**MarkLogic Sample Authoring Application for Microsoft Excel®**

**MarkLogic Corporation**

1

\

Release 1.0

November, 2011

Last Revised: 1.0-1, November, 2011

© Copyright 2002-2012 by MarkLogic Corporation. All rights reserved worldwide.

Contents

[**1.0** **Introduction** 3](#_Toc309478977)

[**2.0** **Up and Running** 4](#_Toc309478978)

[**3.0** **Custom Configuration** 6](#_Toc309478979)

[3.1 Tagging 6](#_Toc309478980)

[3.2 Associating Custom Metadata 10](#_Toc309478981)

[3.3 Search and Reuse 13](#_Toc309478982)

[3.4 Macros 18](#_Toc309478983)

[**4.0** **Other Files of Interest** 19](#_Toc309478984)

1. **Introduction**

The MarkLogic Authoring Sample App for Microsoft Excel® is a sample application, built using The MarkLogic Toolkit for Microsoft Excel 2.0-1. Please download and install the Toolkit before reading this guide. The guide included with the Toolkit provides details on how to install and configure the Add-in for Excel as well as Sample applications.

*Authoring Sample Application Requirements:*

1. Office 2007 or Office 2010
2. MarkLogic Toolkit for Micrsoft Excel 2.0-1

*If you are already have the MarkLogic Toolkit for Microsoft Excel and the Authoring Sample App installed and wish to jump right in and start using the application as-is, jump to section* ***Section 2.0*** *Up and Running to get started.*

The Toolkits are built to simplify the use of Open XML and to provide a jumpstart to developers who are working with Office 2007/2010, Open XML, and MarkLogic Server.

The Authoring Sample Application was created to provide a rich example of the type of application developers can build using the MarkLogic Toolkit for Excel®. Please note, this is just one example of the type of application you can build using the Toolkit. Also, the application is not the Toolkit, nor does it encompass all Toolkit functionality. This application was built as a solution to common use cases. The Authoring application allows authors to:

* enrich Excel workbooks by tagging workbooks, worksheets, and worksheet components
* associate custom metadata with those identified components and edit
* search those identified components once saved in MarkLogic Server
* reuse components from workbooks saved in MarkLogic Server within the workbook being authored while retaining all associated metadata across documents

The functionality of the application is detailed online, in blog posts, and in alternate guides found at developer.marklogic.com. However, to use the application out of the box, some minor configuration is required.

**Section 2.0** of this guide details what files you will need to update to use the Authoring sample application as-is.

The application is also configurable. We understand not everyone will want to code their own Office Toolkit application, so **Section 3.0** details what configuration files are available, and how they can be updated to change the display and functionality of the application without modifying any code.

Finally, in **Section 4.0**, we provide details on files of interest to those developers who want to dive in and just hack this thing.

1. **Up and Running**

To use the Sample application as-is, you will need to set the URL for the application in 3 places:

1. The URL registry entry for the Add-in
2. <Application-Root>\xlAuthor\js\authoring.js
3. <Application-Root>\xlAuthor\config\config.xqy

You will see below that you can minimize the required configuration to 2 places for deployment.

For our examples below, we’ll assume we are going to install the Authoring sample application on the HTTP Server at port 8000, the default Server MarkLogic makes available on install.\*

1. The root directory for applications on the HTTP Server on port 8000 is:

<Server-Install-Directory>\Docs

1. So we place the Author directory for the Sample application at:

<Server-Install-Directory>\Docs\xlAuthor

*\*Note: In MarkLogic 5, the Docs directory referenced by the HTTP Server on port 8000 is now subject to a URL rewriter being used for other MarkLogic applications. To be safe, you’ll want to create your own application server on your own port and create your own directory on the filesystem to install this application.*

*If you’re unfamiliar with creating and configuring HTTP Servers in MarkLogic, see the HTTP Server overview in the Administrator’s guide available at* [*http://developer.marklogic.com*](http://developer.marklogic.com)*.*

*Example directory for steps above: <Server-Install-Directory>\YourDocs\xlAuthor*

**URL registry entry for the Add-in**

Per the Toolkit for Excel guide, we know we can configure the .msi to include the URL for our application. We also know that if we’ve already installed the toolkit, we can just run regedit and set the following key to the desired URL:

HKEY\_CURRENT\_USER/MarkLogicAddinConfiguration/Excel/URL

For our example the URL value could be: <http://marklogic.myserver.com:8023/xlAuthor> where the HTTP Server is running on the machine marklogic.myserver.com and is running on port 8030.

