

The OpenXML Libraries

Table of Contents

Introduction	4
Generating Excel Files	5
Sample 1	6
Template Generator	12
Sample 1-1	13
ttMeta TEMP-TABLE	15
ttMeta Options	15
Sample 1-2	16
Additional Excel Samples	17
Useful Excel Tips	18
Managing Named Ranges	18
Freeze Panes	18
Repeat Header Rows (for Printing)	18
Generating Word Files	19
Sample 2	20
Additional Word Samples	27
Paste Behavior	28
Design Recommendations	30
Use a Main Part for Every Page	30
Use Tables for Lists	30
Put Header and Rows in a Single Table	30
Avoid Part Borders Overwriting Each Other	30
Field Formats	31
Numbers	31
Dates	31
Text	31
Advanced Word Options	32
Using Images	32
Using Barcodes	33
Using HTML and RTF Values	34
Useful Word Tips	35
Managing Bookmarks	35

Repeat Header Rows	35
Keep Table Rows Together	35
Keep Paragraph Lines Together.....	36
Embed Fonts.....	36
View Table Gridlines.....	36
Making a Document Read-Only.....	36
Converting and Printing	37
Converting	38
Sample 3	38
Printing	39
Sample 4	39
Open Office Compatibility Issues	41
Right-to-Left Issues.....	41

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.

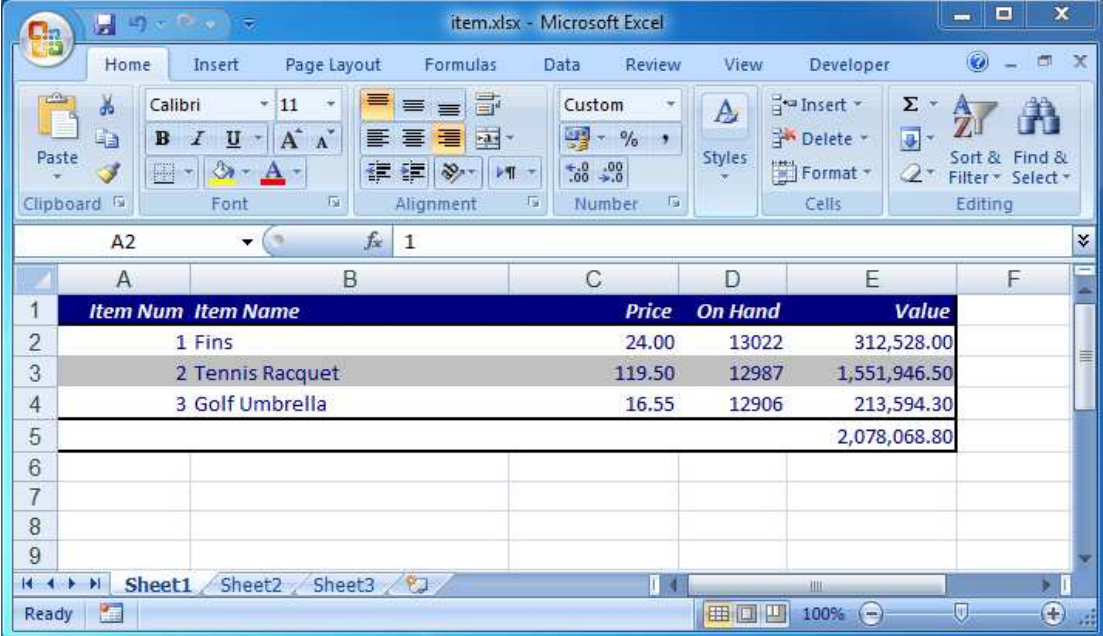
On the Progress side use the OpenXML Libraries to create a new Excel file from the template you just created and replace the sample data with data from a Progress query (it will actually delete the sample data rows and insert the rows from the query) while still keeping the sheet formats and styles, 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.



Item Num	Item Name	Price	On Hand	Value
1	Fins	24.00	13022	312,528.00
2	Tennis Racquet	119.50	12987	1,551,946.50
3	Golf Umbrella	16.55	12906	213,594.30
				2,078,068.80

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 then 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 */

{slib/booxml/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.

Generating Word Files

Just like when generating an Excel file you always start by designing your template (a regular .DOCX file).

A template is made up of parts that you mark by highlighting an area and saving it as a Bookmark (see Managing Bookmarks in the Useful Word Tips section). A part can have sub parts inside it, sub parts can have sub parts inside them and so on (you can have as many levels of sub parts as you'd like).

For example: a Sales Order form with a main Order part covering the entire page with an OrderLine sub part inside it (that will be repeated for every OrderLine) followed by a Total Line sub part. You could even add a Comments sub part inside the main Order part, an Introduction page part and a Summary page part (separated by page breaks) etc.

