The OpenXML Libraries

Table of Contents

Introduction 3

Generating Excel Files 4

Sample 1 5

Template Generator 11

Sample 1-1 12

ttMeta TEMP-TABLE 14

ttMeta Options 14

Sample 1-2 15

Additional Excel Samples 16

Useful Excel Tips 17

Managing Named Ranges 17

Freeze Panes 17

Repeat Header Rows (for Printing) 17

Converting and Printing 18

Converting 19

Sample 3 19

Printing 20

Sample 4 20

Open Office Compatibility Issues 21

Right-to-Left Issues 21

Introduction

I'm sure you've all seen by now the Microsoft Office 2007 (Microsoft Office 2010 and going forward) new default file formats .XLSX, .DOCX, .PPTX etc. files but I don't know if you know that if you rename the file and add a .ZIP to the end of the file name, you can open the file and see that it's a ZIP file with XML files inside it.

The Microsoft Office 2007 new default file formats are a set of XML/markup languages: a SpreadSheet markup language used for creating Excel files, a WordProcessing language for Word, a Presentation markup language for PowerPoint etc. and a set of conventions how the XML files are packaged inside the ZIP file, that are collectively called Office Open XML or OpenXML for short or just the OOXML acronym.

When I first set out to write the OpenXML Libraries above everything else I wanted to write a solution that will let me do the design in a visual tool and write the queries in Progress, and that's still the idea how to use the OpenXML Libraries: do the design in Microsoft Office and merge the data in Progress.

Note: *OpenXML is supported by basically every modern Office suite out there. Microsoft Office 2000 and 2003 (through a freely downloadable service pack), Open Office (starting with version 3.0), LibreOffice, Google Docs, Apple iWorks, IBM Lotus Notes, Corel WordPerfect and many more. I can even open OpenXML files on my iPhone.*

You can also use Open Office or another Office suite to design and open your OpenXML files.

Generating Excel Files

Excel is ideal for working with tabular table like data that can also be linked to charts, pivot tables etc. but if you ever try to create forms like invoices, sales orders, letters etc. in Excel you'll realize very quickly you need to use Word to create forms.

To generate an Excel file you always start by designing a template (or downloading one off the internet) with table like sheet(s) and filling them with several rows of sample data (usually 10 or so rows). Although filling the sheet with sample data is optional, it lets you see how the cell formats, styles etc. will look like with values in them, add and check calculated columns and summaries, link charts and pivot tables etc.

Note: Templates in this context refer to how the file is used. Use a regular Excel Workbook (.XLSX) file, there's no need to save the file as an Excel Template file (.XLST).

The current version of the OpenXML Libraries only supports regular Excel and Word files (.XLSX, .DOCX). Support for template and macro enabled Excel and Word files (.XLST, .XLSM, .DOCT, .DOCM) is planned to be added.

The only thing left to do to finish the template is name the columns using named ranges (see Maintaining Named Ranges in the Useful Excel Tips section) so they can be mapped to the fields in your Progress query. If the column names and the query field names are the same, mapping is done automatically.

