to create a report and insert its worksheets into the workbook specified in the WorkbookName parameter. Then this other workbook can be saved under the name specified in the NewWorkbookName parameter. You can also use this method to create a report and save it under new name taken from the NewWorkbookName parameter. The WorkbookName parameter should be empty in this case. Examples:

```
// Create a report and save it in the new workbook
Report.ReportTo('', 'C:\Accounting\GrossRevenue.xls');

// Create a report and add it to an existing workbook
Report.ReportTo('C:\Accounting\MonthlyBalance.xls');

// Create a report, add it to an existing workbook, and save it under new name
Report.ReportTo('C:\Accounting\ ProgressiveBalance.xls.xls',
  'C:\Accounting\ProgressiveBalance_Sep2001.xls.xls');
```

Events

The pseudo-code below describes the algorithm of report generation:

```
procedure TxlAbstractReport.Report;
var
  Raised: boolean;
  i: integer;
begin
  Raised := true;
  try
    try
      Connect;
      if IsRefreshParams then
        RefreshParams(True);
      BeforeBuild;
      DoOnBeforeBuild;
      MacroProcessing(mtBefore, MacroBefore);
      Parse;
      Params.Build;
      for i := 0 to DataSources.Count - 1 do
        if DataSources[i].RangeType = rtNoRange then begin
          DataSources[i].Build;
        end;
      for i := 0 to DataSources.Count - 1 do
        if DataSources[i].RangeType = rtRange then begin
          DataSources[i].Build;
        end;
      for i := 0 to DataSources.Count - 1 do
        if DataSources[i].RangeType = rtRangeRoot then begin
          DataSources[i].BuildRoot;
        end;
      MacroProcessing(mtAfter, MacroAfter);
      DoOnAfterBuild;
      OptionsProcessing;
      AfterBuild;
      Show;
    except
      on E: Exception do begin
        DoOnError(E, Raised);
        ErrorProcessing(E, Raised);
        if Raised then
          raise E;
      end;
    end;
  finally
    Disconnect;
  end;
end;
```

When the process starts (Connect), the XL Report instance gets pointers to all Excel interfaces, opens the template workbook, and adds the XLRpt_TempSheet worksheet and the XLRpt_Module VBA module. Starting from this moment the following interfaces are accessible in XL Report event handlers: IXLSApp, IWorkbooks, IWorkbook, ITempSheet, INames and so on. If the xroRefreshParams option is on, the Params collection is refreshed. Then the OnBeforeBuild event is triggered. Here you can initialize report parameters and/or perform any operations before the datasets will be open. Then the VBA macro specified in the MacroBefore property is invoked. Just before this the OnMacro event is triggered in order to allow you to pass macro parameters. Then the template is analyzed, report parameters are transferred, NoRange-datasets are transferred, and standalone Range-datasets are transferred. Then comes the turn of nested Range-datasets. After data transfer the MacroAfter VBA macro is invoked being preceded with the OnMacro event. The report and sheet options are processed, and the AfterBuild event is triggered. Excel is shown and its interfaces are freed (Disconnect). In

case of any exception, the OnError event is triggered thus allowing you to process the exception. Every step described above is preceded with the OnProgress event.

This is a standard event cycle for XL Report reporting. There are two events that can be triggered in following situations:

- OnTargetBookSheetName – an event triggered before a worksheet is added to the report workbook specified in the ReportTo method. You can change the name of the worksheet in corresponding event handler.
- OnBeforWorkbookSave – an event triggered before the report workbook is saved (if the xroAutoSave option is on). You can change the file and path name of the workbook or cancel saving.

OnAfterBuild

Is triggered after report generation.

```
property OnAfterBuild: TxlReportHandleEvent;
type
  TxlReportHandleEvent = procedure (Report: TxlReport) of object;
```

Use the event handler in order to carry out some additional actions after report generation. All Excel interfaces are accessible here. See also the *OnBeforeBuild* event.

OnBeforeBuild

Is triggered before report generation.

```
property OnBeforeBuild: TxlReportHandleEvent;
type
  TxlReportHandleEvent = procedure (Report: TxlReport) of object;
```

Use the event handler in order to carry out some additional actions after report generation. All Excel interfaces are accessible here. Many XL Report users prepare template workbooks in this event. See also the *OnAfterBuild* event.

