ecmarchitect.com

Alfresco Developer: Working with Custom Content Types

June, 2007

Jeff Potts

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Alfresco Developer: Working with Custom Content Types

June, 2007

Jeff Potts

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.

This article discusses how to create your own custom content model, but it doesn't stop there. What good would a custom content model be if you did nothing exciting with the content? Once we get our example model in place, we'll tweak the web client user interface to expose our custom model, then we'll write some code to create, search for, and delete content.

You should already be familiar with general document management and Alfresco web client concepts. If you want to follow along, 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.

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 2.5 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”.

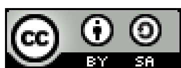
Note that types, properties, constraints, associations, and aspects have names. Names are made unique across the repository by using a namespace specific to the model. The namespace has an abbreviation. So, for example, at Optaros we might define a custom model which declares a namespace with the URI of “http://www.optaros.com/model/content/1.0” and a prefix of “opt”. Any type defined as part of that model would have a name prefixed with “opt:”. We'll talk more about how models are actually defined using XML, 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.

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 2.5 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.

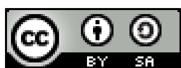
Constraints can be defined once and reused across a model. For example, out-of-the-box, Alfresco makes available a constraint named "cm:filename" that defines a regular expression constraint for file names. If a property in a custom type needs to restrict values to those matching the filename pattern, 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, we might want to store each expense as an individual object. In addition to an Expense Report type we'd also have an Expense type. Using associations we can 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.



This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

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 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.

Aspects

Before we talk about *Aspects*, let's first consider how inheritance works and the implications on our content model. Suppose we are going to use Alfresco 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 our repository is content we want to show in the portal. And, when content is to be displayed in the portal, there are some additional pieces of metadata we need to capture. A simple example might be that we want to know the date and time a piece of content was approved to be shown in the portal.

Using the content modeling concepts discussed so far, we would have only two options. First, we could 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. Second, we could individually define the publish date property only in the content types we knew were going to be published to the portal.

Neither of these are great options. In the first option, we would wind up with 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 we know which content types we want to publish in the portal 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 give us an easy way to encapsulate behavior or business logic we might want to tie to the publish date.

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

Going back to the portal example, we would define a “Portal Displayable” aspect 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.

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

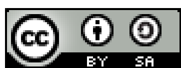
Custom Behavior

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 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. I'll show how to do this in a future article, but you should 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.



This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

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 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*. We haven't talked about exactly how content models are defined yet, 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”.

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

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.

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.
bpmModel.xml	model/bpm/1.0	bpm	d sys cm	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 the two WCM-related model files, the JCR model, and the web client application model. 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,

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

it defines the XML vocabulary Alfresco content model XML files must adhere to.

Custom Model Example

Time for a detailed example. Suppose we work for a company called "SomeCo". SomeCo is a commercial open source company that's 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, we'll 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. Let's assume that after a lot of discussion, SomeCo has elected to go with "straight" Alfresco for this project rather than leveraging Alfresco's WCM features.

The first step is to consider the types and properties we are dealing with. 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.

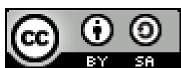
Content that needs to be shown on the web site needs to have a flag that indicates the content is "active" and a date when the content was set to active.

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.

