MarkLogic Toolkits for Office® Lab

Prerequisites: MarkLogic Server installed, CQ configured and available for querying

Creating an MS Office Document from MarkLogic

- Understanding the XQuery APIs
- Creating Word Documents
- Creating Excel Workbooks

TOPIC

Understanding the XQuery APIs

The XQuery APIs that accompany each toolkit are provided to assist you in creating new Office documents as well as updating existing, extracted Office documents in MarkLogic Server. In this unit, we'll be generating documents using the APIs. But before we dive into the XQuery, we should familiarize ourselves a little bit with the XML formats these APIs work with.

The Office document format is named Office Open XML, or concisely, Open XML. This is a bit misleading, as Open XML is not just one format, but 4.

- WordprocessingML is the format for Word documents
- SpreadsheetML is the format for Excel workbooks
- PresentationML is the format used by PowerPoint for presentations
- DrawingML is used for shapes and art and can be found sprinkled and used throughout all the above.

The Office applications are created and maintained by 3 different teams within Microsoft, so unfortunately you won't find many common XML elements used across the formats.

When working with these formats, it's also important to remember that the Word, Excel, and PowerPoint applications are now producers and consumers of XML. But the XML required to generate a well formed and valid Office document often requires much less XML than the XML that will be generated if you were to save the same document in Office.

Keep all this in mind when we review the XQuery API documentation. While each function is listed, and detailed, the functionality of each API, as well as any additional details provided in the documentation, is due to the complexity of each format in relation to the types of things developers want to be able to do when working with these formats.

Understanding the components of an unzipped .docx, .xlsx, and .pptx can be very useful when writing queries in MarkLogic to generate new Office content as well as search and reuse existing Office content.

Open XML 101

We know that a .docx, .xlsx, and .pptx are just zip files containing interrelated XML parts. Each is required to have at minimum, 3 parts.

- *[Content_Types].xml* - A dictionary of content types for all the other parts inside the package
- *_rels/.rels - The main relationships file, tying the various parts of the package together*
- *document.xml (Word), workbook.xml(Excel), presentation.xml (PowerPoint) - The main document part*

The main document part however, may not be the main content part, and to be useful, a document requires content. So while you may construct something that will open in Office using just those 3 parts, you'll want to know the minimum XML files required, the main document part and main content part for each format. When introducing, the main content part however, you'll introduce new dependencies, as such, you'll then require more related XML parts. Let's look at each format in detail.

Filenames listed below include path within respective Office zip package.

WordprocessingML

Creating a document using just the minimum 3 required parts, Word will open with an empty document ready to be authored. This is because in Word, the main document part and main content part are the same.

Minimum XML parts required for document that includes content:3

- [Content_Types].xml
- rels/.rels
- document.xml – **main document part** and **main content part** are the same

SpreadsheetML

Creating a workbook using just the minimum 3 required parts, Excel will open, but there will be no worksheets in the workbook. The Excel application opens and appears empty.

Minimum XML parts required for workbook that includes content: 5

- [Content_Types].xml
- _rels/.rels
- _rels/workbook.xml.rels – relates workbook to sheets
- xl/workbook.xml – **main document part**
- xl/worksheets/sheet#.xml (where # = 1,2,3,…n) – **main content part**

PresentationML

Creating a presentation using just the minimum 3 required parts, PowerPoint will open, but there will be no slides in the presentation. The PowerPoint application opens and appears empty.

Minimum XML parts required for presentation that includes content: 11

- [Content_Types].xml
- rels/.rels
- ppt/presentation.xml – **main document part**
- ppt/_rels/presentation.xml.rels –relates presentation to slides
- ppt/slides/slide#.xml (where # = 1,2,3,…n) – **main content part**
    - a slide is a container for shapes, stored in a shape tree
- ppt/slides/_rels/slide#.xml.rels – relates slide#.xml to slideLayout#.xml
- ppt/slideLayouts/slideLayout#.xml
    - another shape tree that combines with the shape tree within the slide and slideMaster to create the content within a slide
- ppt/slideLayouts/_rels/slideLayouts#.xml.rels –relates slideLayout to slideMaster
- ppt/slideMaster/slideMaster#.xml
    - another shape tree which forms the root of the elements which make up a slide
- ppt/slideMaster/_rels/slideMaster#.xml.rels –relates slideMaster to slideLayout and theme
- ppt/theme/theme.xml

Reminder: Listed above are the minimum files required for generating these documents using their respective XML formats. However, when you click " Save" in Office to save a document of similar content, the number of XML files generated for each format is greater.

The naming of the package files used above and paths you'll see for XML files from extracted Office documents reflects what the Office formats save to as natively within their respective zip packages. The Open XML formats do allow you to create zip packages for Office documents that use your own paths and names for files, so long as the XML is well-formed and valid (You could for example, create a Word document with document.xml renamed as foo.xml in a directory named "bar" you'd just update [Content_Types].xml and .rels accordingly). But just know that when you open your document in Office, the moment you click "Save", your paths and naming will be stored to Office defaults in the respective Office zip package.

The XQuery APIs will jumpstart your development, and also provide guidance towards how Office documents are constructed. This is useful as at some point you may find yourself digging deeper into the XML for a particular format, and with a basic understanding of how a .docx, .xlsx, and .pptx are constructed from interrelated XML files, and the basic structure of the main part in those files that contains the content you care about, you'll be well on your way to achieving your goals.

Let's dig in!

WALKTHROUGH

Exercise 1: View the documentation

1. In your favorite browser, open the following from your Toolkits
    a. MarkLogic-Toolkit-for-Word-1.2-2\docs\xquery-apidoc\ word-processing-ml-support.html
    b. MarkLogic-Toolkit-for-Excel-1.0-3\docs\xquery-apidoc\ spreadsheet-ml-support.html

2. Examine the WordProcessingML Support documentation
3. Look at the function ooxml:document() and it's example
    a. The main body of content in Word is document.xml. This function creates that part
    b. In the example we see function ooxml:create-paragraph() take a look at its definition
    c. Examine the ooxml:paragraph() definition and its example

In Word, a document consists of block-level elements and Inline elements. block-level content provides the main structure of the document and contains inline-content. Examples of block-level content are <w:p> (paragraphs) and <w:tbl> (tables). Many Word documents consist of a series of paragraphs. Paragraphs are composed of runs (<w:r>) of text (<w:t>) and can include images as well as tables.

    d. Look at the function ooxml:create-simple-docx()
    e. Look at the function ooxml:docx-package()
    f. Look at the function ooxml:package()

We can use the API to create our content. We can then either pass it on to the ooxml:create-simple-docx() function which will construct a .docx consisting of the 3 minimum parts for us. If we wish to construct a more complicated document, the API provides constructor functions for the other .docx parts, or we could construct them on our own. We can then pass these XML parts to ooxml:docx-package() to zip them up and construct our binary .docx for us.

Finally, an alternative to saving our Word document as .docx, is to save it as a package (<pkg:package>). This is the result of saving your document as XML in Word and is called OPC (The Open Packaging Convention). The benefit of using this function includes having all our parts in a single document, eliminating the need to write queries that have to join disparate, extracted parts, as well as the format required by our JavaScript API function MLA.insertWordOpenXML().

