**MarkLogic Sample Authoring Application for PowerPoint®**

**MarkLogic Corporation**

1

\

Release 1.0

September, 2010

Last Revised: 1.0-2, November, 2011

© Copyright 2002-2011 by MarkLogic Corporation. All rights reserved worldwide.

Contents

[**1.0** **Introduction** 3](#_Toc273629239)

[**2.0** **Up and Running** 4](#_Toc273629240)

[**3.0** **Custom Configuration** 6](#_Toc273629241)

[3.1 Tagging 6](#_Toc273629242)

[3.2 Associating Custom Metadata 10](#_Toc273629243)

[3.3 Search and Reuse 12](#_Toc273629244)

[**4.0** **Other Files of Interest** 17](#_Toc273629245)

1. **Introduction**

The MarkLogic Authoring Sample App for PowerPoint® is a sample application, built using The MarkLogic Toolkit for PowerPoint 1.1-1. 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 Addin for PowerPoint as well as Sample applications.

*Authoring Sample Application Requirements:*

1. Office 2007 or Office 2010 (32 bit)
2. MarkLogic Toolkit for PowerPoint 1.1-1

*If you are already have the MarkLogic Tookit for PowerPoint 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 PowerPoint®. 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 PowerPoint presentations by tagging presentations, slides, and slide components
* associate custom metadata with those identified components and edit
* search those identified components once saved in MarkLogic Server
* reuse components from presentations saved in MarkLogic Server within the presentation 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>\Author\js\authoring.js
3. <Application-Root>\Author\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\Author

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

Per the Toolkit for PowerPoint 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/PowerPoint/URL

For our example the URL value could be: <http://marklogic.myserver.com:8000/pptAuthor> where the HTTP Server is running on the machine marklogic.myserver.com and is running on port 8000.

If running locally your URL value could be <http://localhost:8000/pptAuthor> .

The PowerPoint Toolkit provides a mechanism for saving directly to MarkLogic from the Office Button. To insure this works properly, you’ll also want to update the following key values with your username and password respectively.

HKEY\_CURRENT\_USER/MarkLogicAddinConfiguration/PowerPoint/User

HKEY\_CURRENT\_USER/MarkLogicAddinConfiguration/PowerPoint/Auth

**\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/pptAuthor”;

***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:8000/pptAuthor/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 PowerPoint you will find the Authoring application available and ready for use.

1. **Custom Configuration**

3 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 pptAuthor.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.

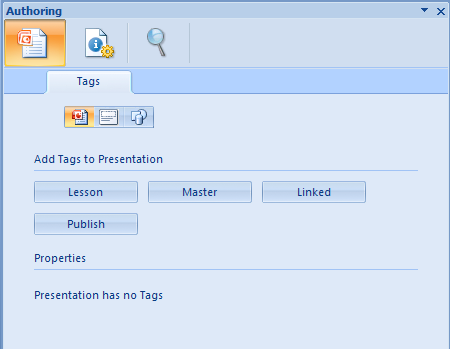
## 3.1 Tagging

Authors can enrich presentations, slides, and slide components in PowerPoint 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.

Content Controls are only available in Word. In PowerPoint, we use Tags for enrichment, but tags are not available in the PowerPoint application natively. They are a programmatic piece of the PowerPoint object model and serialize in the XML when a presentation is saved. We’ve aimed to provide similar functionality to the MarkLogic Sample Authoring App for Word® by providing an application interface to use tags within presentations through the sample application.

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.

tags2.PNG

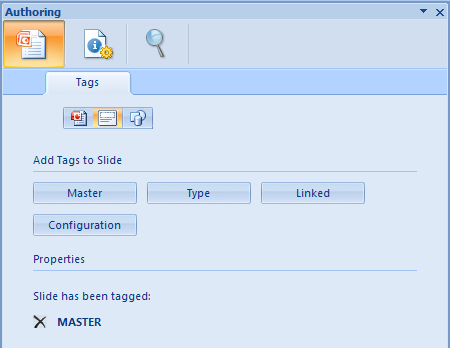
Authors can tag presentations, slides, and slide components (shapes). Clicking the icon for the associated tag type will display the buttons available for that type of object in the task pane. The label of the button is the name of the tag as it will be inserted into the presentation when the button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

As you click the icons, if the selected item (presentation, slide, or shape) has been tagged, those tags will be displayed underneath the tag buttons in the Properties section. Presentation tags are tags for the active presentation, so clicking the presentation icon will always show you all presentation tags. For Slides and Slide Components however, the tags displayed in the Properties panel are for the active slide and active component respectively.

