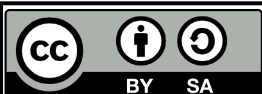
ecmarchitect.com

Alfresco Developer Series
Working with Custom Content Types
2nd Edition
January, 2012

Jeff Potts



This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

Alfresco Developer Series
Working with Custom Content Types
2nd Edition
January, 2012
Jeff Potts

About the Second Edition

This tutorial was originally published in June of 2007. At the time it was written, Alfresco 2.0 was the latest release, although the tutorial continued to be used successfully by many people around the world over the next several years and releases and was eventually used in the Alfresco Developer Guide (Packt, 2008).

Alfresco has changed a lot since 2007. The two biggest changes affecting this tutorial are (1) the addition of Alfresco Share and its subsequent adoption as the preferred web user interface and (2) the Content Management Interoperability Services (CMIS) standard implementation, which can be used to manage content programmatically across the network using a vendor-neutral, language-independent API.

This edition includes updates for Alfresco Share and CMIS. It was written for and tested against Alfresco 4.0, but it should also be useful for those who have not yet moved to 4.0.

I've moved the Alfresco Explorer configuration section to the Appendix and replaced it with how to do the same tasks using Alfresco Share. But I've tested the Alfresco Explorer section against 4.0 so it is still there if you need it.

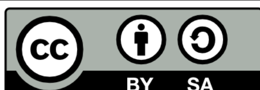
Similarly, I've moved the Alfresco Web Services API to the Appendix, which has also been tested against 4.0, and replaced it with a discussion of how to do the same thing with CMIS using the OpenCMIS Java API from Apache Chemistry. The first edition included a PHP Web Services API example which has been omitted from this edition.

As always, code for everything—the old stuff and the new stuff—is included. See the link at the end of the document if you don't have it.

Alfresco--as a platform and as a company--continues to grow and evolve. We are reaching out to more and more people who are completely new to the platform. If you are new, welcome! I hope you find this edition every bit as helpful as the readers of the first edition did.

Feel free to join the conversation on ecmarchitect.com and in the Alfresco Forums if you have questions.

Thanks for reading -- Jeff



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

About the Second Edition 2
Introduction 5

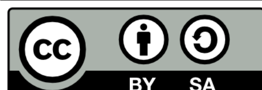
Part 1: Implementing a Custom Content Model 5
Modeling Basics 5
Types 6
Properties 6
Property Types 6
Constraints 7
Associations 7
Aspects 8
Custom Behavior 8
Content Modeling Best Practices 9
Out-of-the-Box Models 10
Creating a Custom Content Model 12
Implementing and Deploying the Model 14

Part 2: Configuring Alfresco Share 18
Configuring the Custom Model in Alfresco Share 18
Adding Custom Types and Aspects to Lists 19
Configuring Forms for Custom Properties 22
Configuring Advanced Search in Alfresco Share 25
Localizing Strings for Custom Content Models 27
Share Configuration Summary 28

Part 3: Working with Content Programmatically 28
Setup 30
Creating Content with OpenCMIS 30
Creating Associations with OpenCMIS 32
Searching for Content with OpenCMIS 33
Queries on Aspect-based Properties 35
Queries Across Multiple Aspects 35
Queries Using Dates 36
Deleting Content with OpenCMIS 38

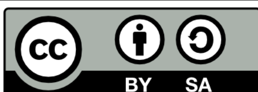
Conclusion 39
Where to Find More Information 39
About the Author 40

Appendix 41
Configuring the Custom Content Model in Alfresco Explorer 41
Property Sheet 42
Create Content/Add Content 43
"Is sub-type" Criteria, Specialize Type Action, Add Aspect Action 43
Advanced Search 44



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

String Externalization *46*
Test the Web Client User Interface Customizations *46*
Using the Web Services API 46
Creating Content with Java Web Services *46*
Creating Associations with Java Web Services *49*
Searching for Content with Java Web Services *50*
Deleting Content with Java Web Services *54*



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Introduction

Alfresco is a flexible platform for developing content management applications. The first step in the process of designing a custom content management application is creating the content model.

The content model Alfresco provides out-of-the-box is fairly comprehensive. In fact, for basic document management needs, you could probably get by with the out-of-the-box model. Of course, you'd be missing out on a lot of the power and functionality that having a model customized for your business needs provides.

Part 1 of this document discusses how to create your own custom content model, but you won't want to stop there. What good would a custom content model be if you did nothing exciting with the content? After an example content model is in place, Part 2 shows how to configure Alfresco Share to expose the custom model in the user interface, then Part 3 shows how to use OpenCMIS, a standard Java API, to create, search for, and delete content. You can safely swap the order of Parts 2 and 3 according to your interest.

You should already be familiar with general document management and Alfresco web client concepts. If you want to follow along with Part 3, you should also know how to write basic Java code. See "Where to find more information" at the end of this document for a link to the code samples that accompany this article.

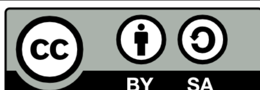
Part 1: Implementing a Custom Content Model

Out-of-the-box, Alfresco gives you folders and content and a few other content types. But you'll probably want to create your own business-specific types. This section discusses how that works.

Modeling Basics

A content model describes the data being stored in the repository. The content model is critical—without it, Alfresco would be little more than a file system. Here is a list of key information the content model provides Alfresco:

- Fundamental data types and how those data types should be persisted to the database. For example, without a content model, Alfresco wouldn't know the difference between a String and a Date.
- Higher order data types like "content" and "folder" as well as custom content types like "Standard Operating Procedure" or "Contract".
- Out-of-the-box aspects like "auditable" and "classifiable" as well as custom aspects like "rate-able" or "comment-able".



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

- Properties (or metadata) specific to each content type.
- Constraints placed on properties (such as property values that must match a certain pattern or property values that must come from a specific list of possible values).
- How to index content for searching.
- Relationships between content types.

Alfresco content models are built using a small set of building blocks: Types, Properties, Property types, Constraints, Associations, and Aspects.

Types

Types are like types or classes in the object-oriented world. They can be used to model business objects, they have properties, and they can inherit from a parent type. “Content”, “Person”, and “Folder” are three important types defined out-of-the-box. Custom types are limited only by your imagination and business requirements. Examples include things like “Expense Report”, “Medical Record”, “Movie”, “Song”, and “Comment”.

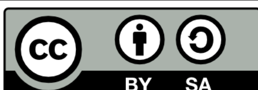
Note that types, properties, constraints, associations, and aspects have names. Names are made unique across the repository by using a namespace specific to the model. The namespace has an abbreviation. Rather than use “Example” or “Foo”, in this tutorial this document assumes Alfresco is being implemented for a fictitious company called SomeCo. So, for example, SomeCo might define a custom model which declares a namespace with the URI of “http://www.someco.com/model/content/1.0” and a prefix of “sc”. Any type defined as part of that model would have a name prefixed with “sc:”. You'll see how models are actually defined using XML shortly, but I wanted to introduce the concept of namespaces and prefixes so you would know what they are when you see them. Using namespaces in this way helps prevent name collisions when content models are shared across repositories. In your project it is definitely important that you use your own namespace.

Properties

Properties are pieces of metadata associated with a particular type. For example, the properties of an Expense Report might include things like “Employee Name”, “Date submitted”, “Project”, “Client”, “Expense Report Number”, “Total amount”, and “Currency”. The Expense Report might also include a “content” property to hold the actual expense report file (maybe it is a PDF or an Excel spreadsheet, for example).

Property Types

Property types (or data types) describe the fundamental types of data the repository will use to store



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

properties. Examples include things like strings, dates, floats, and booleans. Because these data types literally are fundamental, they are pretty much the same for everyone so they are defined for us out-of-the-box. (If you wanted to change the fact that the Alfresco data-type "text" maps to your own custom class rather than java.lang.String, you could, but let's not get ahead of ourselves).

Constraints

Constraints can optionally be used to restrict the value that Alfresco will store in a property. There are four types of constraints available: REGEX, LIST, MINMAX, and LENGTH. REGEX is used to make sure that a property value matches a regular expression pattern. LIST is used to define a list of possible values for a property. MINMAX provides a numeric range for a property value. LENGTH sets a restriction on the length of a string.

Constraints can be defined once and reused across a model. For example, out-of-the-box, Alfresco makes available a constraint named "cm:filename" that defines a regular expression constraint for file names. If a property in a custom type needs to restrict values to those matching the filename pattern, the custom model doesn't have to define the constraint again, it simply refers to the "cm:filename" constraint.

Associations

Associations define relationships between types. Without associations, models would be full of types with properties that store "pointers" to other pieces of content. Going back to the expense report example, each expense report might be stored as an individual object. In addition to an Expense Report type there could also be an Expense type. Associations tell Alfresco about the relationship between an Expense Report and one or more Expenses.

Associations come in two flavors: Peer Associations and Child Associations. (Note that Alfresco refers to Peer Associations simply as "Associations" but I think that's confusing so I'll refer to them with the "Peer" distinction). Peer Associations are just that—they define a relationship between two objects but neither is subordinate to the other. Child Associations, on the other hand, are used when the target of the association (or child) should not exist when the source (or parent) goes away. This works like a cascaded delete in a relational database: Delete the parent and the child goes away.

An out-of-the-box association that's easy to relate to is "cm:contains". The "cm:contains" association defines a Child Association between folders ("cm:folder") and all other objects (instances of "sys:base" or its child types). So, for example, a folder named "Human Resources" (an instance of "cm:folder") would have a "cm:contains" association between itself and all of its immediate children. The children could be instances of custom types like Resume, Policy, or Performance Review.

Another example might be a "Whitepaper" and its "Related Documents". Suppose that a company publishes whitepapers on their web site. The whitepaper might be related to other documents such as

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

product marketing materials or other research. If the relationship between the whitepaper and its related documents is formalized it can be shown in the user interface. To implement this, as part of the Whitepaper content type, you'd define a Peer Association. You could use “sys:base” as the target type to allow any piece of content in the repository to be associated with a Whitepaper or you could restrict the association to a specific type like “cm:content” or “sc:whitepaper”.

Aspects

Before discussing *Aspects*, let's first consider how inheritance works and the implications on the content model. Suppose Alfresco will be used to manage content to be displayed in a portal (quite a common requirement, by the way). Suppose further that only a subset of the content in the repository is content that should be shown in the portal. And, when content is to be displayed in the portal, there are some additional pieces of metadata that need to be captured. A simple example might be a requirement to show the date and time a piece of content was approved.

Using the content modeling concepts discussed so far, thre are only two options. The first option is to define a root content type with the “publish date” property. All subsequent content types would inherit from this root type thus making the publish date available everywhere. The second option is to individually define the publish date property only in the content types that are going to be published to the portal.

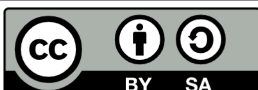
Neither of these are great options. In the first option, there would be a property in each-and-every piece of content in the repository that may or may not ultimately be used which can lead to performance and maintenance problems. The second option isn't much better for a few reasons. First, it assumes the content types to be published in the portal are known ahead of time. Second, it opens up the possibility that the same type of metadata might get defined differently across content types. Last, it doesn't provide an easy way to encapsulate behavior or business logic that might be tied to the publish date.

As you have probably figured out by now, there is a third option that addresses these issues: Aspects. Aspects “cross-cut” the content model with properties and associations by attaching them to content types (or even specific instances of content) when and where they are needed.

Going back to the portal example, a “Portal Displayable” aspect could be defined with a publish date property. The aspect would then be added to any piece of content, regardless of type, that needed to be displayed in the portal.

Custom Behavior

You may find that your custom aspect or custom type needs to have behavior or business logic associated with it. For example, every time an Expense Report is checked in you want to recalculate the total by iterating through the associated Expenses. One option would be to incorporate this logic into rules or actions in the Alfresco web client or your custom web application. But some behavior is so



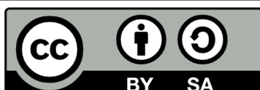
This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

fundamental to the aspect or type that it should really be “bound” to the aspect or type and invoked any time Alfresco works with those objects. If you are curious how this works, read the Custom Behaviors tutorial on ecmarchitect.com. For now, just know that associating business logic with your custom aspects and types (or overriding out-of-the-box behavior) is possible.