Constructing a Word document on the server using ooxml:package(), this document can be inserted into any active Word document at the current cursor position using MLA.insertWordOpenXML(). The content will be serialized into the document.xml being authored. Any required styles, themes, etc. from other package parts, will be appended to the stylex.xml, themes.xml, etc. in the document being authored. The combination of these functions provide us a powerful mechanism for searching, reusing, and generating content into active Word documents.

Note: Word and PowerPoint can both serialize into the OPC format and likewise open documents in OPC format. Word allows us to inject OPC format documents into Word documents actively being authored, PowerPoint however does not. Excel has no support for OPC, so you can't save as OPC XML from Excel, nor open an Excel document in OPC format into Excel.

4. Examine the SpreadsheetML Support documentation
5. Look at the function excel:worksheet() and its Example
    a. A Worksheet is the main content part for an Excel document
    b. A Worksheet contains rows (excel:row()) , which contain cells (excel:cell())
6. Look at the function excel:create-simple-xlsx and its Example
7. Look at the function excel:xlsx-package() and its Example

Similar to Word, we find constructor functions to build our main content part as well as the other XML parts of an Excel workbook. We also find create-simple function that will generate an .xlsx containg our content and the minimum number of XML parts required to create a .xlsx. We also have excel:xlsx-package() for creating more complex workbooks.

To generate documents, we'll need to use the XQuery APIs that accompany each toolkit.

8. Copy MarkLogic-Toolkit-for-Excel-1.0-3\xquery\spreadsheet-ml-support.xqy to <ServerRoot>MarkLogic\Modules\MarkLogic\openxml\ spreadsheet-ml-support.xqy
9. Copy MarkLogic-Toolkit-for-Word-1.2-2\xquery\ word-processing-ml-support.xqy to <ServerRoot>MarkLogic\Modules\MarkLogic\openxml\word-processing-ml-support.xqy

TOPIC

Creating Word Documents

To review, the main body of content in Word is found in the document.xml part. The body of the document is composed of block-level an inline elements. Block-level elements provide the main structure of the document and contain inline-elements. Two of the most frequently used block-level elements include paragraphs(<w:p>) and tables (<w:tbl>).

The document.xml is combined with other XML parts to create a .docx package.

Another block-level element to consider for document generation is <w:altChunk>. The <w:altChunk> element provides us a mechanism for importing content that isn't in WordProcessingML format into our Word document. The chunk is imported from a file located within the .docx package. Once the .docx is opened and the document materialized in Word, on save, the document will be transformed completely to WordProcessingML and the altChunk elements will no longer be present.

<w:altChunk>is meant for **import only**. It facilitates a onetime conversion by Word of the imported content into WordProcessingML.

<w:altChunk> can import content with the following format types:

- text/html
  - A HTML document.
- text/plain
  - A Text document.
- application/xhtml+xml
  - A XHTML document.
- application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml
  - An existing .docx package in binary form. (That's right, we can import other Word documents.)

We include an example in our walkthrough as anyone considering document generation for Word documents should understand how <w:altChunk> works within Word.

WALKTHROUGH

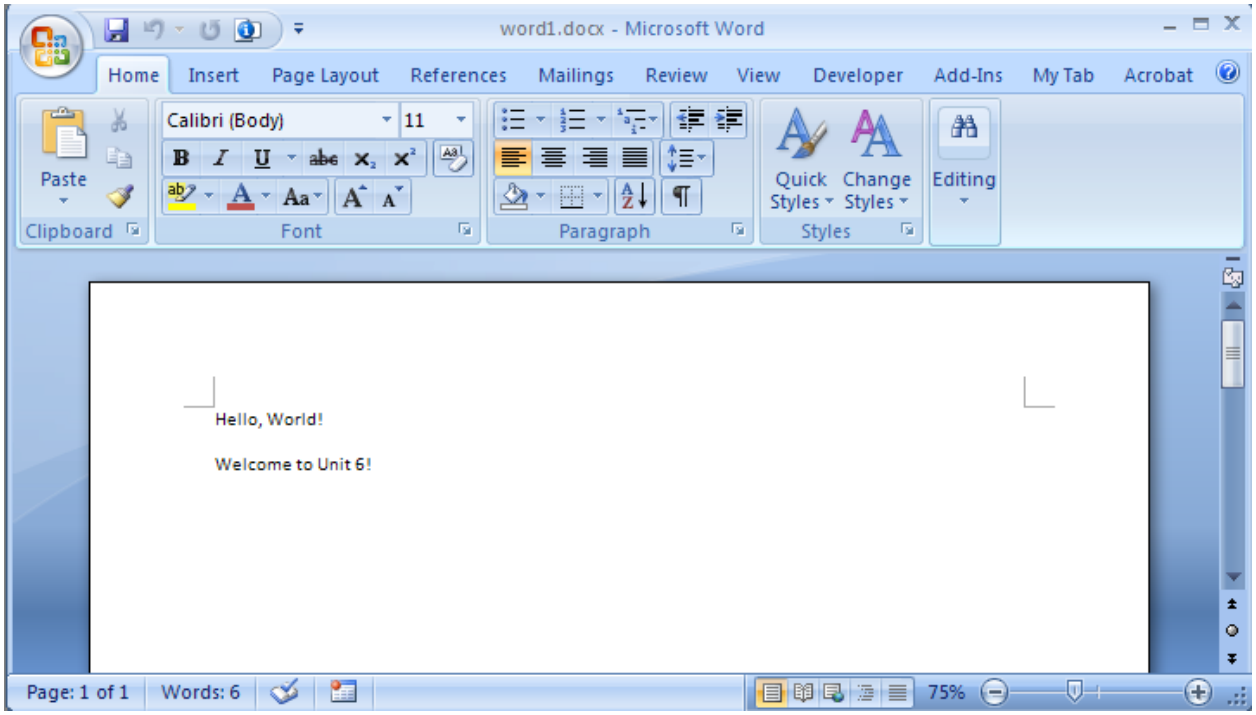
The following examples assume you are using Windows. Adjust your paths accordingly when saving and using documents in the following examples.