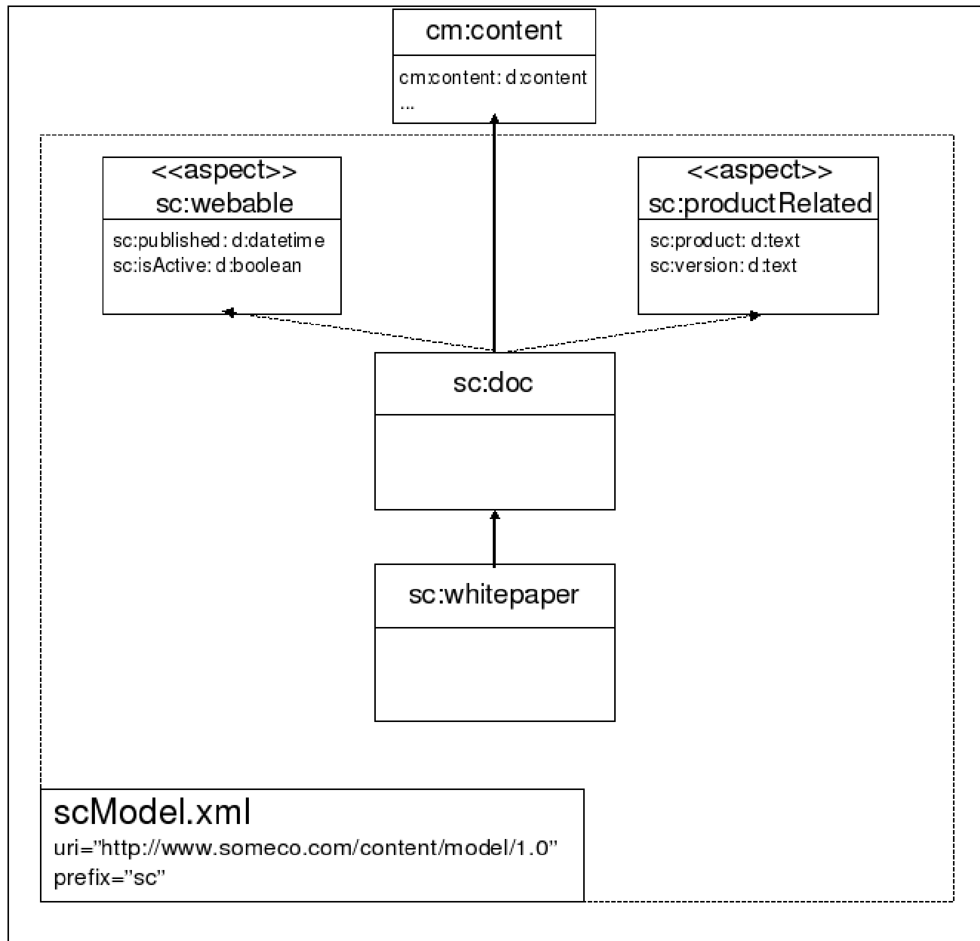
Taking these requirements into consideration, the team comes up with the content model depicted in Drawing 1: SomeCo's initial content model. As the drawing shows, we have defined a common root type called sc:doc with one child, sc:whitepaper. Neither type currently has any properties of their own.

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

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.



This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License



Drawing 1: SomeCo's initial content model

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 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.

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

Implementing and deploying the model

Before we start, a couple of notes about my setup:

- Ubuntu Dapper Drake
- MySQL 4.1
- Tomcat 5.5.x
- Alfresco 2.0.1 Enterprise, WAR-only distribution

Obviously, other operating systems, databases, application servers, and Alfresco versions will work as well.

Okay, let's configure Alfresco to use our new content model. Here are the steps we are going to follow:

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. Ones 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 your Alfresco web client customizations and your 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.

Alfresco loads any file on the classpath that ends with context.xml. All we have to do is create a file that ends with that suffix, and in it, override the bean that refers to our content model. How do we know which bean to extend? Recall from earlier that there is an out-of-the-box content model file called