Content Modeling Best Practices

Now that you know the building blocks of a content model, it makes sense to consider some best practices. Here are the top ten:

1. *Don't change Alfresco's out-of-the-box content model.* If you can possibly avoid it, do not change Alfresco's out-of-the-box content model. Instead, extend it with your own custom content model. If requirements call for several different types of content to be stored in the repository, create a content type for each one that extends from cm:content or from an enterprise-wide root content type.
2. *Consider implementing an enterprise-wide root type.* Although the need for a common ancestor type is lessened through the use of aspects, it still might be a good idea to define an enterprise-wide root content type from which all other content types in the repository inherit if for no other reason than it gives content managers a “catch-all” type to use when no other type will do.
3. *Be conservative early on by adding only what you know you need*. A corollary to that is *prepare yourself to blow away the repository multiple times until the content model stabilizes*. Once you get content in the repository that implements the types in your model, making model additions is easy, but subtractions aren't. Alfresco will complain about “integrity errors” and may make content inaccessible when the content's type or properties don't match the content model definition. When this happens to you (and it will happen) your options are to either (1) leave the old model in place, (2) attempt to export the content, modify the ACP XML file, and re-import, or (3) drop the Alfresco tables, clear the data directory, and start fresh. As long as everyone on the team is aware of this, option three is not a big deal in development, but make sure expectations are set appropriately and have a plan for handling model changes once you get to production. This might be an area where Alfresco will improve in future releases, but for now it is something you have to watch out for.
4. *Avoid unnecessary content model depth.* I am not aware of any Alfresco Content Modeling Commandments that say, “Thou shall not exceed X levels of depth in thine content model lest thou suffer the wrath of poor performance” but it seems logical that degradation would occur at some point. If your model has several levels of depth beyond cm:content, you should at least do a proof-of-concept with a realistic amount of data, software, and hardware to make sure you aren't creating a problem for yourself that might be very difficult to reverse down the road.
5. *Take advantage of aspects.* In addition to the potential performance and overhead savings through the use of aspects, aspects promote reuse across the model, the business logic, and the



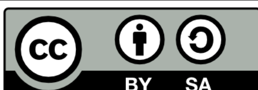
This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

presentation layer. When working on your model you find that two or more content types have properties in common, ask yourself if those properties are being used to describe some higher-level characteristic common across the types that might better be modeled as an aspect.

6. *It may make sense to define types that have no properties or associations.* You may find yourself defining a type that gets everything it needs through either inheritance from a parent type or from an aspect (or both). In those cases you might ask yourself if the "empty" type is really necessary. In my opinion, it should at least be considered. It might be worth it just to distinguish the content from other types of content for search purposes, for example. Or, while you might not have any specialized properties or associations for the content type you could have specialized behavior that's only applicable to instances of the content type.
7. *Remember that folders are types too.* Like everything else in the model, folders are types which means they can be extended. Content that "contains" other content is common. In the earlier expense report example, one way to keep track of the expenses associated with an expense report would be to model the expense report as a sub-type of cm:folder.
8. *Don't be afraid to have more than one content model XML file*. You'll see how content models are defined shortly, but when it is time to implement your model, keep this in mind: It might make sense to segment your models into multiple namespaces and multiple XML files. Names should be descriptive. Don't deploy a model file called "customModel.xml" or "myModel.xml".
9. *Implement a Java class that corresponds to each custom content model you define*. Within each content model Java class, define constants that correspond to model namespaces, type names, property names, aspect names, etc. You'll find yourself referring to the "Qname" of types, properties, and aspects quite often so it helps to have constants defined in an intuitive way.
10. *Use the source!* The out-of-the-box content model is a great example of what's possible. The forumModel and recordsModel have some particularly useful examples. In the next section I'll tell you where the model files live and what's in each so you'll know where to look later when you say to yourself, "Surely, the folks at Alfresco have done this before".

Out-of-the-Box Models

The Alfresco source code is an indispensable reference tool which you should always have at the ready, along with the documentation, wiki, forums, and Jira. With that said, if you are following along with this article but have not yet downloaded the source, you are in luck. The out-of-the-box content model files are written in XML and get deployed with the web client. They can be found in the alfresco.war file in /WEB-INF/classes/alfresco/model. The table below describes several of the model files that can be found in the directory.

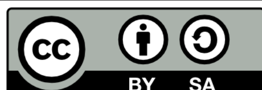


This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

File	Namespaces	Prefix	Imports	Description
dictionaryModel.xml	model/dictionary/1.0	d	None	Fundamental data types used in all other models.
systemModel.xml	model/system/1.0	sys	d	System-level objects like base, store root, and reference.
	system/registry/1.0	reg		
	system/modules/1.0	module		
contentModel.xml	model/content/1.0	cm	d sys	Types and aspects extended most often by your models like Content, Folder, Versionable, and Auditable.
	model/rendition/1.0	rn		
	model/exif/1.0	exif		
	model/audio/1.0	audio		
	model/webdav/1.0	webdav		
bpmModel.xml	model/bpm/1.0	bpm	d sys cm usr	Advanced workflow types. Extend these when writing your own custom advanced workflows.
forumModel.xml	model/forum/1.0	fm	d cm	Types and aspects related to adding discussion threads to objects.

In the interest of brevity, I've left off about 25 other model files. Depending on what you are trying to do with your model, or just to see further examples, you might want to take a look at those at some point.

In addition to the model files the modelSchema.xsd file can be a good reference. As the name suggests, it defines the XML vocabulary Alfresco content model XML files must adhere to.



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Creating a Custom Content Model

Time for a detailed example. As mentioned earlier, suppose Alfresco is being implemented for a fictional company called "SomeCo". Pretend that SomeCo is a commercial open source company behind the ever-popular open source project, "SomeSoftware". SomeCo has decided to re-vamp its web presence by adding new types of content and community functionality to their web site. For this example, let's focus on the white papers SomeCo wants to make available.

SomeCo has selected Alfresco as their Enterprise Content Management solution. In addition to managing the content on the new site, SomeCo wants to use Alfresco to manage all of its rich content. So everything will live in the Alfresco repository and some subset of the company's content will be served up to the external portal.

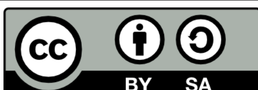
The first step is to consider the types and properties needed. There are some pieces of metadata SomeCo wants to track about all content, regardless of whether or not it will be shown on the web site. All documents will have an audience property that identifies who will be most interested in the content. Documents related to SomeCo's software will have properties identifying the Software Product and Software Version.

Content that needs to be shown on the web site needs to have a flag that indicates the content is "active" and a date when the content was set to active.

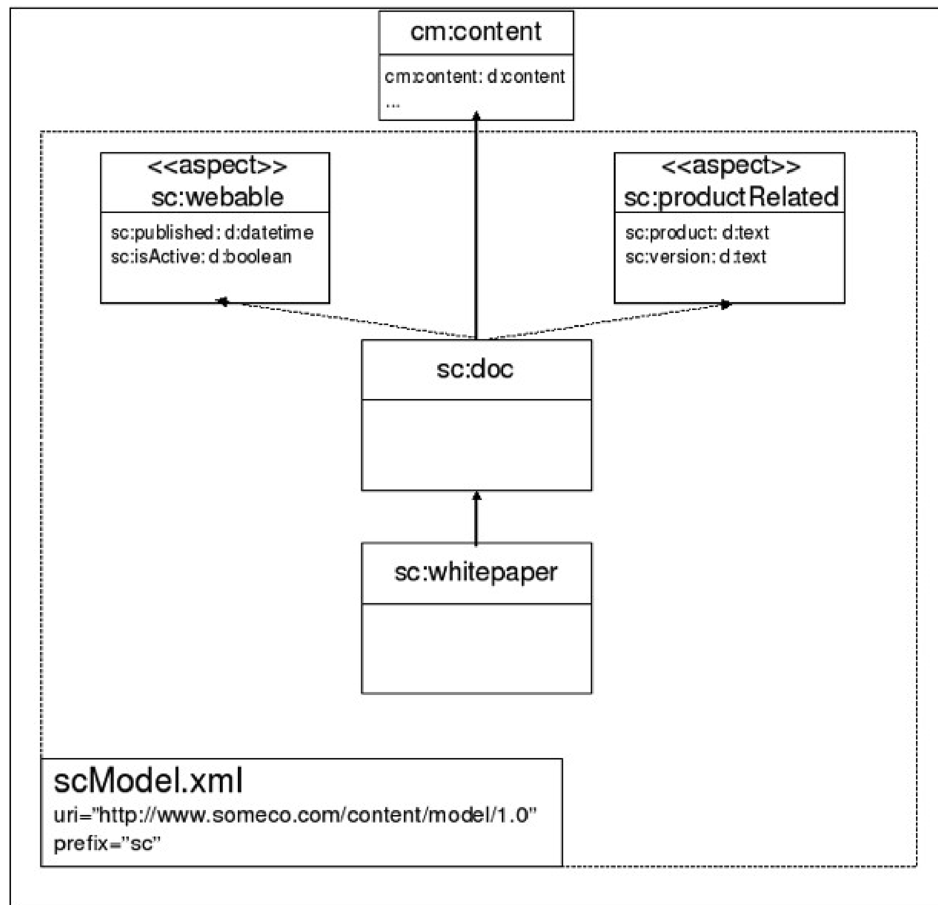
Now let's think about associations. For some documents, SomeCo would like to explicitly define one or more "related documents". On the web site, SomeCo might choose to show a list of related documents at the bottom of a white paper, for example.

Taking these requirements into consideration, the team comes up with the content model depicted in Drawing 1: SomeCo's initial content model. As the drawing shows, there is a common root type called sc:doc with one child, sc:whitepaper. Neither type currently has any properties of their own.

It's not shown on the model diagram, but there is a Peer Association defined as part of sc:doc to keep track of related documents. The target class of the association will be sc:doc because the requirement is to be able to associate any instance of sc:doc or its children with one or more instances of sc:doc or its children.



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License



Drawing 1: SomeCo's initial content model

In addition, there are two aspects. One, the web-able aspect, is used for content that is to be shown on the web site. It contains the active flag and active date. The product-related aspect is used only for content that relates to a SomeCo product. It captures the specific product name the content is related to as well as the product version.

It should be easy to see how the model might be extended over time. The requirements mentioned community features being needed at some point. A rateable aspect could be added along with a rating type. Comments could work the same way, or comments could leverage the existing forums content model.

As new content types are identified they will be added under sc:doc.

Using the aspect to determine whether or not to show the content on the portal is handy, particularly in light of the SomeCo decision to use Alfresco for all of its content management needs. The repository

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

will contain content that may or may not be on the portal. Portal content will be easily-distinguishable from non-portal content by the presence of the “webable” aspect.

Implementing and Deploying the Model

Before starting, here are a couple of notes about my setup:

- Mac OS X Lion 10.7.2
- MySQL 5.1.60 (Macports)
- Tomcat 6.0.32
- Java 1.6.0_29
- Alfresco 4.0.c Community, WAR-only distribution

Other operating systems, databases, application servers, and Alfresco versions should work as well. Content modeling hasn’t changed much since the early days of the product.

Here are the steps to follow when configuring a custom content model:

1. Create an extension directory
2. Create a custom model context file
3. Create a model file
4. Restart Tomcat

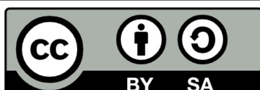
The first step is to **create an extension directory** if you do not already have one. An extension directory keeps your customizations separate from Alfresco's code. This makes it easier to upgrade Alfresco later on.

The extension directory can be anywhere on the web application's classpath. I recommend using two. One should be for server-specific settings such as the repository properties (specifies the database username, password, and connection string) and LDAP authentication context. For Tomcat installations, this extension directory would be $TOMCAT_HOME/shared/classes/alfresco/extension.

The other extension directory is for Alfresco web client customizations and the content model. For that I use the extension directory that is included in the Alfresco web application which is under $TOMCAT_HOME/webapps/alfresco/WEB-INF/classes/alfresco/extension.

It's fine if you want to keep it simple for now and just use one extension directory.

The second step is to **create a custom model context file**. A context file is a file that contains one or more Spring bean configurations. There are several context files used to configure Alfresco Spring beans. Depending on the Alfresco distribution you downloaded you may have a set of sample context files in your extension directory.