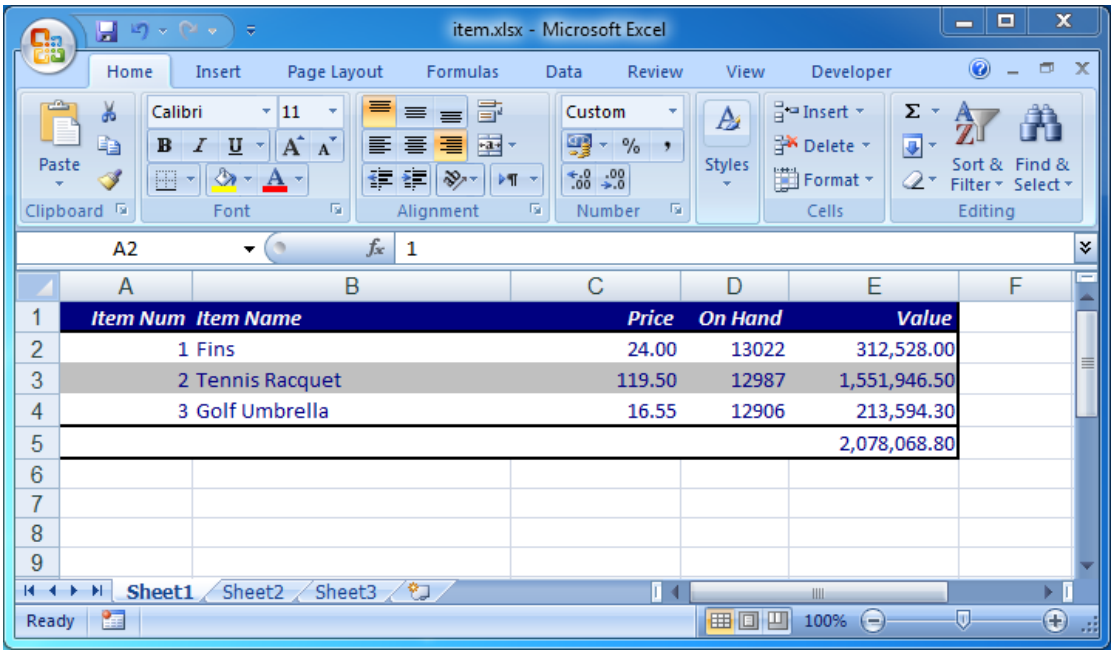
On the Progress side use the OpenXML Libraries to create a new Excel file from the template you just created and replace the sample data with data from a Progress query (it will actually delete the sample data rows and insert the rows from the query) while still keeping the sheet formats and styles, updating cell and range references, recalculating formulas, refreshing chart and pivot tables etc. (charts and images placed under the sample data will be pushed down and those placed to the side will not).

Note: The current version of the OpenXML Libraries supports replacing only one query per sheet. Support for replacing multiple queries in a single sheet is planned to be added in the next release.

Sample 1

In this sample you'll be generating an Excel file for the Item table from the Sports2000 sample database.

You always start by designing your template. Follow the steps below to create the template in the picture.



1. Create a new blank Excel file.
2. Add the values "Item Num", "Item Name", "Price", "On Hand" and "Value" in cells A1 to E1 respectively to add the column labels.
3. Add 3 rows of sample data as shown in the template picture and leave the "Value" column empty. The "Value" column is calculated and not filled.
4. Add the formula "=C2*D2" in cell E2 and copy the formula to cells E3 to E4 to add the calculated "Value" column.
5. Add the summary "=SUM(E2:E4)" in cell E5 to add a summary for the "Value" column. That's it for the data the rest of the steps deal with styling the document.
6. Add the number cell formats "#,##0" for cells A2 to A4, "#,##0.0##" for D2 to D4 and "#,##0.00" for C2 to C4 and E2 to E4 and E5.
7. Add an outside border around cells A1 to E1, A2 to E4 and A5 to E5 for the column labels, data and summary rows.
8. Add a black background and a white foreground color for cells A1 to E1 and a gray background for A3 to E3 for the column labels and an alternating data row.

9. Resize your columns to fit the sample data. You can always go back and readjust the template column sizes.

10. Last thing left to do is add the named ranges "ItemNum", "ItemName", "Price" and "OnHand" for the ranges A2 to A4, B2 to B4, C2 to C4 and D2 to D4 respectively.

11. Save the file as "item.xlsx" in the working directory or anywhere on the PROPATH. For example: C:\OpenEdge\WRK\item.xlsx.

Note: The current version of the OpenXML Libraries does not support automatically resizable columns. Support for automatically resizable columns is planned to be added to the template generator (see the Template Generator section) in the next release.

Note: When sorting a sheet in Excel how the sheet was sorted is not saved in the Excel file. The OpenXML Libraries cannot sort the query in the same way it was sorted because that information is not available.

To tell the OpenXML Libraries to sort the query while designing your template add .Sort<n>[.Descend] to the end of the column named range. For example: if a template has an Order and Price column named ranges and you'd like to sort the query by Order and descending Price change the named ranges to Order.Sort1 and Price.Sort2.Descend.

If you're using or plan to use .Sort<n>[.Descend] for sorting, use dynamic queries (opened using the :QUERY-PREPARE and the :QUERY-OPEN methods) instead of static queries (opened using the OPEN QUERY statement) because Progress does not save the query phrase (in the :QUERY-PREPARE attribute) that the OpenXML Libraries use to change the query phrase sorting expression and reopen the query.

On the Progress side copy and run the sample program below.

```
/* sample1.p */

{slibooxml/slibxlsx.i}

{slib/slibos.i}

{slib/sliberr.i}

define var cError       as char no-undo.
define var cErrorMsg    as char no-undo.
define var cStackTrace  as char no-undo.



{slib/err_try}:

    run xlsx_copyTemplate(
        input "stXlsx",
        input "item.xlsx" ).

    run xlsx_replaceLongRange(
        input "stXlsx",
        input buffer Item:handle,
        input "ItemNum   = Item.ItemNum"
           + ",ItemName  = Item.ItemName"
           + ",Price     = Item.Price"
           + ",OnHand    = Item.OnHand"
        input "",
        input "" ).

    run xlsx_save(
        input "stXlsx",
        input os_getNextFile( session:temp-dir + "item_new.xlsx" ) ).

{slib/err_catch cError cErrorMsg cStackTrace}:

    message
        cErrorMsg
        skip(1)
        cStackTrace
    view-as alert-box.

{slib/err_end}.
```

Explanation

The slibooxml/slibxlsx.i is the OpenXML Library for generating Excel files and the only one from the 3 include files that is required for generating Excel files.

The slib/slibos.i is a general purpose Operating System Library that the sample program later uses (see the os_getNextFile function below).

The slib/sliberr.i is a Structured Error Handling Library compatible with Progress version 9 that uses thrown exceptions and Try/Catch/Finally blocks.

In the sample program if the OpenXML Libraries procedure encounters an error (for example: if the template file is not found) the program will leave the Try block (after {slib/err_try}:) and jump to the Catch block (after {slib/err_catch}:). The Catch block in the sample program displays the error message and a stack trace of where the error happened.

The slib/sliberr.i is optional. If slib/sliberr.i and the Try/Catch/Finally blocks are not used and the OpenXML Libraries procedures encounter an error, they will RETURN ERROR with a RETURN-VALUE of the error message.

In general the OpenXML Libraries takes a more relaxed approach towards errors caused by users designing the Template and a more strict approach towards errors caused by developers writing the Progress code. For example: if a column named range is written incorrectly. the OpenXML Libraries will simply not fill that column and no error will be raised on the other hand if a buffer field is written incorrectly, an error will be raised and the file will not be generated.

The slib/slibos.i and slib/sliberr.i libraries are part of the Progress STandard Libraries (STL) open source project that can be downloaded at http://www.oehive.org/project/lib.

The Progress STandard Libraries (STL) is a huge project that includes among others libraries and utilities for ZIP, HTTP/S, Win API, Timing Events, Code Parsing, Backup/Archive/Restore, Query Optimization, Salesforce.com Integration, Google API and many more. If you're looking for something, look in the Progress STandard Libraries (STL) first.

The OpenXML Libraries uses the Progress STandard Libraries (STL) and a minimal version is included with the product in the slib/ directory.

All the libraries are actually persistent super procedures and not include files. The include file is used to launch a single persistent procedure for the entire session if one is not already running.

As you can see from the sample program it only takes 3 procedures to generate an Excel file.

1. The first xlsx_copyTemplate procedure creates a new Excel file by copying an existing template and has the following parameters:
    1. Stream name: The first parameter in all the OpenXML Libraries procedures is a stream name. Streams allow you to work on multiple files at the same time by using different streams. In most cases this option is not needed and the "stXlsx" stream name is used for generating Excel files.
    2. Template file name: If a relative path is passed, the procedure will search for the file on the PROPATH.

2. The main xlsx_replaceLongRange procedure does most of the work and replaces the sample data with data from the Progress query and has the following parameters:

    1. Stream name: See the previous procedure.

    2. Data source handle: QUERY, TEMP-TABLE, DATASET or even a BUFFER handle to import an entire table.

       To replace multiple sheets data either run the xlsx_replaceLongRange procedure multiple times or pass a DATASET to replace the sheets data for every TEMP-TABLE in the DATASET. TEMP-TABLES's with a DATA-RELATION are joined and treated as a single query.

       A comma separated list of BUFFER, QUERY or TEMP-TABLE handles can be passed as a Progress version 9 alternative to an OpenEdge 10 DATASET.

    3. Named range and buffer field mapping: A comma separated list of <column named range> = <buffer field>, …

       If the column named ranges and the buffer field names are the same, mapping is done automatically. You can also use the buffer.field notation for column named ranges if the field name is ambiguous and there is more the one buffer with the same field name.

       In the sample program (and in most cases) the column named ranges and the buffer field names are the same and the mapping is not required and only added for demonstration purposes. Remove the mapping by either entering a blank "" or null ? value and run the sample program again.

       The xlsx_replaceLongRange procedure uses the mapping to find the Excel sheet the Progress query is mapped to.

    4. Buffer can-do list: The buffer and field can-do lists are a security feature because mapping is done automatically if the column names and query field names are the same, users changing the template can add buffers and fields that may be restricted. Remove the buffer and field restrictions by entering a blank "" or null ? value.

       For example: "!_file,*" to exclude the _file buffer. Note that the ",*" suffix is required for specifying all buffers except _file (for more information see the Progress help on the CAN-DO function).

    5. Field can-do list: See buffer can-do list:

       You can also use the buffer.field notation in the field can-do list if the field name is ambiguous and there is more than one buffer with the same field name.

       For example: "!*rowid*,*" to exclude all fields with "rowid" in the field name or "ItemNum,ItemName,OnHand,Price" to include a specific field list.

3. The last xlsx_save procedure saves the file out to disk and has the following parameters:

    1. Stream name: See the previous procedure.

    2. New file name: If a relative path is passed, the procedure will save the file in the client working directory.

Microsoft Office locks the file while it is being edited. If the OpenXML Libraries tries to save to an existing file that is being locked it will fail causing an error. The os_getNextFile function (from the slib/slibos.i library) adds a counter to the file name if the file already exists. For example: item(2).xlsx.

Template Generator

You can use the template generator to generate a template for a Progress query the first time. You can than make changes to the template, remove columns, add calculated fields, summaries, charts, pivot tables etc. and use that as the template the next time you generate an Excel file for the Progress query.

The template generator generates a table like sheet for a Progress query, adds column labels, converts Progress formats to Microsoft Office cell formats etc. and adds the column named ranges so the Excel file can be mapped to the Progress query.

Note: Be aware that there is an additional overhead associated with using the template generator instead of using an existing template because of the additional step of generating a new template before using it. Try to avoid always generating a new template if possible.

Sample 1-1

```
/* sample1-1.p */

{slibooxml/slibxlsx.i}

{slib/slibos.i}

{slib/sliberr.i}

define var cError       as char no-undo.
define var cErrorMsg    as char no-undo.
define var cStackTrace  as char no-undo.



{slib/err_try}:

    run xlsx_createTemplate(
        input "stXlsx",
        input "samples/templates/blank.xlsx",
        input buffer Item:handle,
        input "",
        input "ItemNum,ItemName,Price,OnHand" ).

    run xlsx_replaceLongRange(
        input "stXlsx",
        input buffer Item:handle,
        input "",
        input "",
        input "" ).

    run xlsx_save(
        input "stXlsx",
        input os_getNextFile( session:temp-dir + "item_new.xlsx" ) ).

{slib/err_catch cError cErrorMsg cStackTrace}:

    message
        cErrorMsg
        skip(1)
        cStackTrace
    view-as alert-box.

{slib/err_end}.
```

Explanation

The sample program continues the previous sample but instead of using an existing template that you've created manually a template will be generated automatically using the template generator.

As you can see there are still only 3 procedures used but the xlsx_copyTemplate procedure is replaced with the xlsx_createTemplate procedure.

The template generator xlsx_createTemplate procedure generates a new template for a Progress query and has the following parameters:

1. Stream name: See the previous procedure.

2. Base template file: The base template is copied and used as the basis for generating the new template. You can use a blank Excel file or even a firm paper with a logo, header, footer etc. for the base template.

   The new generated template is not linked in any way to the base template. If changes are later made to the base template, these changes are not reflected in the templates based on it.

3. Data source handle: BUFFER, QUERY, TEMP-TABLE or DATASET handle.

   To generate a template with multiple sheets pass a DATASET to generate a sheet for every TEMP-TABLE in the DATASET. A comma separated list of BUFFER, QUERY or TEMP-TABLE handles can be passed as a Progress version 9 alternative to an OpenEdge 10 DATASET.

