**MarkLogic Sample Authoring Application for Word®**

**MarkLogic Corporation**

1

\

Release 1.0

May, 2011

Last Revised: 1.0-2, November, 2011

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

1. **Introduction**

The MarkLogic Sample Authoring App for Word® is a sample application, built using the MarkLogic Toolkit for Word 1.2-1. 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 Word as well as sample applications.

*Sample Authoring Application Requirements:*

1. Office 2007
2. MarkLogic Toolkit for Word 1.2-1

*If you already have the MarkLogic Toolkit for Word and the Sample Authoring 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 Sample Authoring Application was created to provide a rich example of the type of application developers can build using the MarkLogic Toolkit for Word®. Please note, this is just one example of the type of application you can build using the Toolkit. Also, the sample application is not the Toolkit, nor does it encompass all of the Toolkit functionality. This application was built as a solution to a specific use-case. The Sample Authoring Application allows authors to:

* enrich Word documents and identify document sections (or components)
* associate custom metadata with those identified components and edit
* search those identified components once saved in MarkLogic Server
* reuse components from documents saved in MarkLogic Server to the document being authored while retaining any associated metadata across documents
* compare documents on the server with the active document using Word’s native merge functionality

The functionality of the Sample Application is detailed online, in blog posts, and in additional guides found at developer.marklogic.com. To use the Sample Application out of the box, some minor configuration is required.

**GUIDE OVERVIEW**

**Section 2.0** of this guide details what files you will need to update to use the Sample Authoring 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 Sample 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 two places for deployment. Let’s quickly look at each area that requires an update.

For our examples below, we’ll assume we are going to install the Sample Authoring Application on the HTTP server at port 8000, the default server MarkLogic makes available on install.

The root directory for applications on the HTTP server on port 8000 is:

<Server-Install-Directory>\Docs

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 Word 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/Word/URL

For our example the URL value could be: <http://marklogic.myserver.com:8000/Author> 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/Author> .

**\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/Author”;

***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/Author/config/";

$config:USER := "user";

$config:PWD := "password"

Update these values to be your HTTP 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 files, the next time you open Word you will find the Sample Authoring Application available and ready for use.

1. **Custom Configuration**

Five configuration files are provided that allow you to customize the look and functionality of the Sample Application without editing any code. They are relatively simple to understand and author.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 Sample Authoring Application. Let’s look at each configuration in detail.

The five configuration files are:

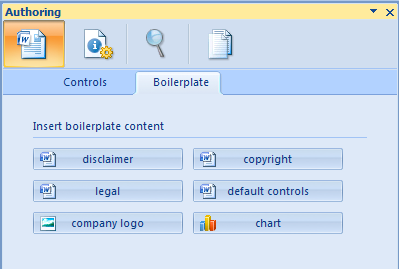
* boilerplate.xml
* controls.xml
* metadata.xml
* search.xml
* compare.xml

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

**boilerplate.xml**

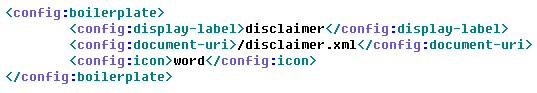
The boilerplate tab within the Sample Authoring App allows you to insert a Word document saved in MarkLogic into the document you are authoring. When you insert a Word document as XML into an active Word document, it is appended to the active document at the current cursor location.

The boilerplate tab in the application looks like:



Clicking a button inserts a Word document from the Server into the active document.

The boilerplate configuration looks like:



config:display-label : the label displayed in the Task Pane on the Boilerplate tab

config:document-uri : the location of the Word document saved as .xml in MarkLogic

config:icon : the image to display on the insert button (must be: word, chart, or image )

For each <config:boilerplate> element in the config file, you will get a button on the Boilerplate tab in the Sample App. The image on the button is noted by the value of <config:icon>. Clicking the button takes the document at the location noted by <config:document-uri> and inserts it at the current cursor position.

You can add as many buttons as you like and/or remove the buttons provided as examples in the sample application. The icons are provided to give the user hints as to the type of document they are inserting. You can think of every piece of content in Word being its own individual document, so you may have Word documents that include boilerplate text, charts, and images for the documents you are authoring.