Inside your parts (and sub parts) you can add fields. You can either use Word Merge Fields, Document Properties etc. or use the OpenXML Libraries {field [optional formatting]} notation. The {field} notation is generally easier and faster to add and also has additional formatting options not available with Merge Fields.

Note: The entire {field [optional formatting]} notation must be in the same font, size, color, everything or the OpenXML Libraries will not recognize the field.

Note: The [optional formatting] uses the Microsoft Office format syntax and not Progress. For example: #,##0.0##, MMM d/yyyy etc. Text, Number and Date are also valid formats values that will produce the default format for these data types.

On the Progress side you generate Word files using cut/paste like commands. When you generate a new Word file from the template you just created the OpenXML Libraries cut away all the parts to the clipboard leaving an empty document. You can then fill the field values of the parts in the clipboard and paste them onto the new document with the field values you just set. Each time a part is pasted all the fields in the clipboard are cleared and ready to be set again.

Note: The clipboard in this context refers to the OpenXML Libraries clipboard (implemented using Progress XML and TEMP-TABLE's) and not the Microsoft Office clipboard (Office is not used to generate the files).

Note: The current version of the OpenXML Libraries does not support automatic field mapping if the template and query fields have the same name in Word like there is in Excel but it's still recommended to use the same field names because support for automatic field mapping in Word is planned to be added in the next release.

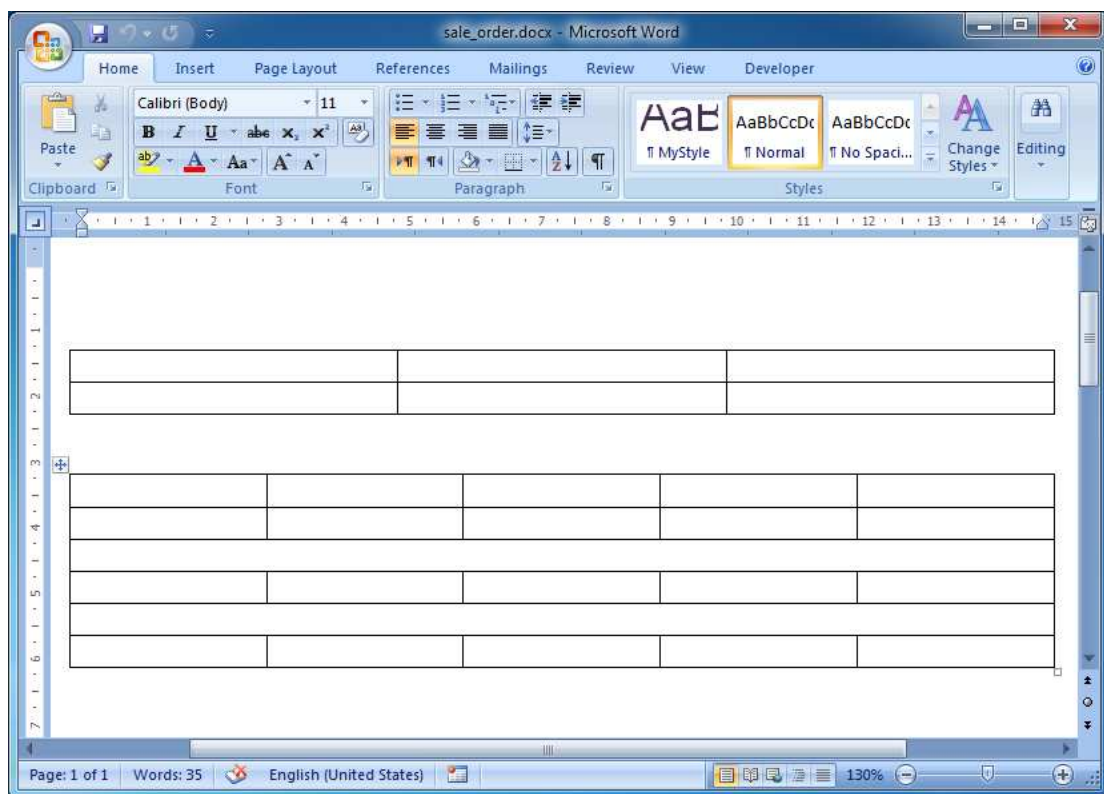
Sample 2

In this sample you'll be generating a Word file for sales orders and lines from the Sports2000 sample database.

Follow the steps below to create the template.