4. Buffer can-do list: Use the buffer and field can-do lists to specify what fields to initially include or exclude from the template.

   See the xlsx_replaceLongRange procedure for more information on using the buffer and field can-do lists.

5. Field can-do list: See the buffer can-do list.

ttMeta TEMP-TABLE

You can pass the ttMeta TEMP-TABLE with additional data like labels, formats, calculated fields and summaries etc. about your Progress query together with it to the Template Generator.

To add ttMeta to your Progress query add the ttMeta BUFFER to your DATASET. The ttMeta TEMP-TABLE handle can be added to a comma separated list of query handles as a Progress version 9 alternative to an OpenEdge 10 DATASET.

ttMeta Options

Sample 1-2

```
/* sample1-1.p */

{slibooxml/slibxlsx.i}

{slib/slibos.i}

{slib/sliberr.i}

define temp-table ttMeta no-undo

    like xlsx_ttMeta.

define var cError       as char no-undo.
define var cErrorMsg    as char no-undo.
define var cStackTrace  as char no-undo.



{slib/err_try}:

    create ttMeta.
    assign
        ttMeta.cObject  = "SheetName"
        ttMeta.cName    = "Item"
        ttMeta.cParam   = "Item List".

    create ttMeta.
    assign
        ttMeta.cObject  = "Calc"
        ttMeta.cName    = "ExtPrice"
        ttMeta.cParam   = "ttItem.Price * Item.OnHand".

    create ttMeta.
    assign
        ttMeta.cObject  = "Label"
        ttMeta.cName    = "ExtPrice"
        ttMeta.cParam   = "Extended!Price".

    create ttMeta.
    assign
        ttMeta.cObject  = "Format"
        ttMeta.cName    = "ExtPrice"
        ttMeta.cParam   = "#,##0.00".

    create ttMeta.
    assign
        ttMeta.cObject  = "Sum"
        ttMeta.cName    = "ExtPrice".
```

```
    run xlsx_createTemplate(
        input "stXlsx",
        input "samples/templates/blank.xlsx",
        input string( buffer Item:handle ) + ","
            + string( buffer ttMeta:handle ),
        input "",
        input "ItemNum,ItemName,Price,OnHand" ).

    run xlsx_replaceLongRange(
        input "stXlsx",
        input buffer Item:handle,
        input "",
        input "",
        input "" ).

    run xlsx_save(
        input "stXlsx",
        input os_getNextFile( session:temp-dir + "item_new.xlsx" ) ).

{slib/err_catch cError cErrorMsg cStackTrace}:

    message
        cErrorMsg
        skip(1)
        cStackTrace
    view-as alert-box.

{slib/err_end}.
```

Explanation

Additional Excel Samples

Inside the samples.zip (included with the product) under the samples/src/excel directory there are additional Excel samples.

Useful Excel Tips

Microsoft Excel and Word have been around since the beginning of the 80's! and have endless amount of options. Below is a list of useful Excel options.

Managing Named Ranges

You can quickly add a named range by highlighting a cell or a range, typing a name in the name box to the left of the formula bar and pressing enter when done. Do not forget to press enter or the named range will not be added! To select from the list of available named ranges (in all the sheets) click on the name box combo-box arrow.

You can delete, edit and add new named ranges in the Data tab, Defined Names group, Name Manager dialog-box (or use the Ctrl-F3 hotkey).

Freeze Panes

If a sheet spans multiple pages and there is a column label row, header information, title etc. that you'd like to keep fixed while scrolling down the list move the cursor below the rows you'd like keep fixed, in the View tab, Window group, Freeze Panes menu and click Freeze Panes.

If you have key columns you'd like to keep fixed while scrolling to the right move the cursor to the right of the columns to be fixed before clicking Freeze Panes.

Repeat Header Rows (for Printing)

Similarly to freeze panes if you have header rows you'd like to repeat for every page when printing instead of scrolling, in the Page Layout tab, Page Setup group, Print Titles dialog-box, Sheet tab, and set the Rows to repeat at top box.