Exercise 2: Generating Word Documents

1. Enter the following into CQ

```
xquery version "1.0-ml";

import module namespace ooxml= "http://marklogic.com/openxml" at
"/MarkLogic/openxml/word-processing-ml-support.xqy";

let $para1 := ooxml:create-paragraph("Hello, World!")
let $para2 := ooxml:create-paragraph("Welcome to Unit 6!")
let $paras := ($para1, $para2)
return xdmp:save("C:\word1.docx",ooxml:create-simple-
docx(ooxml:document(ooxml:body($paras))))
```

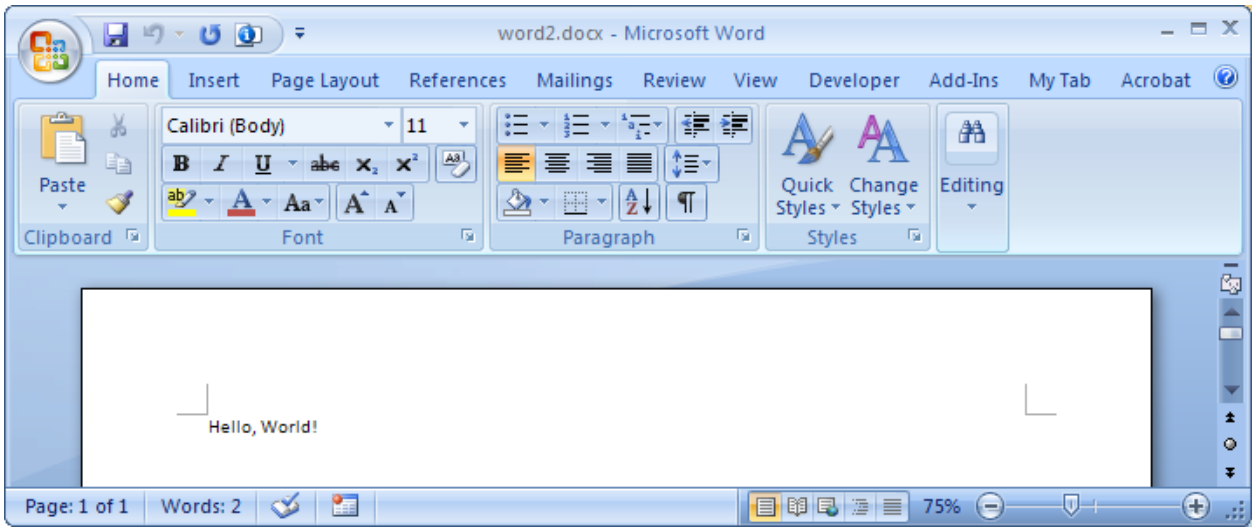
2. Click XML
3. Open C:\word1.docx



4. Enter the following into CQ

```
xquery version "1.0-ml";

import module namespace ooxml= "http://marklogic.com/openxml" at
"/MarkLogic/openxml/word-processing-ml-support.xqy";

let $text := ooxml:text("Hello, ")
let $text2 := ooxml:text("World!")
let $run := ooxml:run(($text,$text2))
let $para := ooxml:paragraph($run)
let $body := ooxml:body($para)
let $document := ooxml:document($body)
return xdmp:save("C:\word2.docx",ooxml:create-simple-docx($document))
```

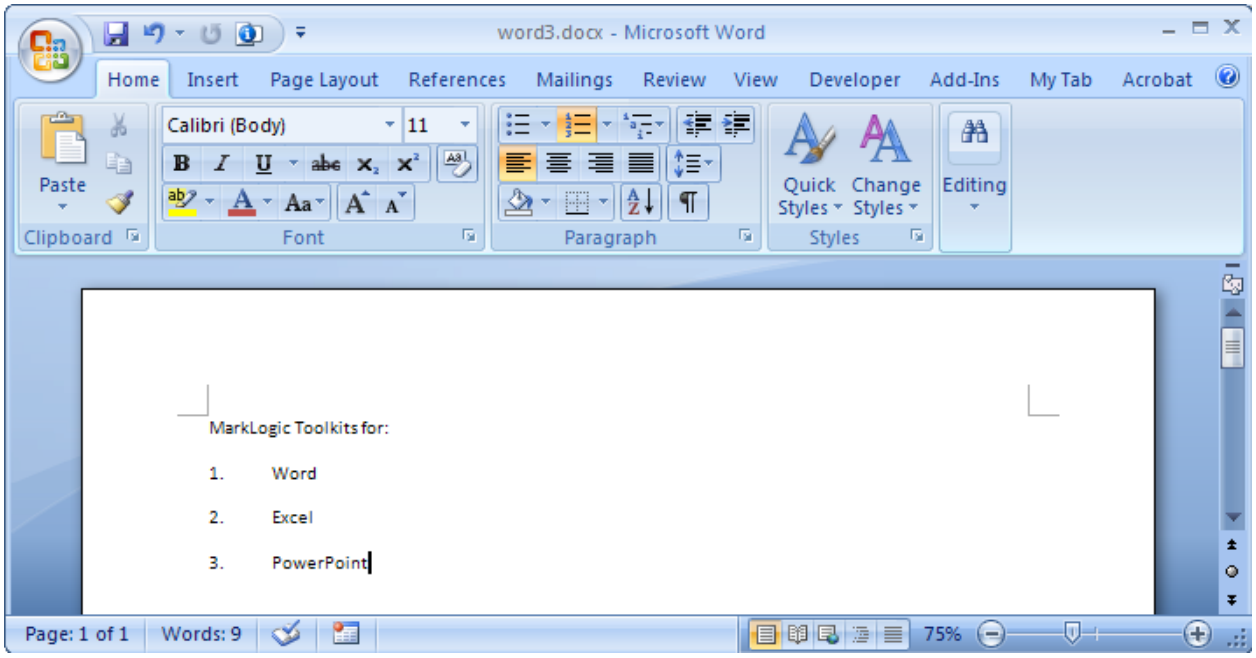
5. Click XML
6. Open the file C:\word2.docx



7. Enter the following into CQ

```
xquery version "1.0-ml";

import module namespace ooxml= "http://marklogic.com/openxml" at
"/MarkLogic/openxml/word-processing-ml-support.xqy";

let $styles := ooxml:list-paragraph-property("1")
let $para1 := ooxml:paragraph((ooxml:run(ooxml:text("MarkLogic
Toolkits for:"))))
let $para2 := ooxml:paragraph((ooxml:run(ooxml:text("Word"))),$styles)
let $para3 :=
ooxml:paragraph((ooxml:run(ooxml:text("Excel"))),$styles)
let $para4 :=
ooxml:paragraph((ooxml:run(ooxml:text("PowerPoint"))),$styles)
let $paras := ($para1, $para2, $para3, $para4)

return xdmp:save("C:\word3.docx",ooxml:create-simple-
docx(ooxml:document(ooxml:body($paras))))
```