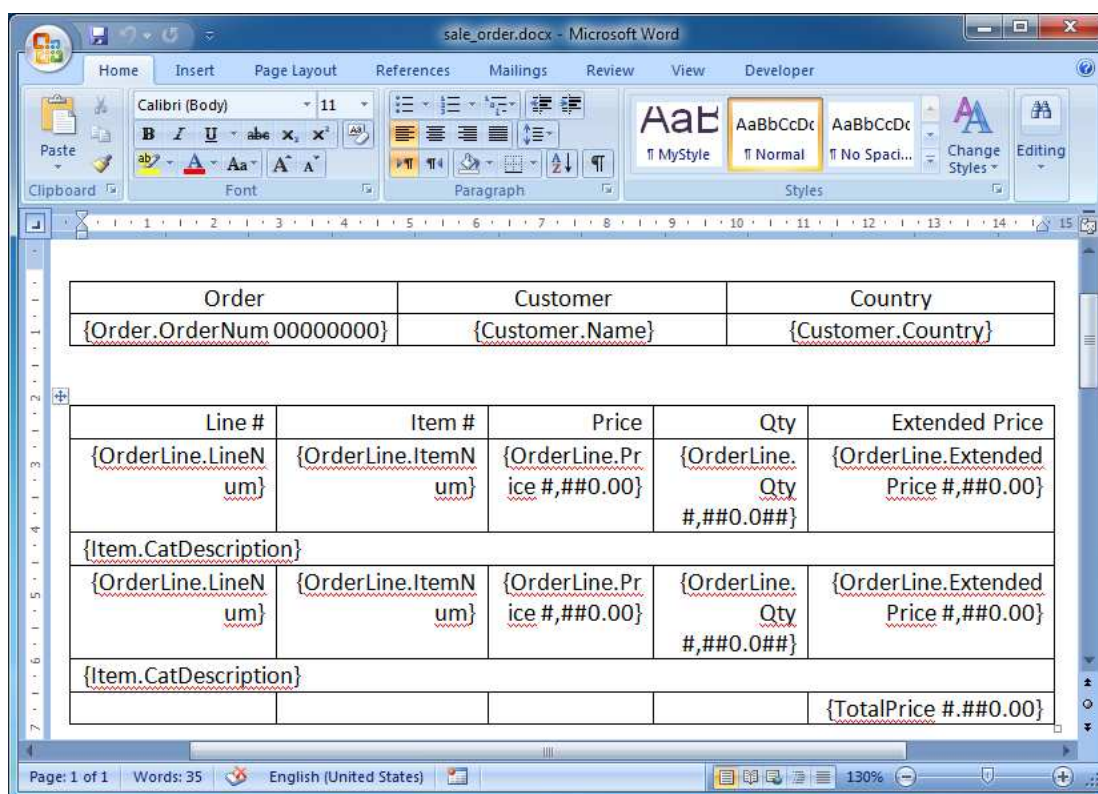
1. Create a new blank Word file.
2. Add a table with 3 columns x 2 rows and a table with 5 columns x 6 rows below it.
3. Merge the third row of the second table. Do the same to the fifth row.

If you followed the steps, your Word file should now look like the picture below.



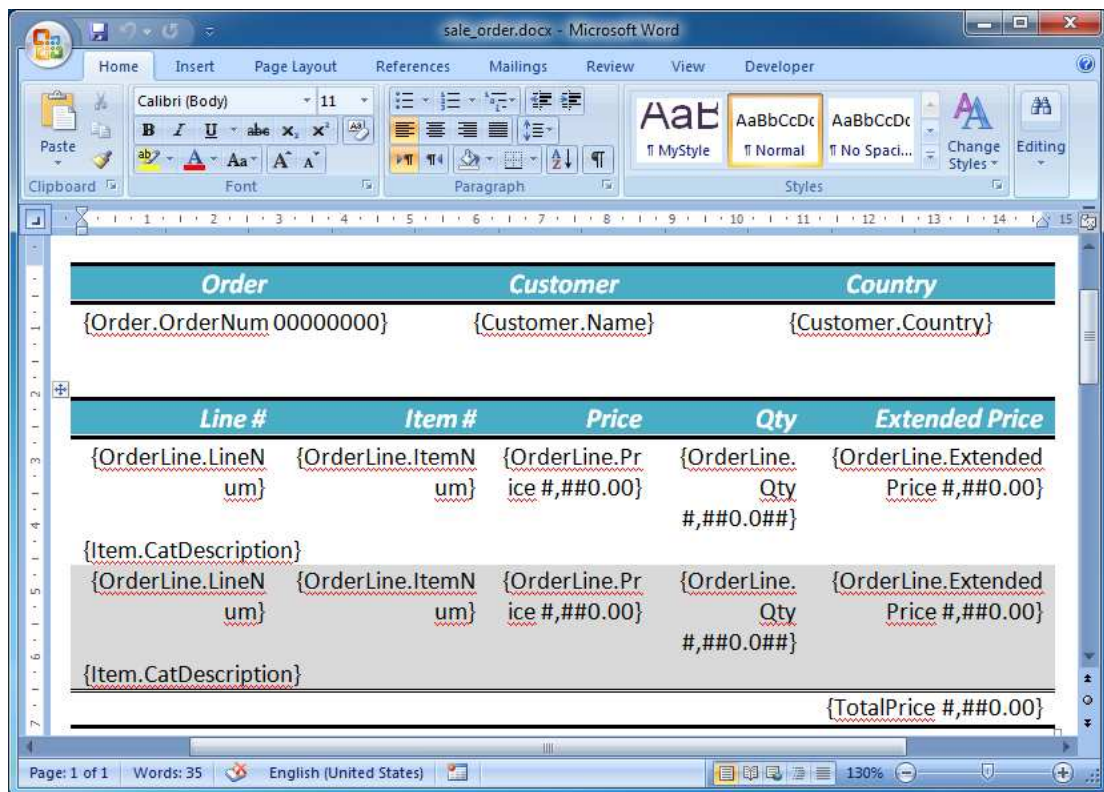
4. Add the values "Order", "Customer" and "Country" in the first row of the first table.
5. Add the following values in the second row of the first table:
 1. {Order.OrderNum 00000000}
 2. {Customer.Name}
 3. {Customer.Country}
6. Add the values "Line #", "Item #", "Price", "Qty" and "Extended Price" in the first row of the second table.
7. Add the following values starting in the second row of the second tables:
 1. {OrderLine.LineNum}
 2. {OrderLine.ItemNum}
 3. {OrderLine.Price #,##0.00}
 4. {OrderLine.Qty #,##0.0##}
 5. {OrderLine.ExtendedPrice #,##0.00}
 6. {Item.CatDescription} in the third row of the second table.
 7. {TotalPrice} in the last column on the last row.
8. Copy/paste the second to third rows to the fourth and fifth rows.
9. Align Top Center the entire first table (in the Layout tab, Alignment group).
10. Align Top Right the entire second table.
11. Align Top Left the third and fifth row of the second table.

