Template Guide

IMPORTANT: This document details the Templating API and Templating Services for Alfresco 2.1 - if you are looking for details on Alfresco 2.0 or 1.4 then please see this document: Template Guide For Alfresco 1.4 and 2.0. Note that the old page will no longer be maintained or updated except to correct any mistakes.

Contents [show]

Introduction

A template is a document that can be applied to an object or objects (e.g. one or more documents from the repository) to produce another document. An example is a FreeMarker or XSLT template file.

The template is written in a specific templating language and the data model consists of objects which are available to the template. Generally the objects would consist of say the current document or folder.

Template + data model = output

Think of the template engine as a mechanism for rendering an output based on a template page and a data model of objects accessible by the template page.

The Alfresco pluggable template system allows for multiple template engines to be used within Alfresco. They are made available through a repository service API (TemplateService), a dedicated servlet for returning templated output (TemplateContentServlet) and via a web-client JSF component (UITemplate) and associated JSP pages.

The core repository TemplateService service is designed to allow the addition of multiple template engines. This is accomplished via Spring XML configuration and the implementation of a simple interface by the developer. Since Alfresco 2.1, languages such as PHP can be used as both a templating and/or scripting language given the appropriate implementation and configuration.

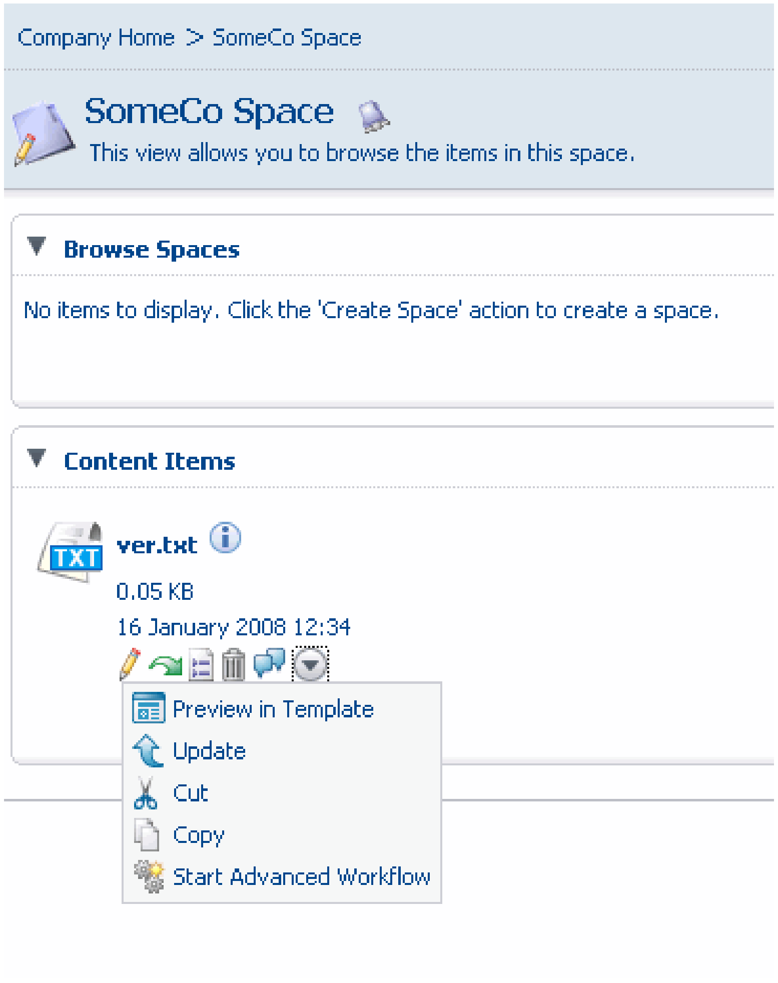
Templates consist of a simple template text file that can be stored either on the ClassPath or in a Repository. The template files are executed against the default or a custom Data Model and the output is directed to an output stream such as an Alfresco document or servlet output or to a web-client page via a JSF UI component.

A JSP page can contain any number of UI Template components, and therefore display any number of different templates and associated models.