OnBeforeWorkbookSave

Is triggered in order to define the path and file name of the report workbook.

```
property OnBeforeWorkbookSave: TxlOnBeforeWorkbookSave;
type
  TxlOnBeforeWorkbookSave = procedure (Report: TxlReport;
    var WorkbookName, WorkbookPath: string; Save: boolean) of object;
```

This event is triggered only if the *xroAutoSave* option is True. *WorkbookName* and *WorkbookPath* contain the path and file name of the report workbook. Both of them are not checked for existence. If the file exists an exception is fired. To save the report set *Save* = *true*, to prevent from saving set *Save* = *false*.

OnBreak

Allows canceling the report generation process.

```
property OnBreak: TxlOnBreak;
type
  TxlOnBreak = procedure (Report: TxlReport;
    var IsBreak: boolean) of object;
```

Use this event in order to allow your users to cancel the report generation process. Set IsBreak to true if it is the case. **Note**. This event doesn't allow canceling long lasting VBA macros, in particular those specified in the MacroBefore and MacroAfter properties.

OnError

Is triggered in case of errors during report generation.

```
property OnError: TxlOnError;
type
  TxlOnError = procedure(Sender: TObject; E: Exception;
    var Raised: boolean) of object;
```

Use this event in order to override the standard exception processing behavior of XL Report. If *IsRaised* = *false* all the raised exceptions will be suppressed in the Report method. Default: *IsRaised* = *true*. Exception type can be determined by use of the *Exception* parameter and Is operator.

OnMacro

Is triggered before calling VBA procedures supplied in *MacroBefore* and *MacroAfter* properties.

```
property OnMacro: TxlOnMacro;
type
  TxlOnMacro = procedure (Report: TxlReport; DataSource: TxlDataSource;
    const AMacroType: TxlMacroType; const AMacroName: string;
    var Arg1, Arg2, Arg3, Arg4, Arg5, Arg6, Arg7, Arg8, Arg9,
    Arg10, Arg11, Arg12, Arg13, Arg14, Arg15, Arg16, Arg17, Arg18,
    Arg19, Arg20, Arg21, Arg22, Arg23, Arg24, Arg25, Arg26, Arg27,
    Arg28, Arg29, Arg30: OLEVariant) of object;
```

Use this event if you need to pass some parameters to the VBA procedure. Assign the parameters' values to *ArgXX* in strict order, otherwise you get an exception. The *DataSource* parameter contains *nil*. If you make *OnMacro* event handler to be a common event handler for several instances of *TxlReport* class, you can use *Tag* property of the class and *AMacroName* – the name of the VBA procedure to be processed.

OnProgress

Accompanies every step of report generation.

```
property OnProgress: TxlOnProgress;
type
  TxlOnProgress = procedure (Report: TxlReport;
    const Position, Max: integer) of object;
```

Use this event to visualize report generation. *Max* contains the number of report generation steps. *Position* contains the number of the current step. Because template options are analyzed in consecutive order, so *Max* can be changed during report generation. After report generation, this event is triggered again with *Position* and *Max* both equal to zero. Example:

```
procedure TfrmEvents1.xlReportProgress(Report: TObject; Position, Max: Integer);
begin
  pbReport.Min := 0;
  pbReport.Max := Max;
  pbReport.Position := Position;
end;
```

OnTargetBookSheetName

Is triggered in order to define the name of the worksheet being added to an existing workbook with the *ReportTo* method.

```
property OnTargetBookSheetName: TxlOnTargetBookSheetName;
type
  TxlOnTargetBookSheetName = procedure (Report: TxlReport;
    ISheet: IxlWorksheet; ITargetWorkbook: IxlWorkbook;
    var WorksheetName: string) of object;
```

This event is triggered when the template worksheet is added to the target workbook . You can rename the worksheet by supplying the new name in the WorksheetName parameter. ISheet contains the interface to the worksheet, ITargetWorkbook contains the interface to the target workbook.