If you followed the steps, your Word file should now look like the picture below.



12. Remove the borders from the two tables.
13. Add a thick border above and below the first row of the first table. Do the same to the first row of the second table (see picture below).
14. Add a double border above and thick border below the last row of the second table (see picture below).
15. Add a blue aqua background color and white foreground color to the first row of the first table. Do the same to the first row of the second table.
16. Add a light gray background color (Darker 15%) to the fourth and the fifth row of the second table.
17. Bold, italicize and increase the font size by one point to the first row of the first table. Do the same to the first row of the second table.
18. Mark Repeat Header Rows the first row of the second table (see the Repeat Header Rows in the Useful Word Tips section).
19. Mark Keep with next the second row of the second table (see Keep Table Rows Together in the Useful Word Tips section). Do the same to the fourth row.

If you followed the steps, your Word file should now look like the picture below.



20. Add an "Order" bookmark that covers the entire page.
21. Add an "OrderLine1" bookmark that covers the second to third rows of the second table.
22. Add an "OrderLine2" bookmark that covers the fourth to fifth rows of the second table.
23. Add an "TotalLine" bookmark that covers the last row of the second table.
24. Save the file as "sale_order.docx" in the working directory or anywhere on the PROPATH. For example: C:\OpenEdge\WRK\sale_order.docx.

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

```

/* sample2.p */

{slibooxml/slibdocx.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 docx_copyTemplate(
    input "stDocx",
    input "sale_order.docx" ).

  for each Order
    where Order.OrderNum >= 1
      and Order.OrderNum <= 10
      no-lock,

      first Customer of Order
      no-lock

      by Order.OrderNum:

        run docx_setClipboardValue(
          input "stDocx",
          input "Order",
          input "Order.OrderNum",
          input Order.OrderNum ).

        run docx_setClipboardValue(
          input "stDocx",
          input "Order",
          input "Customer.Name",
          input Customer.Name ).

        run docx_setClipboardValue(
          input "stDocx",
          input "Order",
          input "Customer.Country",
          input Customer.Country ).

        run docx_paste(
          input "stDocx",
          input "Order" ).

```



```

for each OrderLine of Order
  no-lock,

  first Item of OrderLine
  no-lock

  by OrderLine.LineNum:

    accumulate OrderLine.ExtendedPrice ( total ).

    run docx_setClipboardValue(
      input "stDocx",
      input "OrderLine",
      input "OrderLine.LineNum",
      input OrderLine.LineNum ).

    run docx_setClipboardValue(
      input "stDocx",
      input "OrderLine",
      input "OrderLine.ItemNum",
      input OrderLine.ItemNum ).

    run docx_setClipboardValue(
      input "stDocx",
      input "OrderLine",
      input "Item.CatDescription",
      input Item.CatDescription ).

    run docx_setClipboardValue(
      input "stDocx",
      input "OrderLine",
      input "OrderLine.Qty",
      input OrderLine.Qty ).

    run docx_setClipboardValue(
      input "stDocx",
      input "OrderLine",
      input "OrderLine.Price",
      input OrderLine.Price ).

    run docx_setClipboardValue(
      input "stDocx",
      input "OrderLine",
      input "OrderLine.ExtendedPrice",
      input OrderLine.ExtendedPrice ).

    run docx_paste(
      input "stDocx",
      input "OrderLine" ).

end. /* each OrderLine */

run docx_setClipboardValue(
  input "stDocx",
  input "TotalLine",
  input "TotalPrice",
  input ( accum total OrderLine.ExtendedPrice ) ).

run docx_paste(
  input "stDocx",
  input "TotalLine" ).

end. /* for each */