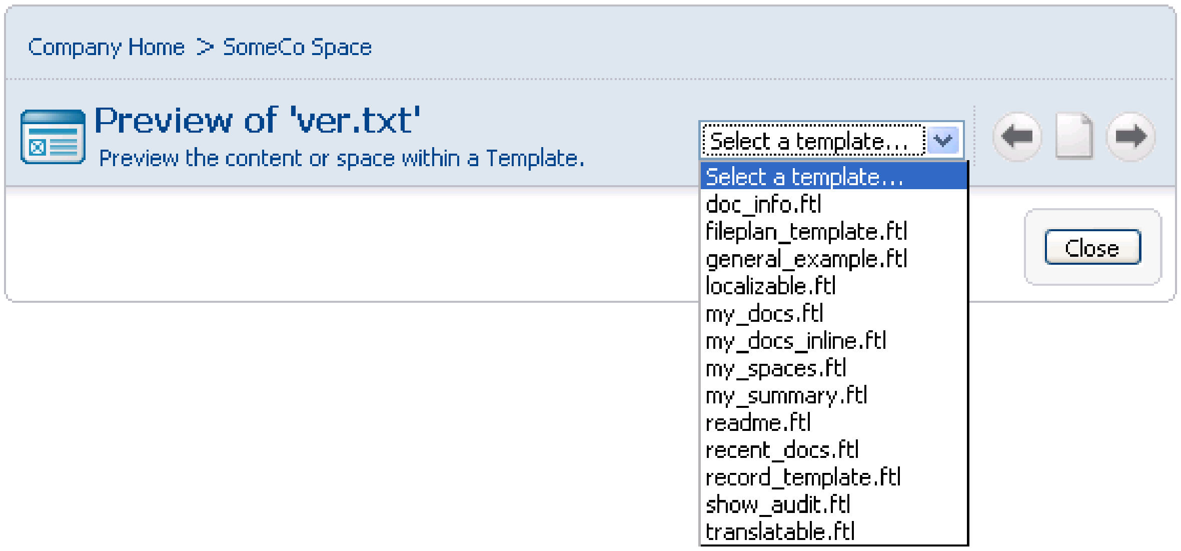
Viewing Templates in the Web-Client

Templates can be applied to Documents and Space objects by following these steps:

1. Create a template file (a good start is to copy-and-paste an example from the Examples section below or copy one from the /web-client/config/templates directory in the Alfresco source code distribution).
2. Add the template file you created to the "Company Home/Data Dictionary/Presentation Templates" folder in your repository. Several examples are already provided here as part of the default installed system. The examples make good starting points for understanding how a template is written.
3. Click the "Preview In Template" action next to a Document or a Space in the main browsing screen.


Company Home > SomeCo Space
SomeCo Space
This view allows you to browse the items in this space.
Browse Spaces
No items to display. Click the 'Create Space' action to create a space.
Content Items
ver.txt
0.05 KB
16 January 2008 12:34
Preview in Template
Update
Cut
Copy
Start Advanced Workflow

1. The screen that appears has a drop-down menu in the top right of the screen. This shows a list of the available templates found the Content Templates folder as above. When you select one it is applied to the current document.



Custom Template Views

It is possible to apply a template as a "custom view" for a Space in the web-client UI. A Custom View can be set up using the *View Details* action screen for a Space. So select the template you wish to be used as a Custom View from the View Details page. This enables the *Custom View* view mode in the main browsing screen for that Space.

Template Framework Architecture

Template Processing Servlet

A special servlet is provided that renders the output of a template+model directly to the response. The guide to URL Addressability contains examples on how to access and define URLs for the servlet. This is also a great way to test your templates before making them public. Templates placed anywhere in the repository can be accessed by the servlet - you only need to create the URL once and keep reloading the page to test your template output.

Template Files

Template files can be stored either on the class path (for example in *alfresco/config/templates*) or in the repository store. The location of the template file is a mandatory attribute to the Template JSF component and also mandatory parameter on the template servlet URL.

Each file will be specific to a particular template engine. The Template Service will pick the appropriate engine based on the file extension. By default FreeMarker template files will be used. FreeMarker documentation can be found here:

- FreeMarker template language reference
- FreeMarker page designer manual.

The template engine is not tied to any output file format, templates can output entire HTML files, snippets of HTML, XML or any other format as desired. The template servlet mentioned above supports a URL parameter to change the MIME type of the response stream as required.

Examples showing FreeMarker templating language and how to bind against the Alfresco data model are shown below.