So if you were to click the slide tag icon, and start rifling through your deck to inspect slides, the Properties panel will update automatically to show you the tags associated with that Slide. Likewise, if you select the slide component icon and start clicking components within a slide, the Properties 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 presentation as ‘Lesson’. We click the button, the tag is inserted, and now our screen looks like the following:

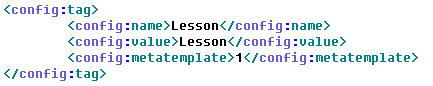


If you were to click the ‘Lesson’ button again, no tag would be added to the presentation, as it has already been tagged ‘Lesson’. For this example, we could add ‘Master’, ‘Linked’, and ‘Publish’ tags if we wished though. If we did, we’d see those displayed under ‘Lesson’ in the properties as well.

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 PowerPoint 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:presentation, config:slide, or config:shape, 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 presentation using the config:name and config:value as the tag name and value and a custom XML part will also be added to the presentation 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: Tags are essentially lists in PowerPoint. There is no hierarchy, or concept of embedding tags within tags. PowerPoint 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 .pptx 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.

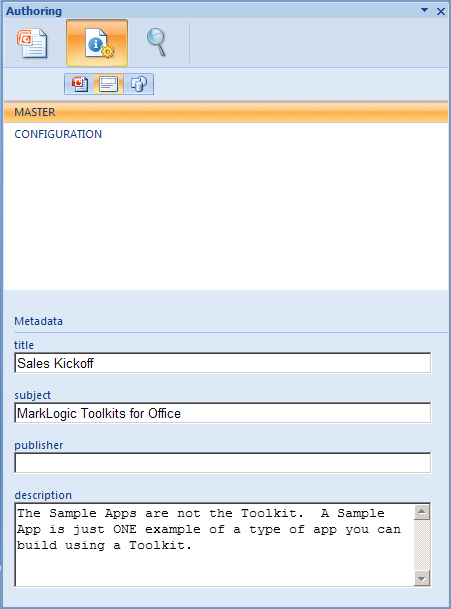
The possible elements available for simple dublin core metadata are:

|  |  |  |
| --- | --- | --- |
| 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 presentation, slide, or slide component icons.

In this example, the slide icon is selected, and there are 2 tags associated with the active slide: ‘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 slide we were looking at in the presentation, the tag lists would update to display tags associated with those slides. Also, when we select component, as we click through the components on the active slide, the tags associated with those components will be displayed. Selecting a tag from the list will display it’s 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 .pptx package.

## 3.3 Search and Reuse

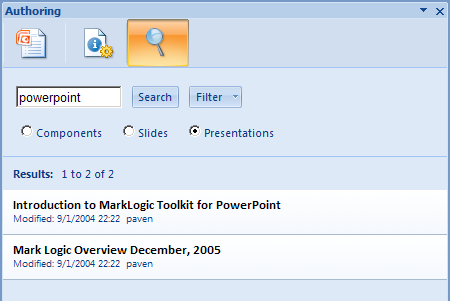
As PowerPoint is more visual, (we usually hunt for slides and cue off an image or text to find what we want), we want to explain what search results look like in the sample application.

When we search, we can search on Presentations, Slides, and Slide Components.

A search on presentations will bring up a list of presentations for any presentation containing the text provided as the search value. The title of the presentation displayed in the results list comes from the document properties of the saved presentation. If the presentation does not have a title, the URI of the saved .pptx in MarkLogic is displayed. Clicking on the title will open the presentation.

Underneath the presentation title metadata is provided. This too comes from the document properties within the .pptx package.

Presentation Search:



When we search on Slides, any slide from any presentation containing the search text is returned. Search results will start with slide’s presentation title and metadata. Following this you’ll note a slide icon, helping to inform you’re working with slides, and not slide components. Next to the icon is a snippet from the returned slide. Below the snippet is a thumbnail of the slide returned. And underneath the thumbnail are buttons for ‘insert’ and ‘undo’.

Slide Search:



Clicking insert will insert the slide into the presentation being authored. Remember, if the slide has tags and/or tagged slide components, any associated metadata parts will be retained from the source and saved to the .pptx currently being authored.

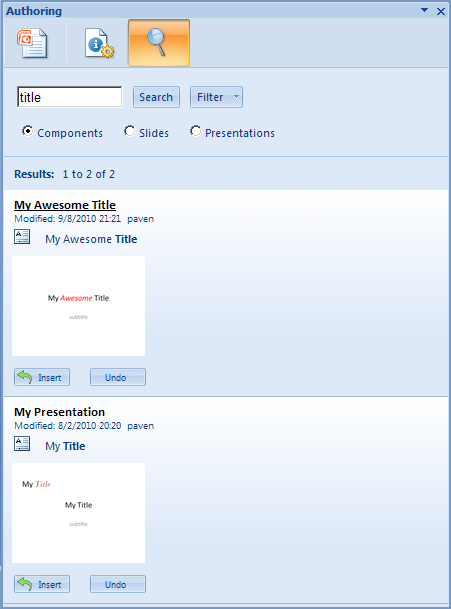
The undo button will remove the inserted slide as well as any custom XML parts that were added to the pptx package.

Note: the undo button is only active after its associated slide (or slide component) has been inserted.

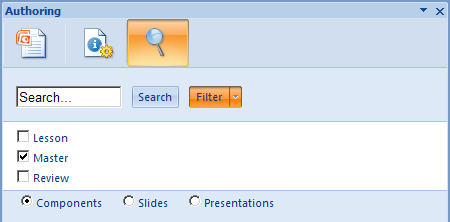
WHERE DO THE THUMBNAILS COME FROM?

When you save to MarkLogic from the Office button within PowerPoint, the .pptx as well as images for each slide in the .pptx are saved to MarkLogic.

When we search on components, we use the same slide images. The icon will either be a textbox or an image. The combination of the thumbnail, snippet, and icon help to inform which component will be inserted. But you may mistakenly insert the wrong component. That’s ok, just click undo and you can insert another.



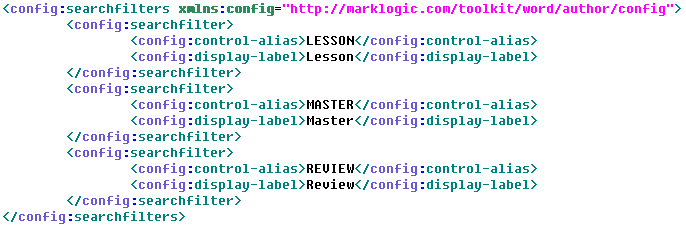
You can edit the available search filters within this file. Search filters can be seen and set on the search tab.

****

Here we see 3 possible filters. If a filter is checked, then a search will only be performed to look for document components that are identified by those checked items.

**search.xml**

The configuration for the above looks like:



config:searchfilter : each entry results in a new list item with checkbox in the filter list

config:control-alias : the value that will be used to search to filter document components

config:display-label : the label to display in the application search filter option list

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

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

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 other 2 files should never change, as it provides the framework APIs that facilitate communication between the document being authored and the html page displayed in the Add-in task pane.

In a perfect world, the above would be completely true, but for this application there is one caveat: MarkLogicPowerPointEventSupport.js is currently tightly coupled to the user code found in authoring.js. An area for improvement would be to create an event handling layer, so developers could register event listeners in authoring.js, as they would for any normal custom events listeners defined when using javascript. We chose not to do that at this time, but have it logged for a future enhancement.

But now that you know this is the case, if you want to capture application events using MarkLogicPowerPointEventSupport.js, you will need to define a function to call out to from each function defined for each event within this file to get the functionality you require.

The nineteen events captured are:

windowSelectionChange()

windowBeforeRightClick()

windowBeforeDoubleClick()

presentationClose()

presentationSave()

presentationOpen()

newPresentation()

presentationNewSlide()

windowActivate()

windowDeactivate()

slideShowBegin()

slideShowNextBuild()

slideShowNextSlide()

slideShowEnd()

presentationPrint()

slideSelectionChanged()

colorSchemeChanged()

presentationBeforeSave()

slideShowNextClick()

You can see how we use windowSelectionChange() and slideSelectionChange() for updating the display for ‘Properties’ on the tag palette as well as the tag list view in the Metadata panel.

**/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!