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

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

We want our model to be registered with the dictionary as well so we'll just extend this bean in our own context file and add our model to the list. Create a file called in the extension directory called someco-model-context.xml and add 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="extension.dictionaryBootstrap" parent="dictionaryModelBootstrap"
depends-on="dictionaryBootstrap">
        <property name="models">
            <list>
                <value>alfresco/extension/scModel.xml</value>
            </list>
        </property>
    </bean>
</beans>
```

Listing 2: someco-model-context.xml

Again, it doesn't matter what the file is called as long as it ends with “context.xml”.

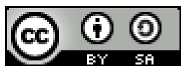
Next, it is time to **create a model file** that implements the content model we defined earlier. Create a new XML file in the extension directory. Make sure the name matches what you specified in the someco-model-context.xml file. In our example, the file should be named scModel.xml.

Referring back to Drawing 1: SomeCo's initial content model, it looks like we'll need two aspects, each with two properties, and two content types. Take a look at the contentModel.xml we found earlier and

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

use it as a reference to build our scModel.xml file. The result should look 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>Optaros</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>
                        </target>
                   </association>
              </associations>
              <mandatory-aspects>
                   <aspect>cm:generalclassifiable</aspect>
              </mandatory-aspects>
         </type>

         <type name="sc:whitepaper">
              <title>Someco Whitepaper</title>
```



This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

```
            <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.

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”.

Exposing the custom model in the web client user interface

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

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

good idea to do just that to make sure the model behaves like you expect. Before we get to that, though, let's talk about what we need to do in order to be able to work with our new model in the Alfresco web client user interface.

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.

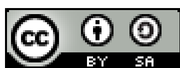
All of these are handled through a file called web-client-config. Looking at the web-client project in the Alfresco source, you can see the out-of-the-box web-client-config.xml file under config/alfresco. The web-client-config.xml file is a proprietary file composed of numerous "config" elements. Each 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, create a web-client-config-custom.xml file in your extension directory with a root element called "alfresco-config". 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.

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:



This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

Drawing 2: Custom properties on the properties sheet

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>
```

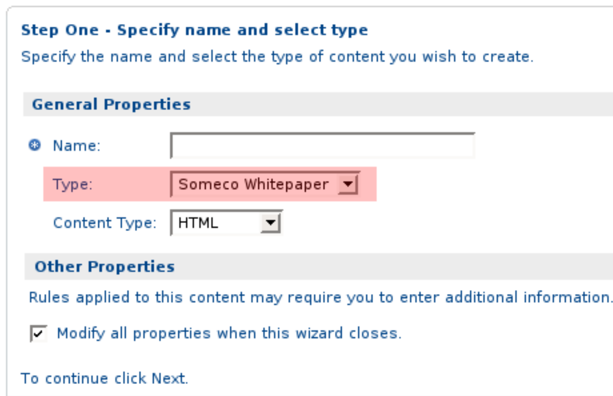
Listing 4: Snippet from web-client-config-custom.xml

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 we'll see where the localized strings reside.

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:

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License



Drawing 3: Custom type in the add content dialog

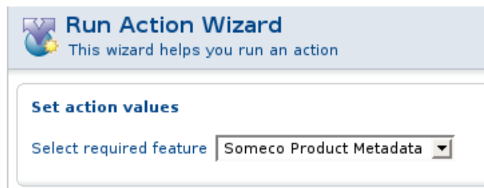
To add content types to the list of available types in the create content and add content dialogs, use the "string-compare" evaluator and the "Content Wizards" condition.

```
<!--  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>
```

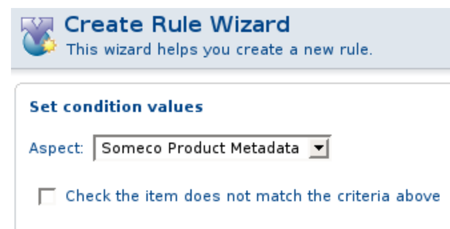
Listing 5: Snippet from web-client-config-custom.xml

"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:



Drawing 5: Custom aspect in "add aspect" dialog



Drawing 4: Custom aspect in "has aspect" dialog

These customizations are all part of the same config element. The "Action Wizards" config has several

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

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 6: 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:

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

Drawing 6: 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>
```

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

Listing 7: Snippet from web-client-config-custom.xml

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. 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, we have four properties we need labels for. 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 8: 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.

Working with content programmatically

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

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

Solution type	Language	Alfresco API
	Language Java/JSP	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 separate Alfresco repository	Java PHP COM/.Net Any language that supports Web Services	Alfresco Web Services API JCR API over JNDI
Server-side Scripts	JavaScript	Alfresco JavaScript API

In this tutorial, we'll focus on using the Alfresco Web Services API with Java and PHP. The Alfresco Wiki and the Alfresco SDK samples are good examples of programming against the Alfresco API. Some of the examples in this document will look similar to the SDK samples, but hopefully provide a more complete end-to-end example.

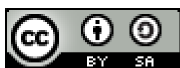
Creating content programmatically with Java

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 we want the content to reside
- Create a series of CML objects that encapsulate the operations we want to execute
- Execute the CML
- Dump the results

Let's look at the Java code that does this first and then the same thing written with PHP.

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.



This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

Also note that all of this code is contained within a try-catch-finally block (we end the session 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:" + getRootFolder();
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 9: 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");

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

Listing 10: Snippet from SomeCoDataCreator.java

Now CML comes into play. I'm not sure what it stands for, but CML objects can be used to execute various content operations. In this case, we'll use 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

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

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 11: Snippet from SomeCoDataCreator.java

To execute the CML we instantiate 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);
Reference docRef = results[0].getDestination();
dumpUpdateResults(results);
```

Listing 12: 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();
```

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

```
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 13: 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 14: Command line results after running SomeCoDataCreator.java

Creating content programmatically with PHP

Java is not a requirement--you could use any language that understands web services to work with the repository. For example, Alfresco includes a PHP API. Setting up PHP and getting it to work with Alfresco is beyond the scope of this article (See “Where to Find More Information”), but for the curious, here's how to do the same thing we just did in Java, in PHP.

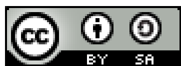
```
<?php
  require_once "Alfresco/Service/Session.php";
  require_once "Alfresco/Service/SpacesStore.php";
  require_once "Alfresco/Service/Node.php";

...snip...

  function createContent($username, $password, $rootFolder, $contentType,
$contentName) {

   // Start and create the session
   $repository = new Repository("http://localhost:8080/alfresco/api");
   $ticket = $repository->authenticate($username, $password);
   $session = $repository->createSession($ticket);

   $store = new Store($session, "SpacesStore");

   $folderPath = "/app:company_home/cm:" . $rootFolder;

   // Grab a reference to the SomeCo folder
   $results = $session->query($store, 'PATH:"' . $folderPath . '"');
   $rootFolderNode = $results[0];
```

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

```
   if ($rootFolderNode == null) {
      echo "Root folder node (" . $folderPath . ") is null<br>";
      exit;
   }

   $timestamp = time();

   $newNode = $rootFolderNode-
>createChild("{http://www.someco.com/model/content/1.0}" . $contentType,
"cm_contains", "{http://www.someco.com/model/content/1.0}" . $contentType . "_" .
$timestamp );

   if ($newNode == null) {
      echo "New node is null<br>";
      exit;
   }

   // Add the two aspects
   $newNode->addAspect("{http://www.someco.com/model/content/1.0}webable");
   $newNode->addAspect("{http://www.someco.com/model/content/1.0}productRelated");

   echo "Aspects added<br>";

   // Set the properties
   $properties = $newNode->getProperties();

   $properties["{http://www.alfresco.org/model/content/1.0}name"] = $contentName .
" (" . $timestamp . ")";
   $properties["{http://www.someco.com/model/content/1.0}isActive"] = "true";
   $properties["{http://www.someco.com/model/content/1.0}published"] = "2007-04-
01T00:00:00.000-05:00";

   $newNode->setProperties($properties);

   echo "Props set<br>";

   $newNode->setContent("cm_content", "text/plain", "UTF-8", "This is a sample " .
$contentType . " document named " . $contentName);

   echo "Content set<br>";

   $session->save();

   echo "Saved changes to " . $newNode->getId() . "<br>";
  }
?>
```