If running locally your URL value could be <http://localhost:8023/xlAuthor> .

**\Author\js\authoring.js**

In this file you’ll find the variable SERVER. Change the value to the URL for the application.

Example:

var SERVER=”http://localhost:8023/xlAuthor”;

***Note:*** you could update this to use MLA.getConfiguration(); This returns a MLA.Config object which includes a URL property, which is the URL found in the registry for the Add-in.

var myconfig = MLA.getConfiguration();

var SERVER = myconfig.url;

**\Author\config\config.xqy**

At the top of this file you will find 3 variables:

$config:CONFIG-PATH := "http://localhost:8023/xlAuthor/config/";

$config:USER := "user";

$config:PWD := "password"

Update these values to be your Server, and the credentials for that Server. We are getting the configuration files from the Server using xdmp:document-get(). We’ve implemented the sample this way for demonstration purposes, and for simplicity of install for most users.

We understand there are various Security considerations and reasons you may not want to to use xdmp:document-get(), or have credentials hardcoded in the .xqy. You also may want to move the configuration files onto the Server, where they might even be generated from other documents, queries, or schemas. Updating config.xqy to meet your specific requirements should be relatively simple.

So that’s it. After you’ve updated these 3 items, the next time you open Excel you will find the Authoring application available and ready for use.

1. **Custom Configuration**

Three configuration files are provided that allow you to customize the look and functionality of the Authoring sample application without editing any code. They are relatively simple to understand and xlAuthor.xsd is provided to allow you to validate any custom edits you make to these files. All files can be found in the /config directory of the Authoring application. Let’s look at each configuration in detail.

The 3 configuration files are:

* tags.xml
* metadata.xml
* search.xml

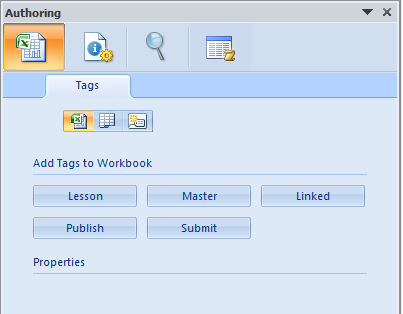
We are examining the files in this order so they will make more sense within the context of the application.

## 3.1 Tagging

Authors can enrich workbooks, worksheets, and sheet components in Excel by using tags. We provide a tags palette to enable organizations to predefine the labels for enrichment, as well as define what types of tags are available for use. Using the configuration tags.xml file allows us to create numerous ways to enrich presentations at the click of a button.

The configuration file defines the buttons that will be available on your tag palette, and also generates the javascript functions required by those buttons to insert the appropriate tags.

The tags palette looks like:



The first thing to notice is that there are 3 types of items to tag available.



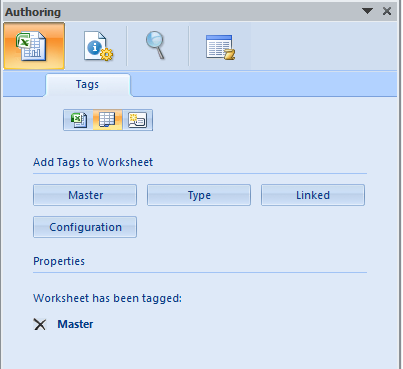
Authors can tag workbooks, worksheets, and sheet components (charts and named ranges). Clicking the icon for the associated tag type will display the buttons available for that type of object in the task pane. The label of the button is the name of the tag as it will be inserted into the workbook when the button is clicked. Once the workbook is saved to MarkLogic Server, this will be a value applications can then search on.

As you click the icons, if the selected item (workbook, worksheet, or sheet component) has been tagged, those tags will be displayed underneath the tag buttons in the Properties section. Workbook tags are tags for the active workbook, so clicking the workbook icon will always show you all workbook tags. For worksheets and sheet components however, the tags displayed in the Properties panel are for the active worksheet and active component respectively.

So if you were to click the worksheet tag icon, and start rifling through your workbook to inspect worksheets, the Properties panel will update automatically to show you the tags associated with that active worksheet. Likewise, if you select the sheet component icon and start clicking components within a worksheet, the Properties panel will update to show you what tags are associated with the selected components.

The number of buttons available for each type of tag can vary depending on your requirements. For this application, an object can only be tagged with a given tag label once.