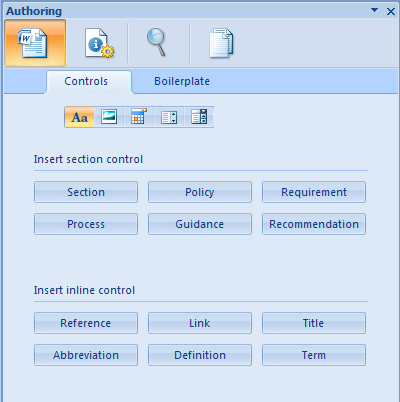
**controls.xml**

Authors can enrich text in Word, as well as add pre-defined pickers and chunks of content into an active document by using Content Controls. We provide a controls palette to enable organizations to predefine the labels for enrichment, as well as define what types of pickers and content chunks are available for insert. Using the configuration controls.xml file allows us to create numerous ways to enrich documents as well as create complex pre-defined forms and lists that we can insert a document at the click of a button.

Content Controls are available in Word under the Developer tab, but this tab is not enabled by default, and there is no control over the types of labels being used. Also, creating complex controls which embed other controls can be tedious. So we’ve aimed to simplify things with our control palette and controls.xml file.

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

The controls palette looks like:



The first thing to notice is that there are five types of controls available.

controls-2.PNG

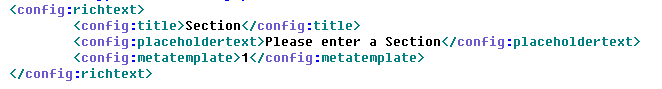
The five types of controls are: Rich Text, Images, Calendar Picker, DropDown List, and ComboBox. Clicking the icon for the associate control type, will display buttons available for that type of control in the task pane. The label of the button is the name of the control as it will be inserted into the document when the button is clicked. Once the document is saved to MarkLogic Server, this will be the value applications can search on once documents enriched using Content Controls are saved back to MarkLogic Server.

The number of buttons available for each type of control can vary depending on your requirements. As each control is different, we’ll need to look at each control individually to understand the configuration.

In controls.xml, the first elements to identify are config:section and config:inline. Only RichText controls can embed other controls, so only RichText controls will be defined under config:section. A RichText control can define embedded elements of inline types, but all other types will be found under config:inline.

**Rich Text Content Controls**

The configuration for a simple rich text button looks like:



For each Rich Text button in the control palette under ‘Insert Section Control’, we’ll find a config:richtext element. This element has the following children:

config:title : the button label and value to be inserted as the ‘title’ of the Content Control

config:placeholdertext: text to provide as prompt if inserted into a document without content

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

A user can either:

1. Select some content in a Word document, and click a richtext button
2. Insert the control into a Word document at the cursor location

In the latter case, the placeholder text will be displayed until a user fills in the control with new content.

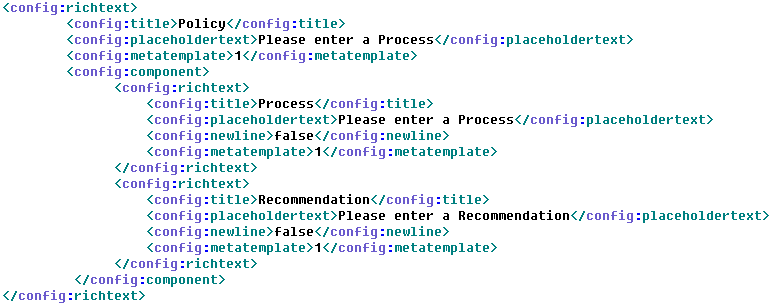
In both cases, the control inserted into the document has a ‘title’ property that is the same as the label of the button, and a custom xml metadata part is added to the .docx package and associated with the control. Which metadata form is added for the control is determined by the metadata template value as it relates to the id we will see in metadata.xml.

So there are two use cases for Content Controls:

1. Enriching already authored docs
2. Inserting controls to request users fill out specific sections with new content

config:richtext has an optional child element config:component, which can be used to note embedded controls.