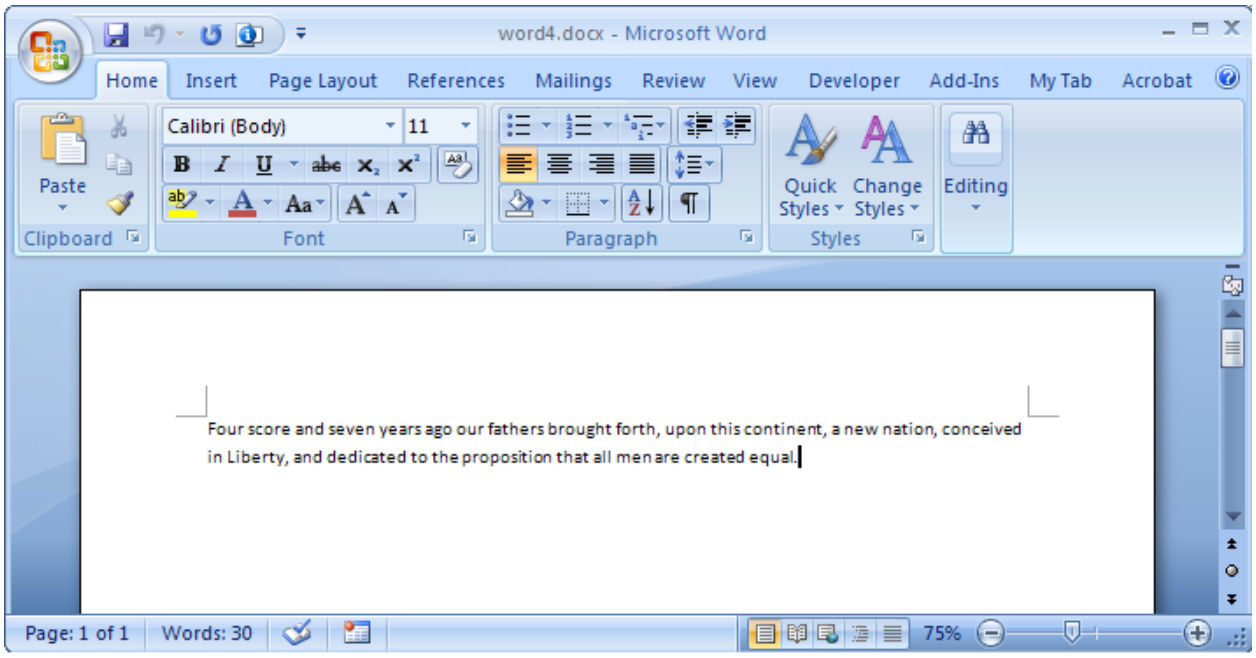
8. Click XML
9. Open C:\word3.docx



10. Enter the following into CQ

```
xquery version "1.0-ml";
import module namespace ooxml= "http://marklogic.com/openxml" at
"/MarkLogic/openxml/word-processing-ml-support.xqy";

let $content-types:= ooxml:simple-content-types()
let $rels := ooxml:package-rels()
let $para := ooxml:create-paragraph("Four score and seven years ago
our fathers brought forth, upon this continent, a new nation,
conceived in Liberty, and dedicated to the proposition that all men
are created equal.")

let $document := ooxml:document(ooxml:body($para))
let $package := ooxml:docx-package($content-types, $rels, $document)

return xdmp:save("C:\word4.docx", $package)
```

11. Click XML
12. Open C:\word4.docx



Finally, we'll create a document using <w:altChunk> . Our document will import a wiki entry, a text file, and a .docx.

13. Copy word\import.txt to C:\import.txt on your file system
14. Load word\import.docx to the Documents database

In CQ with Toolkits selected as content source (adjust path to file accordingly):

```
xquery version "1.0-ml";

let $filename := "/import.docx"
return xdmp:document-load("C:\documents\unit6\import.docx",
        <options xmlns="xdmp:document-load">
             <uri>{$filename}</uri>
        </options>)
```

15. Create a file C:\Program Files\MarkLogic\Docs\altChunk.xqy
16. Open altChunk.xqy and enter the following
    a. Adjust path for import.txt accordingly
    b. If you don't have wifi access, you won't be able to import the wiki entry

```
xquery version "1.0-ml";
declare namespace gso = "generate-simple-ooxml-alt";
declare namespace html ="http://www.w3.org/1999/xhtml";

declare function gso:generate-simple-ooxml-alt(
  $content-types as node(),
  $rels as node(),
  $document as node(),
  $documentxmlrels as node(),
  $importedhtml as node(),
  $txt as node(),
  $docx as node()
) as binary()
{
let $manifest := <parts xmlns="xdmp:zip">
                    <part>[Content_Types].xml</part>
                    <part>_rels/.rels</part>
                    <part>word/document.xml</part>
                    <part>word/_rels/document.xml.rels</part>
                    <part>word/import.htm</part>
                    <part>word/import.txt</part>
                    <part>word/import.docx</part>
                  </parts>
let $parts := ($content-types, $rels, $document, $documentxmlrels,
$importedhtml, $txt, $docx)
  return
    xdmp:zip-create($manifest, $parts)
};



let $content-types :=
  <Types xmlns="http://schemas.openxmlformats.org/package/2006/content-
types">
    <Default Extension="rels" ContentType="application/vnd.openxmlformats-
package.relationships+xml"/>
      <Default Extension="xml" ContentType="application/xml" />
      <Override PartName="/word/document.xml"
ContentType="application/vnd.openxmlformats-
officedocument.wordprocessingml.document.main+xml" />
        <Default Extension="htm" ContentType="application/xhtml+xml"/>
        <Default Extension="txt" ContentType="text/plain"/>
        <Default Extension="docx"
ContentType="application/vnd.openxmlformats-
officedocument.wordprocessingml.document.main+xml"/>
  </Types>

let $rels :=
  <Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Id="rId1"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/off
iceDocument" Target="word/document.xml"/>
  </Relationships>
```

```
let $document :=
  <w:document
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
>
   <w:body>
       <w:altChunk r:id="altChunk1" />
       <w:altChunk r:id="altChunk2" />
       <w:altChunk r:id="altChunk3" />
       <w:p><w:r><w:t>Coolest document ever!</w:t></w:r></w:p>
   </w:body>
  </w:document>

let $documentxmlrels :=
<Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="altChunk1" TargetMode="Internal"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/aFC
hunk" Target="import.htm" />
<Relationship Id="altChunk2" TargetMode="Internal"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/aFC
hunk" Target="import.txt" />
<Relationship Id="altChunk3" TargetMode="Internal"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/aFC
hunk" Target="import.docx" />
</Relationships>

let $docx :=  fn:doc("/import.docx")

let $txt :=  xdmp:document-get("C:\documents\unit6\import.txt")

let $html := <html><body>
              <h1>MarkLogic</h1>{
xdmp:tidy(
xdmp:http-get("http://en.wikipedia.org/wiki/Mark_Logic",<options
xmlns="xdmp:document-get">
    <repair>full</repair>
  </options>)//*:div[@id="bodyContent"]//*:p[2]
          )[2]}
</body></html>

let $package := gso:generate-simple-ooxml-alt($content-types, $rels,
$document,$documentxmlrels, $html,$txt/text(), $docx)

let $filename :=  "hello-world.docx"
let $disposition := concat("attachment; filename=""",$filename,"""")
let $x := xdmp:add-response-header("Content-Disposition", $disposition)
let $x:= xdmp:set-response-content-type("application/vnd.openxmlformats-
officedocument.wordprocessingml.document")
 return
    $package
```

17. In IE navigate to http://localhost:8000/altChunk.xqy
18. When prompted, choose to open the document in Word

Given our understanding of WordProcessingML we used <w:altChunk> to import a .docx saved in MarkLogic, a .txt file on the local filesystem, and the MarkLogic web page from Wikipedia into a brand new .docx document that included our own constructed paragraph.

Constructing our .docx, we created our own [Content_Types].xml to include the file types for the .docx, .txt, and XHTML parts. We related these to the main content part document.xml by creating our own document.xml.rels file to include these relationships. Finally, we created our own function to zip up the document parts for us so we could deliver the final .docx package. It was that simple.

