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, tlReadGraphicData, 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:

AddAutoFilter()	Adds a filter to the column range
AddColumnFilter()	Sets the specific filter for a column
AddGroupByColumn()	Adds a column group level to the selection
AddGroupByRow()	Adds a row group level to the selection

AddHyperLinkFile()	Adds a new hyperlink to an external file
AddHyperLinkSheet()	Adds a new hyperlink to another cell range within the workbook
AddImage()	Adds an image to the sheet
AddStyleProtection()	Sets the style's protection values (locked and hidden)
ClearAutoFilter()	Clears the column filter for the sheet
ConvertRangeToColumnRowValues()	Converts a given range notation to row and column values
CopyStyle()	Copies the style to a new style Id
DeleteHyperLink()	Deletes the selected hyperlink from the sheet
DeleteImage()	Deletes an image from the sheet
GetColumnHidden()	Returns the column hidden setting
GetDisplayGridLines()	Gets the display setting for showing/hiding grid lines in the sheet
GetImageDimensions()	Gets the image height and width dimensions for inserting into a sheet
GetImageRelationshipId()	Gets the relationship Id for an image based on the workbook, sheet and position
GetSheetIndex()	Gets the internal sheet index from the sheet name for a given workbook
GetSheetProtection()	Returns the sheet protection settings in an object
GetWorkbookProtection()	Gets the workbook protection settings
OpenXlsxWorkbookSheet()	Opens a selected worksheet in a XLXS workbook; always sets the opened sheet as sheet1
SetColumnHidden()	Sets the column hidden setting
SetDisplayGridLines()	Sets the display setting for showing/hiding grid lines in the sheet
SetTabColor()	Sets the tab color of the selected sheet in the workbook
SetSheetGroupSettings()	Sets the row and column summary settings (roll-up or roll-down)
SetSheetProtection()	Sets the sheet protection settings

SetWorkbookProtection()	Sets the workbook protection settings
UnGroupByColumn()	Removes a column group level from the selection
UnGroupByRow()	Removes a row group level from the selection

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 and to add some new features.

Corrections were made to the methods SaveGridToWorkbook() and SaveGridToWorkbookEx(). The first correction is the class's handling of the grid column properties for Format and InputMask. I have tried to retain the original intent of the formatting when exporting to Excel. So please report an issue if you find any problems when the grid column formatting is exported.

Another correction involves illegal characters that are not permitted in the spreadsheet tab title which I had been checking for (the code converts illegal characters to underscore). However, if the tab title

string passed had these illegal characters encoded as an xml equilvalent then it would cause an error on trying to open the spreadsheet. I have added additional checks to remove illegal characters if they are encoded. I also corrected the title being saved to be encoded for XML to properly encode valid characters such as the & symbol.

Another correction was made to SaveGridToWorkbookEx() for handling calculated field values in the grid.

Another correction/enhancement was made to ensure the temp directory name for the workbook create is unique and not used (suggestion by Doug Hennig).

Another correction was identified in the method SetCellBorderEx() and is now corrected.

Another correction was made in the AddFilesToZip() method for detecting Win10 OS provided by starfiresg1.

Another correction was made in the handling of cell formulas. Formulas can be stored in a shared mode in order to reduce the spreadsheet size (this is normal behavior by Microsoft Excel) when they are similar in definition and only differ by the column or row numbers. These formulas are then stored differently in the sheet.xml as compared to other non-shared formulas. Changes have been made to the class read methods to properly account for the shared formulas. This also impacted the GetCellFormula() method which had to be updated to retrieve the formula when it is shared. This class will not store similar formulas as shared – they are stored individually in the cells.

Another correction was made in the methods GetCellNumberFormat() and GetCellNumberFormatText(). Due to changes in the handling of the numeric formatting of cells from a previous release, I introduced an error to these two methods which was not caught during testing. These two methods now correctly return the values.

A suggestion by Doug Hennig on the saving and restoring of the SET POINT and SET SEPARATOR commands only if needed was added to increase performance in writing to the spreadsheet. Doug found out that these two commands have a high overhead in changing the settings. Excel stores numeric values internally with the US standard of a separator being a comma and the decimal being a period; hence, these settings have to be changed when the reverse in being used.

The following methods have been added:

DeleteCell()	Deletes the selected cell
DeleteColumn()	Deletes the selected column
DeleteRow()	Deletes the selected row
GetNamedRange()	Returns the specific named range within the workbook.
GetNamedRanges()	Returns all the named ranges within the workbook.
GetRowHidden()	Gets the row hidden setting

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.
SetRowHidden()	Sets the row hidden setting

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, new optional parameters have been added for specifying the sheet to output to (sheet must exist); and the beginning row and column to write the cell values to.
SaveGridToWorkbookEx()	New optional parameters have been added for specifying the beginning row and column to write the cell values to.
SaveTableToWorkbook()	New optional parameters have been added for specifying the sheet to output to (sheet must exist); and the beginning row and column to write the cell values to.
SaveTableToWorkbookEx()	New optional parameters have been added for specifying the beginning row and column to write the cell values to.

The storing of string values into the internal cursors has also been changed in this release. In order to maximize speed output for the grid and table to workbook methods, I write the string values directly into the sheet.xml as in-line text (instead of writing to the strings.xml and then a reference id in the sheet.xml). I have now incorporated this same logic into the methods that create and write to cell values. It is only on the reading of an existing spreadsheet that the strings.xml cursor will be populated; writing new string values to the cells will use the in-line method.

On my laptop (Lenovo Intel Core i7-4510 dual CPU @ 2GHz with 16G RAM), I am able to create a spreadsheet with 100,000 string cell values (10,000 rows by 10 columns) with a 31% reduction in time as compared to using the strings.xml storage method. See the sample program StringsTimeTest.prg to test the difference between Release 33 beta 9 and Release 33 beta 10 (you will need to adjust the paths to the classes for the NEWOBJECT command.

Additionally, the documentation has corrections added for some of the methods, and the PDF bookmarks has been updated to add the methods listed alphabetically as well as by function groups.

Release 34

Release 34 is released to correct a number of bugs identified and to make some improvements. The primary documentation for the class needs to be updated as I have found some errors and omissions. Check the class for the methods and parameter usage.

Bugs corrected in this release:

- When an existing spreadsheet is imported into the internal cursors and the sharedstring.xml file indicates that the string is to retain whitespace (but not formatted). This is read in correctly; however, when writing the string back to the sharedstring.xml file, the method WriteStringsXML() was not writing the value from the correct cursor (was incorrectly selecting the value from the xl_strformat cursor instead of the xl_strings cursor.

- When the xlsx file to be created was located on a share drive. Apparently, the windows code for the zip creation would fail. Doug Hennig discovered this and suggested writing the zip file to a temporary location on the local machine and then using the move API command to move it to the correct location with the xlsx extension.

- Incorrect writing of the sheet name to the workbook.xml file. I check for invalid characters that are in the sheet name (as defined by MS Excel; the ECMA Open Office does not provide much guidance that I found) but I was not converting the string into the proper xml format for writing to the workbook.xml file. As such, when a user would enter the sheet name using the < or > characters (which are valid per MS Excel), I was not writing those characters using the proper escape sequence in the xml file. Also, when reading the sheet name, I did not convert it from the xml escaped sequence to text – it was stored 'as-is' into the xl_sheets cursor. Both of these circumstances have been corrected to read and write the correct string type.

- When a Named Range is assigned to be local to a spreadsheet, I was incorrectly setting the internal sheet id which is required to be zero based index; this is corrected. This was identified by "zulqasar" as an issue on GitHub.

- The RETURN statement in the method GetWorkbookFileName() had been somehow removed from Release 31 (affected releases 32 onward); the RETURN statement has been restored. This was identified by "CHamlinPNP" as an issue on GitHub.

- The handling of the currency symbol was hard coded to use the $ sign in the save grid method; this is corrected to now use the VFP setting for the currency symbol. This was identified and solution provided by Doug Hennig.

Improvements added in this release:

- The handling of the theme attribute in cell fill color definitions for existing spreadsheets. This attribute was being ignored during the reading of an existing spreadsheet which caused the cell fill color values to not be set during the save workbook method. Subsequently, the spreadsheet when opened would show the affected cells as a black filled cell. The cell fill color theme attribute is now being stored and written for an existing workbook.

- Doug Hennig also provided improved handling of numeric field value formatting when exporting a grid to xlsx file. The decimal symbol was being incorrectly applied everytime when the grid formatting should not include the decimal. Also, blank values are being written now for grid formatting that should be blank for a zero value.

- Added a new optional parameter to the SetCellValue() method. This parameter is tlAppend and is used for appending the value to the existing cell text if the cell text is character; otherwise, this parameter is ignored.

- Added checking for reserved names when assigning Named Ranges. This was identified by "zulqasar" as an issue on GitHub. Reserved names are now stored with the correct internal prefix of "_xlnm.".