TxlDataSource class

Unit: xlReport

This class represents an item of the collection (a *TCollectionItem* descendant) used to bind a dataset to a report (i.e. with the instance of *TxlReport*). Its main purpose is to describe a data set supplying data for a report. Create instances of this class using methods of the *DataSources* collection or the *TxlReport.AddDataSet* method. Fill the properties needed while editing the form or at run-time. Here are the properties:

- *DataSet* – a dataset, an instance or a descendant of the *TDataSet* class;
- *Range* – the name of the range whose cells should be filled with the dataset data;
- *Options* – range options;
- *MacroAfter* / *MacroBefore* – macro's names that will be executed after/before data transfer.

Example:

```
Report.DataSources[0].MacroBefore := '';
Report.DataSources[0].Range := 'CustRange';
Report.DataSources[0].MacroAfter := 'Module1.BuildPivotTable';
Report.DataSources[0].DataSet := qryCustomers;
```

Because this class is not designed to be used alone, it doesn't contain published methods. In addition to properties, the *TxlDataSource* class contains several events that can be useful in some reports.

Properties

Alias	published	read/write

A pseudonym for a dataset.

```
property Alias: string;
```

Is used in field formulas instead of referencing to *DataSource.Name* property. Useful in case of a report built on several datasets of the same name. *Alias* is equal to *Name* by default.

Column	public	read only

Returns the column number of the leftmost column of the range.

```
property Column: integer;
```

A range in Excel is described by its top left corner coordinates (Row, Column) and dimensions (RowCount, ColCount). This properties can be useful in event handlers of the TxlReport and TxlDataSource classes.

ColCount	public	read only

Returns the number of columns in the range.

```
property ColCount: integer;
```

A range in Excel is described by its top left corner coordinates (Row, Column) and dimensions (RowCount, ColCount). This properties can be useful in event handlers of the TxlReport and TxlDataSource classes.

DataSet	published	read/write

Binds a dataset to a report.

```
property DataSet: TDataSet;
```

Specify the name of an existing dataset. At run-time use the following syntax:

```
Report.DataSources[0].DataSet := Table1;
```

If *DataSet* is *nil* then arbitrary data transfer is assumed and *OnGetDataSourceInfo, OnGetFieldInfo* and *OnGetRecord* events are triggered.

Enabled published read/write

Enables (disables) the data transfer from the corresponding dataset.

```
property Enabled: boolean;
```

This property can be useful in report debugging.

MacroAfter published read/write

Contains the full name of the VBA procedure invoked after data transfer.

```
property MacroAfter: string;
```

The full name contains the module name, dot, and the procedure name itself. The procedure must be declared public. Example:

```
Report.DataSources[0].MacroAfter := 'Module1.BuildChart';
```

See also *OnMacro* event.

MacroBefore published read/write

Contains the full name of the VBA procedure that must be invoked after data transfer.

```
property MacroBefore: string;
```

The full name contains the module name, dot, and the procedure name itself. The procedure must be declared public. Example:

```
Report.DataSources[0].MacroBefore := 'Module1.DoBuildRange';
```

See also *OnMacro* event.

MasterSource public read/write

Is used in order to specify the range, containing the current range (specified in the Range property

```
property MasterSource: TxlDataSource;
```

Use this property in order to create master-detail relationships between datasets. In order to break a master-detail relationship, specify nil. Even if you specify master-detail relationships correctly you have to create a correct template – all ranges should be nested correctly.

MasterSourceName published read/write

Is used in order to specify the range, containing the current range (specified in the Range property

```
property MasterSourceName: string;
```

Use this case-insensitive property in order to create master-detail relationships between datasets. In order to specify the name of the master-range use the Alias of the master-range. In order to break a master-detail relationship, specify the empty string. Even if you specify master-detail relationships correctly you have to create a correct template – all ranges should be nested correctly.

Options published read/write

Contains various settings of the dataset and its range.

```
property Options: TxlRangeOptionsSet;
  default [xrgoAutoOpen, xrgoPreserveRowHeight]
type
  TxlRangeOptions = (xrgoAutoOpen, xrgoAutoClose, xrgoPreserveRowHeight);
  TxlRangeOptionsSet = set of TxlRangeOptions;
```

xrgoAutoClose (default True)

Setting this option to True will automatically close the dataset specified in the DataSet property immediately after its use. If this option is off the dataset will be left open.

xrgoAutoOpen (default True)

Setting this option to True will automatically open the dataset specified in the DataSet property. If this option is False and a dataset is not open – an exception is raised.

xrgoPreserveRowHeight (default True)

While processing Range-datasets, XL Report inserts rows in a range. By default, the height of each row is set according to the same of the template. This can cause slowing down the report generation speed. Set this option to False in order to avoid the slowing.