Template Models

A model is an object or a hierarchy of Java Bean objects from which a template file retrieves values that can be used for output. The model is like the API for the template page - it provides the top-level objects from which properties and values can be accessed. Alfresco provides a default model provided by the JSF Template component and template servlet. It is also possible to define a custom model which will be merged into the default model to provide additional model objects.

Default Model

The default model provides a number of common root objects useful for most templates. Most of the objects are known as "TemplateNode" objects and wrap the notion of an Alfresco "Node" (such as a folder or document in the repository). This provides a rich OO layer to make it easy to display common Alfresco concepts such as node properties, aspect values and content.

If you are accessing a template through the basic web-client UITemplate component (this is the common case when developers use this component directly), then the following named objects are provided by default at the root of the model:

companyhome	The Company Home template node.
userhome	Current users Home Space template node.
person	Node representing the current users Person object.

If you are accessing a template through the web-client UI using a Custom View for a space or the Space Preview action or via URL using the Template Servlet then the following named object is also provided:

space	The current space template node.

If you are accessing a template through the Custom View for a document or the Document Preview action or via URL using the Template Servlet then the following named objects are also provided:

document	The current document template node.
template	The node representing the template itself.

These "TemplateNode" objects have their own API for accessing common Alfresco concepts such as properties, aspects, associations and content as detailed in the TemplateNode Model API section below.

If you are accessing a template via the Template Servlet then the following special object is also available:

args	A Map of any URL parameters passed via the Template Content Servlet. This is a neat way to pass additional parameters to your template. FreeMarker has built-in method to parse integers and check for the existence of values which can be used to make your template even more interactive and powerful.

```
<#assign keys = args?keys>
<#list keys as arg>
   ${arg}
</#list>
```

Various other root objects are also always available providing a full set of rich functional objects to the template developer:

session	Session related information. Provides a single value **session.ticket** for the current authentication ticket. This is useful when generating some Alfresco URLs for accessing outside of the web-client.
classification	Read access to classifications and root categories, see the Classification section.
url	Provides a single property **url.context** which can be used to retrieve the Alfresco web-server context path, such as `/alfresco` - useful when generating URL links to objects. Note: Not available when using the Template as a Custom View on a space.
workflow	Read access to information on workflow objects and the currently executing workflows for a user, see the Workflow section.
avm	Read access to Alfresco WCM store objects (known as the AVM). It is possible to access the user details, forms, files and folders of a named WCM web-project using this API. This means it is possible to build templates that display content from files in the staging area or sandbox of a web project based website. See the AVM section.

The various default model objects can be accessed directly from the root of a model in your template, for example to display the *name* property of the *userhome* object:

```
userhome.properties.name
```

The Alfresco node model is built dynamically as you access it, therefore you can write statements such as:

```
userhome.children[1].children[0].children[2].name
```

Important Note: It should be noted that the FreeMarker template engine is very strict on the access of empty or *null*values! Unlike many templating or scripting languages, that display empty values or assume FALSE as a default value for a missing property, the FreeMarker engine will instead throw an exception and abort the rendering of the template. To help you build stable templates, most of the *TemplateNode* API calls provided by the default model that return Maps or Sequences (lists) of items will return empty Maps and Sequences instead of null. Also if a null value may be returned by a call (e.g. from accessing a Map to find a value by name), you should use the FreeMarker built-in *exists* method to check for null first, thus:

```
<#if mynode?exists>
  <#if mynode.assocs["cm:translations"]?exists>
     ${mynode.assocs["cm:translations"][0].content}
  </#if>
</#if>
```

This checks for the existence of *mynode*, then checks for the existence of a *translation* association before attempting to access the translation.

TemplateNode Model API

These objects, and any subsequent child node objects provide access to the common Alfresco concepts such as properties, aspects and associations. The following template API is provided:

properties

A map of the properties for the node, such as **userhome.properties.name**. Properties may return several different types of objects. This depends entirely on the underlying property type in the repository. If the property is multi-value, the result will be a sequence which can be indexed like any other sequence or array. If the result is an unknown or unsupported type, the toString() result is generally used. Therefore, the result will mostly be a String type. If the property can potentially contain a 'null' value, take care when accessing it and use the *exists* FreeMarker built-in method to check for null values before accessing.

Date and Boolean property values should be handled carefully. Use of the FreeMarker built-in methods *is_date* and*is_boolean* can be used to check if the page developer is unsure of the property value type. These values can then be formatted as appropriate.

Refer to the FreeMarker Template Cookbook for examples.