```

```

run docx_save(
  input "stDocx",
  input os_getNextFile( session:temp-dir + "sale_order_new.docx" ) ).
{slib/err_catch cError cErrorMsg cStackTrace}:

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

{slib/err_end}.

```

Explanation

The Word program starts the same way the Excel program starts creating a new file by copying an existing template with an almost identical procedure name with the same parameters.

The program has an outer Order query that sets the field values and pastes the Order main part that starts in a new page every time it's pasted (see the Paste Behavior section).

And an inner OrderLine query that sets the field values and pastes the OrderLine sub part. Although the OrderLine part does not exist you can split a sub part into two parts with 1 and 2 suffixes for alternating sub parts in your template while still using the same sub part name in your program (see the Paste Behavior section).

The Word program ends the same way the Excel program ends saving the file out to disk with an almost identical procedure name with the same parameters.

The `docx_setClipboardValue` procedure sets clipboard parts field values 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 "stDocx" stream name is used for generating Word files.
2. Part name:

If a blank "" or null ? value is passed the procedure will set the field values for the specified field name in all parts.
3. Field name.
4. Field value.

The docx_paste procedure pastes clipboard parts onto the new document with the field values you just set and has the following parameters:

1. Stream name: See the previous procedure.
2. Part name: See the previous procedure.

If a blank "" or null ? value is passed the procedure will paste all the parts out. This is sometimes used for forms with no parts like simple letters although forms with no parts are not recommended (see the Design Recommendation section).

Additional Word Samples

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

Paste Behavior

Templates are always made up of parts even if you don't define any parts because pages are parts themselves so even a blank template has one page part. A template is broken into pages by page breaks and not the page size which can be changed according to the printing device you're using. Page part names are their page number although you should never paste pages directly because if a program pastes a page and that page is moved, the program would also have to be updated making the code and design tied together.

*Note: To find the parts and fields in a template use `docx_getPartList` and `docx_getFieldList` procedures (see the *OpenXML Libraries Reference* for more details).*

The paste procedure always continues where it left off, the last part that was pasted. If the part being pasted is a sub part of the last part that was pasted, or one of his parents he's added to them. If not the part is added to the end of the document.

For example: a template with a main Order part covering the entire page, an OrderLine sub part and a Comments sub part at the bottom of the page inside it and a Summary page part. If you paste the Order part and then the OrderLine part, it's added to the Order part. If you then paste the Comment part, it's added to the Order part. Finally if you paste the Summary page part it's added to the end of the document.

If you're pasting a sub part without pasting his parent(s) first, the missing parents are added automatically before adding the sub part. For example: using the same template, if you paste the OrderLine sub part without pasting the Order part first, the Order part will be added automatically before adding the OrderLine part.

Every time a top level part (that is not a sub part of any other part, except the page it's in) is pasted it starts in a new page (actually the page part it's in is added along with it). For example: using the same template, every time the Order part is pasted it starts in a new page.

Every time a sub part is pasted the sub part is repeated continuously, in the same place it's in the parent part. For example: using the same template, if you paste several OrderLine sub parts and several Comment sub parts, the OrderLine sub parts is repeated continuously in the same place it's in the Order part and the Comment sub part is repeated continuously in the same place at the bottom of the page separately from the OrderLine sub part.

But what happens if you want to mix sub parts together? A typical example is alternating sub parts. If the sub parts in the template are continuous and placed one right after the other, you can mix the sub parts in the order they are pasted.

For example: using the same template, if instead of having one OrderLine sub part you had an OrderLine1 sub part and an OrderLine2 sub part right after the other for rows with different colors and you pasted alternating between OrderLine1 and OrderLine2 sub parts, the generated Word file will have zebra like striped rows.

Note: To find out if parts are continuous, open the Insert Bookmark dialog-box, Sort by: Location and go down the list highlighting the bookmarks and make sure there are no spaces between the parts (see Managing Bookmarks in the Useful Word Tips section).

Continuing the previous example you can generate alternating sub parts by splitting a sub part into two parts with 1 and 2 suffixes in your template while still using the same sub part name for setting the field values and pasting on the Progress side. The OpenXML Libraries will automatically set the field values and alternate between the two sub parts. This way you can design your template with alternating sub parts or not without making changes to your code, keeping your design and data separated.