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Alfresco loads any file on the classpath that ends with "context.xml". To create custom Spring configuration files, create a file that ends with that suffix, and in it, override the appropriate bean. In this case, the one that refers to the content model. How do you know which bean to extend? Recall from earlier that there is an out-of-the-box content model file called contentModel.xml. If you have the source handy, go to the repository project and do a recursive grep for the string "contentModel.xml". You'll find that it is referenced in a file called config/alfresco/core-services-context.xml as shown below.

```
    <bean id="dictionaryBootstrap" parent="dictionaryModelBootstrap" depends-
on="resourceBundles">
        <property name="models">
            <list>
                <!-- System models  -->
                <value>alfresco/model/dictionaryModel.xml</value>
                <value>alfresco/model/systemModel.xml</value>
                <value>org/alfresco/repo/security/authentication/userModel.xml</va
lue>
                <!-- Content models -->
                <value>alfresco/model/contentModel.xml</value>
                <value>alfresco/model/bpmModel.xml</value>
    ...
```

Listing 1: core-services-context.xml

The custom content model needs to be registered with the dictionary as well so to do that, just extend this bean in a custom context file and add the custom content model to the list. By convention, the context file should end with "model-context.xml", so for SomeCo, the context file is in the extension directory ($TOMCAT_HOME/webapps/alfresco/WEB-INF/classes/alfresco/extension) and is called someco-model-context.xml. It contains the following:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE beans PUBLIC '-//SPRING//DTD BEAN//EN'
'http://www.springframework.org/dtd/spring-beans.dtd'>

<beans>
    <!-- Registration of new models -->
    <bean id="someco.dictionaryBootstrap" parent="dictionaryModelBootstrap"
depends-on="dictionaryBootstrap">
        <property name="models">
            <list>
                <value>alfresco/extension/model/scModel.xml</value>
            </list>
        </property>
    </bean>
</beans>
```

Listing 2: someco-model-context.xml

The next step is to **create a model file** that implements the content model defined earlier. Over time

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

there may be a lot of files in the extension directory, so it is a good idea to create a directory called "model" in the extension directory. Custom models live in that directory as XML. The name matches what was specified in the someco-model-context.xml file. In this example, the file should be named scModel.xml.

Referring back to Drawing 1: SomeCo's initial content model, there are two aspects, each with two properties, and two content types. You can use the out-of-the-box contentModel.xml discussed earlier as a reference to build your own custom model. In this example, scModel.xml looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Definition of new Model -->
<model name="sc:somecomodel" xmlns="http://www.alfresco.org/model/dictionary/1.0">

    <!-- Optional meta-data about the model -->
    <description>Someco Model</description>
    <author>Jeff Potts</author>
    <version>1.0</version>

    <!-- Imports are required to allow references to definitions in other models
-->
    <imports>
        <!-- Import Alfresco Dictionary Definitions -->
        <import uri="http://www.alfresco.org/model/dictionary/1.0" prefix="d" />
        <!-- Import Alfresco Content Domain Model Definitions -->
        <import uri="http://www.alfresco.org/model/content/1.0" prefix="cm" />
    </imports>

    <!-- Introduction of new namespaces defined by this model -->
    <namespaces>
        <namespace uri="http://www.someco.com/model/content/1.0" prefix="sc" />
    </namespaces>

    <types>
        <!-- Enterprise-wide generic document type -->
        <type name="sc:doc">
            <title>Someco Document</title>
            <parent>cm:content</parent>
            <associations>
                <association name="sc:relatedDocuments">
                    <title>Related Documents</title>
                    <source>
                        <mandatory>false</mandatory>
                        <many>true</many>
                    </source>
                    <target>
                        <class>sc:doc</class>
                        <mandatory>false</mandatory>
                        <many>true</many>
```

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
                    </target>
                </association>
            </associations>
            <mandatory-aspects>
                <aspect>cm:generalclassifiable</aspect>
            </mandatory-aspects>
        </type>

        <type name="sc:whitepaper">
            <title>Someco Whitepaper</title>
            <parent>sc:doc</parent>
        </type>
    </types>

    <aspects>
        <aspect name="sc:webable">
            <title>Someco Webable</title>
            <properties>
                <property name="sc:published">
                    <type>d:date</type>
                </property>
                <property name="sc:isActive">
                    <type>d:boolean</type>
                    <default>false</default>
                </property>
            </properties>
        </aspect>

        <aspect name="sc:productRelated">
            <title>Someco Product Metadata</title>
            <properties>
                <property name="sc:product">
                    <type>d:text</type>
                    <mandatory>true</mandatory>
                </property>
                <property name="sc:version">
                    <type>d:text</type>
                    <mandatory>true</mandatory>
                </property>
            </properties>
        </aspect>
    </aspects>
</model>
```

Listing 3: scModel.xml

Here's an important note about the content model schema that may save you some time: Order matters. For example, if you move the “associations” element after “mandatory-aspects” Alfresco won't be able to parse your model. Refer to the modelSchema.xsd referenced earlier to determine the expected order.

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

The final step is to **restart Tomcat** so that Alfresco will load our custom model. Watch the log during the restart. You should see no errors related to loading the custom model. If there is a problem, the message usually looks something like, “Could not import bootstrap model”.

Part 2: Configuring Alfresco Share

Now that the model is defined, you could begin using it right away by writing code against one of Alfresco's API's that creates instances of your custom types, adds aspects, etc. In practice it is usually a good idea to do just that to make sure the model behaves like you expect. In fact, if you just can’t wait to see some code to create, query, and update content, skip to Part 3. For everyone else, let's talk about how to work with a custom model in the Alfresco web clients.

Configuring the Custom Model in Alfresco Share

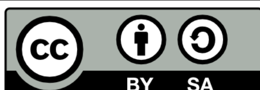
There are two Alfresco web clients to consider. One is called Alfresco Explorer. It has been around since the early days of the product and is based on JavaServer Faces (JSF). The other is called Alfresco Share. Share was introduced in Alfresco 3.0 and is based on Spring Surf and the Yahoo UI (YUI) library. Nowadays, most people starting out with Alfresco will use Alfresco Share. At this point, almost all functionality available in Explorer is available in Share, so unless you just love JSF or you have existing customizations in Explorer that you won’t be migrating to Share any time soon, you should start with Alfresco Share.

Of course, it is possible to configure both Alfresco Explorer and Alfresco Share to leverage your custom content model. If you’d like to do that as a learning exercise, go for it. The instructions for customizing Alfresco Explorer are included in the Appendix.

Let’s continue by configuring Alfresco Share to work with our custom content model.

First, think about the Alfresco Share client and all of the places the content model customizations need to show up:

- **“Is sub-type” criteria**. When a user configures a rule on a space and uses content types as a criteria, the custom types should be a choice in the list of possible content types.
- **“Has aspect” criteria**. When a user configures a rule on a space and uses the presence of an aspect as a criteria, the custom aspects should be a choice in the list of possible aspects.
- **Change type action**. When a user runs the “specialize type” action, either as part of a rule configuration or through the “Change Type” UI action, the custom types should be available.
- **Add aspect**. When a user runs the “add aspect” action, either as part of a rule configuration or through the “Manage Aspects” UI action, the custom aspects should be available.
- **Document Details**. When a user looks at the document details page for a piece of content



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

stored as one of the custom types or with one of the custom aspects attached, the properties section should show the custom properties.

- **Edit Properties**. When a user edits the properties of a piece of content, either with the "full" form or the abbreviated form, the appropriate properties should be shown.
- **Advanced search**. When a user runs an advanced search, they should be able to restrict search results to instances of our custom types and/or content with specific values for the properties of our custom types.

Let's look at each of the areas mentioned above in order to understand how the custom content model can be exposed in the user interface.

Before doing so, a brief word about setup. The Alfresco repository and the Alfresco Explorer web client run in the same "alfresco" web application (WAR). The Alfresco Share web client runs in its own "share" WAR. You're about to add some configuration to the Alfresco Share WAR, and some of that configuration requires a restart. Because Share restarts faster than the repository, and because most production environments have them separated anyway, I like to use two Tomcats on my local machine for developing and testing Share customizations. My Tomcat server running on 8080 is for the Alfresco repository, Alfresco Explorer, and out-of-the-box Alfresco Share. My Tomcat server running on 8081 is for customized Alfresco Share. You're free to do whatever feels right. Just know that, for this section, when I say, "Restart Alfresco Share Tomcat," you'll know why I'm making the distinction.

If you are used to working with the Alfresco Explorer user interface configuration, this will feel eerily familiar. Alfresco Share has a similar configuration file called share-config-custom.xml. The share-config-custom.xml file is a proprietary file composed of numerous "config" elements. Each config element has an "evaluator" and a "condition". Extending the web-client-config.xml file is a matter of knowing which evaluator/condition to use.

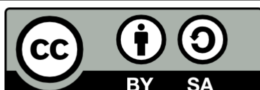
And, similar to the Alfresco repository WAR, Share has an extension directory that keeps your configuration separate from the rest of the web application. In Share, the extension directory is called "web-extension".

Adding Custom Types and Aspects to Lists

The first four items in our list have to do with telling the Share user interface about custom types and aspects. A single set of configuration will take care of all four items.

If you are following along and have not done so already, create a new file called share-config-custom.xml in $TOMCAT_HOME/webapps/share/WEB-INF/classes/alfresco/web-extension with the following content:

```
<alfresco-config>
    <!-- Document Library config section -->
    <config evaluator="string-compare" condition="DocumentLibrary">
```



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
    </config>
</alfresco-config>
```

Listing 4: The start of a custom Share form config, share-config-custom.xml

Now add an “aspects” element as a child of the “config” element to identify the aspects that are visible, addable, and removeable:

```
        <aspects>
            <!-- Aspects that a user can see -->
            <visible>
                <aspect name="sc:webable" />
                <aspect name="sc:productRelated" />
            </visible>

            <!-- Aspects that a user can add. Same as "visible" if left empty -->
            <addable>
            </addable>

            <!-- Aspects that a user can remove. Same as "visible" if left empty
-->
            <removeable>
            </removeable>
        </aspects>
```

Listing 5: Configuring Share for custom aspects

As the comments suggest, the addable and removeable elements can remain empty if the list is the same.

That takes care of aspects. Let’s configure the types. Add a “types” element as a sibling to the “aspects” element you just added with the following content:

```
         <types>
             <type name="cm:content">
                <subtype name="sc:doc" />
                <subtype name="sc:whitepaper" />
             </type>
             <type name="sc:doc">
                <subtype name="sc:whitepaper" />
             </type>
        </types>
```

Listing 6: Configuring Share for custom types

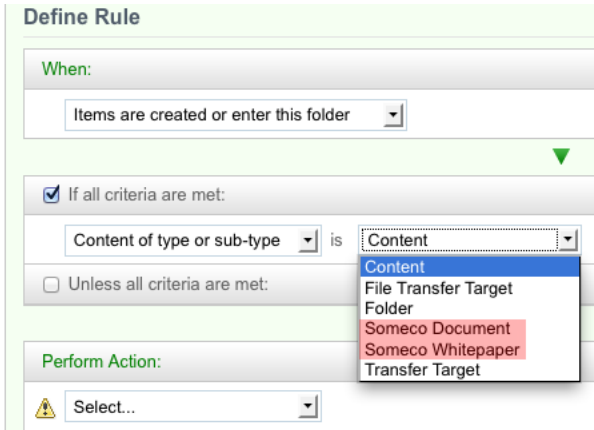
Notice how “sc:whitepaper” appears in the list twice—once as a subtype of cm:content and once as a subtype of sc:doc. This allows a user to specialize directly to the sc:whitepaper type regardless of whether the original type is cm:content or sc:doc.

Now restart Alfresco Share Tomcat and let’s see what’s different.

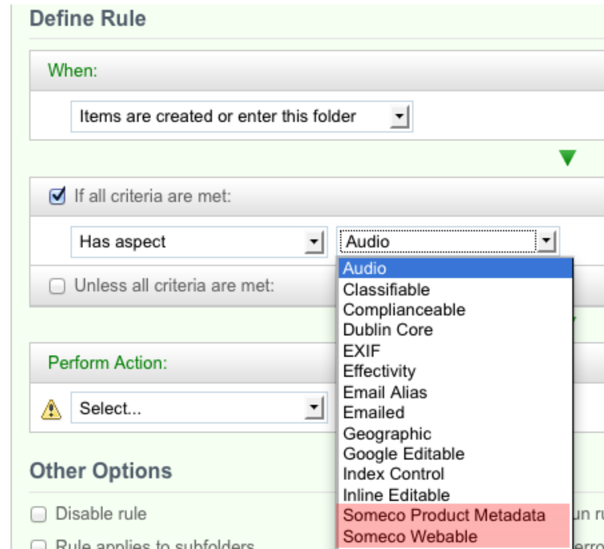
What’s different is that the “is sub-type” dropdown in the rule configuration panel now has our custom

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

types and the "has aspect" dropdown now has our custom aspects.