Listing 15: Contents of index.php

Running the PHP script in a web browser produces:



This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

```
Aspects added
Props set
Content set
Saved changes to 5a8dac5e-1314-11dc-ab93-3b56af79ba48
```

Listing 16: Results of running index.php

Creating associations

Now let's switch back to Java and 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,
        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 17: Snippet from SomeCoDataRelater.java

The last line calls a method that queries the associations for a given reference. This should dump the association we 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
```

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

```
bd0bd57d-160c-11dc-a66f-bb03ffd77ac6
{http://www.alfresco.org/model/content/1.0}name:Test Document 2 (1181340487582)
```

Listing 18: Results of running SomeCoDataRelater.java

Now you can use the Alfresco Web Client to view the associations. Remember the web-client-config-custom.xml file? We told it that any time we view the property sheet for sc:doc or sc:whitepaper objects, 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 programmatically

Now that we have some content in the repository we can test out Alfresco's full-text search engine, Lucene. Content in the repository is automatically indexed by Lucene and query strings use the Lucene query syntax to find content based on full-text contents, property values, and content type. (XPath is also a supported search dialect but we won't cover that here).

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 we want to start the search
- Establish a query object using the Lucene query string
- 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. We'll write a generic method to execute the query so we can call it 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 we'll see the method that calls it for each example query string. First, we set up the Query object and execute 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,
```


This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

```
false);

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

Listing 19: Generic method for executing queries from SomeCoDataQueries.java

Next, we iterate 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());
                }
             }
             results.add(contentResult);
        } //next row
   } // end if
   return results;
}
```

Listing 20: 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/" +
   getRootFolderUuid() + "\" +TEXT:\"sample\"";
```

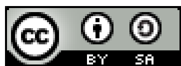

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

```
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 21: Snippet from SomeCoDataQueries.java showing calls to getQueryResults

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
```



This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

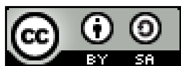
```
----------------------
Result 2:
id=1fe9cf04-160b-11dc-a66f-bb03ffd77ac6
name=Test Document (1181339794431)
created=2007-06-08T16:56:35.028-05:00
----------------------
Result 3:
id=1355e60e-160b-11dc-a66f-bb03ffd77ac6
name=Test Whitepaper (1181339773331)
created=2007-06-08T16:56:13.932-05:00
======================
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)
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 22: 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 programmatically

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, we'll create CMLDelete objects for each result, and then we'll execute the CML to perform the delete.



This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

```
// 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];
     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 23: 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.

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 24: Command line results for SomeCoDataCleaner.java

This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License

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 client user interface, and how to work with content via the web services API using both Java and PHP. I've thrown in a few recommendations that will hopefully save you some time or at least spark some discussion.

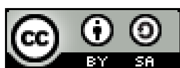
There's plenty of additional data model-related customizations to cover in future articles including custom behavior, custom metadata extractors, custom transformers, and custom data renderers.

Where to find more information

- The complete source code for these examples is available here from ecmarchitect.com.
- The Alfresco SDK comes with a samples for working with web services, the JCR, and the foundation API.
- The Search Documentation on the Alfresco wiki provides query examples using both Lucene and XPath.
- See "Using Alfresco's PHP API" on ecmarchitect.com for pointers on getting Alfresco working from PHP.
- For deployment help, see the Client Configuration Guide and 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, see the ecmarchitect.com tutorial on Custom Actions.

About the Author

Jeff Potts is the Enterprise Content Management Practice Lead at Optaros, a leading Open Source and Next Generation Internet consultancy. Jeff has nearly fifteen years of experience implementing content management, collaboration, and other knowledge management technologies for a variety of Fortune 500 companies. Jeff lives in Dallas, Texas with his wife and two kids. Read more at ecmarchitect.com.



This work is licensed under the Creative Commons Attribution-Share Alike 2.5 License