For example: using the same example with the OrderLine1 and OrderLine2 sub parts while still using the OrderLine sub part name for setting the field values and pasting, the OpenXML Libraries will automatically alternate between OrderLine1 and OrderLine2 and generate the same file as the previous example.

Design Recommendations

Use a Main Part for Every Page

It's recommended to use a main part for every page covering the entire page for the top level parts that need to start in a new page and put the repeatable sub parts inside them.

It's not recommended to put fields directly in pages or paste pages directly because if a program pastes a page and that page is moved, the program would also have to be updated making the code and design tied together.

It's not recommended to use several top level parts in a page because it complicates the design and makes paging hard to control.

It's not recommended to create templates with no parts even for very basic forms like letters because what happens quite often in the real world nothing stays simple.

Use Tables for Lists

It's recommended to use tables for lists even if the lists are more paragraph like than table like. Using tables produces the most symmetric results, allows to easily align columns and allows to repeat any header information, title etc. if the list overflows onto the next page (see Repeat Header Rows in the Useful Word Tips section).

Put Header and Rows in a Single Table

If you'd like to repeat the column label row, header information, title etc. if the table overflows onto the next page put the rows and header rows in the same table (even if there are blank rows between them which can easily be resolved using a blank table row without borders) and set the repeat header rows (see Repeat Header Rows in the Useful Word Tips section).

Avoid Part Borders Overwriting Each Other

When designing a template if for example you have a table with a column label row, rows and a summary row that you'd like to have different borders and avoid them overwriting each other, you can add spacer parts between them. The spacer parts separate between the parts and allow you to define different borders for parts without them overwriting each other and because they are parts themselves they keep the parts continuous so they can be mixed (see the Paste Behavior section).

When the Word file is generated the spacer parts will be cut away and not used and the wider borders will overwrite the thinner borders. For example: if you have a column label row and summary row with wider borders then the rows thinner borders, the column label row and summary row will overwrite the rows borders.

Field Formats

Numbers

The field optional formatting for numbers uses the Microsoft Office syntax and not Progress. For example: "#,##0.0##", "#0%" or "#,##0.0#[RED]-#,##0.0#" (without the "" quotes).

In the template the number formats are always entered in the American numeric format. In the generated Word file the numbers are displayed according to the OpenXML Libraries numeric format.

The OpenXML numeric format defaults to the SESSION:NUMERIC-FORMAT and can be changed using the docx_setNumericFormat procedure that has one parameter for the numeric format. Valid numeric formats are "American", "European" and ? for setting the numeric format back to the session format.

This approach allows for sharing templates created in different countries with different numeric formats.

Dates

Just like numbers the field optional formatting for dates uses the Microsoft Office syntax. For example: "MMM dd, yyyy", "dddd MM/yyyy" or "hh:mm:ss".

In the template you can optionally use the Progress 99/99/99 date format and it's variations. In the generated Word file the dates will be displayed according to the OpenXML Libraries date format.

The OpenXML date format defaults to the SESSION:DATE-FORMAT and can be changed using the docx_setDateFormat procedure that has one parameter for the date format. Typical values include "mdy", "dmy" and ? for setting the date format back to the session format.

Text

The field optional formatting for text accepts Progress character formats. For example: "x(2)", "!(10)" or "xx-xxx-xxxx"

Advanced Word Options

Using Images

The OpenXML Libraries treats images as just another type of field. In the template insert a placeholder image, right-click the image, select Size, Alt Text tab and give it a name in the Alternative text box using the {<field> [optional formatting]} notation. On the Progress side use the regular docx_setClipboardValue procedure using the image name in the field name and the image path in the field value.

The optional formatting for images are use-image-size: to resize the new inserted image to their original size and is also the default format if no other format is specified and keep-size: to keep the template placeholder image size.

Images in most cases are saved inside the Word file (also called embedded images) but there is also an option to save only a reference to the file (also called linked images) to drastically reduce the Word file size but if the Word file is sent to someone outside the office or anyone without access to the image files the images will not be displayed.

The OpenXML Libraries uses embedded images by default to use linked images pass a file:/// URL scheme image path (simply add a "file:///" prefix to your path). For example: file:///c:\dir\image.jpg. It's recommended to use embedded images. Use linked images only when necessary.

Note: The OpenXML Libraries searches for the image file even for linked files when the image is inserted and if the file cannot be found no image is inserted and the image placeholder remains blank.

Note: The current version of the OpenXML Libraries only supports .JPEG and .PNG images which are the only images supported in OpenXML. Even If you insert other image types in Office they will be converted to .JPEG/.PNG images and saved inside the OpenXML file. Support for other image types is planned to be added.