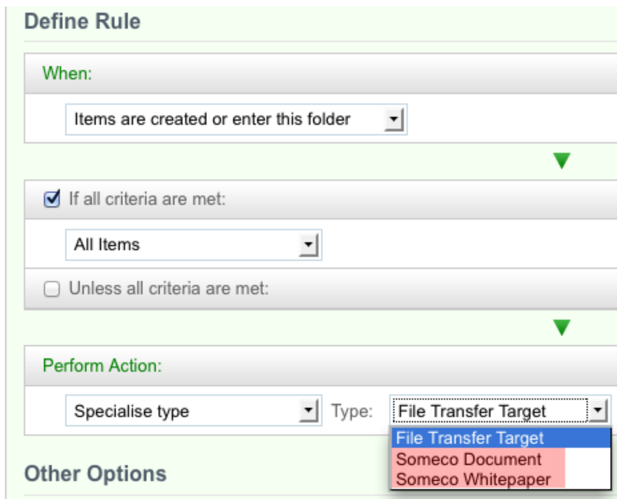


Drawing 2: Rule configuration with custom types

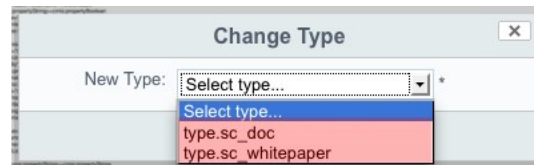


Drawing 3: Rule configuration with custom aspects

The "change type" dropdown, in both the rule configuration and UI action, lists our custom types:



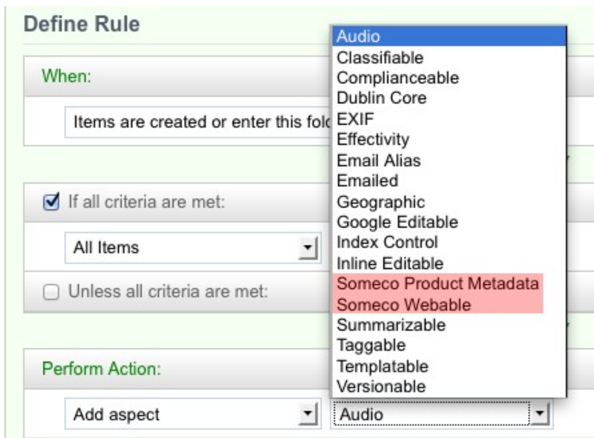
Drawing 4: Specialize type action with in rule config



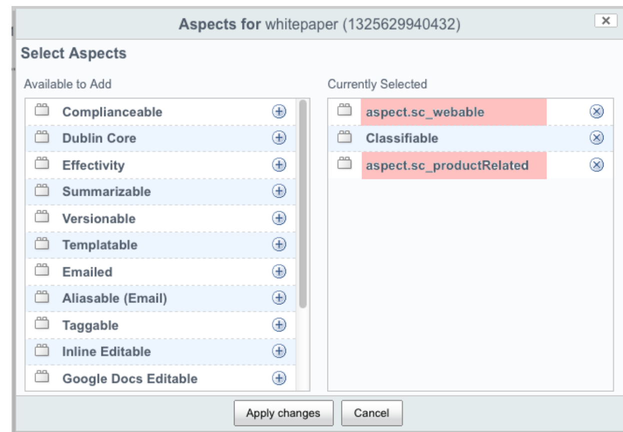
Drawing 5: Manage types UI action with custom types

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

And the “add aspect” list, in both the rule configuration and UI action, lists our custom aspects:



Drawing 6: Add aspect action with custom actions



Drawing 7: Manage aspects UI action with custom aspects

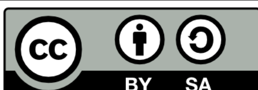
You’ll notice that in the rule configuration the types and aspects are shown with their localized names while in the dialogs the strings are shown as not yet localized (their ID is displayed instead of their localized string). That’s because the strings for Share haven’t yet been localized and some services, like the rule configuration, will pull those strings from the content model XML. If it is annoying you, jump to the localization section and come back.

Configuring Forms for Custom Properties

You may have already noticed that if you look at the details page for an instance of a custom type, sc:whitepaper, for example, you’ll see that Alfresco Share has already figured out that there are some custom properties on the content object. But if you compare the properties for an instance of sc:whitepaper to that of an instance of cm:content there are other properties being shown for sc:whitepaper in addition to our custom properties.

Content properties can be edited using either the default form, which is what you get when you click “Edit Properties” from the document details page, or the edit metadata popup dialog, which is what you get when you click “Edit Properties” from the document library page. If you compare the edit metadata form for sc:whitepaper versus cm:content, just like the default form, you’ll see that there is a big discrepency.

What’s going on here is that Share is looking for a form configuration for sc:whitepaper. Finding none,



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

it is just listing all of the properties of the object. Clearly something prettier is needed for both the default form and the edit metadata form, so let’s fix that.

Configuring the form service for a custom type

Alfresco Share uses the Form Service to decide which properties to show for a given form, how to lay out forms, and the control to use for each property. Let’s take a look at the out-of-the-box form configuration for cm:content. It resides in $TOMCAT_HOME/webapps/share/WEB-INF/classes/alfresco/share-form-config.xml.

If you open that file and search for ‘condition=”cm:content”’ you’ll find two config elements that contain a total of six form configurations. One config element, identified by the “node-type” evaluator, is for forms dealing with existing nodes. The config element with the “model-type” evaluator is for forms used to create new nodes.

Let’s focus on existing nodes for now. The three forms in that config element are:

- Default (form element with no id attribute). This is displayed when you open the document details.
- A form called “doclib-simple-metadata”. This is used when you edit properties from the document library.
- A form called “doclib-inline-edit”. This is used when you click the “Inline Edit” UI action.

Suppose whitepapers need to have the same metadata displayed as instances of cm:content. To do that, copy the entire config element and all of its children into share-config-custom.xml as a sibling of the Document Library config element you added to that file earlier. Then, change the condition from “cm:content” to “sc:whitepaper”.

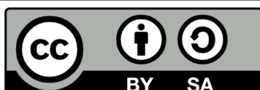
If you save the file and restart Tomcat, you’ll see that the property sets match for sc:whitepaper and cm:content across all forms.

Configuring the form service for custom properties

The document details for a whitepaper looks better, but it isn’t showing the custom metadata. In this particular example, our model defines an association called sc:relatedDocuments which sc:whitepaper inherits from sc:doc and four properties across two custom aspects.

For the properties defined in aspects, you have a choice. You can either add the properties to every form definition you want them to appear in. Or, you can add an aspect configuration so that they’ll automatically be displayed for any object that includes that aspect. The advantage of the former is that you can have fine-grained control over where those fields appear whereas if you choose the latter route, the form service will decide where to include your fields.

This example takes the latter route. Edit the share-config-custom.xml file and add the following aspect configuration for sc:webable:



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
    <config evaluator="aspect" condition="sc:webable">
        <forms>
            <form>
                <field-visibility>
                    <show id="sc:published" />
                    <show id="sc:isActive" />
                </field-visibility>
                <appearance>
                    <field id="sc:published" label-id="prop.sc_published" />
                    <field id="sc:isActive" label-id="prop.sc_isActive" />
                </appearance>
            </form>
        </forms>
    </config>
```

Listing 7: Configuring a default form for the sc:webable aspect

You can add the aspect configuration for sc:productRelated following the same pattern.

Notice that there are two elements you have to worry about—the field-visibility element defines which properties are on the form while the appearance element defines how those properties are rendered. In this example I'm showing only a label-id, but there are a lot of options here. For example, if you wanted to override the component used to display the property, this is where you'd do it.

Let's take care of the sc:relatedDocuments association. It's not defined in an aspect, so it is added directly to the form configuration for sc:whitepaper. It probably makes sense for the related documents property to be shown on the default form and the edit metadata popup dialog. To do that, you're going to modify the default form configuration element for "sc:whitepaper" that you created earlier. First add a "show" element just before the closing "field-visibility" tag:

```
               <!-- cm:geographic aspect -->
               <show id="cm:latitude" />
               <show id="cm:longitude" />

               <!-- sc:doc -->
               <show id="sc:relatedDocuments" />
            </field-visibility>
```

Listing 8: Configuring the default whitepaper form to show the relatedDocuments association

Then, add a new "field" element just before the closing "appearance" tag:

```
               <field id="cm:addressees" read-only="true" />
               <field id="cm:sentdate" read-only="true" />
               <field id="cm:subjectline" read-only="true" />
               <field id="sc:relatedDocuments" label-
id="assoc.sc_relatedDocuments" />
            </appearance>
```

Listing 9: Configuring the default whitepaper form to show the relatedDocuments association

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Now do the same thing for the doclib-simple-metadata form.

After restarting Alfresco Share Tomcat, you should see all four custom properties and the related documents association in the document details page. On the edit metadata pop-up dialog you should see the related documents association.

Configuring Advanced Search in Alfresco Share

The advanced search form in Alfresco Share allows end-users to first select what they are looking for and then specify both full-text and specific property values to search for depending on the content type selected. Out-of-the-box, the search form includes two types: cm:content and cm:folder.

Users need to be able to search specifically for SomeCo Whitepapers, so the first step is to add the "sc:whitepaper" type to the list. Like the other Share form configuration covered thus far, the configuration goes in share-config-custom.xml. In this case, the condition is "AdvancedSearch".

```
<config replace="true" evaluator="string-compare" condition="AdvancedSearch">
    <advanced-search>
        <!-- Forms for the advanced search type list -->
        <forms>
            <form labelId="search.form.label.cm_content"
descriptionId="search.form.desc.cm_content">cm:content</form>
            <form labelId="search.form.label.cm_folder"
descriptionId="search.form.desc.cm_folder">cm:folder</form>
            <form labelId="type.sc_whitepaper"
descriptionId="search.form.desc.sc_whitepaper">sc:whitepaper</form>
        </forms>
    </advanced-search>
</config>
```

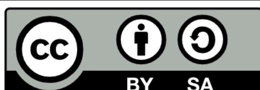
Listing 10: Configuring the forms list for Share Advanced Search

Notice that the list of advanced search forms replaces the out-of-the-box list. If the list only had "sc:whitepaper" and left out "cm:content" and "cm:folder", Share users would not be able to search for plain content or folders.

The next step is to tell Share which form to use when a given type is selected. Recall earlier that there are two sets of form configuration—one for existing nodes (evaluator of "node-type") and one for new nodes (evaluator of "model-type"). Search forms go in the "model-type" evaluator.

The search form for whitepapers should be the same as the one for plain content, but should include the four properties we've defined in our aspects. The easiest way to do this is to copy the "cm:content" search form from the out-of-the-box form configuration into share-config-custom.xml and then modify it to suit our needs. The code listing below shows this, with modifications in bold:

```
<!-- sc:whitepaper type (new nodes) -->
<config evaluator="model-type" condition="sc:whitepaper">
    <forms>
```



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
        <!-- Search form -->
        <form id="search">
            <field-visibility>
                <show id="cm:name" />
                <show id="cm:title" force="true" />
                <show id="cm:description" force="true" />
                <show id="mimetype" />
                <show id="cm:modified" />
                <show id="cm:modifier" />
                <!-- sc:productRelated -->
                <show id="sc:product" />
                <show id="sc:version" />
                <!-- sc:webable -->
                <show id="sc:isActive" />
                <show id="sc:published" />
            </field-visibility>
            <appearance>
                <field id="mimetype">
                    <control
template="/org/alfresco/components/form/controls/mimetype.ftl" />
                </field>
                <field id="cm:modifier">
                    <control>
                        <control-param name="forceEditable">true</control-param>
                    </control>
                </field>
                <field id="cm:modified">
                    <control
template="/org/alfresco/components/form/controls/daterange.ftl" />
                </field>
                <!-- sc:productRelated -->
                <field id="sc:product" label-id="prop.sc_product">
                    <control
template="/org/alfresco/components/form/controls/textfield.ftl" />
                </field>
                <field id="sc:version" label-id="prop.sc_version">
                    <control
template="/org/alfresco/components/form/controls/textfield.ftl" />
                </field>
                <!-- sc:webable -->
                <field id="sc:isActive" label-id="prop.sc_isActive">
                    <control
template="/org/alfresco/components/form/controls/checkbox.ftl" />
                </field>
                <field id="sc:published" label-id="prop.sc_published">
                    <control
template="/org/alfresco/components/form/controls/daterange.ftl" />
                </field>
```

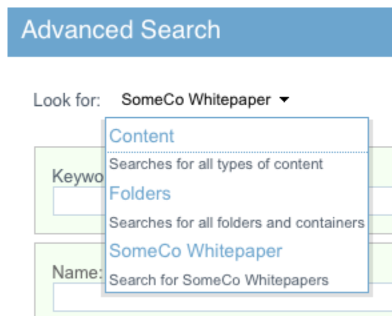

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
            </appearance>
        </form>
    </forms>
</config>
```

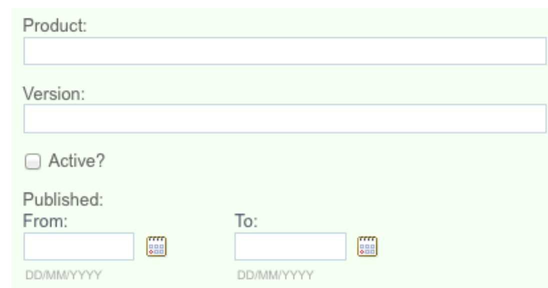
Listing 11: Configuring the search form for sc:whitepaper

It is important to note that the field elements in the appearance section require the form control to be specified. If it is not specified, the field will not show up on the search form.

After deploying this configuration and restarting, the SomeCo Whitepaper type is included in the advanced search dropdown, and four fields for the aspect properties are shown in the form.



Drawing 8: Search by Type



Drawing 9: Search custom properties

Localizing Strings for Custom Content Models

We’ve put off localizing the form labels until now. To fix this, create a file in web-extension called “custom-slingshot-application-context.xml” with the following content:

```
<?xml version='1.0' encoding='UTF-8'?><!DOCTYPE beans PUBLIC '-//SPRING//DTD
BEAN//EN' 'http://www.springframework.org/dtd/spring-beans.dtd'>

<beans>
   <!-- Add Someco messages -->
   <bean id="someco.resources"
class="org.springframework.extensions.surf.util.ResourceBundleBootstrapComponent">
      <property name="resourceBundles">
         <list>
            <value>alfresco.web-extension.messages.scModel</value>
         </list>
      </property>
   </bean>
</beans>
```


This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Listing 12: Adding a new Spring bean with a custom localized strings bundle

Now create a new directory in web-extension called “messages” and in that directory create a new file called “scModel.properties” with the following content:

```
#sc:doc
type.sc_doc=SomeCo Doc
assoc.sc_relatedDocuments=Related Documents

#sc:whitepaper
type.sc_whitepaper=SomeCo Whitepaper
search.form.desc.sc_whitepaper=Search for SomeCo Whitepapers

#sc:webable
aspect.sc_webable=SomeCo Webable
prop.sc_published=Published
prop.sc_isActive=Active?