Range — published — read/write

Determines the name of the range in whose cells you want to transfer the data from the *DataSet*.

```
property Range: string;
```

To transfer the data into a certain range of workbook you must name the range and specify it in the *Range* property. The cells of this range should contain field formulas on the **=DataSetName_FieldName** stencil. We describe format of XL Report ranges in **Developer's Guide**. If *Range* property is empty, XL Report will transfer to report workbook the current record only.

RangeType — public — read only

Returns the type of the range.

```
property RangeType: TxlRangeType;

type
  TxlRangeType = (rtNoRange, rtRange, rtRangeRoot,
    rtRangeMaster, rtRangeDetail);
```

In order to provide additional possibilities of data transfer and processing, XL Report allows using range and column options. Some of the options can be used in certain range types only.

- **rtNoRange –** NoRange-dataset;
- **rtRange –** a standalone Range;
- **rtRangeRoot –** a top-level range including one or more nested ranges;
- **rtRangeMaster –** a range nested into another range and including one or more nested ranges;
- **rtRangeDetail –** a range nested into another range and having no nested ranges.

Report — public — read only

Points to the instance of *TxlReport* whose *DataSources* collection this instance of TxlDataSource belongs to.

```
property Report: TxlReport;
```

Row	public	read only

Returns the column number of the leftmost column of the range.

```
property Row: integer;
```

A range in Excel is described by its top left corner coordinates (Row, Column) and dimensions (RowCount, ColCount). This properties can be useful in event handlers of the TxlReport and TxlDataSource classes.

RowCount	public	read only

Returns the number of columns in the range.

```
property RowCount: integer;
```

A range in Excel is described by its top left corner coordinates (Row, Column) and dimensions (RowCount, ColCount). This properties can be useful in event handlers of the TxlReport and TxlDataSource classes.

Tag	published	read/write

This is a standard property of VCL component classes.

```
property Tag: integer;
```

Use it as you wish.

Excel Type Library interfaces

IName	public	read only

Points to the interface of the Name object (Excel) referencing the range.

```
property IName: IxlName;
type
  IxlName = Excel8G2.Name;
```

This interface allows accessing the Name object that refer to the range whose name is specified in the Range property. Can be used in event handlers only in order to carry out some additional actions during report generation.

INames	public	read only

Points to *Workbook.Names* interface.

```
property INames: IxlNames;
type
  IxlNames = Excel8G2.Names;
```

This interface allows accessing the *Names* collection of the report workbook. Can be used in event handlers only in order to carry out some additional actions during report generation.

IRange	public	read only

Points to the interface of the range for a Range-dataset.

```
property IRange: IxlRange;
type
  IxlRange = Excel8G2.Range;
```

This property allows accessing the range of a Range-dataset. Can be used in event handlers only in order to carry out some additional actions during report generation.

ITempSheet	public	read only

Points to the temporary sheet (XLRpt_TempSheet) in the report.

```
property ITempSheet: IxlWorksheet;
type
  IxlWorksheet = Excel8G2._Worksheet;
```

This interface allows accessing the hidden worksheet inserted by XL Report during report generation. Can be used in event handlers only in order to carry out some additional actions during report generation.

IWorkbook public read only

Points to the *Workbook* interface.

```
property IWorkbook: IxlWorkbook;
type
  IxlWorkbook = Excel8G2._Workbook;
```

This property allows accessing the report workbook. Can be used in event handlers only in order to carry out some additional actions during report generation.

IWorkbooks public read only

Points to the interface of the *Application.Workbooks* collection.

```
property IWorkbooks: IxlWorkbooks;
type
  IxlWorkbooks = Excel8G2.Workbooks;
```

This property allows accessing the collection of open workbooks. Can be used in event handlers only in order to carry out some additional actions during report generation.