Note: The OpenXML Libraries does not support images in compatibility mode because of their many drawbacks, most importantly Word files with even a hundred images may take many minutes and even more to open. To find if the Word file is in compatibility mode, check if the title bar displays [Compatibility Mode]. To remove compatibility mode, select Save as from the menu, Word Document and in the Save as dialog-box make sure the Maintain compatibility with Word 97-2003 checkbox is not selected.

Using Barcodes

Barcodes are essentially a font that displays the characters as a series of bars and spaces. Before being displayed using the barcode font the value is also encoded, adding a prefix, checksum suffix etc. depending on the barcode type being used. Some of the most popular barcode types are Code 128A, Code 128B and Code 39 although almost all modern barcode readers can read all the major barcode types and more.

To use barcodes you'll first need to install the barcode font on your machine. There is a Windows Code 128B barcode font included with the Product in `slib\bin\Code128bWin.ttf`. Open the Control Panel folder, Appearance and Personalization, Fonts folder and drag the font file into the Fonts folder.

In the template add a field using just the `{field}` notation with no optional formatting. For example: `{barcode}`. (Merge Fields can also be used). Highlight the field (including `{}` brackets) and change the font to Code128bWin, the new Code 128B barcode font.

On the Progress side before setting the field value the value needs to be encoded for that barcode type. There is a Code 128B encoding procedure included with the product in `slib/getbarcode128b.p`. The `slib/getbarcode128b.p` procedure has an input parameter for passing the original value and an output parameter that returns the encoded value. Run the `slib/getbarcode128b.p` first to encode the value and use the encoded value with the `docx_setClipboardValue` procedure.

Note: The barcode font can be embedded in the Word file so the file can be used on Linux or other machines where the font is not installed (see Embed fonts in the Useful Word Tips section).

Note: Built-in support for the major barcode types is planned to be added. The OpenXML Libraries will let you set the barcode type using the optional formatting in the `{field [optional formatting]}` notation, for example: `{barcode code128b}` and will embed the font and do the encoding automatically when generating the Word file.

Using HTML and RTF Values

The OpenXML Libraries support inserting HTML and RTF field values instead of plain text that can be used to add formatting like fonts, bolding, colors etc. all the way up to inserting mini documents with bullet lists, tables etc.

To add HTML or RTF values in the template add a field using the {field [optional formatting]} notation and use either HTML or RTF in the optional formatting. For example: {Comments HTML}. On the Progress side simply insert a valid HTML or RTF value in the field value using the regular docx_setClipboardValue procedure.

Note: The docx_setClipboardAltChunk procedure can be used to set HTML or RTF values. The docx_setClipboardAltChunk has the same parameters and accepts LONGCHAR field values for inserting very large HTML or RTF values (larger then 32K). AltChunk is the OpenXML specification part name used for embedding HTML and RTF.

The HTML tag styles, fonts, colors etc. are inherited from the document Word Styles with a similar name, in the Home tab, Styles group. For example: the <h1></h1> tags inherit the Heading 1 style, the <p></p> tags inherit the Normal style, tags inherit the Strong style.

Just like tags can be combined, styles can also be combined. For example: <p></p> combines the Normal and Strong styles. You can also create new styles in your document and set the HTML tag class property to the style name. For example: create a style in Word and call it MyStyle and insert the HTML tag: <p class="mystyle"><p> (note that the class property value must be lowercase).

Although you can also use CSS together with HTML to add styles, fonts, colors etc. it is recommended to do all the design and styles in Word and the data in Progress, keeping the design and data separated.

Note: HTML values in Word files are only supported starting with Microsoft Office 2007. This is a Microsoft Office limit not an OpenXML Libraries limit. RTF values do not have that limit.

Note: HTML and RTF fields are paragraph level fields meaning that just like paragraphs they have a line break before and after them and cannot be used in the middle of a line.

Note: The HTML <html><body></body></html> tags (including DTD and other headers) or the RTF {\rtf1\ansi\deff0} definitions in HTML and RTF fields are optional. If they're missing the OpenXML Libraries will add them automatically.

Useful Word Tips

Managing Bookmarks

To add a bookmark highlight an area, in the Insert tab, Links group, click Bookmark that opens the Bookmark dialog-box, type a name for the bookmark in the Bookmark name: box and click the Add button.

To delete a bookmark in the Bookmark dialog-box select a bookmark and click the Delete button.

To highlight a bookmark in the Bookmark dialog-box select a bookmark and click the Go To button.

To sort the list of bookmarks in the Bookmarks dialog-box, Sort by: radio buttons and select Name or Location. You can use sort by name to find a bookmark and sort by location to check if parts are continuous and there are no spaces between them.

Repeat Header Rows

If a table overflows onto the next page and there is a column label row, header information, title etc. that needs to be repeated for every page, highlight the top rows that you'd like to repeat, in the Layout tab, Data group and click the Repeat Header Rows.