Converting and Printing

Writing a program that can render Office files (for converting and printing) would be equal to rewriting Office which is little out of the scope of this project. The OpenXML Libraries uses Microsoft Office or Open Office to open a file and print or save as a different file type. Microsoft Office is used on Windows and Open Office is used on UNIX/Linux or as a free alternative to Microsoft Office on Windows. Microsoft Office is 100% compatibile with OpenXML files

And Open Office compatibility with OpenXML is very good and improving with every release but is not perfect.

The OpenXML Libraries also supports the odf-converter-integrator tool for conversion. The odf-converter-integrator is an open source project from Novell in co-operation with Microsoft. The odf-converter-integrator can be downloaded at
http://katana.oooninja.com/w/odf-converter-integrator/download

The odf-converter-integrator currently produces better results converting Word .DOCX files to Open Office .ODT files than Open Office. The OpenXML Libraries uses the odf-converter-integrator to convert the Word files to Open Office files before opening them in Open Office (The OpenXML Libraries can also use the odf-converter-integrator alone to convert OpenXML files to Open Office files and vice versa). If you are using Open Office it is recommended to also install the odf-converter-integrator to improve results.

Note: the file types supported depends on the tools used. The libraries can use both etc.

Note: microsoft office compatibility pack for earlier versions. Not about the service pack etc.

Note: Microsoft office 2007 sp1 support for pdf, xps.

Note: open office version required 3.0, recommended the latest version. Open office version 3.1 had issues with Excel Pivot Tables.

Note: Converting and printing the first file may take longer because Microsoft Office or Open Office has to be loaded first. To make Open Office load quicker, in the Tools menu, Options, click on Memory and change Number of steps to 30, Use for OpenOffice.org to 128MB, Memory per object to 20MB, Number of objects to 20 and select Load OpenOffice.org during system start-up checkbox (the last quick start option is only available on Windows).

Converting

Sample 3

```
/* sample3.p */

{slibooxml/slibooxml.i}

{slib/slibos.i}

{slib/sliberr.i}

define var cError       as char no-undo.
define var cErrorMsg    as char no-undo.
define var cStackTrace  as char no-undo.



{slib/err_try}:

    run ooxml_convert(
        input "item.xlsx",
        input os_getNextFile( session:temp-dir + "item.pdf" ),
        input ? ).

    run ooxml_convert(
        input "sale_order.docx",
        input os_getNextFile( session:temp-dir + "sale_order.pdf" ),
        input ? ).

{slib/err_catch cError cErrorMsg cStackTrace}:

    message
        cErrorMsg
        skip(1)
        cStackTrace
    view-as alert-box.

{slib/err_end}.
```

Explanation

The slibooxml/slibooxml.i is a general purpose OpenXML Library used for printing and converting.

The ooxml_convert procedure converts one file type (even files that are not OpenXML files) to other file types and has the following parameters:

1. Source file.
2. Target file.
3. Optional formatting.

The ooxml_convert procedure identifies the file types to convert based on the file extension. For example: run ooxml_convert("test.docx", "test.pdf") converts a Word file to a PDF file.

Printing

Sample 4

```
/* sample3.p */

{slibooxml/slibooxml.i}

{slib/slibos.i}

{slib/sliberr.i}

define var cError       as char no-undo.
define var cErrorMsg    as char no-undo.
define var cStackTrace  as char no-undo.



{slib/err_try}:

    run ooxml_print(
        input "sale_order.docx",
        input "<printer-name>",
        input ? ).

{slib/err_catch cError cErrorMsg cStackTrace}:

    message
        cErrorMsg
        skip(1)
        cStackTrace
    view-as alert-box.

{slib/err_end}.
```

Explanation

The ooxml_print procedure prints a file out to a printer (even files that are not OpenXML files) and has the following parameters:

1. Source file.
2. Printer name.
3. Number of copies:

   The number of copies parameter is optional. If a zero 0 or null ? value is passed. a single copy will be printed.
4. Optional formatting.

Open Office Compatibility Issues

Open Office compatibility with OpenXML is very good but it's not perfect. Open Office compatibility has improved since it was first released in version 3.0 and it's improving with every new release. Below is list of issues and fixes.

This section covers Open Office compatibility and not another Office suite because it is probably the most widely used Office suite after Microsoft Office for working with OpenXML files and the OpenXML Libraries uses it for converting and printing.

Right-to-Left Issues

If you're using tables in Word, right-click the table, Table Properties, Table tab and make sure the Table direction is left-to-right.