IWorksheet public read only

Points to the *Worksheet* interface.

```
property IWorksheet: IxlWorksheet;
type
  IxlWorksheet = Excel8G2._Worksheet;
```

This property allows accessing the *Worksheet* interface of the worksheet containing the Range. Can be used in event handlers only in order to carry out some additional actions during report generation.

IWorksheets public read only

Points to the interface of the *Worksheets* collection.

```
property IWorksheets: IxlWorksheets;
type
  IxlWorksheets = Excel8G2.Sheets;
```

This property allows accessing the collection of worksheets of the report workbook. Can be used in event handlers only in order to carry out some additional actions during report generation.

IXLSApp public read only

Points to the *Excel.Application* interface.

```
property IXLSApp: IxlApplication;
type
  IxlApplication = Excel8G2._Application;
```

This property allows accessing the *Excel.Application* interface. Can be used in event handlers only in order to carry out some additional actions during report generation.

Events

The pseudo-code below describes the algorithm of data transfer for a given DataSource.

```
procedure TxlAbstractDataSource.Build;
begin
  if Enabled then
    try
      Connect;
      DoOnBeforeDataTransfer;
      MacroProcessing(mtBefore, MacroBefore);
      if Assigned(DataSet) then begin
        // data from DataSet
        if xrgoAutoOpen in Options then
          DataSet.Open;
      end
      else begin
        // unbound data
        DoOnGetDataSourceInfo;
        for i := 1 to FieldsCount do
          DoOnGetFieldInfo;
      end;
      Parse;
      if Assigned(DataSet) then
        // data from DataSet
        PutData
      else begin
        // unbound data
        Eof := false;
        while not Eof do
          GetRecord(RecBuf, Eof);
      end;
      MacroProcessing(mtAfter, MacroAfter);
      DoOnAfterDataTransfer;
      OptionsProcessing;
    finally
      if Assigned(DataSet) and (xrgoAutoClose in Options) then
        DataSet.Close;
      Disconnect;
    end;
end;
```

When the process starts (Connect), pointers to all Excel interfaces are initialized. Then the OnBeforeDataTransfer event is triggered and the macro specified in the MacroBefore property is invoked. If the dataset isn't open and xrgoAutoOpen is on, the dataset is open. If the dataset isn't assigned, it is assumed that arbitrary data should be transferred to the report and the OnGetDataSourceInfo event is triggered. You use this event in order to define the structure of data and the number of fields. Each field causes triggering of the OnGetFieldInfo event where you supply XL Report with the field type. The template range is parsed and data are transferred In case of arbitrary data, each "record" is accompanied with triggering of the OnGetRecord event. The macro specified in the MacroBefore property is invoked and the OnAfterDataTransfer event is triggered. If the xrgoAutoClose option is on, the dataset is closed. All Excel interfaces are freed (Disconnect).

OnAfterDataTransfer

Is triggered after data transfer.

```
property OnAfterDataTransfer: TxlDataTransferHandleEvent;
type
  TxlDataTransferHandleEvent = procedure (DataSource: TxlDataSource) of object;
```

Use this event to carry out any additional actions after data transfer. This also is the place where you can use the Excel interfaces described in corresponding sections of this Help. See also *OnBeforeDataTransfer*.

OnBeforeDataTransfer

Is triggered before data transfer.

```
property OnBeforeDataTransfer: TxlDataTransferHandleEvent;
type
  TxlDataTransferHandleEvent = procedure (DataSource: TxlDataSource) of object;
```

Use this event to carry out any additional actions before data transfer. This also is the place where you can use the Excel interfaces described in corresponding sections of this Help. See also *OnAfterDataTransfer*.

OnGetDataSourceInfo

Allows describing the structure of arbitrary data.

```
property OnGetDataSourceInfo: TxlGetDataSourceInfo;
type
  TxlGetDataSourceInfo = procedure (DataSource: TxlDataSource;
    var FieldCount: integer) of object;
```

XL Report operates on tabular data so you have to define your data structure (even if you work with single record). If the *DataSet* property of the *DataSources* collection item is empty, this event is triggered. You define the number of fields in your by passing it in the FieldCount parameter. This causes triggering the OnGetFieldInfo event FieldCount times.