Example: We want to tag our worksheet as ‘Master’. We click the button, the tag is inserted, and now our screen looks like the following:

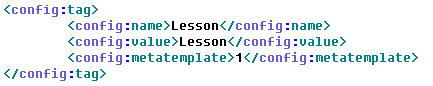


If you were to click the ‘Master’ button again, no tag would be added to the worksheet, as it has already been tagged ‘Master’. For this example, we could add ‘Type’, ‘Linked’, and ‘Configuration’ tags if we wished though. If we did, we’d see those displayed under ‘Master’ in the properties as well.

There is a delete icon next to the tag we’ve added. If we click this, the Tag will be removed. Remember, there is no tag interface in Excel other than what we’ve provided with the Sample App, so management of tags is done completely from within the Sample Application.

**tags.xml**

The configuration for tags looks like:



For each tag button in the control palette under ‘Add Tags to :”, we’ll find a config:tag element. The parent of this element will either be config:workbook, config:worksheet, or config:component, identifying which tag pallete it is associated with and which type of object the button can tag.

The children elements of config:tag are:

config:name : the tag name, also used as the button label

config:value: the tag value

config:metatemplate: the id of the metadata template found within metadata.xml to be associated with the inserted control

When a user clicks a button, a tag will be inserted into the workbook using the config:name and config:value as the tag name and value and a custom XML part will also be added to the workbook and associated with the inserted tag.

Which metadata form is added for the tag is determined by the metadata template value as it relates to the tag, which we will see in the next section when we discuss metadata.xml.

So for tags displayed in the Properties panel, when you click the delete button to remove a tag, you are also removing the tag’s associated custom metadata part.

Note: We use tags essentially as lists in Excel. There is no hierarchy, or concept of embedding tags within tags. For the Sample Authoring App for Excel, we have followed PowerPoint’s native functionality with respect to Tags (as Excel has no concept of Tags, and PowerPoint does.) So similar to how enrichment is defined natively by PowerPoint, the Sample Authoring App for Excel will not allow you to add a Tag with a specified name and value to an object if a Tag for that object with that specified name and value already exists.

## 3.2 Associating Custom Metadata

Whenever a Tag is added to the document being authored, a custom metadata part will be added to the .xlsx package and associated with the added Tag. These custom parts can be associated explicitly. If no association is defined, a default custom part is added.

We are using Dublin Core Metadata for the Authoring sample application.

**metadata.xml**

The current configuration looks like:

****

You can add N number of config:template elements. These @id is used to map the config:template child metadata form to the config:metatemplate value found in the associated content control definition within controls.xml.

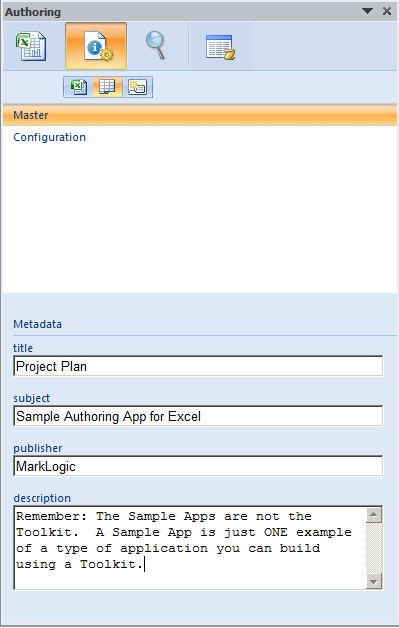
The possible elements available for simple dublin core metadata are:

|  |  |  |
| --- | --- | --- |
| dc:title | dc:creator | dc:subject |
| dc:description | dc:publisher | dc:contributor |
| dc:date | dc:type | dc:format |
| dc:identifier | dc:source | dc:language |
| dc:relation | dc:coverage | dc:rights |

More info on Dublin core elements can be found here: <http://dublincore.org/documents/dces/>

Note: For the Sample Application, the first dc:identifier and dc:description elements in each form will not be displayed or accessible to users. We use those 2 elements internally with our application for associating the metadata part with a tag as well as supporting the roundtripping of saved slide components.

The metadata form can be found on the metadata panel.



Notice across the top, we again can select which tags to examine by selecting either the workbook, worksheet, or sheet component icons.

In this example, the worksheet icon is selected, and there are 2 tags associated with the active worksheet: ‘Master’ and ‘Configuration’. The ‘Master’ tag is selected and so the form for its associated metadata part is displayed below, displaying metadata information we can add and edit.

Again, if we were to change the worksheet we were looking at in the workbook, the tag lists would update to display tags associated with those worksheets. Also, when we select a component, as we click through the components on the active worksheet, the tags associated with those components will be displayed. Selecting a tag from the list will display its associated metadata form.