The Toolkit APIs are intended to jumpstart development with WordProcessingML and while they include a lot of functionality, are not comprehensive. Depending on your goals you may have to dig deeper into the format and may end up writing your own functions to enable the functionality you require.

TOPIC

Creating Excel Workbooks

After reading Office Open XML 101 and reviewing the SpreadsheetML API, we have an idea of how to construct Word documents on the Server. Now we'll put that into action.

To review, the main body of content in Excel is found in the sheet#.xml part, where # is the number of the sheet in the workbook. The main document part is workbook.xml, which contains references to the worksheets for the workbook and is related to them through relationships files (.rels, workbook.xml.rels, etc.)

The sheet#.xml is combined with other XML parts to create a .xlsx package.

The body of content for a worksheet is found in the <sheetData> element within sheet#.xml. <sheetData> contains rows (<row>), which contains cells (<c>).

Additionally, something very useful in Excel is the application of Named Ranges to identify a contiguous selection of rows and cells. In SpreadsheetML, this is done through use of the table element (<table>). (<table>) elements may have an optional child (<autofilter>) element. These application of these elements to our .xlsx package manifest themselves for the workbook in Excel as a selectable range in a dropdown list of named ranges for the workbook and as a header filter selection for columns included in the named range respectively.

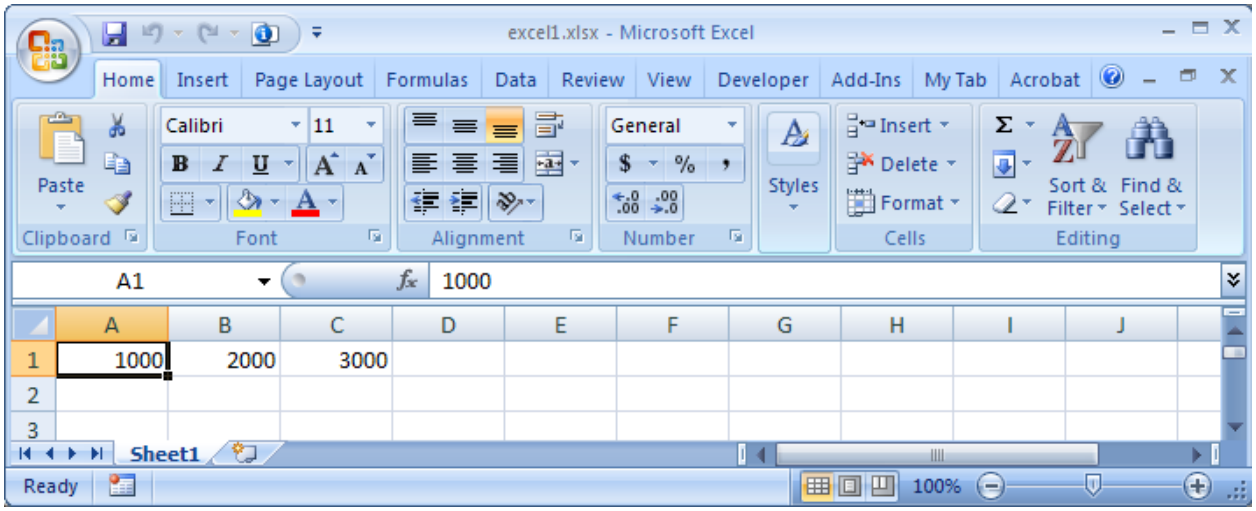
WALKTHROUGH

Exercise 3: Generating Excel Workbooks

1. Enter the following into CQ

```
xquery version "1.0-ml";
import module namespace excel= "http://marklogic.com/openxml/excel"
       at "/MarkLogic/openxml/spreadsheet-ml-support.xqy";
let $cells := ((excel:cell("A1",1000),
                excel:cell("B1",2000),
                excel:cell("C1",3000)))
let $row := excel:row($cells)
return xdmp:save("C:\excel1.xlsx",excel:create-simple-
xlsx(excel:worksheet($row)))
```

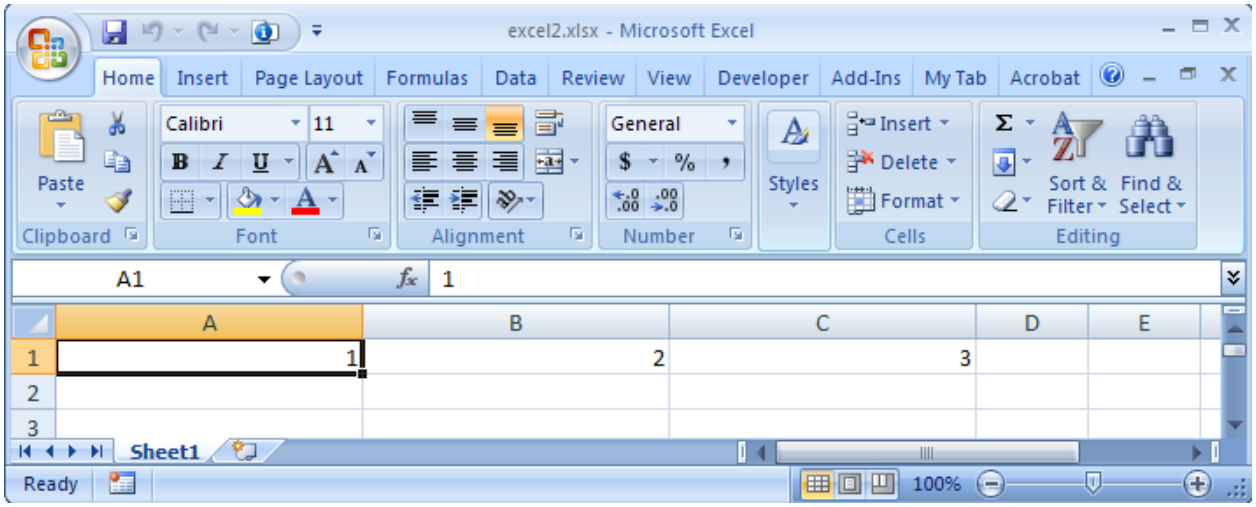
2. Click XML
3. Open C:\excel1.xlsx



4. Enter the following into CQ:

```
xquery version "1.0-ml";
import module namespace excel= "http://marklogic.com/openxml/excel"
       at "/MarkLogic/openxml/spreadsheet-ml-support.xqy";

let $cells := ((excel:cell("A1",1),
                excel:cell("B1",2),
                excel:cell("C1",3)))
let $row := excel:row($cells)
let $colwidths := excel:column-width((25,25,25))
let $worksheet := excel:worksheet($row,$colwidths)

let $package := excel:create-simple-xlsx($worksheet)

return xdmp:save("C:\excel2.xlsx",$package)
```