#sc:productRelated
aspect.sc_productRelated=SomeCo Product Related
prop.sc_product=Product
prop.sc_version=Version
```

Listing 13: Localized strings for the custom model

After restarting Alfresco Share Tomcat you should see that the properties have the localized labels.

Share Configuration Summary

You’ve seen that configuring Alfresco Share for your custom content model essentially involves adding XML to the share-config-custom.xml file and creating a properties file for your localized strings. All of this lives under the “web-extension” directory in the Share web application.

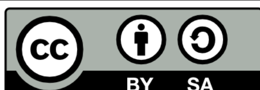
There are other things you might like to do to the Share user interface, but these are beyond the scope of this document:

- Add custom content types to the Create Content menu
- Add custom properties to the document library sort criteria
- Add custom properties to the document library data grid

Now let’s turn our attention from the front-end to the back-end to understand how to create, query, update, and delete content using code running remotely from the Alfresco server.

Part 3: Working with Content Programmatically

So far we've created a custom model and we've exposed the model in the Alfresco web clients. For



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

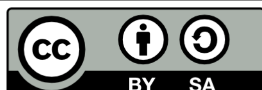
simple document management solutions, this may be enough. Often, code will also required. It could be code in a web application that needs to work with the repository, code that implements custom behavior for custom content types, or code that implements Alfresco web client customizations.

There are several API's available depending on what you want to do. The table below outlines the choices:

Solution type	Language	Alfresco API
Alfresco web client user interface customizations	Freemarker Templating Language Java/JSP	Alfresco Freemarker API Alfresco Foundation API
Custom applications with an embedded Alfresco repository (Repository runs in the same process as the application)	Java	Alfresco Foundation API JCR API
Custom applications using a remote Alfresco repository	Java, PHP, .NET, or any language that supports Web Services	Alfresco Web Services API CMIS (Web Services binding)
	Java, Python, PHP, .NET or any language that can make RESTful calls over HTTP/S	Alfresco JavaScript API (Custom Spring Surf Web Scripts) CMIS (ATOM Pub binding)
Server-side Scripts	JavaScript	Alfresco JavaScript API

The first edition of this tutorial focused on the Alfresco Web Services API with Java and PHP. However, a better approach exists thanks to Alfresco’s support for the Content Management Interoperability Services (CMIS) standard. So let’s look at CMIS examples for creating, updating, querying, and deleting content. The client API’s available from the Apache Chemistry project offer a variety of languages to choose from. This tutorial focuses on Java.

If you’d rather not use CMIS for some reason, the Alfresco Web Services examples for Java still work and are included in the Appendix.



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Setup

These examples are simple enough that I don't expect you to be copying-and-pasting from this document into your own source code. But you may want to do that or you may be curious about my setup.

I use Eclipse. In my Eclipse workspace I have imported the Alfresco SDK samples, which includes the SDK AlfrescoEmbedded and SDK AlfrescoRemote projects. Alfresco 4.0.c Community has a few issues with the SDK. In short, you may find yourself needing to re-add the dependent JARs in the embedded and remote projects before anything will build cleanly.

The AlfrescoEmbedded project contains the OpenCMIS JARs and the OpenCMIS extension JAR. Obviously, you could just download OpenCMIS and use those JARs—if you are developing a pure CMIS client you do not need the Alfresco SDK.

The AlfrescoRemote project contains the Alfresco Web Services API JARs. This is only used if you compile and run the Java Web Services examples in the Appendix.

If you do not use Eclipse, I've made sure that all examples will compile and run cleanly from the command line using Ant. All you need to do in that case is download and unpack the Alfresco SDK, then specify the SDK location in the build.properties file in this article's source code.

The source code that accompanies this document is in a single Eclipse project. I am using a simple Ant build script for the Java examples as well as to make it easy to quickly deploy the content model and user interface configuration from Part 1 and Part 2. In real life, you'll want to package and deploy everything as an AMP, but I'll save the AMP discussion for another day.

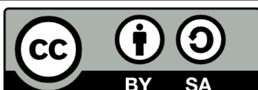
All of the Java code in this tutorial assumes you are executing against a local Alfresco repository running on localhost at port 8080. It also assumes you've deployed the SomeCo content model from Part 1.

Creating Content with OpenCMIS

The code we're going to use for creating content is a port of the Alfresco Web Services based code from the first edition of this tutorial, which is, in turn, almost exactly the same code that comes with the Alfresco SDK Samples.

The goal here is to create a runnable class that accepts arguments for the username, password, folder in which to create the content, type of content we're creating, and a name for the new content. I've left out the main method as well as the code that establishes the session, but you can see the full class in its entirety if you download the code that accompanies this document (See "Where to Find More Information").

The first thing the code does is grab a session. The getSession method is inherited from a class called



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

CMISExampleBase which is used for all of the CMIS examples in this document. Next, it gets a reference to the folder where the content will be created. The timestamp is incorporated into the content name so that if the code is run multiple times, the object names will be unique.

```
Session session = getSession();

// Grab a reference to the folder where we want to create content
Folder folder = (Folder) session.getObjectByPath("/" + getFolderName());

String timeStamp = new Long(System.currentTimeMillis()).toString();
String filename = getContentName() + " (" + timeStamp + ")";
```

Listing 14: Snippet from SomeCoCMISDataCreator.java

Next, the code sets up the properties that will be set on the new document. It creates a Map of Strings and Objects to hold the property names and values.

```
// Create a Map of objects with the props we want to set
Map <String, Object> properties = new HashMap<String, Object>();
// Following sets the content type and adds the webable and productRelated aspects
// This works because we are using the OpenCMIS extension for Alfresco
properties.put(PropertyIds.OBJECT_TYPE_ID,
               "D:sc:whitepaper,P:sc:webable,P:sc:productRelated");
properties.put(PropertyIds.NAME, filename);
properties.put("sc:isActive", true);
GregorianCalendar publishDate = new GregorianCalendar(2007,4,1,5,0);
properties.put("sc:published", publishDate);
```

Listing 15: Snippet from SomeCoCMISDataCreator.java

Notice that instead of listing a single value for the object type ID, a comma-separated list is being passed in. This is the content type followed by the aspects that need to be added. This is possible because the code leverages an Alfresco-specific extension that allows us to work with aspects. It is also important to point out that, in CMIS, document types begin with “D:” while policy types begin with “P:”. (CMIS 1.0 doesn’t have a native concept of aspects—what Alfresco calls an aspect, CMIS 1.0 calls a “Policy”--just go with it).

The next step is to prepare the content that will be set on the new object. This is a matter of calling the ContentStreamImpl constructor with the file name, length, mimetype, and an InputStream based on the content.

```
String docText = "This is a sample " + getContentType() + " document called " +
       getContentName();
byte[] content = docText.getBytes();
InputStream stream = new ByteArrayInputStream(content);
ContentStream contentStream = new ContentStreamImpl(filename,
       BigInteger.valueOf(content.length), "text/plain", stream);
```

Listing 16: Snippet from SomeCoCMISDataCreator.java

Finally, the code tells the folder to create a new document and passes in the properties, content stream,

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

and versioning state, then dumps the length of the content that was created.

```
Document doc = folder.createDocument(properties, contentStream,
     VersioningState.MAJOR);
System.out.println("Content Length: " + doc.getContentStreamLength());
```

Listing 17: Snippet from SomeCoCMISDataCreator.java

The sample code that accompanies this tutorial includes a very basic Ant build file. If you have Ant, you can compile and run the data creation example by typing the following in the root of the source code directory:

```
ant cmis-data-creator
```

Listing 18: Running SomeCoCMISDataCreator.java using Ant

Running the Java snippet produces:

```
Created: workspace://SpacesStore/4f1725a0-db29-4b0f-8fc8-ea625cf49356;1.0
Content Length: 59
```

Listing 19: Command line results after running SomeCoCMISDataCreator.java

Creating Associations with OpenCMIS

Now let's write a class to create the “related documents” association between two documents.

The class accepts a source object ID and a target object ID as arguments. The code creates a map of properties containing the association type, source ID, and target ID. Note that the association type is preceded by “R:” when working with CMIS.

```
Session session = getSession();

// Create a Map of objects with the props we want to set
Map <String, String> properties = new HashMap<String, String>();
properties.put(PropertyIds.OBJECT_TYPE_ID, "R:sc:relatedDocuments");
properties.put(PropertyIds.SOURCE_ID, getSourceObjectId());
properties.put(PropertyIds.TARGET_ID, getTargetObjectId());
try {
    session.createRelationship(properties);
} catch (Exception e) {
    System.out.println("Oops, something unexpected happened. Maybe the rel already
exists?");
}
```

Listing 20: Snippet from SomeCoCMISDataRelater.java

The last half of the method dumps the associations of the source object. The trick here is that when you make the getObject call to instantiate the object, you will not get back the relationships by default. The OperationContext that is being instantiated makes sure that happens.

```
// Dump the object's associations
OperationContext oc = new OperationContextImpl();
```


This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
oc.setIncludeRelationships(IncludeRelationships.SOURCE);
Document sourceDoc = (Document) session.getObject(
              session.createObjectId(getSourceObjectId()),
              oc);
List<Relationship> relList = sourceDoc.getRelationships();
System.out.println("Associations of objectId:" + getSourceObjectId());
for (Relationship rel : relList) {
    System.out.println("    " + rel.getTarget().getId());
}
```

Listing 21: Snippet from SomeCoCMISDataRelater.java

The last line calls a method that queries the associations for a given reference. This should dump the association that was just created plus any other associations that have been created for this object.

If you want to run this on your own using the Ant task, it would look something like this, with your own values for the source and target object ID's:

```
ant cmis-data-relater -DsourceId="workspace://SpacesStore/7f7b2d2c-95de-406b-b987-adbc00051bde;1.0" -DtargetId="workspace://SpacesStore/b53dabe9-355b-4ade-a5e9-cd4dffa75916;1.0"
```

Listing 22: Running SomeCoCMISDataRelater.java using Ant

Running the Java snippet as above, assuming no other relationships exist on the source object produces:

```
Associations of objectId: workspace://SpacesStore/7f7b2d2c-95de-406b-b987-adbc00051bde;1.0
    workspace://SpacesStore/b53dabe9-355b-4ade-a5e9-cd4dffa75916;1.0
```

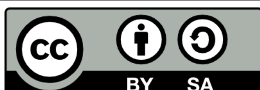
Listing 23: Results of running SomeCoCMISDataRelater.java

Now you can use the Alfresco Share Client to view the associations. Remember the share-config-custom.xml file? It says that any time the default form or edit metadata form is used for sc:whitepaper objects, the sc:relatedDocuments associations should be shown. Alternatively, the Node Browser, available in the Alfresco Share Administration Console, is a handy way to view associations.

Searching for Content with OpenCMIS

Now that there are some instances of SomeCo's custom types in the repository it is time to write code that will query for it. Alfresco 4.0 ships with two options for search. The first option is to use Lucene, which is the embedded search engine that has always shipped with Alfresco. The second option is to use Solr. If you upgraded an existing pre-4.0 installation and made no other changes, you are using Lucene. If you installed 4.0 using the installer, you are using Solr. You can switch from one to the other—refer to the documentation to find out how.

Regardless of the search engine option you've chosen, content in the repository is automatically indexed by Alfresco. You can execute searches to find content based on full-text, property values,



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

content types, and folder paths.

There are a number of supported search dialects. Prior to 3.4, the most common dialect was Lucene. Starting with 3.4, Alfresco introduced a new dialect called Alfresco FTS which is an abstraction that means you don't have to learn the specific Lucene search syntax. CMIS has its own query language which looks a lot like SQL. XPath is also an option, although it's not used that often.