If the type of the property is a NodeRef object (d:noderef in the content model), the template model will automatically convert the property type into another TemplateNode object. This means the page writer can continue to dynamically walk the object hierarchy for that node. For example, if a document node has a NodeRef property called "locale", you could execute the following to retrieve the name of the node the property references

```
Locale ${document.properties.locale.properties.name}
```

children

A sequence (list) of the child nodes, such as **mynode.children[0]**.

assocs

A map of the target associations for the node. Each entry in the map contains a sequence of the Node objects on the end of the association. For example, **mynode.assocs["cm:translations"][0]**.

childAssocs

A map of the child associations for the node. Each entry in the map contains a sequence of the Node objects on the end of the child association. For example, **myforumnode.childAssocs["fm:discussion"][0]**.

aspects

A sequence of the aspects (as QName strings) applied to the node.

hasAspect(String aspectName)

A function that returns *true* if a node has the specified aspect.

For example,

```
 <#if userhome.hasAspect("cm:templatable")>...</#if>
```

isContainer

True if the node is a folder node; otherwise, false.

isDocument

True if the node is a content node; otherwise, false.

isCategory

True if the node is a category node; otherwise, false.

content

Returns the content of the node as a string.

url

The url to the content stream for this node.

downloadUrl

The url to the content stream for this node as an HTTP1.1 attachment object.

displayPath

The display path to this node. Take care when accessing the display path if the current user does not have permissions to view all of it. A fix has been made in Alfresco 2.1.1E so that the display path can always be obtained safely for any user.

icon16

The small icon image for this node.

icon32

The large icon image for this node.

icon64

The extra large icon image for this node.

mimetype

The mimetype encoding for content attached to this node.

size

The size in bytes of content attached to this node.

isLocked

True if the node is locked; otherwise, false.

id

GUID for the node.

nodeRef

NodeRef string for the node.

name

Shortcut access to the *name* property.

type

Fully qualified QName type of the node.

typeShort

Prefix string, or "short" QName type of the node. **(v3.2 or newer)**

parent

The parent node can be null if this is the root node. Take care when accessing the parent node if the current user does not have permissions to view it. A fix has been made in Alfresco 2.1.1E so that the parent node can always be obtained to allow a hasPermission() check to be applied before accessing it in a template.

permissions

Sequence of the permissions explicitly applied to this node. Strings returned are of the format ***[ALLOWED|DENIED];[USERNAME|GROUPNAME];PERMISSION'***. So for example, **ALLOWED;kevinr;Consumer** can be easily tokenized on the ';' character.

inheritsPermissions

True if the node inherits its parent node permissions, false if the permissions are applied specifically.

hasPermission(permission)

A function that returns true if the current user has the specified permission on the node. For example,

```
<#if userhome.hasPermission("Write")>...</#if>
```

childrenByXPath

Returns a map capable of executing an Xpath query to find child nodes, such as**companyhome.childrenByXPath["*[@cm:name='Data Dictionary']/*"]**. The map executes an XPath search against the current node and returns a sequence of the nodes as results of the query.

childByNamePath

Returns a map capable of returning a single child node found by name path, such as**companyhome.childByNamePath["Data Dictionary/Content Templates"]**. Under the covers, this method is building a XPath and executing a search against the "cm:name" attribute on children of the current node. This method allows you to find a specific child node if you know its name.

Note that the previous API calls use the node they are executed against as the current context for the query. For example, if you have a folder node called "myfolder" and you execute the call**myfolder.childByNamePath["MyChildFolder"]**, the search tries to find a folder called "MyChildFolder" as the child of the myfolder node.

childrenBySavedSearch

Returns a map capable of executing a search based on a previously Saved Search object. It returns a sequence of child nodes which represent the objects from the results of the search. For example,

```
companyhome.childrenBySavedSearch["workspace://SpacesStore/92005879-996a-11da-bfbc-f7140598adfe"]
```

The value specified must be a NodeRef to an existing Saved Search object.

childrenByLuceneSearch

Returns a map capable of executing a search *against the entire repository* based on a Lucene search string. It returns a sequence of nodes which represent the objects from the results of the search, such as**companyhome.childrenByLuceneSearch["TEXT:alfresco* AND TEXT:tutorial*"]**. The value can be any valid Lucene search string supported by Alfresco. Note that you may need to escape Lucene specific characters. Note that the entire repository is searched: the current node is only used as an access point to retrieve the search object.

Refer to the FreeMarker Template Cookbook for examples.

nodeByReference

Returns a map capable of executing a search for a single node by NodeRef reference. This method allows you to find a node if you have the full NodeRef string or NodeRef object. Refer to the FreeMarker Template Cookbook for examples.

Advanced TemplateNode API

The following values are available but are only required for special use cases - you can safely ignore these if you don't know what they mean.

qnamePath	QName based Path to the node. This is useful for building Lucene PATH: style queries that constrain to a path location.
primaryParentAssoc	**ChildAssociationRef** instance for the node.
auditTrail	return a sequence of **AuditInfo** objects representing the Audit Trail for a node. This is only available if auditing is active for the repository.

Version History

Meta-data and content for previous version of a versioned document node can be obtained through the version history API.

versionHistory	returns a sequence of Version History record objects for a versioned TemplateNode.

Each Version History record object has the following API:

id	GUID for the node.
nodeRef	NodeRef string for the node.
name	*name* property of the node version record.
type	fully qualified QName type of the node.
createdDate	created date of the version.
creator	creator of the version.
versionLabel	version label of the version record.
isMajorVersion	boolean true if this was a major version.
description	version history description.
url	url to the content stream for the frozen content state.

In addition the **properties** and **aspect** APIs as described above for TemplateNode are available which return the frozen history state of the properties and aspects for the node version record.

For examples, please see the FreeMarker Template Cookbook.

Classification

The **classification** root object provides read access to classifications and root categories.

getRootCategories(String aspectQName)	function to get a sequence of root categories for a classification.
getAllCategoryNodes(String aspectQName)	function to get a sequence of the category nodes for a classification.
allClassificationAspects	return a sequence of QName objects of all classification aspects.

The following extended node API methods are provided to work with Category node objects.

categoryMembers	get all members of a category.
subCategories	get all sub-categories of a category.
membersAndSubCategories	get all sub-categories and members of a category.
immediateCategoryMembers	get all immediate members of a category.
immediateSubCategories	get all immediate sub-categories of a category.
immediateMembersAndSubCategories	get all immediate sub-categories of a category.

For examples, please see the FreeMarker Template Cookbook.

XML Content Processing Model API

The FreeMarker language supports XML DOM processing using either DOM functional or macro style declarative operations. The Alfresco TemplateNode API has been extended to provide access to the FreeMarker DOM model objects.

xmlNodeModel

returns the XML DOM model object for the content of the node. If the node content is valid XML, and the XML can be parsed ok, then this method returns the root of the DOM for this node. The DOM can be walked and processed using the syntax as per the FreeMarker XML Processing Guide. Also see the Examples section below.

For examples, please see the FreeMarker Template Cookbook.

Workflow

The **workflow** root object provides read access to the in-progress and finished tasks for the current user. It also provides a function to lookup a single task by its task ID. The functions described mostly return **WorkflowTaskItem**objects (see below).

assignedTasks	the sequence WorkflowTaskItem objects representing the assigned tasks for the current user.
pooledTasks	the sequence WorkflowTaskItem objects representing the pooled tasks for the current user.
completedTasks	the sequence WorkflowTaskItem objects representing the pooled tasks for the current user.
getTaskById(taskId)	function to return a single task given a known task ID.

For examples, please see the FreeMarker Template Cookbook.

WorkflowTaskItem API

type	Workflow task type value.
qnameType	Underlying QName model type of the workflow task.
name	Task name value.
description	Task description value.
id	Task id
isCompleted	Boolean value true if the task has been completed.
startDate	Start Date of the workflow task.
transitions	Returns a Map of the available task transition names to transition Labels and Ids.
initiator	Returns a **TemplateNode** representing the user who initiated the workflow task.
outcome	The outcome label from a completed task.
package	Returns the NodeRef to the workflow package node.
packageResources	Returns a sequence of the node resources from the package attached to this workflow task.
properties	A map of all the properties for the task, includes all appropriate BPM model properties.

AVM

Warning: AVM Deprecation

The AVM is no longer being actively developed by Alfresco Engineering and Enterprise support subscriptions for the AVM are no longer being offered to new customers. New projects requiring Web Content Management features may want to consider leveraging a CMIS-based solution such as Web Quick Start or the File System Transfer Receiver. The topic AVM Decommissioning collects useful information for migrating off of the AVM. The AVM was removed from the product in version 5.0.

The Alfresco Versioning Machine (AVM) API provides access to Web Content Management (WCM) stores and their associated file and folder nodes. A WCM project is divided into "stores" such as the Staging Store and various User Sandbox stores and the child nodes of these stores. The store contents has a well defined structure. Developers should read the associated wiki pages to get familiar with the structure and the naming convention used for staging and user stores.

avm.stores	returns all store objects in the AVM
avm.lookupStore(storeid)	function to returns the store object for the specified store id
avm.lookupStoreRoot(storeid)	function to return the root node for the specified store
avm.lookupNode(path)	function to return a single avm node given the full path including store to the node
avm.webappsFolderPath	returns the well known root path to the avm webapp folder for a store
avm.getModifiedItems(storeId, username, webapp)	function to return a sequence of node representing the modified items for the specified user sandbox store for a specific webapp in the store
avm.stagingStore(storeId)	function to return staging store name for a specific store Id.
avm.userSandboxStore(storeId, username)	function to return the user sandbox store name for a specific store Id and user.
	function to return the preview url to the staging store for the specified store

avm.websiteStagingUrl(storeId)	ID.
avm.websiteUserSandboxUrl(storeId, username)	function to return the preview URL to the user sandbox for the specified store ID and username.
avm.assetUrl(path)	function to return the preview URL to the specified fully qualified avm path asset.
avm.assetUrl(storeId, path)	function to return the preview URL to the specified store asset path (asset path is store relative)

AVM Store API

The Store objects returned by the methods above have the following additional API:

store.id	internal ID of the store.
store.name	store name.
store.creator	user who created the store.
store.createdDate	creation date of the store.
store.lookupRoot	returns store root node.
store.lookupNode(path)	function to return a node within the store given the store relative path (relative to avm webapp folder root).
store.luceneSearch(query)	function to execute a Lucene search against the store and return a sequence of nodes as the result.

AVM Node API

The AVM specific node objects returned by the methods above have the following API:

node.version	Version of the node, generally this will be -1 == HEAD revision.
node.path	Fully qualified AVM path to the node
node.parentPath	Fully qualified AVM path to the parent of this node.
node.name	Node name property.
node.type	The model QName type of the node.
node.id	The GUID for the node.
node.nodeRef	The Alfresco NodeRef for the node.
node.isDeleted	True if the node is within a layer and has been deleted.
node.isContainer	True if the node is a folder.
node.isDocument	True if the node is a document.
node.isLocked	Returns true if the node is currently locked.
node.isLockOwner	Returns true if the node is locked and current user is the lock owner.
node.hasLockAccess	Returns true if this user can perform operations on the node when locked. This is true if the item is either unlocked, or locked and the current user is the lock owner, or locked and the current user has Content Manager role in the associated web project.
node.parent	Reference to the parent AVM node.

AVM nodes also support the standard properties, aspects, children, content and permissions API as per theTemplateNode Model API as above.

Default Model Methods

Custom template methods can be added to the FreeMarker language for use on template pages. The default model provides the following additional methods:

hasAspect(TemplateNode, String) - will return the integer value 1 if the TemplateNode supplied has the aspect with the supplied QName String, else the integer value 0 will be returned.

dateCompare(DateA, DateB) - Compare two dates to see if they differ. Returns 1 if DateA if greater than DateB, else returns 0.

dateCompare(DateA, DateB, Number) - Compare two dates to see if they differ by the specified number of milliseconds. Returns 1 if DateA is greater than DateB by at least the Number value in milliseconds, else returns 0.

dateCompare(dateA, dateB, millis, test) - Same as above, but the 'test' variable is one of the following strings ">", "<", "==" - greater than, less than or equal - as the test to perform.

incrementDate(date, millis) - Increment a date by the specified number of milliseconds, returns the new Date.

message(String) - will return the I18N message string (resolved for current user Locale setting) for the specified String message ID.

cropContent(TemplateNode, length) - returns the first N characters from the content stream for the specified node.

'shortQName(String) - *returns the short, or prefix, version of a long QName.*

More information on adding your own custom methods: [FreeMarker Custom Methods]. It should be noted that the return values for all custom methods are rather limited. It is only possible to return String, number or date object. This is why the custom method described above does not return a boolean value as you might expect. You must also extend the Template Service Configuration to include your custom template method.

Current date

As Freemarker is a templating language there is no such variable as "today". So the current date (as a new Date() Java object) can be accessed in a template as the "date" object in the root of the model. For example:

```
<#assign datetimeformat="EEE, dd MMM yyyy HH:mm:ss zzz">
${date?string(datetimeformat)}
```

JSP Page

Minimum JSP code required to display a template using the JSF Template component:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="/WEB-INF/repo.tld" prefix="r" %>

<html>
 <body>

 <f:view>
   <h:form>
     <r:template template="alfresco/templates/userhome_docs.ftl" />
   </h:form>
 </f:view>

 </body>
</html>
```

Examples

For examples, please see the FreeMarker Template Cookbook.

Template Infrastructure Design and Custom Models

How is Templating integrated into Alfresco?

The repository service providing templating features is called the Template Service. It is a typical Alfresco repository service accessed via a Spring managed bean with the name of **TemplateService**. Only repository developers will be interested in accessing the **TemplateService** directly. Those more interested in writing scripts themselves should skip this section and jump to TemplateNode Model API.

Template Service Configuration

The available template engines can be configured in the Alfresco Spring config file *template-services-context.xml*. You should download the Alfresco SDK to examine the structure of this file.

In Alfresco 2.1 the mechanism for adding additional templating engines has been greatly improved. It is now possible to completely replace or augment the existing template implemenation with others such as PHP or Ruby. The correct templating implementation will be automatically selected by the TemplateService when executing a template based on the file extension of the template being executed. If it cannot resolve the engine to use, then the developer can specify it explicitly when calling the TemplateService.

The **baseTemplateProcessor** bean is the base definition for all template engine implementations. By default theFreeMarker Template Engine is available and will be used as the default engine during calls to the template services.

Further engines can be added by developers. A Java based PHP engine has been integrated and is a good example of this, it is available as an AMP download for addition into an existing Alfresco 2.1 installation.

Custom Root Objects

By default a number of special root objects are provided in the default Alfresco model, these are fully detailed as above in the API sections. However it is possible to configure in your own additional root objects to make them available to all templates. In the *template-services-context.xml* file there is a base bean definition which should be extended by all custom root objects:

```
<bean id="baseTemplateImplementation" abstract="true" init-method="register">
    <property name="processor">
        <ref bean="freeMarkerProcessor"/>
    </property>
</bean>
```

Your custom root bean classes must extend the org.alfresco.repo.template.BaseTemplateProcessorExtension class. An example of extending this would be the **Workflow** root object as configured in the same file:

```
<bean id="workflowTemplateExtension" parent="baseTemplateImplementation" class="org.alfresco.repo.template.Workflow">
    <property name="extensionName">
        <value>workflow</value>
    </property>
    <property name="serviceRegistry">
        <ref bean="ServiceRegistry"/>
    </property>
</bean>
```

The usual Spring config is available to set various services as required into your POJO bean class definition. You can add addition bean implementations by extending the config in the usual way.

Custom Models

Developers can create custom models to be bound into the JSF Template component for use by templates. Custom models should be provided as a **java.util.Map** implementation and will be automatically merged into the default model. This means all root objects in your custom model will be accessable at the same level as the default model root objects.

Custom models can contain any object valid for your particular choice of template engine. By default the FreeMarker supports most standard Java types and Collections, see the FreeMarker data model reference for more info.

If you wish to expose Node objects in your model, it is recommended that you follow the same pattern as used by the default model and simply add NodeRef instances to your model, as these objects automatically get converted to the appropriate template model types by the various engines to provide the API described above. Any other objects can be added to your custom model as you see fit - they will not get converted by the template engine if they are not recognised.

To use a custom model and use it with the UI Template component, create your Map in a JSF Bean and bind a method returning it in the normal way to your template component on a page:

```
<r:template template="alfresco/templates/example.ftl" model="#{MyBean.templateModel}" />
```

Example Bean code to return a model:

```
   /**
    * Returns a model for use by a template on the Document Details page.
    *
    * @return model containing current document and current space info.
    */
   public Map getTemplateModel()
   {
      Map<String, Object> model = new HashMap<String, Object>(4, 1.0f);

      model.put("document", getDocument().getNodeRef());
      model.put("space", this.navigator.getCurrentNode().getNodeRef());
      model.put(TemplateService.KEY_IMAGE_RESOLVER, imageResolver);

      return model;
   }
```

As mentioned, the model will be merged into the default model.