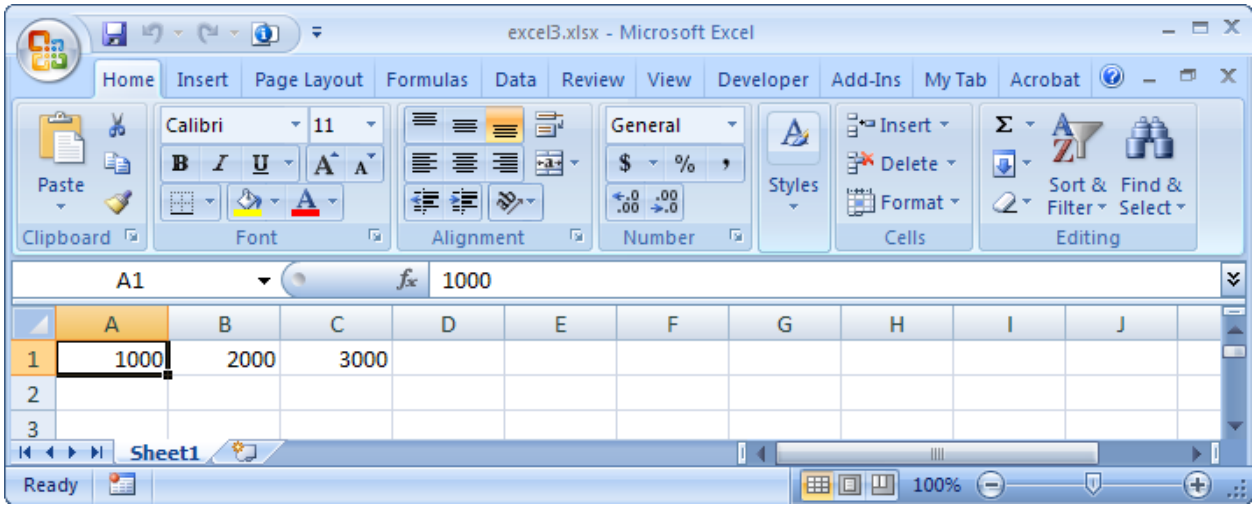
5. Click XML
6. Open C:\excel2.xlsx



7. Enter the following into CQ:

```
xquery version "1.0-ml";
import module namespace excel= "http://marklogic.com/openxml/excel"
       at "/MarkLogic/openxml/spreadsheet-ml-support.xqy";

let $cells := ((excel:cell("A1",1000),
                excel:cell("B1",2000),
                excel:cell("C1",3000)))
let $row := excel:row($cells)
let $worksheets := excel:worksheet($row)
let $ws-count := fn:count($worksheets)
let $content-types := excel:content-types($ws-count,0)
let $workbook := excel:workbook($ws-count)
let $rels :=  excel:package-rels()
let $workbookrels :=  excel:workbook-rels($ws-count)
let $package := excel:xlsx-package($content-types, $workbook, $rels,
$workbookrels, $worksheets)

return xdmp:save("C:\excel3.xlsx",$package)
```

