VFPxWorkbookXlsx Release Notes

Class for Reading/Writing XLSX Format Files

Release 10

VFPxWorkbookXlsx class Release 10 is being released in order to correct several identified bugs by users. Additionally, new methods are provided for page layout of the sheets. A new method is also released for defining custom numeric formats.

There were a number of bug fixes that was identified and now fixed in this release which is the main purpose of releasing this class at time.

Addtionally, there is a new method for reading an existing XLSX file into the working cursors and then supporting methods to extract the individual cell values and formatting. However, this XLSX read method is still in development and not yet ready for use. This is the method OpenXlsxWorkbook() – only use if you want to try it but the method will not necessarily read all the numeric or date values correctly at this point. I did not want to hold this release of VFPxWorkbookXlsx class for the completion of the reading of the XLSX file due to the other bug fixes.

Release 11

Release 11 is released to correct several identified bugs by users and to provide a first working release of reading an existing XLSX file into the class. Methods are provided to retrieve the sheet data and cell data/formatting. Methods are also provided to read a XLSX file directly to a cursor or table. Changes can be made to the cells and then re-saved to the existing file or to a new file.

The color determination of cell formatting is not yet accurate; I am not yet sure if I am reading the value incorrectly or writing the value incorrectly. This is still 'work in progress' as the color written seems to be the inverse of the original color (Red and Blue values reversed).

The data environment for the class is now private per the instance of the class. This keeps the cursor names from possible naming conflicts with other applications. The switching of data environments between the host form and this class is managed by VFPxWorkbookXlsx. VFPxWorkbookXlsx identifies the 'default' data session based on the assigned data session of the host form. If VFPxWorkbookXlsx is created as an instance independent of a form (i.e., via NEWOBJECT() command), then the default data session is Id=1.

Release 12

Release 12 is released to correct the incorrect color being written and read by the class for cell formatting.

Release 13

Release 13 is released to correct some bugs in using a class defined private data session for the cursors and the use of some currencies other than USD (in particular Euro and British Pounds). The work for the these currencies was provided by Richard Kaye. Richard also provided feedback on

some of the data session errors that are corrected. Also added support for logical data types – this data type was overlooked when writing the SetCellValue() method. The boolean values written to the xlsx file are controlled by the property TrueFalseValue which is a pipe separated string of the true and false value; by default the value is “Yes|No”.

Release 14

Release 14 is released to restore the SET DELETED setting after performing a workbook save. The previous behavior was to set it to OFF; now it restores to the setting prior to calling the save method.

Release 15

Release 15 is released to fix a stupid bug – the DataSession was not being set to the class session for the SetCellWordWrap() method. Most likely a code completion error in using intellisense and selecting the wrong method in the drop down list. Thanks to Benny Thomas for finding this problem.

Release 16

Release 16 is released to fix bugs and change the private DataSession behavior as well as adding some more functionality.

There were several DEFINEs that were repeated names and now corrected. Corrected a bug in the Set…Range methods where the default DataSession was not reset in the event a set returned not set. Corrected the value assignment for a General field type (pointed out by Doug Hennig) to be an empty string; i.e. “”; I made this assignment in the method SetCellValue based on the value data type.

The intent of the private DataSession was to prevent the collision of using the same cursor name in the class and by others in their forms/classes. My naming convention for cursors is to prefix them with a c underscore; i.e., c_strings. This naming convention could be used by others as well. So I attempted to prevent this by a private DataSession. But this has proven to be troublesome in having to constantly switch to the private DataSession and then back to the default DataSession. The most particular problem was when a grid is to be outputted to a spreadsheet. Having the cursors in their own DataSession and the grid’s cursorsource in another DataSession causes the grid to loose its CursorSource setting. Additionally, there are performance delays in the constant switching of DataSessions. So in short, I have decided to remove the private DataSession and rename the working cursors with an xl_ prefix. Hopefully, this naming convenion is not being commonly used by others in cursor or alias naming.

The following methods have been added:

GetWorkbookFileName()
OpenCreatedXlsxFile()
GetSheetName()
SetSheetVisibility()
AddNamedRange()
ClearCellValidation()
ClearNamedRange()
GetCellValidation()
GetValidation()
GetValidationList()
SetCellValidation()

Release 17

Release 17 is released to fix issues with opening existing workbooks. There were errors associated with extracting the cell formatting. To resolve this I decided to re-write the extraction code and remove the usage of the MS DOM XML parser. Instead, I wrote my own XML parser methods in pure VFP code. This has the advantage of lower memory overhead. In doing this I restructured the cursors for holding the format information. They are actually now in line with the data structures in the styles.xml that is used by the workbook. This restructuring of code should also have the benefit of faster writing of workbooks as there are now considerably less SEEKs being performed to retrieve the cell formatting during the save workbook method.

Additionally, the methods that has color definition for cell fonts, fills, and borders changed to account for the property "indexed". From the documentation, "*ECMA-376, 4th Edition, Office Open XML Formats – Fundamentals and Markup Language Reference*", the "indexed" property is retained for backward compatibility. However, apparently Microsoft Excel will save color definitions with this property (I am testing with version 2010) so I am not sure how far backward compatibility it is. Anyway, the open methods now account for this color property. This necessitated the changing of some methods that impact color definitions.

This restructuring of code also necessitated the changes to the parameters of some of the methods. These include the following methods:

GetCellBorders()	Return object changed
GetCellFill()	Return object changed; previously was a single color value, now it is returned as an object with property values
GetCustomNumericFormat()	New parameter added; Workbook Id
SetCellFill()	New/changed parameters; Color now split by FgColor (foreground) and BgColor (background) values and added pattern type for fill
SetCellFillRange()	New/changed parameters; Color now split by FgColor (foreground) and BgColor (background) values

The following methods have been added:

SetDefaultBorder()	Allows user to set the default border to be used globaly in the workbook.
SetDefaultFill()	Allows user to set the default cell fill to be used globaly in the workbook.
SetDefaultFont()	Allows user to set the default cell font/size to be used globaly in the workbook.
GetCellIndent()	Gets the cell indentation amount.
SetCellIndent()	Sets the cell indentation amount.
GetCellTextRotation()	Gets the cell text rotation amount.
SetCellTextRotation()	Sets the cell text rotation amount.
AddInLineFontObject()	Adds an in-line character definition to the base in-line font definition object.
CreateInLineFormatText()	Creates the base in-line font object for assigning a text string in a cell to have its characters to be individually formatted (works with method AddInLineFontObject()).
GetInLineFormatText()	Gets the in-line font definition for a cell text string.
SetCellInLineFormatText ()	Saves an in-line font definition for a text string in a cell.

Release 18

Release 18 is released to add new methods and fix bugs. Also, included in this release are improvements in speed for creating spreadsheets with cell formatting. I have also started the ground-work for maintaining all content in an existing spreadsheet such as images, external referenced spreadsheets, shapes, etc.; however, this feature is not yet ready for this release (I have elected to release this code without this in order for more users to gain benefit from other improvements).

The following properties have been added:

SaveCurrencyAsNumeric	Indicates whether to save a currency value as a currency value or as a numeric value. Boolean value (default is False). Feature request by Tony Federer.

The following methods have been added:

AddIndexColor()	Adds a new indexed color definition to a workbook
AddMruColor()	Adds a new MRU color definition to a workbook
CellFormatPainter()	Copies the formatting of a source cell to a range of cells. This is faster than formatting individually each cell.
GetCellNumberFormatText()	Returns the cell numeric format as a text string
GetNumberOfSheets()	Returns the number of sheets for the given workbook.
GetRowMaxColumn()	Returns the maximum row column for the given row

The following methods have been added for managing cell styles (see section on Cell Styles in the documentation):

AddStyleBorders()	Adds to a cell style the cell border definition
AddStyleFill()	Adds to a cell style the cell fill definition
AddStyleFont()	Adds to a cell style the cell font definition
AddStyleHorizAlignment()	Adds to a cell style the cell horizontal alignment definition
AddStyleVertAlignment()	Adds to a cell style the cell vertical alignment definition
AddStyleIndent()	Adds to a cell style the cell indent definition