The configuration for the Policy button in the Sample Application looks like:



Here we see that we can identify other RichText Controls within RichText. This Policy control will insert with an embedded Process Control, with a sibling embedded Recommendation control. A more complex example is provided in controls.xml. A control for Annex has been commented out, but is available for you to use and test.

The only other new element we see is config:newline. The value for this element is true or false. If true, controls will insert with a newline following them, so embedded controls are stacked on top of each other on insert. If set false, controls will append as embedded siblings. The values for config:newline for any sibling controls must be either all true, or all false.

Using the configuration, you can create complex forms from RichText controls that you can insert at the click of a button in the palette.

*NOTE: This is just one way to make forms of controls available. You can always create a document of Content Controls and save it as boilerplate as well. We use the ContentControl events to capture when new controls are added to the document. So inserting from boilerplate, you can still associate and add metadata parts to the controls being added to the document.*

Finally, config:richtext elements that are children of config:inline will insert under ‘Insert inline control’ in the control palette. These controls can embed other controls as well.

**Picture Content Controls**

A picture control allows you to insert a Content Control into the document that allows you insert an image from the client fileystem into the control. The benefit is having the label available for future searches.

The configuration for Picture Content Controls looks like:



For each config:image element, a button will be displayed in the palette under the picture tab.

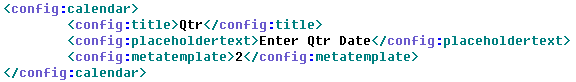
config:title : the label on the button, and the title of control upon insert into the document

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

**Calendar Content Controls**

These buttons will insert calendar pickers into the active document at the current cursor location.

The configuration for Calendar Content Controls looks like:



For each config:calendar element, a button will be displayed in the palette under the calendar tab.

config:title : the label on the button, and the title of control upon insert into the document

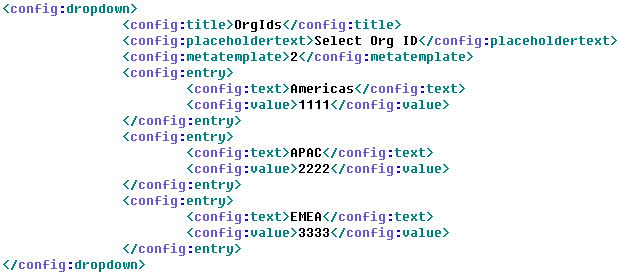
config:placeholdertext : text to provide as prompt as it inserts into a document without content

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

**DropDown Content Controls**

These buttons are found under the DropDown tab in the control palette. Clicking one will insert a dropdown list of selectable entries into the active document. The properties of the control, as well as the items in the list are all configurable.

The configuration for DropDown Content Controls looks like:

****

For each config:dropdown element, a button will be displayed in the palette under the calendar tab.

config:title : the label on the button, and the title of control upon insert into the document

config:placeholdertext : text to provide as prompt as it inserts into a document without content

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

config:entry : a dropdown list selection.

config:text : the text authors will see as the available selection in the dropdown list

config:value : an alternate value for the control. Users would have to open the control properties to view in this Word. Alternately, we have this value available for search and analysis once we save the document to MarkLogic.

You can add as many config:entry entries as you like.

**ComboBox Content Controls**

The configuration for ComboBox Content Controls looks like:



The only difference for the configuration is in the name of the parent element, config:combo. The rest of the details are similar to that of DropDown Content Controls.

The difference is really visible in the behavior of the control in the document. DropDown controls will only allow you to select items from the DropDown selection list, while ComboBox allows users to add arbitrary entries to the selection list.

**metadata.xml**

Whenever a Content Control is added to the document being authored, a custom metadata part will be added to the .docx package and associated with the added control. 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 Sample Application.

The current configuration looks like:

****

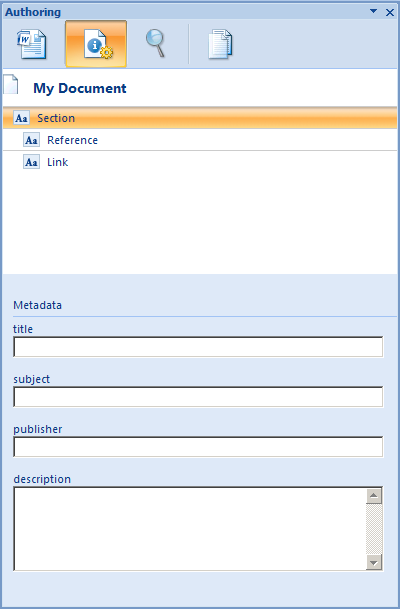
You can add N number of config:template elements. These @id is used to map the config:template child metadata form to the config:metatemplate value found in the associated Content Control definition within controls.xml.

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/>

The metadata form can be found on the metadata panel.

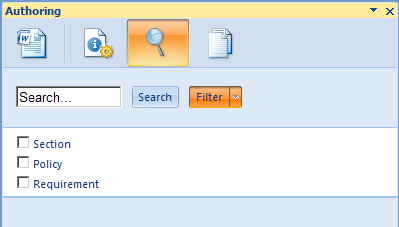


When the user either A) enters the associated control in the active document, or B) selects the section from the control navigation tree, the form is displayed below the control navigation tree.

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 .docx package.

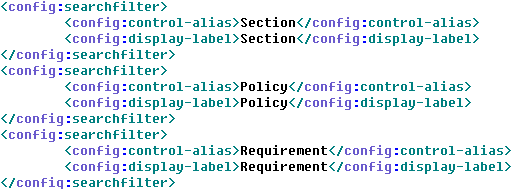
**search.xml**

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

****

Here we see three 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.

The configuration for the above looks like:



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

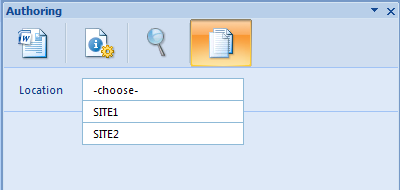
config:control-alias : the value that will be used to search to filter document components

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

In the XML a Content Control manifests itself in document.xml, the main body of content for a Word document in the .docx package, as a structured document tag (w:sdt). The value that labels a w:sdt, is actually the attribute of a child element (w:sdt/w:sdtPr/w:alias/@w:val). Filters apply an element-attribute-value query to the search, which is by default applied to //w:sdt for the search from this pane.

**compare.xml**

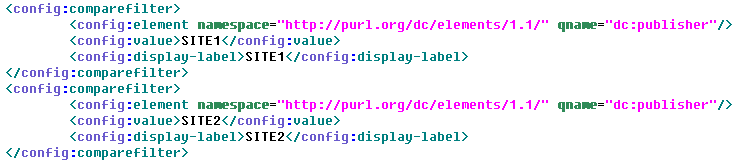
This final configuration file sets the values for the dropdown list on the compare tab.

****

This panel searches for Word documents in MarkLogic based on metadata parts within the .docx package.

Search on the search pane is on text within document components, and brings over associated metadata parts, inserting the component into the document, and the metadata parts into the active .docx package. Search on this pane, the Compare tab, is on the metadata parts for those components. Choosing a value from the dropdown returns search results that allow the user to open the Word document on the Server within Word alongside the document being authored within Word’s native merge functionality.

The configuration file looks like:

****

config:comparefilter : each entry results in a new list item available in the dropdown

config:element : the search is performed for //dc:metadata and setting @namespace and @qname constructs an element-value-query() to apply to the search so you can search on a particular element value within your metadata form.

config:value : the text value to query for the config:element

config:display-label : the label to display to the user in the dropdown list on the compare tab

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

/js/MarkLogicContentControlSupport.js – captures Content Control 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 two 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: MarkLogicContentControlSupport.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 Content Control events using MarkLogicContentControlSupport.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 five events captured are:

contentControlOnEnter()

contentControlOnExit()

contentControlAfterAdd()

contentControlBeforeDelete()

contentControlBeforeContentUpdate()\*

contentControlBeforeStoreUpdate()\*

\*These last two fire when you have a Content Control mapped to values in a customXML part, which is not part of the functionality of the Sample App.

You can see how we use these events in the Sample Application for adding and deleting custom metadata parts to the .docx package, as well as updating the display for ‘Properties’ on the control palette as well as the treeview 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 webBrowser 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!