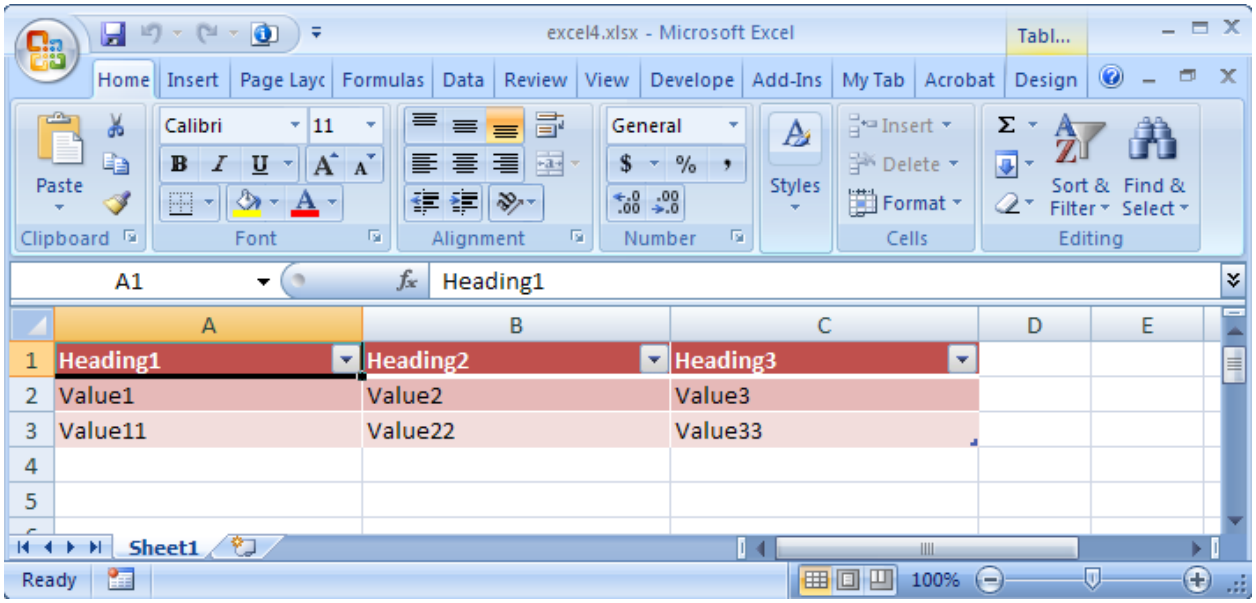
8. Click XML
9. Open C:\excel3.xlsx



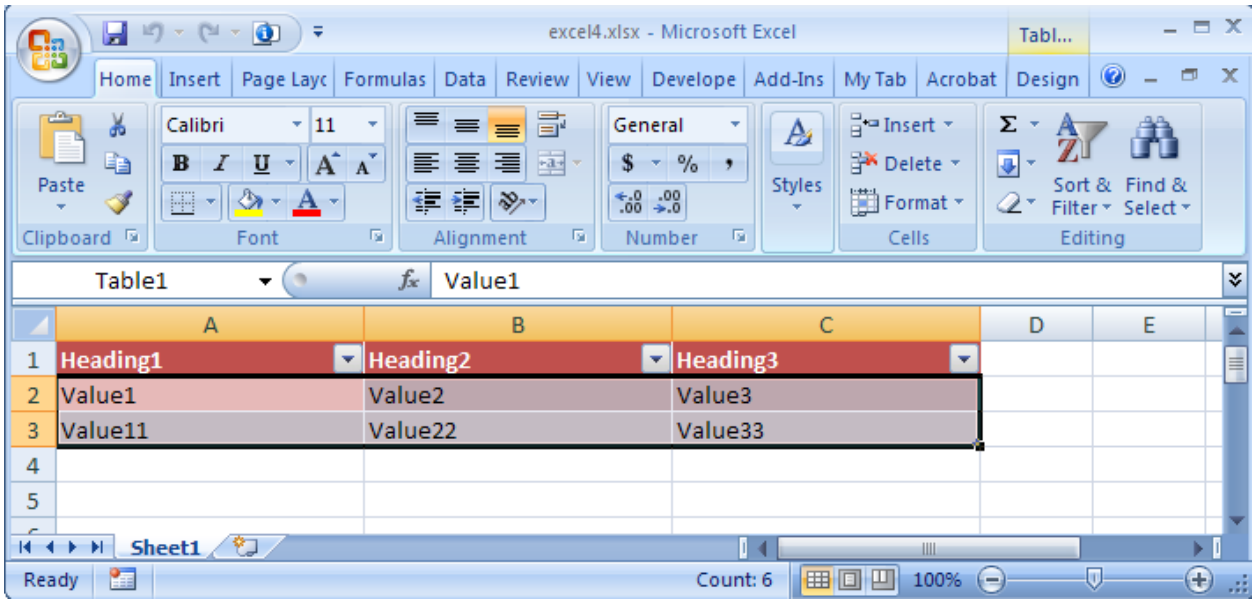
10. Enter the following into CQ:

```
xquery version "1.0-ml";
import module namespace excel= "http://marklogic.com/openxml/excel"
       at "/MarkLogic/openxml/spreadsheet-ml-support.xqy";

let $cells1 := ((excel:cell("A1","Heading1"),
                 excel:cell("B1","Heading2"),
                 excel:cell("C1","Heading3")))
let $row1 := excel:row($cells1)

let $cells2 := ((excel:cell("A2","Value1"),
                 excel:cell("B2","Value2"),
                 excel:cell("C2","Value3")))
let $row2 := excel:row($cells2)

let $cells3 := ((excel:cell("A3","Value11"),
                 excel:cell("B3","Value22"),
                 excel:cell("C3","Value33")))
let $row3 := excel:row($cells2)

let $colwidths := excel:column-width((25,25,25))

let $worksheets := excel:worksheet(($row1,$row2, $row3),$colwidths,1)
let $ws-count := fn:count($worksheets)
let $content-types := excel:content-types($ws-count,1)
let $workbook := excel:workbook($ws-count)
let $rels :=  excel:package-rels()
let $workbookrels :=  excel:workbook-rels($ws-count)
let $worksheetrels := excel:worksheet-rels(1,1)
let $table :=
excel:table(1,"A1:C2",("Heading1","Heading2","Heading3"),xs:boolean("t
rue"), xs:boolean("true"))

let $package := excel:xlsx-package($content-types, $workbook, $rels,
$workbookrels, $worksheets, $worksheetrels, $table)

return ($table, xdmp:save("C:\excel4.xlsx",$package))
```

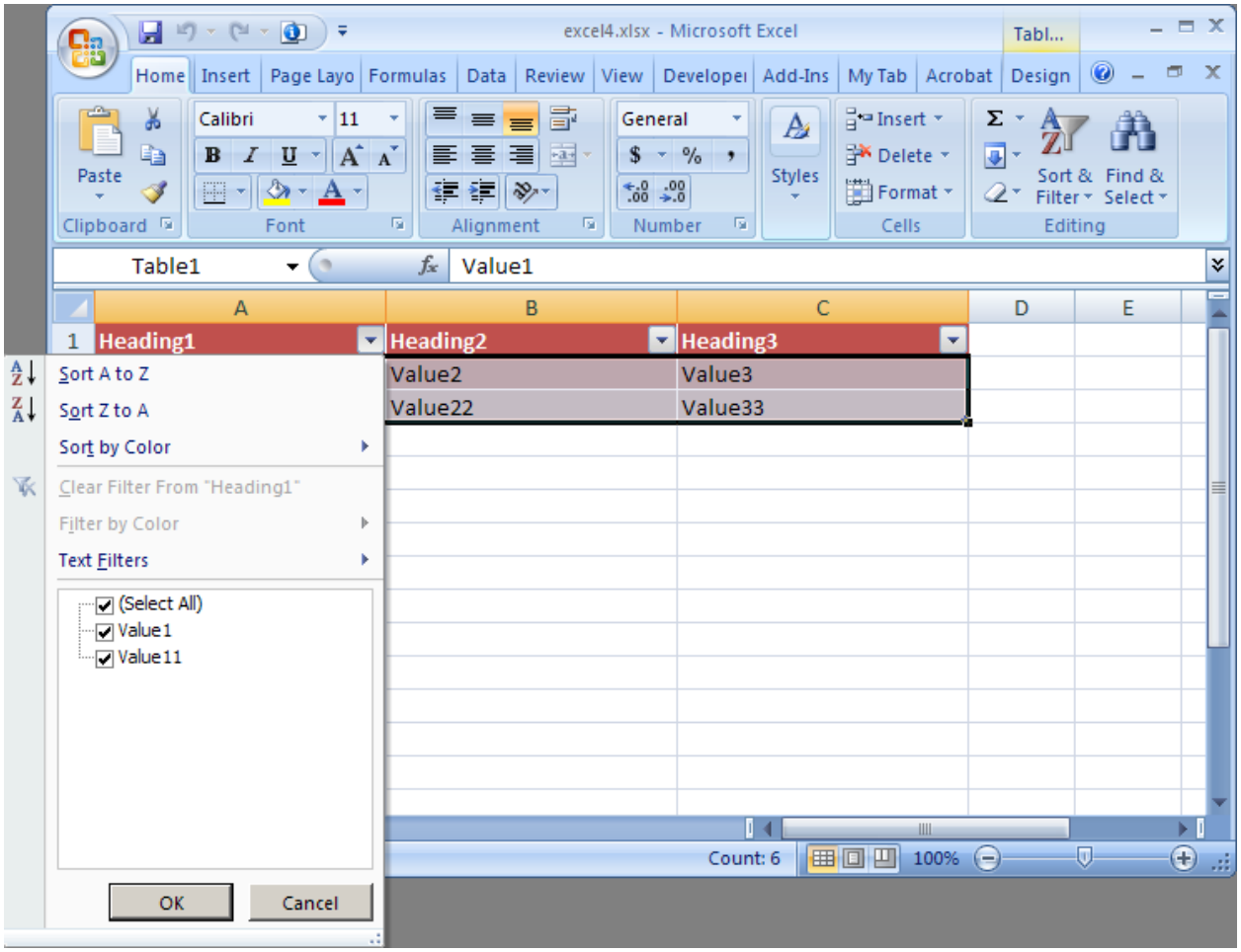
11. Click XML
12. Open C:\excel4.xlsx



13. Click on the down arrow next to A1 in the upper left to view our named range “Table1” and select it.



14. Next click any down arrow next to any of our headings to reveal the filter list.



If you uncheck any item in the dropdown filter and click "OK", the row containing the unchecked value is then made invisible in Excel as the filter is applied to the entire range. If we were to create a separate named range for each column, then the filter would apply only to values within that column.

Note: Named ranges added to the workbook are named by count ("Table1","Table2", "Table3", etc.). This is a function of the API, and not Excel. But now that you know what functions to look at, you can change that depending on your requirements.

R1C1 vs A1 notation

There are 2 ways to identify a cell in Excel. One is by using A1 notation, where a letter is used for the column identifier, immediately followed by a number which indicates the row position for the same cell.

A1 notation is what Excel uses by default within each worksheet for identifying cells.

R1C1 notation refers to the row number, followed by a column number. An example of this notation is: R2C2 (identifies cell at row 2 column 2).

In the above examples, we explicitly defined our cells for simplicity. A more likely scenario would be to dynamically generate our cell column and row position numbers from within loops or position of our query results. We can pass a row index and a column index to the `excel:r1c1-to-a1()` function and have it generate A1 notation for us.
Sometimes your query may fetch rows.

15. Enter the following in CQ and evaluate.

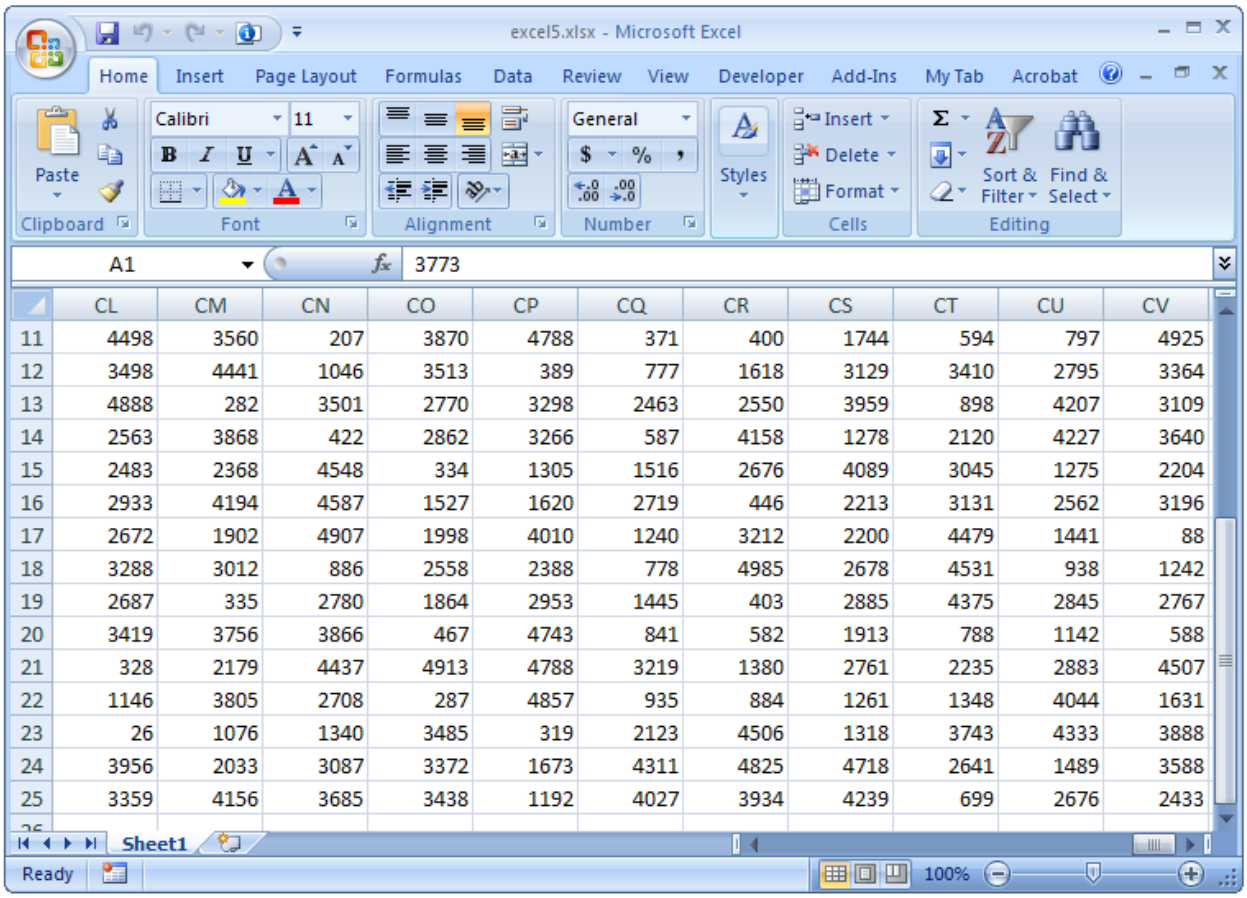
```
xquery version "1.0-ml";
import module namespace excel= "http://marklogic.com/openxml/excel"
       at "/MarkLogic/openxml/spreadsheet-ml-support.xqy";

declare namespace ms =
"http://schemas.openxmlformats.org/spreadsheetml/2006/main";

let $rows:= for $j in 1 to 25
            return excel:row(
                       for $i in 1 to 100
                       return
                           excel:cell(
                                   excel:r1c1-to-a1($j,$i),
                                   xdmp:random(5000)
                                     )
                            )
return xdmp:save("C:\excel5.xlsx",
                 excel:create-simple-xlsx(excel:worksheet($rows)))
```

Other times you may be starting with columns.

16. Enter the following in CQ and evaluate

```
xquery version "1.0-ml";
import module namespace excel= "http://marklogic.com/openxml/excel"
       at "/MarkLogic/openxml/spreadsheet-ml-support.xqy";

declare namespace ms =
"http://schemas.openxmlformats.org/spreadsheetml/2006/main";

let $columns :=  for $i in 1 to 50
                 let $cells:= for $j in 1 to 50
                              return excel:cell(
                                          excel:r1c1-to-a1($j,$i),
                                          xdmp:random(5000))
                 return <column>{$cells}</column>

let $rows := for $i in 1 to fn:count($columns)
              return  excel:row($columns/ms:c[$i])
return xdmp:save("C:\excel6.xlsx",
                 excel:create-simple-xlsx(excel:worksheet($rows)))
```

17. Open excel5.xlsx and excel6.xlsx to view the results



	CL	CM	CN	CO	CP	CQ	CR	CS	CT	CU	CV
11	4498	3560	207	3870	4788	371	400	1744	594	797	4925
12	3498	4441	1046	3513	389	777	1618	3129	3410	2795	3364
13	4888	282	3501	2770	3298	2463	2550	3959	898	4207	3109
14	2563	3868	422	2862	3266	587	4158	1278	2120	4227	3640
15	2483	2368	4548	334	1305	1516	2676	4089	3045	1275	2204
16	2933	4194	4587	1527	1620	2719	446	2213	3131	2562	3196
17	2672	1902	4907	1998	4010	1240	3212	2200	4479	1441	88
18	3288	3012	886	2558	2388	778	4985	2678	4531	938	1242
19	2687	335	2780	1864	2953	1445	403	2885	4375	2845	2767
20	3419	3756	3866	467	4743	841	582	1913	788	1142	588
21	328	2179	4437	4913	4788	3219	1380	2761	2235	2883	4507
22	1146	3805	2708	287	4857	935	884	1261	1348	4044	1631
23	26	1076	1340	3485	319	2123	4506	1318	3743	4333	3888
24	3956	2033	3087	3372	1673	4311	4825	4718	2641	1489	3588
25	3359	4156	3685	3438	1192	4027	3934	4239	699	2676	2433

Conclusion

To learn more about the Toolkits for Word, Excel, and PowerPoint, visit us at:

http://developer.marklogic.com