Note: For the Repeat Header Rows to work right-click the table, Table Properties, Table tab, and make sure the Text wrapping option is set to None. Of course the rows and header rows you'd like to repeat must be in the same table.

Unfortunately Repeat Header Rows does not work with nested tables which greatly limits nested tables use. This is a Microsoft Office limit not an OpenXML Libraries limit.

Keep Table Rows Together

If a table overflows and there are parts with multiple rows that you'd like to keep together and not split at the point where the page breaks, highlight all the rows except for the last row, in the Home tab, Paragraph properties (Paragraph group bottom right corner), Line and Page Breaks tab and select Keep with next checkbox (make sure Keep with next is not selected for the last row).

Note: Unfortunately Microsoft Office does not have a Keep with previous option which has been requested by Office users for years and could be useful for example with total line so the line will not be alone if it overflows onto the next page. Because we recognize that this is an important feature support for Keep with previous option is planned to be added to the OpenXML Libraries in the next release.

Keep Paragraph Lines Together

If you have a paragraph with multiple lines that you'd like to keep together and not split at the point where the page breaks, highlight the paragraph, in the Home tab, Paragraph properties (Paragraph group bottom right corner), Line and Page Breaks tab and select Keep lines together checkbox.

The Keep lines together option only works on a single paragraph. If you have several paragraphs that you'd like to keep together, highlight all the paragraphs except for the last paragraph, in the Home tab, Paragraph properties (Paragraph group bottom right corner), Line and Page Breaks tab and select Keep with next checkbox (make sure Keep with next is not selected for the last paragraphs).

Embed Fonts

If you're using barcode fonts or internationalization fonts etc. and you'd like to open the file on Linux or send it to someone on the other side of the world that may not have the fonts installed, you can embed the fonts in the Word file and make sure they can be displayed, click the Office button (the Office icon at the top left corner), Word options (at the bottom right corner), Save options, select Embed fonts in the file checkbox, deselect Embed only the characters used in the document checkbox (while you're designing your document some characters may not be used that may be added later when generating the Word file) and select Do not embed common system fonts checkbox.

Note: Unfortunately you cannot embed fonts in Excel files which is something you need to be aware of if you're using barcode or internationalization fonts in Excel.

View Table Gridlines

Table borders are not always marked and the table structure is not always clear. A typical example is when tables are used for layouts. To show the table gridlines, in the Layout tab, Table group, click the View Gridlines.

Making a Document Read-Only

You can protect a Word document from being modified (used for working with Word forms) in the Review tab, Protect group, in the Protect Document menu, select Restrict Formatting and Editing, select Limit formatting to a selection of styles, click Yes, Start Enforcing Protection, select the Password option and enter a password.

Although it's easy to bypass this protection (which was never intended for this use) by copying and pasting the entire document into a new unprotected Word document (disabling copy/pasting can be done using Macros which is both cumbersome and not supported with the current version of the OpenXML Libraries).

Using PDF files isn't secure either because although PDF files can only be read with Acrobat Reader, PDF files can be edited with the full version of Acrobat. Almost any protection would take your average geek just a few minutes to bypass. The only safe thing left to do is print the document on paper.

Converting and Printing

Writing a program that can render Office files (for converting and printing) would be equal to rewriting Office which is out of the scope of this project. The current version of the OpenXML Libraries mainly relies on Open Office for converting and printing which is available on both UNIX/Linux and Windows.

The file types supported for printing and converting depends on the file types supported for opening and saving by the printing and converting tools.

Note: Support for Microsoft Office on Windows for converting and printing is planned to be added in the next release. The OpenXML Libraries will detect and use whatever application is available and according to what application will produce the best results.

Note: The OpenXML Libraries also supports the odf-converter-integrator for printing and converting. The odf-converter-integrator is an open source project from Novell in cooperation with Microsoft and is the recommended tool in some document conversions.

The odf-converter-integrator can be downloaded at
<http://katana.oooninja.com/w/odf-converter-integrator/download>

Note: Converting and printing the first file may take longer because Open Office has to be loaded first. To make Open Office loads 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" ) ).

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

{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.

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 ?,
        input ?,
        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. Paper format: For example: "A4", "A5" or "LETTER" etc.

The paper format parameter is optional. If a blank "" or null ? value is passed, a default value will be used if one is not specified in the document.

4. Paper orientation: "PORTRAIT" or "LANDSCAPE".

The paper orientation parameter is optional. If a blank "" or null ? value is passed, a default value will be used if one is not specified in the document.

5. 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.

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.

Note: Support for Microsoft Office on Windows for converting and printing is planned to be added in the next release that will guarantee 100% compatibility but in most cases will require doing the conversion or printing on the client side.

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.