The form is created by taking the local name for the elements in the form, and creating text fields for entry. As the user enters content, whenever they change entry fields, the values are saved within the metadata part in the .xlsx package.

## 3.3 Search and Reuse

When we search, we can search on Worksheet Components, Worksheets, and Macros.

A search on worksheet components will bring up a list containing table snippets for named ranges and images of charts. Assuming you’ve saved workbooks containing tagged components to MarkLogic Server. The search on named range components is done against the worksheets of the extracted .xlsx within MarkLogic. The search on chart components is done against the chart’s associated metadata form that was also extracted from the .xlsx package.

The title of the workbook containing the component item displayed in the results list comes from the document properties of the saved workbook. If the workbook does not have a title, the URI of the saved .xlsx in MarkLogic is displayed. Clicking on the title will open the workbook.

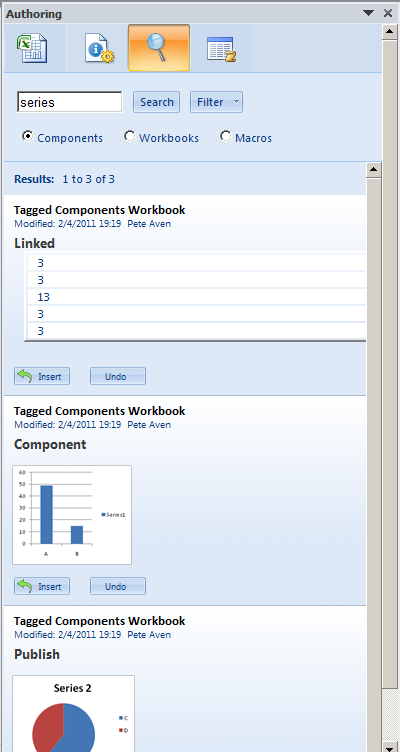
Underneath the workbook title metadata is provided. This too comes from the document properties within the .xlsx package.

Below this is the name of the tag used to identify the component, along with either a table snippet displayed for named range components, or an image of the chart found.

Finally, underneath the search result you’ll find insert and undo buttons, allowing you to add and remove your search results and their associated metadata to the document you’re authoring.

Note: the undo button is only active after its associated component has been inserted. When you click undo, you remove the component from the worksheet, and also remove it’s associated custom XML part from the active document.

Component Search:

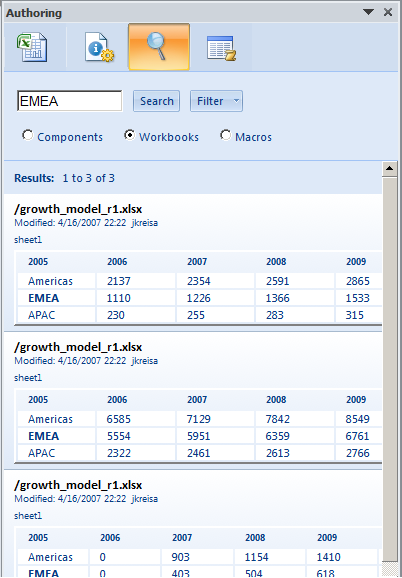


WHERE DO THE CHART THUMBNAILS COME FROM?

When you tag a chart, you essentially save an image of the chart within the custom XML part you associate with it for maintaining your custom metadata. The chart is saved as a base 64 encoded string. This is done so that you may share a snapshot of the chart across multiple applications.

Worksheet Search:

When we search on worksheets, any worksheet from any workbook containing the search text is returned. Search results will start with worksheet’s title and metadata. Following this you’ll note the name of the sheet in the source .xlsx. Finally, a table snippet is provided. The row containing the highlighted search hit is returned, along with the row preceding and following it. Clicking the title of the search result will again open the source workbook found in MarkLogic.

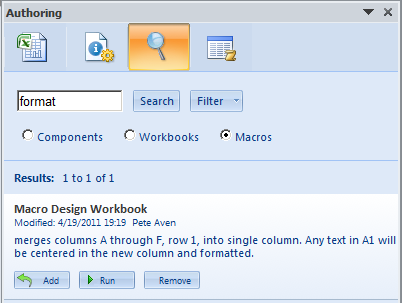


Macro Search:

We can search on macro code, or any metadata associated with macros contained in workbooks saved to MarkLogic Server. How this is done will make more sense when we examine the macro tab in the next section. But for now, just know you can do it.