If you are writing code against an Alfresco repository that is 3.4 or higher, you should use either Alfresco FTS or CMIS Query Language, if possible. The first edition of this document walked you through some sample queries using Lucene. Those are in the Appendix if you need them. This edition uses the exact same set of examples, but uses CMIS Query Language instead of Lucene.

Just like the content creation code, the class will be a runnable Java application that accepts the username, password, and folder name as arguments. It includes a generic method used to execute a query which is called repeatedly with multiple query examples.

If you are following along, you should either run the content creation code a few times or create some content manually so you can test out the queries more thoroughly.

Let's take a look at the generic query execution method first, then the method that calls it for each example query string.

The getQueryResults method is pretty straightforward. It returns a list of CmisObject objects. It instantiates those objects by iterating over the query results, grabbing the objectId from each result, and making a getObject() call on the session.

```
public List<CmisObject> getQueryResults(String queryString) {
    List<CmisObject> objList = new ArrayList<CmisObject>();
    Session session = getSession();

    // execute query
    ItemIterable<QueryResult> results = session.query(queryString, false);

    for (QueryResult qResult : results) {
        PropertyData<?> propData = qResult.getPropertyById("cmis:objectId");
        String objectId = (String) propData.getFirstValue();
        CmisObject obj = session.getObject(session.createObjectId(objectId));
        objList.add(obj);
    }

    return objList;
};
```

Listing 24: Generic method for executing queries from SomeCoCMISDataQueries.java

The doExamples() method then executes a series of example queries and dumps the results. The first two queries are simple. One returns every instance of "sc:doc" including instances of types that inherit from "sc:doc". The second one finds any objects residing in the folder passed in that contains the word

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

"sample" anywhere in the content. Notice the SQL-like syntax of CMIS Query Language. It basically treats content types as if they were tables.

```
System.out.println(RESULT_SEP);
System.out.println("Finding content of type:" +
     SomeCoModel.TYPE_SC_DOC.toString());
queryString = "select * from sc:doc";
dumpQueryResults(getQueryResults(queryString));

System.out.println(RESULT_SEP);
System.out.println("Find content in the root folder with text like 'sample'");
queryString = "select * from cmis:document where contains('sample') and
     in_folder('" + getFolderId() + "')";
dumpQueryResults(getQueryResults(queryString));
```

Listing 25: Two basic CMIS queries

You might have noticed the getFolderId() call. The "in_folder" predicate expects an object ID. So the getFolderId() method does a query to find the object ID of the folder that was passed in as an argument to the class. It would be nice if you could do this in a single query, but you can't.

Queries on Aspect-based Properties

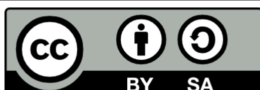
The next query looks for active content. This is when it starts to get interesting because the property that tracks whether or not a piece of content is active, "sc:isActive", is defined on an aspect. The CMIS specification allows for joins in queries. But Alfresco does not support joins *except* in the special case of aspects. In Alfresco CMIS, joins are used to relate a base type to one of its aspects. That allows you to use an aspect-based property in a where clause.

```
System.out.println(RESULT_SEP);
System.out.println("Find active content");
queryString = "select d.*, w.* from cmis:document as d join sc:webable as w on
     d.cmis:objectId = w.cmis:objectId where w.sc:isActive = True";
dumpQueryResults(getQueryResults(queryString));
```

Listing 26: Snippet from SomeCoCMISDataQueries.java showing calls to getQueryResults

Queries Across Multiple Aspects

The next query shows another special case. In this example the goal is to find the active content that has a product property set to a specific value. That's a challenge because the "sc:isActive" property is defined by the "sc:webable" aspect while the "sc:product" property is defined by a different aspect, "sc:productRelated". Unfortunately, there is no good way to get these results in a single query. The solution used here is to write a method called getSubQueryResults() that accepts two queries as arguments. The method runs the first query and then builds an IN predicate using the object IDs that come back, which it appends to the second query before invoking it.



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
System.out.println(RESULT_SEP);
System.out.println("Find active content with a product equal to 'SomePortal'");
String queryString1 = "select d.cmis:objectId from cmis:document as d join
    sc:productRelated as p on d.cmis:objectId = p.cmis:objectId " +
    "where p.sc:product = 'SomePortal'";
String queryString2 = "select d.cmis:objectId from cmis:document as d join
    sc:webable as w on d.cmis:objectId = w.cmis:objectId " +
          "where w.sc:isActive = True";
dumpQueryResults(getSubQueryResults(queryString1, queryString2));
```

Listing 27: Snippet from SomeCoCMISDataQueries.java showing calls to getQueryResults

Queries Using Dates

The last query uses the aspect join trick to do a date range search on instances of “sc:whitepaper” published between a specific range.

```
System.out.println(RESULT_SEP);
System.out.println("Find content of type sc:whitepaper published between 1/1/2006
    and 6/1/2007");
queryString = "select d.cmis:objectId, w.sc:published from sc:whitepaper as d join
    sc:webable as w on d.cmis:objectId = w.cmis:objectId " +
    "where w.sc:published > TIMESTAMP '2006-01-01T00:00:00.000-05:00' " +
    "  and w.sc:published < TIMESTAMP '2007-06-02T00:00:00.000-05:00'";
dumpQueryResults(getQueryResults(queryString));
```

Listing 28: Snippet from SomeCoCMISDataQueries.java showing calls to getQueryResults

If you want to compile and run this on your own machine, you can use one of the Ant tasks in the build.xml file included in the code that accompanies this article. Just type:

```
ant cmis-data-queries
```

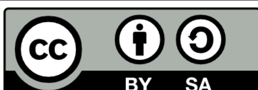
Listing 29: Running SomeCoCMISDataQueries.java using Ant

Your results will vary based on how much content you've created and the values you've set in the content properties, but when I ran my test, the results were as shown below.

```
======================
Finding content of type:doc
----------------------
Result 1:
id:workspace://SpacesStore/7f7b2d2c-95de-406b-b987-adbc00051bde;1.0
name:whitepaper (1325799329291)
created:Jan 5, 2012 3:35:29 PM
----------------------
Result 2:
id:workspace://SpacesStore/e4e6baac-7a58-40ed-a10b-ea703d444c97;1.0
name:whitepaper (1325800839908)
created:Jan 5, 2012 4:00:39 PM
----------------------
```


This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
Result 3:
id:workspace://SpacesStore/b53dabe9-355b-4ade-a5e9-cd4dffa75916;1.0
name:whitepaper (1325799336650)
created:Jan 5, 2012 3:35:36 PM
======================
Find content in the root folder with text like 'sample'
----------------------
Result 1:
id:workspace://SpacesStore/7f7b2d2c-95de-406b-b987-adbc00051bde;1.0
name:whitepaper (1325799329291)
created:Jan 5, 2012 3:35:29 PM
----------------------
Result 2:
id:workspace://SpacesStore/e4e6baac-7a58-40ed-a10b-ea703d444c97;1.0
name:whitepaper (1325800839908)
created:Jan 5, 2012 4:00:39 PM
----------------------
Result 3:
id:workspace://SpacesStore/b53dabe9-355b-4ade-a5e9-cd4dffa75916;1.0
name:whitepaper (1325799336650)
created:Jan 5, 2012 3:35:36 PM
======================
Find active content
----------------------
Result 1:
id:workspace://SpacesStore/7f7b2d2c-95de-406b-b987-adbc00051bde;1.0
name:whitepaper (1325799329291)
created:Jan 5, 2012 3:35:29 PM
----------------------
Result 2:
id:workspace://SpacesStore/e4e6baac-7a58-40ed-a10b-ea703d444c97;1.0
name:whitepaper (1325800839908)
created:Jan 5, 2012 4:00:39 PM
----------------------
Result 3:
id:workspace://SpacesStore/b53dabe9-355b-4ade-a5e9-cd4dffa75916;1.0
name:whitepaper (1325799336650)
created:Jan 5, 2012 3:35:36 PM
======================
Find active content with a product equal to 'SomePortal'
----------------------
Result 1:
id:workspace://SpacesStore/b53dabe9-355b-4ade-a5e9-cd4dffa75916;1.0
name:whitepaper (1325799336650)
created:Jan 5, 2012 3:35:36 PM
======================
Find content of type sc:whitepaper published between 1/1/2006 and 6/1/2007
----------------------
```



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
Result 1:
id:workspace://SpacesStore/7f7b2d2c-95de-406b-b987-adbc00051bde;1.0
name:whitepaper (1325799329291)
created:Jan 5, 2012 3:35:29 PM
----------------------
Result 2:
id:workspace://SpacesStore/e4e6baac-7a58-40ed-a10b-ea703d444c97;1.0
name:whitepaper (1325800839908)
created:Jan 5, 2012 4:00:39 PM
----------------------
Result 3:
id:workspace://SpacesStore/b53dabe9-355b-4ade-a5e9-cd4dffa75916;1.0
name:whitepaper (1325799336650)
created:Jan 5, 2012 3:35:36 PM
```

Listing 30: Command line results for SomeCoCMISDataQueries.java

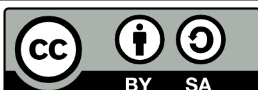
Deleting Content with OpenCMIS

Now it is time to clean up after ourselves by deleting content from the repository. The delete logic is similar to the search logic except that instead of dumping the results, the CmisObject's delete() method gets called on every hit that is returned.

```
Session session = getSession();

// execute query
String queryString = "select * from sc:doc";
ItemIterable<QueryResult> results = session.query(queryString, false);

// if we found some rows, create an array of DeleteCML objects
if (results.getTotalNumItems() >= 0)
    System.out.println("Found " + results.getTotalNumItems() + " objects to
delete.");

for (QueryResult qResult : results) {
    PropertyData<?> propData = qResult.getPropertyById("cmis:objectId");
    String objectId = (String) propData.getFirstValue();
    CmisObject obj = session.getObject(session.createObjectId(objectId));
    obj.delete(true);
    System.out.println("Deleted: " + objectId);
}
System.out.println("Done!");
```

Listing 31: Code snippet from SomeCoCMISDataCleaner.java

Note that this code deletes every object in the repository of type sc:doc *and instances of child types.* You would definitely want to "think twice and cut once" if you were running this code on a production repository, particularly if you were using a broad content type like cm:content.



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Similar to the other examples, you can compile and run this on your own by executing the following:

```
ant cmis-data-cleaner
```

Listing 32: Running SomeCoCMISDataCleaner.java using Ant

Again, your results will vary based on the content you've created but in my repository, running the code results in the following:

```
Deleted: workspace://SpacesStore/78322173-a603-456c-a891-22a9279626c2;1.0
Deleted: workspace://SpacesStore/217afd99-4f0c-4760-8b93-7c23fbc908f3;1.0
Done!
```

Listing 33: Command line results for SomeCoDataCleaner.java

You'll notice that the System.out.println that displays the number of results is not shown in the output. That's because I used the Atom Pub binding and the getTotalNumItems() call always returns -1 for that binding when run against Alfresco 4.0.c. You'll find discrepencies like that between the two bindings.

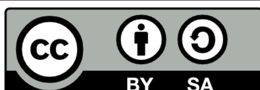
Conclusion

This article has shown how to extend Alfresco's out-of-the-box content model with your own business-specific content types, how to expose those types, aspects, and properties in the Alfresco web clients, and how to work with content via OpenCMIS, the Java API for CMIS that is part of Apache Chemistry. I've thrown in a few recommendations that will hopefully save you some time or at least spark some discussion.

There's plenty of additional data model-related customizations to cover in future articles including custom behaviors, custom metadata extractors, custom transformers, and custom data renderers.

Where to Find More Information

- The complete source code for these examples is available here from ecmarchitect.com.
- The Alfresco SDK includes everything you need to compile Java that works with Alfresco.
- Official documentation for both Enterprise and Community is available at docs.alfresco.com.
- Share Extras has many examples of deeper Share customization.
- The Search-related pages on the Alfresco wiki provide query examples using both Lucene and XPath.
- See "Getting Started with CMIS" on ecmarchitect.com for a brief introduction to CMIS. The Alfresco CMIS page is also a great resource.
- The Apache Chemistry Home Page has examples and source code that works with CMIS.
- For deployment help, see Packaging and Deploying Extensions in the Alfresco wiki.
- For general development help, see the Developer Guide.
- For help customizing the data dictionary, see the Data Dictionary wiki page.
- If you are ready to cover new ground, try another ecmarchitect.com tutorial in the Alfresco



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

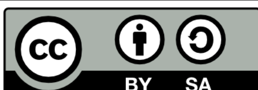
Developer Series.

About the Author

Jeff Potts is the Chief Community Officer for Alfresco Software. Jeff has been a recognized and award-winning leader in the Alfresco community for many years. He has over 18 years of content management and collaboration experience, most of that having come from senior leadership positions in consulting organizations.

Jeff has made many contributions to the Alfresco community since he started working with Alfresco in 2005. Examples range from code, to tutorials and informative blog posts, to code camps, meetups, and conference sessions. In 2008, Jeff wrote the Alfresco Developer Guide, the first developer-focused book on Alfresco. He has also been active in the Apache Chemistry project where he leads the development of cmislib, the Python API for CMIS.

Read more at Jeff's blog, ecmarchitect.com.



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Appendix

Configuring the Custom Content Model in Alfresco Explorer

Alfresco Explorer is the original web client Alfresco shipped with the product. As such, it hasn't seen much attention in recent releases. Still, for one reason or another, you might need to configure the Alfresco Explorer client for your custom content model, so let's see how to do that.

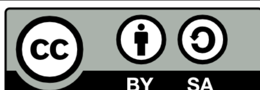
First, think about the web client and all of the places the content model customizations need to show up:

- **Property Sheet**. When a user looks at a property sheet for a piece of content stored as one of the custom types or with one of the custom aspects attached, the property sheet should show the custom properties.
- **Create content/add content**. When a user clicks Create or Add Content, the custom types should be a choice in the list of content types.
- **"Is sub-type" criteria**. When a user configures a rule on a space and uses content types as a criteria, the custom types should be a choice in the list of possible content types.
- **Specialize type action**. When a user runs the "specialize type" action, the custom types should be available.
- **Add aspect**. When a user runs the "add aspect" action, the custom aspects should be available.
- **Advanced search**. When a user runs an advanced search, they should be able to restrict search results to instances of our custom types and/or content with specific values for the properties of our custom types.

All of these are handled through a file called web-client-config. You can see the out-of-the-box web-client-config.xml file under $TOMCAT_HOME/webapps/alfresco/WEB-INF/classes/alfresco. The web-client-config.xml file is a proprietary file composed of numerous "config" elements. Each config element has an "evaluator" and a "condition". Extending the web-client-config.xml file is a matter of knowing which evaluator/condition to use.

To customize the user interface, you'll create a web-client-config-custom.xml file in your extension directory with a root element called "alfresco-config". Then, you'll add the appropriate config elements as children of alfresco-config.

Let's look at each of the areas mentioned above in order to understand how the custom content model can be exposed in the user interface.



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Property Sheet

When a user looks at a property sheet for a piece of content stored as one of the custom types or with one of the custom aspects attached, the property sheet should show the custom properties as shown below:

Drawing 10: Customer properties on the property sheet

If you have not already created a file called web-client-config-custom.xml in your extension directory with the following content, do so now:

```
<alfresco-config>
</alfresco-config>
```

Listing 34: Starter web-client-config-custom.xml

To add properties to property sheets use the “aspect-name” evaluator for aspects and “node-type” for content types. The snippet below shows the config for the sc:webable aspect. The sc:productRelated aspect would be similar.

```
    <!--  add webable aspect properties to property sheet -->
    <config evaluator="aspect-name" condition="sc:webable">
        <property-sheet>
            <show-property name="sc:published" display-label-id="published" />
            <show-property name="sc:isActive" display-label-id="isActive" read-
only="true" />
            <show-association name="sc:relatedDocuments" />
        </property-sheet>
    </config>
```

Listing 35: Snippet from web-client-config-custom.xml

Note the display-label-id attribute. You could specify the label in this file by using the label attribute, but a better practice is to externalize the string so the interface could be localized if needed. At the end of this section you'll see where the localized strings reside.

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Create Content/Add Content

When a user clicks Create or Add Content, the custom types should be a choice in the list of content types as shown below:

Drawing 11: Custom type in the add content dialog

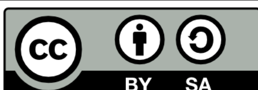
To add content types to the list of available types in the create content and add content dialogs, use the "string-compare" evaluator and the "Content Wizards" condition.

```
    <!--  add someco types to add content list -->
    <config evaluator="string-compare" condition="Content Wizards">
        <content-types>
            <type name="sc:doc" />
            <type name="sc:whitepaper" />
        </content-types>
    </config>
```

Listing 36: Snippet from web-client-config-custom.xml

"Is sub-type" Criteria, Specialize Type Action, Add Aspect Action

When a user configures a rule on a space and uses content types or aspects as a criteria, or runs an action related to types or aspects, the appropriate list of types and aspects should be displayed as shown below:



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Drawing 12: Custom aspect in "add aspect" dialog

Drawing 13: Custom aspect in "has aspect" dialog

These customizations are all part of the same config element. The “Action Wizards” config has several child elements that can be used. The “aspects” element defines the list of aspects shown when the “add aspect” action is configured. The “subtypes” element lists types that show up in the dropdown when configuring the content type criteria for a rule. The “specialise-types” element (note the UK spelling) lists the types available to the “specialize type” action.

```
    <config evaluator="string-compare" condition="Action Wizards">
        <!-- The list of aspects to show in the add/remove features action -->
        <!-- and the has-aspect condition -->
        <aspects>
            <aspect name="sc:webable"/>
            <aspect name="sc:productRelated"/>
        </aspects>

        <!-- The list of types shown in the is-subtype condition -->
        <subtypes>
            <type name="sc:doc" />
            <type name="sc:whitepaper" />
        </subtypes>

        <!-- The list of content and/or folder types shown in the specialise-type
action -->
        <specialise-types>
            <type name="sc:doc" />
            <type name="sc:whitepaper" />
        </specialise-types>
    </config>
```

Listing 37: Snippet from web-client-config-custom.xml

Advanced Search

When a user runs an advanced search, they should be able to restrict search results to instances of our custom types and/or content with specific values for the properties of our custom types as shown below:

Drawing 14: Custom types and properties in advanced search

The “Advanced Search” config specifies which content types and properties can be used to refine an advanced search result set.

```
    <config evaluator="string-compare" condition="Advanced Search">
        <advanced-search>
            <content-types>
                <type name="sc:doc" />
                <type name="sc:whitepaper" />
            </content-types>
            <custom-properties>
                <meta-data aspect="sc:webable" property="sc:published" display-
label-id="published" />
                <meta-data aspect="sc:webable" property="sc:isActive" display-
label-id="isActive" />
                <meta-data aspect="sc:productRelated" property="sc:product"
display-label-id="product" />
                <meta-data aspect="sc:productRelated" property="sc:version"
display-label-id="version" />
            </custom-properties>
        </advanced-search>
    </config>
```

Listing 38: Snippet from web-client-config-custom.xml

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

String Externalization

The last step in exposing our custom content model to the user interface is externalizing the strings used for labels in the interface. This is accomplished by creating a webclient.properties file in the extension directory. The properties file is a standard resource bundle. It holds key-value pairs. The keys match the “display-label-id” attribute values in the web-client-config-custom.xml file.

In our example, there are four properties that need labels. The entire contents of our webclient.properties file would be as follows:

```
#sc:webable
published=Published
isActive=Active?

#sc:productRelated
product=Product
version=Version
```

Listing 39: webclient.properties

Test the Web Client User Interface Customizations

Now that your web-client-custom-config.xml and webclient.properties files have been modified and placed in your extension directory, you should be able to restart Tomcat and see your changes. If you don't, check the Tomcat log for error messages, then re-check your model and web client files.

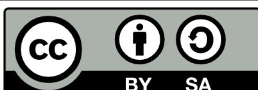
Using the Web Services API

If you can use CMIS, you should. Your code will be portable across other CMIS-compliant repositories and you’ll be able to leverage client libraries that are widely used and actively developed. If you can’t use CMIS and you have your heart set on using SOAP-based Web Services, you can use Alfresco’s Web Services API. This section includes examples of that API. Some of the examples in this section will look similar to the samples shipped with the SDK, but hopefully these provide a more complete end-to-end example.

Creating Content with Java Web Services

First, let's create some content and add some aspects. The general gist is that we're going to:

- Authenticate
- Get a reference to the folder where the content should reside
- Create a series of Content Manipulation Language (CML) objects that encapsulate the operations to execute



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

- Execute the CML
- Dump the results

Let's use Java to do this, but you could use any language that can use Web Services.

The code we're going to use for creating content is almost exactly the same code that comes with the Alfresco SDK Samples. But I think it is helpful to break it down here.

Also note that all of this code is contained within a try-catch-finally block (the session is closed in finally) but I've left it out here for brevity. The class is runnable with arguments being passed in for the username, password, folder in which to create the content, type of content we're creating, and a name for the new content. Again, I've left that out but you can see the full class in its entirety if you download the code that accompanies this document (See “Where to Find More Information”).

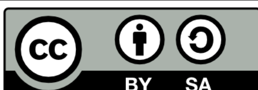
The first thing the code does is start a session. Next, it gets a reference to the folder where the content will be created. The timestamp is incorporated into the content name so that if the code is run multiple times, the object names will be unique.

```
AuthenticationUtils.startSession(getUser(), getPassword());

Store storeRef = new Store(Constants.WORKSPACE_STORE, "SpacesStore");
String folderPath = "/app:company_home/cm:" + getFolderName();
String timeStamp = new Long(System.currentTimeMillis()).toString();
ParentReference docParent = new ParentReference(
    storeRef,
    null,
    folderPath,
    Constants.ASSOC_CONTAINS,
    Constants.createQNameString(SomeCoModel.NAMESPACE_SOMECO_CONTENT_MODEL,
getContentName() + timeStamp));
```

Listing 40: Snippet from SomeCoDataCreator.java

Next, create an array of NamedValue objects for the properties and their values. Take special note of that date-time format.

```
NamedValue nameValue = Utils.createNamedValue(Constants.PROP_NAME, getContentName()
+ " (" + timeStamp + ")");;

NamedValue activeValue = Utils.createNamedValue(
    Constants.createQNameString(SomeCoModel.NAMESPACE_SOMECO_CONTENT_MODEL,
SomeCoModel.PROP_IS_ACTIVE),
    "true");

NamedValue publishDateValue = Utils.createNamedValue(
    Constants.createQNameString(SomeCoModel.NAMESPACE_SOMECO_CONTENT_MODEL,
SomeCoModel.PROP_PUBLISHED),
    "2007-04-01T00:00:00.000-05:00");
```



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
NamedValue[] contentProps = new NamedValue[] {nameValue, activeValue,
publishDateValue};
```

Listing 41: Snippet from SomeCoDataCreator.java

Now Content Manipulation Language (CML) comes into play. CML objects can be used to execute various content operations. In this case, the code uses CMLCreate and CMLAddAspect objects that create the node and add aspects to it. Note the "ref1" string. That's an arbitrary reference that Alfresco uses to relate the CML statements. Without it, Alfresco wouldn't know which content object to add the aspects to.

```
CMLCreate createDoc = new CMLCreate(
    "ref1",
    docParent,
    null,
    null,
    null,
    Constants.createQNameString(SomeCoModel.NAMESPACE_SOMECO_CONTENT_MODEL,
SomeCoModel.TYPE_SC_DOC),
    contentProps);

CMLAddAspect addWebableAspectToDoc = new CMLAddAspect(
    Constants.createQNameString(SomeCoModel.NAMESPACE_SOMECO_CONTENT_MODEL,
SomeCoModel.ASPECT_SC_WEBABLE),
    null,
    null,
    "ref1");

CMLAddAspect addProductRelatedAspectToDoc = new CMLAddAspect(
    Constants.createQNameString(SomeCoModel.NAMESPACE_SOMECO_CONTENT_MODEL,
SomeCoModel.ASPECT_SC_PRODUCT_RELATED),
    null,
    null,
    "ref1");
```

Listing 42: Snippet from SomeCoDataCreator.java

To execute the CML the code instantiates a new CML object and pass the create and add aspect arrays to the CML object's setter methods. The CML object is passed to the update method of the Repository Service which returns an UpdateResult array. The dumpUpdateResults method just iterates through the UpdateResult array and writes some information to sysout.

```
// Construct CML Block
CML cml = new CML();
cml.setCreate(new CMLCreate[] {createDoc});
cml.setAddAspect(new CMLAddAspect[] {addWebableAspectToDoc,
addProductRelatedAspectToDoc});

// Execute CML Block
UpdateResult[] results = WebServiceFactory.getRepositoryService().update(cml);
```

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
Reference docRef = results[0].getDestination();
dumpUpdateResults(results);
```

Listing 43: Snippet from SomeCoDataCreator.java

The last step writes some text content to the newly-created content node. This example uses a String for the content but it could just as easily write the bytes from a file on the local file system.