AddStyleNumericFormat()	Adds to a cell style the cell numeric format definition
AddStyleTextRotation()	Adds to a cell style the cell text rotation definition
AddStyleWordWrap()	Adds to a cell style the cell wordwrap definition
CreateFormatStyle()	Creates a new formatting style definition to be applied to cells
GetCellStyle()	Returns the selected cell style definition Id
SetCellStyle()	Sets the selected cell style definition Id
SetCellStyleRange()	Sets a range of cells the cell style definition Id

The following events have been added:

OnShowErrorMessage()	Allows BINDEVENTS to be used to bind to the event in order to display a user an error message
OnShowStatusMessage()	Allows BINDEVENTS to be used to bind to the event in order to display a user progress message during open and save spreadsheet methods

The following enhancements have been added:

Speed improvements	The protected method GetNextId() has been changed for speed enhancements. Previously a call to this method would perform a SQL – Select on the passed cursor to determine the last used Id value; now these values are stored as properties and accessed via a ASCAN() call. The basis for this enhancement was from Tony Federer.
Cell limits checking	Methods that assign cell values or formatting now enforce row and column max limits
String limits checking	Character string limits for assigning to cell values and to header values are now enforced

Set currency value to cell	Value can be saved as a currency or as a numeric value (float) based on setting for property SaveCurrencyAsNumeric (default is currency)
Named Ranges	Named Ranges are now supported when opening an existing workbook
`CELL_FORMAT_CURR_EURO_RED`	New currency format for Euro with negative values in red
`CELL_FORMAT_CURR_POUNDS_RED`	New currency format for Pounds with negative values in red

The following bugs have been fixed:

Setting the cell format to the class defined `CELL_FORMAT_CURRENCY_RED` numeric format would not format the decimals correctly and set a period at the end of each value displayed. Corrected; this class defined format now restricts the number of decimal places to two. The internal numeric format code `CELL_FORMAT_CURRENCY_RED` has been changed to include parenthesis for negative amounts in addition to the red color.

When document properties were set these were being stored as text without conversion to XML format into the spreadsheet. Characters such as '&' would cause an error in the spreadsheet due to invalid character in the XML value. These are now stored after conversion to XML format.

The method SaveTableToWorkbook() did not save the table records based on the current index set; this is now corrected to save the table records based on index order. This change was contributed by Tony Federer.

SaveWorkbook() method now includes a SET SAFETY OFF and restore of previous setting to prevent unwanted file prompts. This change was contributed by Tony Federer.

The protected method GetXMLString() now removes ASCII characters 0 through 31 except for 9 (TAB), 10 (LF) and 13 (CR). This change was contributed by Tony Federer.

Spreadsheets that had a very large number of columns was failing in the method ColumnIndexToAscii() due to the code logic. The logic is now changed to support the max column width allowed by specification.

How the cell numeric formatting was being saved has been changed to streamline the read of a cell formatting and the assigning of a cell formatting; earlier method introduced problems during reading of cell numeric formatting for existing workbooks.

When adding a Named Range for just a column or just a row, the code would incorrectly add a 0 reference for the row when only a column is referenced or when only a row is referenced; now fixed.

The following method parameter defaults have been changed:

SetCellFill() and SetCellFillRange()	These methods previously required the tnBColor (background color) value to be specified; this is now defaulted to RGB(255,255,255) if not specified.

Release 19

Release 19 is released to fix bugs in two methods. I had added events for allowing progress to be displayed. In the methods SaveTabletoWorkbook() and SaveGridToWorkbook() I had added these events with the first name of the event that I set; however, later I decided to change the name of the event and missed updating the name in these two events. The event names has been changed.

In changing the event names in these two methods, I have added a new tnMode value which is 3. The definition of this mode is as follows:

```
nMode = 3; saving an xlsx file
tnStage = 0; start of write of data to cell values
tnStage = 1-n; indicates saving cell values
tnStage = -1; end of write of data
```

In the first call (tnStage=0), the total number of rows is also sent as a parameter.

Release 20

Release 20 is released to fix a typo bug pointed out by rkaye (on VFPx).

Release 21

Release 21 is released to fix the conversion of a string to an XML string and reverse. I ran into this bug this afternoon in some of my own work. In the protected method GetXMLString() I changed the method of converting a string to xml representation using a DOM processor; specifically

```
this.oXDOM = CREATEOBJECT('MSXML2.DOMDocument')
```

This is initialized in Init() method and if fails (MSXML not installed) then I revert to the older VFP code. Then I use createTextNode method to create the xml string. This works except again for embedded double-quotes which are retained and not converted to " which I would have expected to have been converted. I also changed to now explicitly change double-quotes and remove any ASCII character below 32 decimal (except for 9, 10, and 13) whether using the DOM or previous VFP code.

I also found a potential infinite loop in the GetStringXML() method. If the string had an embedded coded value of `&#nnn;` where nnn is greater than 255, then this method went into an infinite loop. I found a case where the nnn value was greater than 255. I have now changed this method to now skip the conversion of this coded value and continue processing. The DOM based method of converting correctly handles the conversion.

Release 22

Release 22 is released to fix read problems for internation characters (or in general double-byte characters). I needed to translate the strings from UTF-8 representation to double-byte. This was brought to my attention by Lubos Kopecny where he was experiencing incorrect strings when reading an existing xlsx file. I believe I have this now corrected. Another fix in this conversion was for the use of higher ASCII characters such as the registration mark or copyright mark – these were not displaying correctly (had an extra character); with this fix these characters now also display correctly.

Release 23

Release 23 is released to fix a number of bugs and code logic. The first bug is in the return value for float cell values. The second bug is when workbooks were deleted not all references were deleted and the internal Id values was not reset; this affected DeleteWorkbook() and DeleteAllWorkbooks() methods.

The code logic flaw was in how I was determining if a string is to be added to the internal strings table (the XLSX format specifies to add a single entry for a string into the strings.xml which is then referenced by Id). I was using the first 240 characters of the string (padded via PADR() function) to determine a match; however, if two strings are longer but share the first 240 characters, then this method fails. I am now using SYS(2007) function to return a CRC32 checksum value which is then concatenated with the first 230 (minus the length of the checksum value) characters of the string for the final checksum value and then stored into an indexed field in the strings cursor.

The following methods have been added:

InsertCell()	Inserts a new cell into the sheet
InsertColumn()	Inserts a new column into the sheet
InsertRow()	Inserts a new row into the sheet
SaveGridToWorkbookEx()	Enhanced version of method that will write directly to the XLXS format bypassing the internal cursors for a higher performance in creating a workbook
SaveTableToWorkbookEx()	Enhanced version of method that will write directly to the XLXS format bypassing the internal cursors for a higher performance in creating a workbook

Release 24

Release 24 is released to fix two bugs in the direct write methods added in Release 23. These methods were not writing accented characters correctly; the problem was how the sheet.xml file was being converted to UTF-8. Now fixed. The second bug was that I was referring to the wrong array column number for the field titles when exporting a table and the fields were not passed; now fixed.

Release 25

Release 25 is released to bugs in the direct write methods added in Release 23 and to incorporate some suggested changes.

In R24 I did not correct all the UTF-8 file conversions and I believe I now have this corrected – I created a separate method to do this and now use this method for all conversions.

I added a new property DefaultFontSize that is initially set to 11 for the XLXS font size value. You can now change this with the DefaultFont (name value). I should have named the property 'DefaultFont' as 'DefaultFontName' to be more consistent but did not change it so as not to break any existing code.

In the SaveGridToWorkbook() (older) method I have made changes suggested by Doug Hennig to incorporate the grid cell formatting into the output to the XLSX file. Also, a small performance enhancement was suggested by Doug Hennig in the SetCellValue() method that is incorporated.

In the SaveTableToWorkbook() and SaveTableToWorkbookEx() methods I have added code to get the field captions defined in a database table if available for the column titles; it will default to using the field names.

Both the SaveGridToWorkbookEx() and SaveTableToWorkbookEx() now uses the DefaultFont and DefaultFontSize property values to define the font definition for cells that are not numeric. Excel does not read the default font definition in the saved Styles.xml but will always default to the font defined by Excel (Tahoe, 11pt); apparently I cannot override this behavior in the XLSX file. I am saving the font definition as an in-line string format which can be only assigned for non-numeric cells.

I also now check for a cell data type that is numeric but formatted as a string value when opening a XLSX file. If this is found, the value is stored as a string value.

Release 26

Release 26 is released to fix a bug in the GetCellValue() method and in the SaveGridToWorkbookEx() / SaveTableToWorkbookEx() methods.

In the GetCellValue() method when a cell value was formatted in the workbook as a numeric format with decimal places, but the value was actually entered without any decimal values or decimal point, the method would cause an exception on the CAST() statement due to incorrect data type.

The SaveGridToWorkbookEx() / SaveTableToWorkbookEx() methods were not correctly handling the decimal separator and decimal point values for these settings when not set to a ',' and '.' respectively (this affects the European countries). I am now saving these settings, changing to USA standard settings and then restoring back to user defaults.

Release 27

Release 27 is released to incorporate a suggestion by Matt Slay for creating the spreadsheet column order based on the grid display order in the methods SaveGridToWorkbook() and SaveGridToWorkbookEx(). Also a bug in the SaveGridToWorkbookEx() method was found when saving the grid without the top row frozen (this would cause an unreadable spreadsheet). This is now fixed.

Release 28

Release 28 is released to fix a bug introduced by copy-n-paste from Release 27. Wben I added the code to output a grid to a spreadsheet in the SaveGridtoWorkbook() method, I copied it from the SaveGridtoWorkbookEx() method; however, I failed to initialize a variable. I read somewhere that most bugs are introduced by copy-n-paste and it would be better to always retype – but I usually take the copy-n-paste route…

I also added to the SaveGridtoWorkbook() and SaveGridtoWorkbookEx() methods to now not export columns that are hidden in the grid. This is controlled by a new parameter on these methods:

```
LPARAMETERS toGrid, tcFileName, tlFreeze, tcSheetName, tlInclHiddenCols

LPARAMETERS toGrid, txWB, tlFreeze, tlSaveWB, tcSheetName, tlInclHiddenCols
```

The value for tlInclHiddenCols will be defaulted to True which will result in the behavior before this addition; i.e., all columns will be exported. I also want to provide a special thank you to Matt Slay for suggesting/helping with this new feature and providing feedback in debugging.

Release 29

Release 29 is released to add several suggestions by Doug Hennig which restores the data environments. Additionally I added code to correctly open a spreadsheet that was created with in-line text which was identified by Micheal Hogan. I also did a small optimization in the SaveGridToWorkbookEx() and SaveTableToWorkbookEx() methods by moving the test for the default font and size (IF statement); previously this test was for each cell and now it only does it once for the sheet.

Release 30

Release 30 is released to add a suggestion by Doug Hennig to check the length of sheet names when creating a workbook via SaveGridToWorkbook(), SaveGridToWorkbookEx(), SaveTableToWorkbook(), and SaveTableToWorkbookEx() methods. Additionally, I added a check to remove any illegal characters from the sheet name in these methods (these checks were already in the AddSheet() method). I also added a new property AutoTrimSheetName which dictates where the sheet name will be automatically truncated to the maximum length or will fail the method; the default value is True (truncate name).

Release 31

Release 31 is released to correct some bugs in the class. Dan Lauer identified several issues: large integer handling and a bug in the AddSheet() method – both of these incorporated his solution. Doug Hennig also identified several issues when saving a grid directly to a workbook, then reopening the workbook and adding additional sheets. These issues have also been corrected.

The SaveGridToWorkbook() method behavior has been changed. Previously all columns were formatted to the column defined font name and font size. However, if the column contained date value type then the resulting output was a numeric value instead of the date (offset from 1/1/1900). So to correct this, I test for the date data type and only set the cell font formatting for non-date data types.