OnGetFieldInfo

Allows defining the field name and type for arbitrary data.

```
property OnGetFieldInfo: TxlGetFieldInfo;
type
  TxlGetFieldInfo = procedure (DataSource: TxlDataSource;
    const FieldIndex: integer;
    var FieldName: string; var FieldType: TxlDataType) of object;
```

XL Report operates on tabular data so you have to define your data structure (even if you work with single record). If the *DataSet* property of the *DataSources* collection item is empty, this event is triggered. You define the number of fields in your by passing it in the FieldCount parameter. This causes triggering the OnGetFieldInfo event FieldCount times. XL Report field types are:

```
type
  TxlDataType = (xdNotSupported, xdInteger, xdBoolean, xdFloat,
    xdDateTime, xdString);
```

OnGetRecord

Is triggered for every "record" of an arbitrary data dataset.

```
property OnGetRecord: TxlGetRecord;
type
  TxlGetRecord = procedure (DataSource: TxlDataSource;
    const RecNo: integer;
    var Values: OLEVariant; var EOF: boolean) of object;
```

OnMacro

Is triggered before invoking the VBA procedure specified in *MacroBefore* or *MacroAfter* properties.

```
property OnMacro: TxlOnMacro;
type
  TxlOnMacro = procedure (Report: TxlReport; DataSource: TxlDataSource;
    const AMacroType: TxlMacroType; const AMacroName: string;
    var Arg1, Arg2, Arg3, Arg4, Arg5, Arg6, Arg7, Arg8, Arg9,
    Arg10, Arg11, Arg12, Arg13, Arg14, Arg15, Arg16, Arg17, Arg18,
    Arg19, Arg20, Arg21, Arg22, Arg23, Arg24, Arg25, Arg26, Arg27,
    Arg28, Arg29, Arg30: OLEVariant) of object;
```

Use this event in order to pass parameters to the VBA procedure. Assign the parameters' values to *ArgXX* in strict order, otherwise you will get an exception. *DataSource* points to the instance of *TxlDataSource* corresponding to this event. If you make *OnMacro* event handler to be a common event handler for several instances of *TxlDataSource* class, you can use the *Tag* property of the class and *AMacroName* – the name of the VBA procedure to be invoked.

TxlDataSources class

Unit: xlReport

The *TxlDataSources* class implements a collection (*TCollection* descendant) binding a report and datasets. It differs from any VCL collections in a few details – *Items* and *Report* properties, and *Add* method. Its main purpose is to unite all datasets in a list and to show this list in an appropriate property of a report – *TxlReport* instance.

Properties

Items	public	read/write

Accesses a collection item by index.

```
property Items[Index: integer]: TxlDataSource;
```

Report	public	read only

Points to the instance of *TxlReport* that owns the collection.

```
property Report: TxlReport;
```

Methods

Add	public

Adds a new item to the collection.

```
function Add: TxlDataSource;
```

TxlReportParam class

Unit: xlReport

The *TxlReportParam* class implements an item of the *TxlReportParams* collection allowing transferring report parameters to the report. You create instances of this class using methods of *TxlReportParams*. Fill its properties either at design-time or at run-time.

Properties

Name	published	read/write

Sets or returns the name of the report parameter.

```
property Name: string;
```

Use this property in order to name the report parameter. Default value is the empty string. The name must be a unique string, otherwise you will get an exception. Use the report parameter in a report via the following construction =XLRPARAMS_ParamName.

Report	public	read only

Points to the instance of *TxlReport* that owns the parameter.

```
property Report: TxlReport;
```

Value	published	read/write

Sets or returns the value of the report parameter.

```
property Value: Variant;
```

Use this property in order to transfer OLEVariant-compatible values to the report.

TxlReportParams class

Unit: xlReport

The *TxlReportParams* class implements the collection of report parameters. It is accessible via the *Params* property of the *TxlReport* class.

Properties

Items	public	read/write

Accesses a collection item by index.

```
property Items[Index: integer]: TxlReportParam;
```

Use the *ParamByName* property of the *TxlReport* class in order to access parameters by their names.

Report	public	read only

Points to the instance of *TxlReport* that owns the collection.

```
property Report: TxlReport;
```

Methods

Add	public

Adds a new item to the collection.

```
function Add: TxlReportParam;
```