- Added additional rules to sheet naming for invalid names.

In this release I have also made a significant change to how an existing workbook is being read in. Before the code would re-read the xml files multiple times to extract segments. Now the read process only reads the xml file once and stores the segments into a memory object. This object is then processed to assign the xml segment information into the cursors. In my current testing with the workbook created by the Demo() method (a mult-sheet workbook), the original open method code took approximately 19 seconds to process; with the new code the workbook processing now takes approximately 8 seconds.

Release 35

Release 35 is released to correct a bug identified in the freeze rows or column method. This bug did not cause a problem unless the workbook just created by the class was read in and then subsequently saved again. This caused the header lines to be hidden by repeating over with the scrolling lines (caused by the Split keyword). This has now been corrected.

The following methods have been added:

ClearPrintArea()	Clears the print area for the selected sheet
GetPrintArea()	Gets the print area for the selected sheet
SetPrintArea()	Sets the print area for the selected sheet

The setting of the print area was available via the AddNamedRange() method; however, these methods should make it easier to manipulate the print area for a sheet.

Release 36

Release 36 is released to correct a bug identified by Doug Hennig when opening a workbook and then subsequently saving it. The strings were being corrupted during the open processing of the sheet cell values. This specific bug occurred due to numeric values being formatted as a text value. Excel does not store the numeric value as a string but rather only marks it as a string formatted value. However, the VFPxWorkbookXlsx class method for processing the sheet xml was storing it as a text value into the xl_strings cursor and assigning a string Id. These added string Ids were then overlapping with the existing Ids from the sharedstrings.xml file causing the corrupted sequence of Ids. This has now been corrected by also storing cells formatted as a string value by format id and not by string id.

Release 37

Release 37 is released to correct a bug identified by Thierry Penninckx. Thierry saved a cursor directly to a workbook using the method SaveTableToWorkbookEx(). This method is designed to write everything directly to the sheet1.xml for speed; so it does not create a styles.xml file nor does it create the sharedstrings.xml file. After the workbook was created, it was then opened by the method OpenXlsxWorkbook() and subsequently changed (row inserted, values assigned, and cell formatting) and resaved. Excel would then report the workbook in error and perform a repair.

So, when the workbook was now read-in to the class, no basic styles definition was defined (the CreateWorkbook() method does perform this). The inserting of the row and assignment of the cell values made this apparent due to my defaulting of the cell format index. This defaulted index points to the styles.xml file which did not exist. Excel raised the error and removed the style information from the affected cells during the repair. Therefore, I made several corrections – the checking for the styles based definition when creating a new style to be applied and how the format index is assigned when the values are assigned. Note that if you use the older depreciated methods for assigning cell formatting, this checking for the styles.xml base definition is not performed, and will cause an error. I currently have no plans to fix these older methods.

Release 38

Release 38 is released to correct date-time format codes that was identified by Cesar Chalom. I was not adding the locale code identifier which was causing incorrect display of the formatting; this is now checked and added as needed. I also corrected the Demo() method examples to now use the current styles methods for cell formatting; it was using the deprecated methods.

Release 39

Release 39 is released to correct a bug in the SaveTableToWorkbookEx() method. Additional new methods were also added for defining conditional cell formatting.

The workbooks created with SaveTableToWorkbookEx() method were able to be opened with Excel; however, if you tried to add a new sheet to the workbook an error would be displayed resulting in not being able to add the sheet. You could do a ‘save as’ or make a change and then save, close Excel, and then reopen the workbook to correct the problem (which is not good…). This error of adding a sheet was caused by the freeze row attribute setting; I have now corrected this problem.

Another bug was discovered by RezaUzer where I had a copy-n-paste error – the DeleteColumn() method had the parameter value for sheet as tnSheet; whereas, the method code used tnSh. This has been corrected.

During helping another user in opening an existing workbook and making changes to it, I discovered a bug in saving the workbook tab names. I needed to convert the spreadsheet workbook.xml file from UTF-8 to the current codepage setting. I have made the corrections that affected both OpenXlsxWorkbook() and OpenXlsxWorkbookSheet() methods.

Color scale, bar, and general conditional cell formatting is now supported (icon set formatting is not yet supported). The conditional formatting methods are (see the main documentation for detailed information on the parameters):

AddBarConditionalFormatting()	Adds a bar type conditional formatting
AddColorScaleConditionalFormatting()	Adds a color scale type conditional formatting (2-color or 3-color)
AddConditionalFormatting()	Adds top/bottom/greater than/less than, formula based conditional formattting

New methods for assigning Table Formatting were also added:

AddTableFormatting()	Adds a table formatting to a range of columns/rows
AddTableFormatColumn()	Adds the column definition to the table format
AddTableFormatColumnFormula()	Adds a formula definition to the table format
AddTableFormatColumnLabel()	Adds a Column Label in the Totals Row to the table format
ClearTableFormatting()	Clears the specified table formatting
SetIgnoreWarnings()	Sets the cell ignore warnings indicator

The Demo() method shows the usage of these three conditional formatting methods and the Table Formatting methods in new sheets.

The following methods have been added:

SetSheetRightToLeft()	Sets the sheet to be right-to-left or left-to-right. Default setting is Left-to-Right.
SetSheetShowZeros()	Sets the sheet to show or hide zero values in cells. Default is True (show values).

The following methods have been enhanced or changed:

AddAutoFilter()	New parameter was added for specifying the row for the auto filters (defaults to 1)
AddHyperLinkFile()	Changed by removing the check for the file existing; this check prevents adding a hyperlink for a file that exists on an Internet URL.

AddSheet()	Two new parameters have been added. The first parameter is to support Right-to-Left layout. If set to True will set the sheet property for right-to-left layout; default is left-to-right (False). The second parameter is to support the display setting for cells with zero values; if set to True (default) will display zero values, set to False will hide zero values.
ReadSheetXML()	Loading of an existing workbook with Auto Filters set will be read in for each sheet and subsequently saved. The sheet attributes for Right-to-Left and Show Zeros are also being processed.
SetColumnBestFit()	Is now working and will be applied to the columns specified in each sheet.
SaveGridToWorkbook()	Two new parameters have been added. The first parameter is to support Right-to-Left layout. If set to True will set the sheet property for right-to-left layout; default is left-to-right (False). The second parameter is to support the display setting for cells with zero values; if set to True (default) will display zero values, set to False will hide zero values.
SaveGridToWorkbookEx()	Three new parameters have been added. The first parameter is to set the table format; however, the totals row is not added to the table format. The second parameter supports setting the sheet to Right-to-Left (if value is True; default is False). The third parameter is to support the display setting for cells with zero values; if set to True (default) will display zero values, set to False will hide zero values.
SaveTableToWorkbook()	Two new parameters have been added. The first parameter is to support Right-to-Left layout. If set to True will set the sheet property for right-to-left layout; default is left-to-right (False). The second parameter is to support the display setting for cells with zero values; if set to True (default) will display zero values, set to False will hide zero values.
SaveTableToWorkbookEx()	Four new parameters have been added. The first parameter allows the columns to be best fitted. The second parameter sets the table format (again, totals row is not added). The third parameter supports setting the sheet to Right-to-Left (if value is True; default is False). The fourth parameter is to support the display setting for cells with zero values; if set to True (default) will display zero values, set to False will hide zero values.

SetCellValue()	Empty date values are not saved. I found that if an empty date value was saved, Excel would display the cell with the value “ / / “ instead of a blank value.

A new utility method was also added:

ConvertColumnRowValuesToRange()	Converts the numeric begin column/row and end column/row values to range notation

A new class has also been added to the class library VFPXWorkbookXLSX.vcx. This is a VFP Report listener class, Xlsx_Listener, for creating workbooks directly from a Report (frx). This listener does not support Label reports (multiple columns). This class is currently under development and currently supports the following report bands:

- Title band
- Page Header band
- Group Header band
- Detail Header band
- Detail band
- Detail Footer band
- Group Footer band
- Page Footer band

The report objects that are written to the workbook are labels (objtype=5) and fields (objtype=8); all other objtypes are ignored. If the group header band is configured to start on a new page, the listener will start the group in a new sheet. The column layout is determined from the Detail band object layout. Formatting is also supported for the object font name, size, fore color, fill color, bold and italic settings.

The Page Footer band sets the page footer for print-out of the workbook. There are three regions that the workbooks support, left side, center, and right side; the report objects will be assigned to one of these regions based on the horizontal position in the band; i.e., left <33% of width, center >33% and <67% of width; right >67% of width.

Reports that have more than one column are not being supported at this time. I plan on supporting multi-detail bands; however, this has not yet been tested to see how it will output.

There are some listener properties that can be set prior to calling the report; see the main.prg that is included in the zip archive. There is also a project included that I am using to test the class as I develop it. It must be run from the APP file to be demonstrated. Full documentation will be provided once this class is completed. The listener include files must be added to your project.

I would appreciate any feedback or problems that are encountered during your testing. If a problem is encounted, please set the listener property DebugMode=.T. and rerun the report. This will output a table file with the same name as the report (FRX) file. Please send the report (frx) files, the outputted table files (DBF), and the generated workbook (xlsx) file when you report any issues/bugs. Note that the outputted table file will contain the same field content as the report; if there is any sensitive data contained, please replace the data in the field cellvalue before sending to me.

Release 40

Release 40 is released to add the ability to get a cell value for a cell that has a formula expression. This cell value is only available when the spreadsheet has been calculated by a spreadsheet program; the create of a cell formula does not assign the cell value.

Release 41

Release 41 is released to correct defects reported by users in the issues log. No new functionality has been introduced. However, some existing methods have been enhanced to either correct behavior, add additional capability, or faster performance.

When inserting or deleting rows, columns, and cells, the row/column/cell would be inserted or deleted; however, the adjustment to affected merged cells was not correctly accounted for. The behavior of Excel during inserting or deleting of single cells that affects merged cells is now being duplicated.

When reading an existing workbook into the class via the OpenXlsxWorkbook() method, the existing header and footer definitions was not being retained. The OpenXlsxWorkbook() method now stores the existing header and footer definitions. These settings are written back when saving the workbook.

The method SaveTableToWorkbook() has had more optimization in the code to generate a direct workbook. This method originally would read the table into the class internal cursors and then save the workbook which is a two-pass processing of the data. Now the reading of the table into the cursors and the writing to a workbook occurs in the same sequence of code (single-pass processing). This has reduced the overhead significantly to generate the workbook directly using this method and allowing for the storing of the workbook data for further manipulation. However, the SaveTableToWorkbookEx() method is still the fastest at creating the direct workbook taking around 10 percent of the time of SaveTableToWorkbook() method.

The VFPxWorkbookXlsx class had stored the current system date time in the workbook properties and designated it with a ‘Z’ at the end. However, this was not necessarily the correct time since the actual offset for the timezone is not accounted for. The offset for the system date is now determined and added to the date time setting.

The reading of image placement definitions when opening an existing workbook did not retain the extended segment definitions in the graphic.xml (these extended segment definitions are not documented in the “ECMA Office Open XML Part 1 - Fundamentals and Markup Language Reference” and hence, were not retained). These omissions were causing Excel to reject the graphic.xml file as a whole in some cases. The reading of these definitions is now stored as a whole and re-applied during the writing of the graphic.xml file. The method used for writing back does not prevent being able to add additional images to the sheet.

The methods InsertCell(), InsertRow(), and InsertColumn() have been enhanced with an additional optional parameter for the count of cells, rows, or columns to insert; the default is 1 cell/row/column.

The methods SetPrintFitToHeight() and SetPrintFitToWidth() have been corrected to also set the fittopage setting. The WriteSheetXML() method has been corrected to assign the fitToPage attribute value when setting page width/height settings.

The AddNamedRange() method has been corrected to check for the keyword “PRINT_TITLES” instead of “PRINT_TITLE” (missing S at end).

The SetPrintArea() method has been fixed to allow setting different print ranges for different sheets. It was incorrectly overwriting a previous sheet setting with a new sheet setting.

The OpenXlsxWorkbook() method has been enhanced to read the worksheet protection settings. The class was reading the workbook protection settings; however, there were errors in reading not all the necessary attribute names. Additionally, the WriteWorkbookXML() method has been corrected to write the full list of attribute values for the workbook protection settings.

The values for hashValue, saltValue, and spinCount have been added to the return structure for the method GetSheetProtection(). These must be assigned to the structure when using the SetSheetProtection() method. The structure value toProtection.Password is no longer used; however, if it is assigned and the toProtection.HashValue is not assigned, then the password value is assigned to the hashvalue field. The SetWorkbookProtection() method parameters were increased to account for the required attributes that were missing. The return value from GetWorkBookProtection() method is changed to return a structure object instead of a single boolean value. The WriteSheetXML() has been corrected to write the proper attributes for the sheet protection.

The SetWorkbookProtection() method allows for either two parameters passed or all seven parameters to be passed. This was done to allow for you to remove the workbook protection by just passing the workbook Id value and False (to remove lock). Just passing the second parameter as True, will not assign the protection unless all the hash fields are also assigned.

I tried to create a method that would generate the hash values for the password but have not successfully been able to do this without Excel rejecting the protection settings being assigned. So, I have decided not to provide a means to generate the password hash. The class will allow you to assign the protection; however, you must provide the hash values that are required; i.e., algorithmName, hashValue, saltValue, and spinCount. If you want to assign protection, I suggest creating a spreadsheet in Excel with the desired password and then opening the created xlsx file with a zip tool and use the values that are assigned to the sheet.xml or the workbook.xml files for these attributes. Additionally, I have removed the property DefaultPW from the class.

Release 42

Release 42 is released to add a new method and fix a bug that was introduced for the styles.xml alignment setting for formatting.

On the Foxite forum, one user had a need to be able to create a spreadsheet with the rows and columns transposed. I cloned the method SaveTableToWorkbookEx() and created the new method SaveTableToWorkbookTranspose(). I then made the changes to output the rows and columns transposed. See the user documentation for more information.

The alignment bug was introduced by the changing of the logic for writing the horizonal and vertical cell alignment, the indent setting, the wrap text setting, and the rotation setting when creating the styles.xml in the previous release. This is now corrected.

I also added a new property, TemporaryPath, that defaults to =SYS(2023). You can set this another file location of your choice. All the class temporary files will then be written to that location.

Release 43

Release 43 is released to add new methods for readonly access to workbooks, speed improvements in opening workbooks for read-write, and for fixing several bugs.

A new method, OpenXlsxWorkbookEx(), for opening a workbook in readonly was added. This method is considerably faster than the OpenXlsxWorkbook() method for opening workbooks. The speed difference is due to the following:

- Only getting the stored value in the cell and not doing any data type determination (all cell values are stored as character data type); the string value is retrieved from the sharedstrings.xml file for each cell
- Not reading any other definitions of the workbook.

Since there is no conversion of the stored cell values, if the cell value is a date, datetime, or time value, then the value is stored as a numeric offset (how Excel handles these data types). Therefore, the programmer will need to know what the data type is for each cell in order to consume it correctly. To correctly translate the date, datetime, or time cell values three additional methods have been provided to assist in the conversion to the appropriate data type (the programmer has to call these methods in their code). The new added methods are:

OpenXlsxWorkbookEx()	Opens a workbook in readonly mode. Returns the workbook Id.
GetCellValueEx()	Gets the selected cell value from the readonly workbook. Returns NULL if a value is not assigned to the cell.
ConvertCellValueToDate()	Converts the cell value retrieved using GetCellValueEx() to a Date value.
ConvertCellValueToDateTime()	Converts the cell value retrieved using GetCellValueEx() to a DateTime value.

ConvertCellValueToTime()	Converts the cell value retrieved using GetCellValueEx() to a Time value.

The OpenXlsxWorkbookEx() stores the cell values in the cursor `xl_sheetcells` which can be processed using SCAN-ENDSCAN or you can use the new method GetCellValueEx() to retrieve the cell values. The structure of this cursor is:

```
CREATE CURSOR xl_sheetcells CODEPAGE = (this.CodePage) (workbook I, sheet I, row I, col I, cellvalue M)
INDEX ON BINTOC(workbook)+BINTOC(sheet)+BINTOC(row)+BINTOC(col) TAG cellindex
```

The method OpenXlsxWorkbook() also has improvements in the speed of opening a workbook. My testing on large workbooks the new code is able to open the workbook approximately 60% faster; my testing on small workbooks did not see a noticeable change in speed.

A bug was also found by Sergei when opening a workbook using OpenXlsxWorkbook(). The attribute value for the BestFit for a column was being defaulted to a NULL value when it was not set which caused an exception when saving to the internal cursors since a Boolean data type was expected. This same error was also found in reading other attribute values that expected a Boolean data type in other parts of the code. These assignments of a Boolean data type have been fixed.

The methods SaveTableToWorkbookEx() and SaveTableToWorkbookTranspose() have been updated to ignore Blob and General field types when saving to cell values (correction submitted by J Hernan Canom).

Release 44

Release 44 is released to correct an oversight on my part in the method OpenXlsxWorkbookEx(). I neglected to take in account cells having a formula value and cells having in-line string values. Additionally, I changed the parameter line to now provide an indicator on whether to return a cell formula or the calculated value (if assigned); this new parameter value is added as the third parameter. Hopefully, this parameter line change will not inconvience anyone too much since this is changed a day after the initial release. Additionally, I missed the possibility of the text strings having been converted to XML compatible text and needed to be converted back to normal text; this also has been fixed.