A new method was added as well – SaveMultiGridToWorkbookEx(). This method is similar to SaveGridToWorkbookEx() but allows for multiple grids to be saved to a workbook with one call as separate sheets. See the included form multigriddemo.scx for syntax on how to call. Note the loGrids object must have 4 columns defined. From the included example form's command1.click() event code:

```
LOCAL lcExcel, loGrids
lcExcel = SYS(5) + ADDBS(SYS(2003)) + "NorthWindMultiDemo.xlsx"
loGrids = CREATEOBJECT("Empty")
ADDPROPERTY(loGrids, "Count", 2)
ADDPROPERTY(loGrids, "List[2, 4]", "")
loGrids.List[1, 1] = thisform.Grid1          && Grid object reference
loGrids.List[1, 2] = "Customers"             && Sheet name
loGrids.List[1, 3] = .T.                     && Freeze top row indicator
loGrids.List[1, 4] = .F.                     && Include hidden columns indicator

loGrids.List[2, 1] = thisform.Grid2
loGrids.List[2, 2] = "Employees"
loGrids.List[2, 3] = .T.
loGrids.List[2, 4] = .F.

thisform.clsVFPxWorkbookXLSX.SaveMultiGridToWorkbookEx(loGrids, lcExcel)
WAIT WINDOW "Saved To Excel" NOWAIT
```

Release 32

Release 32 is released to add additional functionality and fix bugs that were identified by users. I have been away for awhile and have had a number of feature requests. These feature requests include hiding grid lines, hyperlinks, column/row grouping, sheet/workbook protection, filter support, setting the tab color on sheets, adding images, and determining the cell formatting from the grid properties. The insert row/column methods were enhanced to mimic the functionality of Excel's cell formatting of the new cells. I have also added support for opening existing macro enabled workbooks (xlsm) and then saving.

Special thanks to Doug Hennig for doing beta testing and providing valuable feedback to solve bug issues in this class. Additionally, some corrections to the documentation were provided by Rick Hodgin (thank you for your contributions).

I have re-written the open workbook methods to vastly improve the performance. To parse the XML structures, I changed from using the VFP command `STREXTRACT()` to a custom solution using `SYS(2600)` which creates a pointer to a string on the heap that I can then access very quickly for scanning/extracting characters. The idea of faster string handling originated on the Universal Thread in a thread started by Rick Hodgin; specifically using `SYS(2600)` came from Christof Wollenhaupt which I adopted. On my system with a specific sheet.xml file (41.5M in size), that is part of a larger workbook, I get the following results (times in seconds):

Selected Segment Extract: 3.370
All Segments Extract: 50.102
Segment Count: 20638

The methods for increased speed in processing the XML are `AddStringToHeap()`, `GetXMLFirstSegment()`, `GetXMLNextSegment()`, `GetXMLSegment()` and `RemoveStringFromHeap()` (all are set to protected in the class). I have created a custom class, `FastStrings.vcx`, that has these methods that can be used by anyone; this class is not needed for VFPxWorkbookXLSX class (methods are incorporated). I had tried to use DOM parsers but found these were generally slower than the built in `STREXTRACT()` function.

I changed the sequence of the code for processing the workbook. This was done to allow for a cleaner approach to opening the workbook and for future inclusion of objects.

I added the ability to read graphic image data for the sheets (as well as to insert via method) during the opening of an existing workbook. I have found that a large number of graphic images can take a while to process so I added a new parameter, tl`ReadGraphicData`, to control if graphic data (images and shapes) are also read into the working cursors for the workbook to the methods `OpenXlsxWorkbook()` and `OpenXlsxWorkbookSheet()` (new method). This parameter was added to reduce the read time of workbooks (to allow not reading graphical data).

I also found that I was not reading the tab color for sheets when opening the workbook; this is now corrected. Another issue that I found was when an existing spreadsheet had a cell error; I was not reading this (I was keeping the cell error value). This would cause an error detection by MS Excel for the cell and prompted a recovery of the workbook when the workbook is saved by VFPxWorkbookXlsx class. I have changed the read process to now not to add the error value of the cell but to leave the cell value empty and retain the cell formatting only (formulas are also retained).

I put the zip file handling code lines into separate methods, `OpenXlsxFileAsZip()` and `AddFilesToZip()`, in order to allow you to override these methods with your own code by subclassing this class. I saw a number of users were using other alternatives to these zip code lines and this way it is cleaner for them to implement their custom code.

The following methods have been added:

AddHyperLinkFile()	Adds a new hyperlink to an external file
AddHyperLinkSheet()	Adds a new hyperlink to another cell range within the workbook
DeleteHyperLink()	Deletes the selected hyperlink from the sheet
SetTabColor()	Sets the tab color of the selected sheet in the workbook
GetSheetIndex()	Gets the internal sheet index from the sheet name for a given workbook
ConvertRangeToColumnRowValues()	Converts a given range notation to row and column values
AddGroupByColumn()	Adds a column group level to the selection
AddGroupByRow()	Adds a row group level to the selection
UnGroupByColumn()	Removes a column group level from the selection
UnGroupByRow()	Removes a row group level from the selection
SetSheetGroupSettings()	Sets the row and column summary settings (roll-up or roll-down)
GetSheetProtection()	Returns the sheet protection settings in an object
SetSheetProtection()	Sets the sheet protection settings
AddStyleProtection()	Sets the style's protection values (locked and hidden)
GetWorkbookProtection()	Gets the workbook protection settings
SetWorkbookProtection()	Sets the workbook protection settings
CopyStyle()	Copies the style to a new style Id
AddImage()	Adds an image to the sheet
DeleteImage()	Deletes an image from the sheet
GetImageRelationshipId()	Gets the relationship Id for an image based on the workbook, sheet and position
GetImageDimensions()	Gets the image height and width dimensions for inserting into a sheet
GetDisplayGridLines()	Gets the display setting for showing/hiding grid lines in the sheet

SetDisplayGridLines()	Sets the display setting for showing/hiding grid lines in the sheet
OpenXlsxWorkbookSheet()	Opens a selected worksheet in a XLXS workbook; always sets the opened sheet as sheet1
AddAutoFilter()	Adds a filter to the column range
ClearAutoFilter()	Clears the column filter for the sheet
AddColumnFilter()	Sets the specific filter for a column
SetColumnHidden()	Sets the column hidden setting
GetColumnHidden()	Returns the column hidden setting

The following methods have been enhanced:

AddFilesToZip()	Internal method for adding the xml files to the zip file. Override this method if you want ot use an alternative way of creating the zip file from the xml files.
InsertColumn()	Retains the column cell formatting from the adjacent column (similar to column insert in Excel)
InsertRow()	Retains the row cell formatting from the adjacent row (similar to row insert in Excel)
OpenXlsxFileAsZip()	Internal method that opens the xlsx file that was copied to a temporary directory location (user's temp setting). Override this method if you want to use an alternative way of opening the zip file and extracting the contents.
OpenXlsxWorkbook()	Support for opening a macro enabled workbook (extension xlsm). A new parameter has been added, tlReadGraphicData, to control the processing of the graphic data (defaulted to True).
SaveGridToWorkbookEx()	Sets the cell format to the same as the grid column property settings. This includes font name, font size, font bold, font italic and format property settings. Added feature to add SUM() formulas to numeric columns (this feature relies on the *column.Tag* property set to the value "SUM").
SaveWorkbook()	Support for saving a macro enabled workbook (extension xlsm) that was previously opened with OpenXlsxWorkbook(). Note there is not any support to create a macro enabled workbook.

Release 33

Release 33 is released to correct a number of bugs identified in the SaveGridToWorkbookEx() when setting the cell formatting based on the Grid column's settings for Format and InputMask properties. Also, an error was found and corrected in the Style methods for setting cell formatting.

The following methods have been added:

GetStyleFormatId()	Gets the format style Id for the given numeric format, font format, and fill format. Will dynamically create a new style if it does not exist.

The following methods have been enhanced:

SaveGridToWorkbook()	This method is enhanced to now include dynamic formatting of cells based on the grid's column dynamic property settings (only DynamicAlignment is not yet supported); special thanks to Doug Hennig for the suggestion and starting code to apply these properties. The testing of dynamic properties is performed only for columns with these property values set.

Additionally, the documentation bookmarks has been updated to add the methods listed alphabetically as well as by function groups.