When we search on macros, the title of the .xlsx that contains them is displayed, but you are not allowed to open the parent workbook. Underneath the title, we have the customary metadata, and below this, a description of the macro as provided within the description section of a custom XML part associated with the macro.

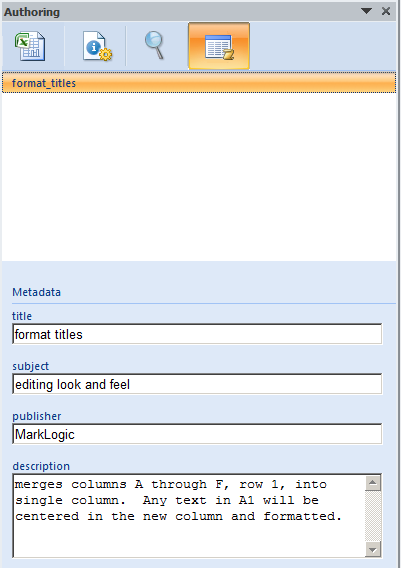
Finally we see 3 buttons that allow us to add the macro to the workbook we’re authoring, run the macro once its added, and remove the macro from the workbook if we determine its not the right one for us.



## 

## 3.4 Macros

If the workbook we’re authoring contains macros, we can go to the macro tab, and see a list of those macros displayed. Associated with each will be a custom XML part, and when we click the name of the macro, its form will be displayed for us to enter information. It is very similar to the metadata tab, but specific to Macros.



Macros are saved as .bin files in a .xlsx package. When using the macro pane, selecting a macro title saves the macro code as text in the associated custom XML part. As Macro titles may not be known before time, the default metadata part specified in metadata.xml is used for each macro selected.

1. **Other Files of Interest**

So maybe the config files don’t afford you all the functionality you require. Well, there are degress of “configurability”, and since this sample is open source, you can pretty much configure anything and everything you’d like. Here we discuss files of interest, the areas you may want to edit or update to create the application you ultimately want.

**/Author/js**

First, let’s discuss framework code vs. user code. The following two files can be considered framework code for Add-in functionality on the client:

/js/MarkLogicExcelAddin.js – functions for getting XML in/out of the document being authored

/js/MarkLogicExcelEventSupport.js – captures application events from the Add-in

/js/MarkLogicExcelEventHandlers.js – called by MarkLogicEventSupport.js, this is where you can add custom event code.

The user, or developer, specific code can be found in:

/js/authoring.js – application specific code

authoring.js is the application specific code that can be modified at will.

The code in the MarkLogicExcelAddin.js and MarkLogicExceEventSupport.js files should never change, as they provide the framework APIs that facilitate communication between the document being authored and the html page displayed in the Add-in task pane.

You can customize event handling however in MarkLogicEventHandlers.js. In some cases, functions here call others in authoring.js, which also can be modified.

The seventeen events captured are:

sheetActivate()

sheetBeforeDoubleClick()

sheetBeforeRightClick()

sheetChange()

sheetDeactivate()

rangeSelected()

-sheetSelectionChange Event only caught when selection is named range

workbookActivate()

workbookAfterXmlExport()

workbookAfterXmlImport()

workbookBeforeXmlExport()

workbookBeforeXmlImport()

workbookBeforeClose()

workbookBeforeSave()

workbookDeactivate()

workbookNewSheet()

workbookOpen()

chartObjectMouseDown()

**/Author/config**

The configuration files discussed in Section 3.0 are found here as well as config.xqy.

This file is of interest as it generates the HTML for everything for everything that is configurable, and for controls the associated JavaScript functions as well.

**Author/search**

/search/search.xqy – used by the Search tab

/search/metadata-search.xqy – used by the Compare tab

If you want to change the search criteria or add additional parameters, you’ll want to look at search.xqy.

If you want to change the metadata to be something other than Dublin core, you can change it in the metadata.xml configuration file, but your search will not change until you update metadata-search.xqy.

**Author/css**

/css/authoring.css

The browser control, when embedded within Office, uses IE8 in compatibility mode. (There is no way to update this unless you update the registry, which we didn’t want to do) By default this mode always adds a scrollbar to the application, even when there’s nothing to scroll.

We didn’t like how this looked, so we’ve set the overflow hidden for <html> and <body>. This will affect certain tabs. So if you add more buttons, properties, or metadata than will fit on a panel, you may not be able to view them unless you unset this property as you’ll have no way to scroll.

**Conclusion**

Happy Hacking!