```
// Nodes are created, now write some content
ContentServiceSoapBindingStub contentService =
WebServiceFactory.getContentService();
ContentFormat contentFormat = new ContentFormat("text/plain", "UTF-8");
String docText = "This is a sample " + getContentType() + " document called " +
getContentName();
Content docContentRef = contentService.write(docRef, Constants.PROP_CONTENT,
docText.getBytes(), contentFormat);
System.out.println("Content Length: " + docContentRef.getLength());
```

Listing 44: Snippet from SomeCoDataCreator.java

Running the Java snippet produces:

```
Command = create; Source = none; Destination = b901941e-12d3-11dc-9bf3-e998e07a8da1
Command = addAspect; Source = b901941e-12d3-11dc-9bf3-e998e07a8da1; Destination =
b901941e-12d3-11dc-9bf3-e998e07a8da1
Command = addAspect; Source = b901941e-12d3-11dc-9bf3-e998e07a8da1; Destination =
b901941e-12d3-11dc-9bf3-e998e07a8da1
Content Length: 26
```

Listing 45: Command line results after running SomeCoDataCreator.java

The sample code that accompanies this tutorial includes a very basic Ant build file. If you have Ant, you can run the data creation example by typing the following in the root of the source code directory:

```
ant data-creator
```

Creating Associations with Java Web Services

Now let's write a class to create the “related documents” association between two documents.

The mechanics are essentially the same. We're going to set up and execute some CML. Our class will accept a source UUID and a target UUID as arguments which are passed in to the CMLCreateAssociation constructor, execute the CML, and then dump the results.

```
Reference docRefSource = new Reference(storeRef, getSourceUuid(), null);
Reference docRefTarget = new Reference(storeRef, getTargetUuid(), null);

CMLCreateAssociation relatedDocAssoc = new CMLCreateAssociation(new Predicate(new
Reference[]{docRefSource}, null, null),
    null,
    new Predicate(new Reference[] {docRefTarget}, null, null),
    null, Constants.createQNameString(SomeCoModel.NAMESPACE_SOMECO_CONTENT_MODEL,
```


This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
        SomeCoModel.ASSN_RELATED_DOCUMENTS));

// Setup CML block
CML cml = new CML();
cml.setCreateAssociation(new CMLCreateAssociation[] {relatedDocAssoc});

// Execute CML Block
UpdateResult[] results = WebServiceFactory.getRepositoryService().update(cml);
    dumpUpdateResults(results);
System.out.println("Associations of sourceUuid:" + getSourceUuid());

dumpAssociations(docRefSource,
Constants.createQNameString(SomeCoModel.NAMESPACE_SOMECO_CONTENT_MODEL,
SomeCoModel.ASSN_RELATED_DOCUMENTS));
```

Listing 46: Snippet from SomeCoDataRelater.java

The last line calls a method that queries the associations for a given reference. This should dump the association that was just created plus any other associations that have been created for this object.

Running the Java snippet produces:

```
Command = createAssociation; Source = 1355e60e-160b-11dc-a66f-bb03ffd77ac6;
Destination = bd0bd57d-160c-11dc-a66f-bb03ffd77ac6
Associations of sourceUuid:1355e60e-160b-11dc-a66f-bb03ffd77ac6
bd0bd57d-160c-11dc-a66f-bb03ffd77ac6
{http://www.alfresco.org/model/content/1.0}name:Test Document 2 (1181340487582)
```

Listing 47: Results of running SomeCoDataRelater.java

If you want to run this on your own using the Ant task, it would look something like this, with your own values for the source and target object ID's:

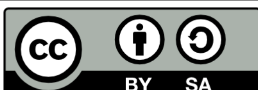
```
ant data-relater -DsourceId=4f23fc5f-ef6c-4089-b733-8fccd8585624
-DtargetId=d38a454f-0ad6-4d07-82e6-57774a1f08fc
```

Now you can use the Alfresco Explorer web client to view the associations. Remember the web-client-config-custom.xml file? It says that any time the property sheet for sc:doc or sc:whitepaper objects is viewed, the sc:relatedDocuments associations should be shown. Alternatively, the Node Browser, available in the Administration Console, is a handy way to view associations.

Searching for Content with Java Web Services

Let's write some code that will show several different examples of Alfresco queries using Lucene. Our code will:

- Authenticate
- Get a reference to the node where to start the search
- Establish a query object using the Lucene query string



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

- Execute the query
- Dump the results

Just like the content creation code, the class will be a runnable Java application that accepts the username, password, and folder name as arguments. The code lives in a try-catch-finally block. The getQueryResults method executes any query so it can be called repeatedly with multiple examples.

If you are following along, you should either run the content creation code a few times or create some content manually so you can test out the queries more thoroughly.

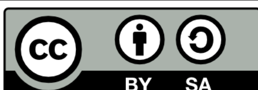
Let's take a look at the generic query execution method first, then the method that calls it for each example query string. First, the code sets up the Query object and then executes the Query using the query() method of the RepositoryService.

```
public List<ContentResult> getQueryResults(String queryString) throws Exception {
    List<ContentResult> results = new ArrayList<ContentResult>();

    Query query = new Query(Constants.QUERY_LANG_LUCENE, queryString );

    // Execute the query
    QueryResult queryResult = getRepositoryService().query(getStoreRef(), query,
false);

    // Display the results
    ResultSet resultSet = queryResult.getResultSet();
    ResultSetRow[] rows = resultSet.getRows();
```

Listing 48: Generic method for executing queries from SomeCoDataQueries.java

Next, the code iterates through the results, extracting property values from the search results and storing them in a helper object called contentResult.

```
    if (rows != null) {
        // Get the infomation from the result set
        for(ResultSetRow row : rows) {
            String nodeId = row.getNode().getId();

            ContentResult contentResult = new ContentResult(nodeId);

            // iterate through the columns of the result set to extract
            // specific named values
            for (NamedValue namedValue : row.getColumns()) {
                if (namedValue.getName().endsWith(Constants.PROP_CREATED) == true)
{
                    contentResult.setCreateDate(namedValue.getValue());
                } else if (namedValue.getName().endsWith(Constants.PROP_NAME) ==
true) {
                    contentResult.setName(namedValue.getValue());
                }
```



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
            }
            results.add(contentResult);
        } //next row
    } // end if
    return results;
}
```

Listing 49: Generic method for executing queries from SomeCoDataQueries.java

Here's the code that calls this method once for each query example.

```
System.out.println(RESULT_SEP);
System.out.println("Finding content of type:" +
    SomeCoModel.TYPE_SC_DOC.toString());
queryString = "+TYPE:\"" +
    Constants.createQNameString(SomeCoModel.NAMESPACE_SOMECO_CONTENT_MODEL,
        SomeCoModel.TYPE_SC_DOC) + "\"";
dumpQueryResults(getQueryResults(queryString));

System.out.println(RESULT_SEP);
System.out.println("Find content in the root folder with text like 'sample'");
queryString = "+PARENT:\"workspace://SpacesStore/" +
    getFolderId() + "\" +TEXT:\"sample\"";
dumpQueryResults(getQueryResults(queryString));

System.out.println(RESULT_SEP);
System.out.println("Find active content");
queryString = "+@sc\\:isActive:true";
dumpQueryResults(getQueryResults(queryString));

System.out.println(RESULT_SEP);
System.out.println("Find active content with a product equal to 'SomePortal'");
queryString = "+@sc\\:isActive:true +@sc\\:product:SomePortal";
dumpQueryResults(getQueryResults(queryString));

System.out.println(RESULT_SEP);
System.out.println("Find content of type sc:whitepaper published between 1/1/2006
and 6/1/2007");
queryString = "+TYPE:\"" +
    Constants.createQNameString(SomeCoModel.NAMESPACE_SOMECO_CONTENT_MODEL,
        SomeCoModel.TYPE_SC_WHITEPAPER) +
    "\"" +
    " +@sc\\:published:[2006\\-01\\-01T00:00:00 TO 2007\\-06\\-01T00:00:00]";
dumpQueryResults(getQueryResults(queryString));
```

Listing 50: Snippet from SomeCoDataQueries.java showing calls to getQueryResults

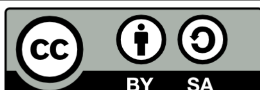
If you want to run this on your own machine, you can use one of the Ant tasks in the build.xml file included in the code that accompanies this article. Just type:

```
ant data-queries
```

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

Your results will vary based on how much content you've created and the values you've set in the content properties, but when I ran my test, the results were as shown below.

```
======================
Finding content of type:doc
----------------------
Result 1:
id=1355e60e-160b-11dc-a66f-bb03ffd77ac6
name=Test Whitepaper (1181339773331)
created=2007-06-08T16:56:13.932-05:00
----------------------
Result 2:
id=bd0bd57d-160c-11dc-a66f-bb03ffd77ac6
name=Test Document 2 (1181340487582)
created=2007-06-08T17:08:08.150-05:00
----------------------
Result 3:
id=1fe9cf04-160b-11dc-a66f-bb03ffd77ac6
name=Test Document (1181339794431)
created=2007-06-08T16:56:35.028-05:00
======================
Find content in the root folder with text like 'sample'
----------------------
Result 1:
id=bd0bd57d-160c-11dc-a66f-bb03ffd77ac6
name=Test Document 2 (1181340487582)
created=2007-06-08T17:08:08.150-05:00
----------------------
Result 2:
id=1fe9cf04-160b-11dc-a66f-bb03ffd77ac6
name=Test Document (1181339794431)
created=2007-06-08T16:56:35.028-05:00
----------------------
Result 3:
id=1355e60e-160b-11dc-a66f-bb03ffd77ac6
name=Test Whitepaper (1181339773331)
created=2007-06-08T16:56:13.932-05:00
======================
Find active content
----------------------
Result 1:
id=bd0bd57d-160c-11dc-a66f-bb03ffd77ac6
name=Test Document 2 (1181340487582)
created=2007-06-08T17:08:08.150-05:00
----------------------
Result 2:
id=1fe9cf04-160b-11dc-a66f-bb03ffd77ac6
name=Test Document (1181339794431)
```



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
created=2007-06-08T16:56:35.028-05:00
----------------------
Result 3:
id=1355e60e-160b-11dc-a66f-bb03ffd77ac6
name=Test Whitepaper (1181339773331)
created=2007-06-08T16:56:13.932-05:00
======================
Find active content with a product equal to 'SomePortal'
======================
Find content of type sc:whitepaper published between 1/1/2006 and 6/1/2007
----------------------
Result 1:
id=1355e60e-160b-11dc-a66f-bb03ffd77ac6
name=Test Whitepaper (1181339773331)
created=2007-06-08T16:56:13.932-05:00
```

Listing 51: Command line results for SomeCoDataQueries.java

There are a couple of other useful tidbits in this class that I've left out of this document such as how to use the ContentService to get the URL for the content and how the UUID for the root folder is retrieved. I encourage you to explore the code that accompanies this guide to see the class in its entirety.

Deleting Content with Java Web Services

Now it is time to clean up after ourselves by deleting content from the repository. Deleting is like searching except that instead of dumping the results, the code creates CMLDelete objects for each result, and then executes the CML to perform the delete.

```
// Create a query object, looking for all items of a particular type
Query query = new Query(Constants.QUERY_LANG_LUCENE, "TYPE:\"" +
    Constants.createQNameString(SomeCoModel.NAMESPACE_SOMECO_CONTENT_MODEL,
        SomeCoModel.TYPE_SC_DOC) "\"");

// Execute the query
QueryResult queryResult = repositoryService.query(storeRef, query, false);

// Get the resultset
ResultSet resultSet = queryResult.getResultSet();
ResultSetRow[] rows = resultSet.getRows();

// if we found some rows, create an array of DeleteCML objects
if (rows != null) {
    System.out.println("Found " + rows.length + " objects to delete.");

    CMLDelete[] deleteCMLArray = new CMLDelete[rows.length];
    for (int index = 0; index < rows.length; index++) {
        ResultSetRow row = rows[index];
```


This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License

```
        deleteCMLArray[index] = new CMLDelete(new Predicate(new Reference[] {new
Reference(storeRef, row.getNode().getId(), null)}, null, null));
    }

    // Construct CML Block
    CML cml = new CML();
    cml.setDelete(deleteCMLArray);

    // Execute CML Block
    UpdateResult[] results =
        WebServiceFactory.getRepositoryService().update(cml);
    dumpUpdateResults(results);

} //end if
```

Listing 52: Code snippet from SomeCoDataCleaner.java

Note that this code deletes every object in the repository of type sc:doc *and its children.* You would definitely want to "think twice and cut once" if you were running this code on a production repository, particularly if you were using a broad content type like cm:content.

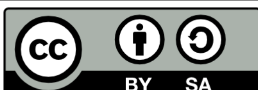
Similar to the other examples, you can run this on your own by executing the following:

```
ant data-cleaner
```

Again, your results will vary based on the content you've created but in my repository, running the code results in the following:

```
Found 2 objects to delete.
Command = delete; Source = b6c3f8b0-12fb-11dc-ab93-3b56af79ba48; Destination = none
Command = delete; Source = d932365a-12fb-11dc-ab93-3b56af79ba48; Destination = none
```

Listing 53: Command line results for SomeCoDataCleaner.java



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported License