Customizing Explorer and Share

This section of the documentation provides guidance and reference for customizing and extending Alfresco Explorer and Share.

Explorer is the older interface, but is still useful for carrying out many tasks and will be familiar to you if you've used earlier versions of Alfresco. Share is the newer interface, and has had a number of features added in this release to enhance extensibility. The look and feel of the Share UI has also been refreshed for this release.

- **Customizing Alfresco Share** [1] Alfresco Share provides a rich web-based collaboration environment for managing documents, wiki content, blogs and more. Share leverages the Alfresco repository to provide content services and uses the Alfresco Surf platform to provide the underlying presentation framework.
- **Customizing Alfresco Explorer** [2] Alfresco Explorer provides a web-based user interface providing document management, collaboration, and administration capabilities.

Parent topic: For Developers [3]

Customizing Alfresco Share

Alfresco Share provides a rich web-based collaboration environment for managing documents, wiki content, blogs and more. Share leverages the Alfresco repository to provide content services and uses the Alfresco Surf platform to provide the underlying presentation framework.

A number of options are available to developers and administrators for customizing Alfresco Share to better fit into their environment. Many of these mechanisms are provided by the underlying Surf framework, therefore a knowledge of Surf is considered useful for anyone wishing to implement substantial customizations.

This section of the documentation looks at customizing Share through configuration files. Customizing Share through more advanced techniques in discussed in the Developing Share Extensions [4] section.

- **Share configuration files** [5] Share can be configured through configuration files. This topic looks at the key configuration files available.
- **Customizing Share with the share-config-custom.xml file** [6] This task describes how to modify the Share custom configuration file.
- **Configuring the Share default port** [7] This section describes how to configure the default port configuration for Alfresco Share.
- **Security policies and filters in Alfresco Share** [8] You can configure a number of policies and filters in Alfresco Share to mitigate security attacks.
- **Configuring Share for mixed user name types** [9] When there is a mix of user name types, for example, some using the @domain in their user name, this may have an impact on the use of Share.
- **Share Document Library** [10] The Share repository document library is a feature that gives full access to the Alfresco repository.
- **Share themes** [11] When you run Alfresco Share, the look and feel is set by a default theme. This section describes how to select one of the alternative themes available in Share, and also how to create and use your own themes for corporate branding.
- **Forms** [12]Alfresco Share presents data view and entry forms throughout its user interface, which are built on the Surf framework. This framework provides a convention for implementing forms.

Parent topic: Customizing Explorer and Share [13]
Related concepts
Developing Share Extensions [4]

Share configuration files

Share can be configured through configuration files. This topic looks at the key configuration files available.

The main Share configuration file is share-config.xml. This can be found in a default installation at tomcat/webapps/share/WEB-INF/classes/alfresco/share-config.xml. While it is possible to change configuration through direct changes to this file this is not recommended as any customizations will be lost if the Share WAR is re-exploded, or you install a new version of Alfresco. To get around this issue it is advisable to make configuration changes to a file outside of the Share WAR. This can be done through the file share-config-custom.xml, which can be found at tomcat/shared/classes/alfresco/web-extension/share-config-custom.xml in the default Alfresco installation. Any changes made here will be applied, and can be saved between reinstalls and the Share WAR exploding.

It should also be noted that it is possible to package a share-config-custom.xml file in a JAR or AMP. In this way you can have multiple share-config-custom.xml files packaged in JARs or AMPs if necessary. JARs will be loaded from the classpath, for example ./tomcat/shared/lib. AMPs will be applied to the Share WAR file.

CAUTION:
The order in which multiple share-config-custom.xml files are applied is not guaranteed in the case where multiple files override the same section of configuration.

Another key Share configuration file is slingshot-application-context.xml which can be found at tomcat/webapps/share/WEB-INF/classes/alfresco/slingshot-application-context.xml in the default Alfresco installation. This loads a number of other configuration files:

```
<!-- Spring Web Scripts -->
<value>classpath:org/springframework/extensions/webscripts/spring-webscripts-config.xml</value>
<value>classpath:META-INF/spring-webscripts-config-custom.xml</value>
<value>jar:*!/META-INF/spring-webscripts-config-custom.xml</value>

<!-- Spring Surf -->
<value>classpath:org/springframework/extensions/surf/spring-surf-config.xml</value>
<value>classpath:org/springframework/extensions/surf/spring-surf-config-remote.xml</value>
<value>classpath:META-INF/spring-surf-config-custom.xml</value>
<value>jar:*!/META-INF/spring-surf-config-custom.xml</value>

<!-- Surf Autowire Support -->
<value>webapp:WEB-INF/surf.xml</value>

<!-- Common form config -->
<value>classpath:alfresco/form-config.xml</value>

<!-- Share default config -->
<value>classpath:alfresco/share-config.xml</value>

<!-- Share help url config -->
<value>classpath:alfresco/share-help-config.xml</value>

<!-- Share form config -->
<value>classpath:alfresco/share-form-config.xml</value>

<!-- Share Document Library config -->
<value>classpath:alfresco/share-documentlibrary-config.xml</value>

<!-- Share Data List form config -->
<value>classpath:alfresco/share-datalist-form-config.xml</value>

<!-- Share workflow form config -->
<value>classpath:alfresco/share-workflow-form-config.xml</value>

<!-- Share CMIS config -->
<value>classpath:alfresco/share-cmis-config.xml</value>

<!-- Share Security config -->
<value>classpath:alfresco/share-security-config.xml</value>

<!-- Share custom config -->
<value>classpath:alfresco/web-extension/share-config-custom.xml</value>
<value>jar:*!/META-INF/share-config-custom.xml</value>
<value>classpath:alfresco/web-extension/share-config-custom-dev.xml</value>
<value>jar:*!/META-INF/share-config-custom-dev.xml</value>
```

Note that the custom configuration files are loaded last, so that they can override existing configuration.
Parent topic: Customizing Alfresco Share [1]

Customizing Share with the share-config-custom.xml file

This task describes how to modify the Share custom configuration file.

To configure the Share application, you can use the custom configuration file named share-config-custom.xml.

1. Open the following file:
   tomcat/shared/classes/alfresco/web-extension/share-config-custom.xml
2. Uncomment any <config> items that you want to enable.
3. Add any <config> items that you want to include.
4. Save the edited file.
5. Restart Alfresco.

Parent topic: Customizing Alfresco Share [1]

Configuring the Share default port

This section describes how to configure the default port configuration for Alfresco Share.

1. Open the <web-extension>/share-config-custom.xml file.
2. Change the port numbers in the following section to meet your needs (for example from 8080 to 8081):

```
<config evaluator="string-compare" condition="Remote">
   <remote>
      <endpoint>
         <id>alfresco-noauth</id>
         <name>Alfresco - unauthenticated access</name>
         <description>Access to Alfresco Repository WebScripts that do not require authentication</description>
         <connector-id>alfresco</connector-id>
         <endpoint-url>http://localhost:8080/alfresco/s</endpoint-url>
         <identity>none</identity>
      </endpoint>
```

```
      <endpoint>
         <id>alfresco</id>
         <name>Alfresco - user access</name>
         <description>Access to Alfresco Repository WebScripts that require user authentication</description>
         <connector-id>alfresco</connector-id>
         <endpoint-url>http://localhost:8080/alfresco/s</endpoint-url>
         <identity>user</identity>
      </endpoint>

      <endpoint>
         <id>alfresco-feed</id>
         <name>Alfresco Feed</name>
         <description>Alfresco Feed - supports basic HTTP authentication via the EndPointProxyServlet</description>
         <connector-id>http</connector-id>
         <endpoint-url>http://localhost:8080/alfresco/s</endpoint-url>
         <basic-auth>true</basic-auth>
         <identity>user</identity>
      </endpoint>

      <endpoint>
         <id>activiti-admin</id>
         <name>Activiti Admin UI - user access</name>
         <description>Access to Activiti Admin UI, that requires user authentication</description>
         <connector-id>activiti-admin-connector</connector-id>
         <endpoint-url>http://localhost:8080/alfresco/activiti-admin</endpoint-url>
         <identity>user</identity>
      </endpoint>
   </remote>
</config>
```

3. Uncomment the following section if you are using NTLM, Kerberos, or external SSO, or an HTTP load balancer:

```
<config evaluator="string-compare" condition="Remote">
   <remote>
      <keystore>
          <path>alfresco/web-extension/alfresco-system.p12</path>
          <type>pkcs12</type>
          <password>alfresco-system</password>
      </keystore>

      <connector>
         <id>alfrescoCookie</id>
         <name>Alfresco Connector</name>
         <description>Connects to an Alfresco instance using cookie-based authentication</description>
         <class>org.alfresco.web.site.servlet.SlingshotAlfrescoConnector</class>
      </connector>

      <connector>
         <id>alfrescoHeader</id>
         <name>Alfresco Connector</name>
         <description>Connects to an Alfresco instance using header and cookie-based authentication</description>
         <class>org.alfresco.web.site.servlet.SlingshotAlfrescoConnector</class>
         <userHeader>SsoUserHeader</userHeader>
      </connector>

      <endpoint>
         <id>alfresco</id>
         <name>Alfresco - user access</name>
         <description>Access to Alfresco Repository WebScripts that require user authentication</description>
         <connector-id>alfrescoCookie</connector-id>
         <endpoint-url>http://localhost:8080/alfresco/wcs</endpoint-url>
         <identity>user</identity>
         <external-auth>true</external-auth>
      </endpoint>
   </remote>
</config>
```

4. Replace all instances of 8080 with the desired port number.
5. Save the file.

Parent topic: Customizing Alfresco Share [1]

Security policies and filters in Alfresco Share

You can configure a number of policies and filters in Alfresco Share to mitigate security attacks.

- **Cross-Site Request Forgery (CSRF) filters in Alfresco Share** [14] You can configure `CSRFPolicy` in Alfresco Share to prevent CSRF attacks that allow malicious requests to be unknowingly loaded by a user.
- **Iframes and phishing attack mitigation in Alfresco Share** [15] You can configure `IFramePolicy` to protect users against a phishing attack, which attempts to acquire information such as user names or passwords by simulating a trustworthy entity.

- **Security filters and clickjacking mitigation in Alfresco Share** [16] You can configure a security filter, `SecurityHeadersPolicy`, that mitigates clickjacking attacks in Alfresco Share.

Parent topic: Customizing Alfresco Share [1]

Cross-Site Request Forgery (CSRF) filters in Alfresco Share

You can configure `CSRFPolicy` in Alfresco Share to prevent CSRF attacks that allow malicious requests to be unknowingly loaded by a user.

You can configure the CSRF filter to run with third party plugins and to stop specific repository services from being accessible directly through the Share proxy.

The filter is implemented in the `org.alfresco.web.site.servlet.CSRFFilter` that reads the `CSRFPolicy` configuration section in `share-security-config.xml`.

`CSRFPolicy` describes how and when the filter mitigates CSRF attacks:

- Each logged in user receives a secret CSRF token.
- The token is communicated to the browser using a `Alfresco-CSRF-Token` cookie.
- When a logged in user performs a POST, PUT or DELETE HTTP request against Alfresco Share the token must be passed in the request using one of the following methods:
  - As a custom HTTP request header called `Alfresco-CSRF-Token`
  - As a URL parameter called `Alfresco-CSRF-Token`
    Note: Usually the header is required, but in some circumstances a header cannot be used and in this case the token can be passed using a URL parameter. The default config only accepts the URL parameter when the `Content-Type` header starts with `multipart/`.
- Every time the logged in user visits a new Share page the token is renewed.
- The filter checks that the referrer and original HTTP request headers match the current domain (if this is present in the request).

Do I need to alter my custom code?

Generally, you should not need to alter your custom code, for example, the following cases need no code alteration:

- You are reading data using GET requests only
- You are using the standard `Alfresco.util.Ajax`, `alfresco/core/CoreXhr` or `Alfresco.forms.Form` javascript classes when creating, updating or deleting data
- You are writing a non-browser client (for example, a mobile application)

However, in these situations you will need to alter your code:

1. You are making an `XMLHttpRequest` with POST, PUT or DELETE methods without using the `Alfresco.util.Ajax` or `alfresco/core/CoreXhr` classes. If you are using the native `XMLHttpRequest` object or a third party library such as jQuery, add code to pass the token, for example:

   ```
   if (Alfresco.util.CSRFPolicy && Alfresco.util.CSRFPolicy.isFilterEnabled())
   {
      xhrHeadersObject[Alfresco.util.CSRFPolicy.getHeader()] = Alfresco.util.CSRFPolicy.getToken();
   }
   ```

   If you are using `YAHOO.util.DataSource` to load data with POST requests, add code similar to this example:

   ```
   if (Alfresco.util.CSRFPolicy && Alfresco.util.CSRFPolicy.isFilterEnabled())
   {
      yuiDataSource.connMgr.initHeader(Alfresco.util.CSRFPolicy.getHeader(), Alfresco.util.CSRFPolicy.getToken(), false);
   }
   ```

2. You are making a form upload with enctype `multipart/form-data` without using `Alfresco.forms.Form`.
   When you upload a file by submitting a form with enctype `multipart/form-data` it is not possible to set a header on the request because it is not possible to set a header on any form submission in the browser. Pass the token as a URL parameter instead. If you are using the `Alfresco.forms.Form` class, this is handled for you automatically, otherwise add the token as a URL parameter, for example:

   ```
    if (Alfresco.util.CSRFPolicy && Alfresco.util.CSRFPolicy.isFilterEnabled())
   {
      url += "?" + Alfresco.util.CSRFPolicy.getParameter() + "=" + encodeURIComponent(Alfresco.util.CSRFPolicy.getToken());
   }
   ```

3. You are using a Flash movie inside Share to send HTTP requests with method POST.

   If you are using a Flash movie to upload files, using the `flash.net.FileReference ActionScript` class to perform a multipart/form-data request, add the token as a URL parameter in your Javascript before passing in the URL to the Flash movie. If your Flash movie is performing application/json or other text based POST requests, using the `flash.net.URLRequest and/or flash.net.navigateToURL` ActionScript classes and methods, pass the token and the name of the header so that it can be set from the Flash movie.

When else might I need to make code updates?

If servers from other domains are allowed to POST requests to your system, then you need to reconfigure `CSRFPolicy` in your `share-config-custom.xml` file so that the token or header is not checked:

1. Copy the `CSRFPolicy` configuration in `share-security-config.xml`.
2. Paste the configuration into your `share-config-custom.xml` file, ensuring that it is replacing the old configuration section:

```
 <config evaluator="string-compare"
      condition="CSRFPolicy" replace="true">
```

3. Copy the following code and add it as the first child of the <filter> element:

```
<rule>
   <request>
      <method>POST</method>
      <path>/page/trusted/call/1|/page/trusted/call/2</path>
   </request>
   <action name="assertReferer">
      <param name="always">false</param>
      <param name="referer">https://www.trustedserver.com/.*</param>
   </action>
   <action name="assertOrigin">
      <param name="always">false</param>
      <param name="origin">https://www.trustedserver.com</param>
   </action>
</rule>
```

The CSRF filter compares the incoming request with the rule request elements to find one that matches and then invokes the defined actions for that rule before normal Share processing begins.
If you want to completely block certain services in the repository, you can add these URLs to the CSRF filter:

1. Copy the `CSRFPolicy` configuration in `share-security-config.xml`.
2. Paste the configuration into your `share-config-custom.xml` file, ensuring that it is replacing the old configuration section:

```
 <config evaluator="string-compare"
      condition="CSRFPolicy" replace="true">
```

3. Copy the following code and add it as the first child of the <filter> element:

```
    <rule>
   <request>
      <path>/proxy/alfresco/acme/special/services/.*</path>
   </request>
   <action name="throwError">
      <param name="message">It is not allowed to access this url from your browser</param>
   </action>
</rule>
```

Parent topic: Security policies and filters in Alfresco Share [8]

Iframes and phishing attack mitigation in Alfresco Share

You can configure `IFramePolicy` to protect users against a phishing attack, which attempts to acquire information such as user names or passwords by simulating a trustworthy entity.

Alfresco allows you to control which domain pages or content are included in Share to create a whitelist of allowed domains. A whitelist is a list of email addresses or IP addresses that are considered to be safe for use within your organisation.

This `IFramePolicy` is applied when Share includes an `<iframe>` tag while constructing the Web View dashlet. The dashlet will allow only those URLs that have been added to the whitelist. Developers can use the `Alfresco.util.IFramePolicy.isUrlAllowed()` method to check if a URL is allowed for custom implementations of a Web View or `<iframe>` tag is included.

Note: If you have a previous installation which includes a URL from a third-party domain, you will get an error message in your production environment prompting you to configure your `IFramePolicy` configuration by adding the domain to the whitelist.
Note: URLs pointing to the same domain, such as documents or wiki pages inside Share, will continue to work as usual by default.

The whitelist of allowed domains is set in the <configRootShare>/classes/alfresco/share-security-config.xml configuration file:

```
<config evaluator="string-compare" condition="IFramePolicy">
 <same-domain>allow</same-domain>
  <cross-domain>
    <url>*</url>
  </cross-domain>
</config>
```

To deny URLs from the current domain, override the existing code in the share-config-custom.xml file with the following code:

```
<config evaluator="string-compare" condition="IFramePolicy" replace="true">
  <same-domain>deny</same-domain>
</config>
```

To allow all cross domain URLs, override the existing code in the share-config-custom.xml file with the following code:

```
<config evaluator="string-compare" condition="IFramePolicy" replace="true">
 <cross-domain>
   <url>*</url>
 </cross-domain>
</config>
```

To allow specific cross domain URLs, override the existing code in the share-config-custom.xml file with the following code:

```
<config evaluator="string-compare" condition="IFramePolicy" replace="true">
 <cross-domain>
```

```
    <url>https://www.owasp.org/</url>
  </cross-domain>
</config>
```

Parent topic: Security policies and filters in Alfresco Share [8]

Security filters and clickjacking mitigation in Alfresco Share

You can configure a security filter, `SecurityHeadersPolicy`, that mitigates clickjacking attacks in Alfresco Share.

`SecurityHeadersPolicy` is a Java Servlet filter that applies HTTP response headers to incoming requests in Alfresco Share. The headers that are returned are defined in a configuration section called `SecurityHeadersPolicy` in `alfresco-security-config.xml`.

Three headers are added by default; `X-Frame-Options`, `X-Content-Type-Options` and `X-XSS-Protection`:

```
<config evaluator="string-compare" condition="SecurityHeadersPolicy">
  <headers>
    <header>
      <name>X-Frame-Options</name>
      <value>SAMEORIGIN</value>
    </header>
    <header>
      <name>X-Content-Type-Options</name>
      <value>nosniff</value>
    </header>
    <header>
      <name>X-XSS-Protection</name>
      <value>1; mode=block</value>
    </header>
  </headers>
</config>
```

X-Frame-Options header

Adding this header to an HTTP response tells the browser whether Share pages are permitted inside iframes. In our default configuration we have set this to `SAMEORIGIN` which means that Share pages are only permitted inside iFrames inside Share or in other web applications that live under the same domain. For example, it is possible to include http://www.acme.com/share inside an iframe on http://www.acme.com/portal.

You can override the configuration and set the header to return `DENY` instead, by placing the following configuration in your `share-config-custom.xml` file:

```
<config evaluator="string-compare" condition="SecurityHeadersPolicy">
  <headers>
    <header>
      <name>X-Frame-Options</name>
      <value>DENY</value>
    </header>
  </headers>
</config>
```

X-Content-Type-Options

This header is valid for Internet Explorer (IE) only. Older versions of IE (8 and below) sniff the content of a returned resource and then execute the content as the content type that IE thinks the resource has, instead of the content type that the server returned. To stop IE from doing this, `nosniff` is returned in the header.

X-XSS-Protection

This header is provided by Internet Explorer (IE) to rectify "sanitization" logic that can be used by an attacker to introduce an XSS flaw on your site.

By default Alfresco Share returns `1; mode=block` for this header, which stops IE from executing sanitized code.

It is also possible to set the value to `0` which stops IE from inspecting the code for XSS attacks.

Adding other headers

Alfresco supports adding other headers to the configuration, for example, the `Strict-Transport-Security` header forces the browser to allow only `https` communication. This header is not provided by Alfresco Share, but can be added by using this code:

```
<config evaluator="string-compare" condition="SecurityHeadersPolicy">
  <headers>
    <header>
      <name>Strict-Transport-Security</name>
      <value>max-age=31536000</value>
    </header>
  </headers>
</config>
```

Parent topic: Security policies and filters in Alfresco Share [8]

Configuring Share for mixed user name types

When there is a mix of user name types, for example, some using the @domain in their user name, this may have an impact on the use of Share.

For example, there may be users with both the @domain and without:

- user2@domain.com
- user1

This configuration enables Share to function correctly when using mixed users types.

1. Open the <web-extension>/share-config-custom.xml file.
2. Add the following bean:

```
<bean id="webframework.slingshot.persister.remote" class="org.springframework.extensions.surf.persister.PathStoreObjectPersister" pa
    <property name="store" ref="webframework.webapp.store.remote" />
    <property name="pathPrefix"><value>alfresco/site-data/${objectTypeIds}</value></property>
    <property name="tenantObjectCache"><value>false</value></property>
</bean>
```

3. Save the file, and restart the Alfresco server.

Parent topic: Customizing Alfresco Share [1]

Share Document Library

The Share repository document library is a feature that gives full access to the Alfresco repository.

The default content structure for Alfresco Share is based on sites, and this does not give full visibility of the content in the repository. By enabling the repository document library configuration setting, you have access to multiple navigation options, for example, folders and categories, tags, and filters. This feature also allows you to recreate and edit text files, for example, within the Data Dictionary.

It is possible to copy or move files to the Share document library without any repository permissions.

The document library is accessed in Share through the Repository, My Files, and Shared Files links in the header, and through the Document Library link in a site. These all kind different views of the complete content repository.

- **Configuring the Repository link** [17] It is possible to control the visibility of the Repository link in Share through configuration. Note the Repository link is always visible to Administrators.
- **Hiding the Share repository link using legacy mode** [18] If required, it is possible to use the Share header menu bar in legacy mode, and hide the Repository link. This means users will only be able to browse content in document libraries of sites that they own, or are a member of.
- **Extending the Alfresco Share Document Library** [19] Alfresco offers a number of extension points for the document library.

Parent topic: Customizing Alfresco Share [1]

Configuring the Repository link

It is possible to control the visibility of the Repository link in Share through configuration. Note the Repository link is always visible to Administrators.

1. Load the file tomcat/shared/classes/alfresco/web-extension/share-config-custom.xml into your favorite editor (assuming you are using the Tomcat application server).
2. Locate the `Repository Library config` section:

```
<!-- Repository Library config section -->
<config evaluator="string-compare" condition="RepositoryLibrary" replace="true">
   <!--
      Root nodeRef or xpath expression for top-level folder.
      e.g. alfresco://user/home, /app:company_home/st:sites/cm:site1
      If using an xpath expression, ensure it is properly ISO9075 encoded here.
   -->
   <root-node>alfresco://company/home</root-node>

   <tree>
      <!--
         Whether the folder Tree component should enumerate child folders or not.
         This is a relatively expensive operation, so should be set to "false" for Repositories with broad folder structures.
      -->
      <evaluate-child-folders>false</evaluate-child-folders>

      <!--
         Optionally limit the number of folders shown in treeview throughout Share.
      -->
      <maximum-folder-count>500</maximum-folder-count>
   </tree>

   <!--
      Whether the link to the Repository Library appears in the header component or not.
   -->
   <visible>true</visible>
</config>
```

3. The configuration that can make the Repository link visible or invisible for non-administrators is the following:

```
<visible>false</visible>
```

   Set to `true` to have the Repository link available to all users.
4. Restart the Alfresco server.

Parent topic: Share Document Library [10]

Hiding the Share repository link using legacy mode

If required, it is possible to use the Share header menu bar in legacy mode, and hide the Repository link. This means users will only be able to browse content in document libraries of sites that they own, or are a member of.

1. Open the <configRootShare>/classes/alfresco/share-config.xml file.
   For example, on Tomcat, the file path for this file will be <TOMCAT_HOME>/webapps/share/WEB-INF/classes/alfresco/share-config.xml
2. Locate the `<!-- Global config section -->` section.
3. Copy the entire `<header></header>` section, and then close the file.
4. Open the <web-extension>/share-config-custom.xml file.
5. Locate `<!-- Global config section -->`.

   If you do not see the global configuration section, copy the full example from the share-config.xml file.
6. Paste the full `<header></header>` section into the `<config replace="true"></config>` section.
7. Set <legacy-mode-true> to true.
8. Locate and comment out the following line from within the <header></header>:

```
<item type="link" id="repository" condition="conditionRepositoryRootNode">/repository</item>
```

9. Save the share-config-custom.xml file, and then restart the Alfresco server.

The Repository link is now not visible in the Share header menu.
Parent topic: Share Document Library [10]

Extending the Alfresco Share Document Library

Alfresco offers a number of extension points for the document library.

This includes:

- Repository tier
- Web tier
- Status indicators
- Metadata templates
- Actions
- Client-side extension points

This documentation also includes a `jsNode` client-side help object reference and a list of out-of-the-box evaluators.

- **Alfresco Share Document Library repository tier** [20] In order to preserve existing customizations and third party add-ons, a parallel set of data web scripts has been developed for Alfresco 4.0 and later to coexist with the previous data web scripts. These new web scripts are located in the `remote-api` project and have URLs starting with `/slingshot/doclib2/`.
- **Alfresco Share Document Library web tier** [21] In versions of Alfresco Share previous to 4.0, the client-side JavaScript requested JSON data from the repository directly via the proxy servlet. From 4.0 onwards, there is a new data web script (at /components/documentlibrary/data/) that requests data from the repository and processes the response based on a configurable set of evaluators before finally returning JSON data to the browser.
- **Override and extension examples** [22] You configure new evaluators via a **web-extension/custom-slingshot-*-context.xml** file, taking the form of bean definitions.
- **Client-side template and action extensions** [23] Two global events are available to make it easier to add new metadata template renders and client-side action handlers.
- **Customizing document library views** [24] Within the document library it is possible to select from a number of views. It is also possible to add custom views to the document library through configuration in the share-documentlibrary-config.xml file.
- **Reference** [25] Reference material.

Parent topic: Share Document Library [10]

Alfresco Share Document Library repository tier

In order to preserve existing customizations and third party add-ons, a parallel set of data web scripts has been developed for Alfresco 4.0 and later to coexist with the previous data web scripts. These new web scripts are located in the `remote-api` project and have URLs starting with `/slingshot/doclib2/`.

There are three extension points supported by the repository data web scripts.

1. Document Library custom response

A custom response appears within the `metadata.custom` section of the JSON response. An example of a cleanly installed service is the `vtiServer` configuration, defined within the slingshot-context.xml file.

The `customResponses` property defines a map of JSON key to custom response bean within the `SlingshotDocLibCustomerResponse` bean definition.

Default `slingshotDocLibCustomResponse` bean configuration:

```
<bean id="slingshotDocLibCustomResponse"
       parent="baseJavaScriptExtension"
       class="org.alfresco.repo.jscript.SlingshotDocLibCustomResponse">
  <property name="extensionName">
     <value>slingshotDocLib</value>
  </property>
  <property name="<b>customResponses</b>">
    <map>
       <entry key="vtiServer">
          <ref bean="doclibCustomVtiServer"/>
       </entry>
    </map>
  </property>
</bean>
```

The bean for returning the vtiServer configuration is defined as:

```
<bean id="doclibCustomVtiServer" class="org.alfresco.repo.jscript.app.VtiServerCustomResponse">
    <property name="scriptUtils">
        <ref bean="utilsScript" />
    </property>
    <property name="sysAdminParams">
        <ref bean="sysAdminParams" />
    </property>
    <property name="port">
        <value>${vti.server.external.port}</value>
    </property>
    <property name="host">
        <value>${vti.server.external.host}</value>
    </property>
</bean>
```

The `VtiServerCustomResponse` class (which implements `CustomResponse`) returns a Serializable object (for example a `LinkedHashMap`) that is serialized into the JSON response by the DocLib web scripts.

This extension point is designed to return useful information that is not specific to any node, for example, the presence of an optional module; whether a subsystem is active or not, etc.

2. Property decorators

The other place the data web scripts may be extended is via the property decorator extension mechanism. These are used by the `org.alfresco.repo.jscript.ApplicationScriptUtils` class and allow properties such as `nodeRefs`, `usernames`, and `dates` to be returned in a much more usable state to the web tier. For example, the Share interface displays `usernames` using First- and Last-name, rather than just the username. By decorating the properties returned in the initial web script request, further requests are not necessary to obtain the missing data.

Partial view of the `applicationScriptUtils` bean configuration:

```
<bean id="applicationScriptUtils" parent="baseJavaScriptExtension"
class="org.alfresco.repo.jscript.ApplicationScriptUtils">
  <property
        name="decoratedProperties">
        <map>
          <entry key="cm:creator">
            <ref bean="usernamePropertyDecorator"/>
          </entry>
          <entry key="cm:modifier">
            <ref bean="usernamePropertyDecorator"/>
          </entry>
```

The `decoratedProperties` property defines a map of short-format `QName` to decorator implementation bean. Each of these decorator beans implements the `PropertyDecorator` interface and returns a serializable map via the `decorate()` method. This map is then serialized into the JSON response for each node being returned by the web script request.

3. Permissions list

The third place the data web script response may be extended is via the list of permissions that are returned for each node. These are defined via the `userPermissions` property on the `applicationScriptUtils` bean. For example:

```
<property name="<b>userPermissions</b>">
  <list>
      <value>CancelCheckOut</value>
      <value>ChangePermissions</value>
      <value>CreateChildren</value>
      <value>Delete</value>
      <value>Write</value>
  </list>
</property>
```

The default set of permissions should not be reduced without fully understanding the impact on actions, indicators, and metadata evaluators already in use throughout Share.
Parent topic: Extending the Alfresco Share Document Library [19]

Alfresco Share Document Library web tier

In versions of Alfresco Share previous to 4.0, the client-side JavaScript requested JSON data from the repository directly via the proxy servlet. From 4.0 onwards, there is a new data web script (at /components/documentlibrary/data/) that requests data from the repository and processes the response based on a configurable set of evaluators before finally returning JSON data to the browser.

All configuration for, and evaluation of, Document Library status indicators, metadata templates, and actions is now on the web tier instead of split between the repository and the browser.

1. Web tier configuration overview

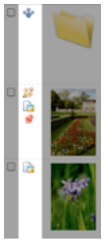
The individual action configuration files (for example documentlist.get.config.xml, document-details.get.config.xml) have been removed and all actions are now defined within common configuration sections.

The new or altered areas of configuration in share-documentlibrary-config.xml are:

`DocumentLibrary`	• Updated for version 4.0 • New `<indicators>` section for configuring status indicators • New `<metadata templates>` for configuring the metadata displayed within the Document Library's "browse" view
`DocLibCustom`	• New to version 4.0 • `<dependencies>` section for defining custom client-side functionality and stylesheets
`DocLibActions`	• New to version 4.0 • `<actions>` section defining all available actions across the various Document Library view pages • `<actionGroups>` that define which (and in what order) actions are to appear on the Document Library pages

Also from version 4.0 onwards is the slingshot-documentlibrary-context.xml file containing all bean definitions for web tier evaluators.

2. Status indicators



- Defined within the `DocumentLibrary` config section, status indicators are small icons typically used to indicate the presence of a marker aspect, or whether a document is in a particular state, for example checked out for editing.
- Indicator images by default are referenced by id: /res/components/documentlibrary/indicators/{id}-16.png unless the name is overridden by the "icon" attribute.
- Tooltip labels are also defaulted by id: `status.{id}` and can be overridden by the `label` attribute.

The status indicators are located in the `<indicators>` config container element with the following structure:

```
<indicator id (index) (icon) (label)>
    <evaluator />
    <labelParam index />
    <override />
</indicator>
```

where:

`<indicator>`	Status indicator element with the following attributes:

	• `id`: Unique indicator id • `index`: Determines display order of this indicator • `icon`: Icon filename; if not specified, `id` is used • `label`: Tooltip i18 label; if not specified, `id` is used
`<evaluator>`	Bean id of evaluator that determines the visibility of the image. The Evaluator extends `org.alfresco.web.evaluator.BaseEvaluator`
`<labelParam>`	Allows placeholder values within i18n label to be replace at runtime with node properties. The value is the replacement string or dot notation path to a node property. The attribute is: • `index`: Index of placeholder value with i18n message
`<override>`	Allows this indicator to override (hide) other indicators that would otherwise be visible. The value is the id of another indicator to override.

Example config

```
<indicator id="google-docs-locked" index="10">
    <evaluator>evaluator.doclib.indicator.googleDocsLocked</evaluator>
    <labelParam index="0">{jsNode.properties.owner.displayName}</labelParam>
    <labelParam index="1">{jsNode.properties.owner.userName}</labelParam>
    <override>locked</override>
</indicator>
```

A note about the `labelParam value`: refactoring on the client-side (JavaScript code) means that a common helper object is available for each node within the Document Library during the rendering cycle, namely `jsNode`. A full reference for this new resource is in jsNode reference. [26]

3. Metadata templates

The metadata template refers to the section of the document "browse" page under the filename. Funtionality introduced in version 4.0 allows this area to be customized with node properties and/or by custom rendering functions.

In a clean install, there are two templates defined: the `default` (fallback) template and one used when rendering working copies. These are both defined within share-documentlibrary-config.xml and can be extended or overridden as required (via share-config-custom.xml).

The metadata templates are located in the `<metadata-templates>` config container element with the following structure:

```
<template id>
    <evaluator />
    <line id (index) (simpleView) (evaluator) />
    <override />
</template>
```

where:

`<template>`	Template element with the following attribute: • `id`: Unique template id
`<evaluator>`	Bean id of evaluator that determines whether the template is to be used for this node or not. The Evaluator extends `org.alfresco.web.evaluator.BaseEvaluator`
`<line>`	Allows placeholder values within i18n label to be replace at runtime with node properties. The value refers either to a node property (such as `cm_description`) or a customer JavaScript renderer. To add a label in front of the property, add the label's i18n messageId after the property value, separated by a space (such as {`cm_description details.description`}. The attributes are: • `id`: Id of the line within the template. Must be unique within this template. • `index`: Optional index for ordering the lines when rendering. • `view`: If set to "simple" or "detailed", then this line will only be rendered when either the simple or detailed view is toggled on, respectively. Leave empty, or omit the attribute for both views. • `evaluator`: Optional evaluator to determine whether this line will be rendered for a node when using the template.

Example config

```
<template id="isPhoto">
   <evaluator>evaluator.doclib.metadata.hasExif</evaluator>
   <line index="10" id="date" view="detailed">{date}{size}</line>
   <line index="20" id="exposure" evaluator="evaluator.doclib.metadata.hasExposure">
      {exposure exif.label.exposure}
   </line>
   <line index="30" id="description" view="detailed">{description}</line>
   <line index="40" id="social" view="detailed">{social}</line>
</template>
```

Custom JavaScript renderers

A renderer can either be a simple property value, or use a custom JavaScript renderer. To register a custom renderer, fire a Bubbling (global) event, passing-in the renderer id and the rendering function:

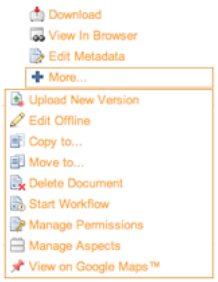
```
if (Alfresco.DocumentList)
{
    YAHOO.Bubbling.fire("registerRenderer",
    {
        propertyName: "renderer id",
        renderer: function(record, label)
        {
           return "...";
        }
    });
}
```

The rendering function should return property escaped HTML.

4. Actions

In versions previous to 4.0, the actions configuration was spread throughout a number of web script XML config files. From 4.0, actions are all now defined globally in the share-documentlibrary-config.xml file, in the `DocLibActions` config section. This means they can be overridden and extended via a share-config-custom.xml file. These customizations can be via AMP, JAR or web-extension folder mechanism, or a mixture of all three.

Actions are also now grouped by view type instead of node “state”.



The actions are located in the `<actions>` config container element with the following structure:

```
<action id type (icon) label>
    <param name />
    <evaluator negate />
    <permissions>
      <permissions allow />
    </permissions>
    <override />
</action>
```

where:

`<action>`	Action config container element with the following attributes: • `id`: Unique action id • `type`: Action type. Currently supported are: javascript, link, pagelink • `icon`: Optionally, override the icon name. If not set, the id is used • `label`: The action's i18n message id
`<param>`	Action parameter elements with the following attribute: • `name`: Parameter name
`<evaluator>`	Bean id of evaluator that determines whether the action is valid for this node or not. Evaluator extends `org.alfresco.web.evaluator.BaseEvaluator` and contains the following attribute:

	• negate: If set to "true", the output of the evaluator is inverted
`<permissions>`	Permission config container element
`<permission>`	List of permissions required for the actions, as defined in the `applicationScriptUtils` bean config with the following attributes: • `allow`: If the permission specifies, the action is allowed • `deny`: If the permission specifies, the action is hidden Only one of the "allow" or "deny" permissions can be specified
`<override>`	If this action should override the visibility of other actions, they are specified using this element.

Example config

```
<!-- Inline edit -->
<action id="document-inline-edit" type="pagelink" label="actions.document.inline-edit">
   <param name="page">inline-edit?nodeRef={node.nodeRef}</param>
   <permissions>
     <permission allow="true">Write</permission>
   </permissions>
   <evaluator>evaluator.doclib.action.inlineEdit</evaluator>
</action>
```

```
<!-- Checkin from Google Docs -->
<action id="document-checkin-from-googledocs" type="javascript" label="actions.document.checkin-google">
     <param name="function">onActionCheckinFromGoogleDocs</param>
     <evaluator>evaluator.doclib.action.googleDocsCheckIn</evaluator>
     <override>document-checkout-to-googledocs</override>
</action>
```

```
<!-- View in Explorer client -->
<action id="view-in-explorer" type="link" label="actions.folder.explorer-view">
  <param name="href">{explorerViewUrl}</param>
  <param name="target">_blank</param>
  <evaluator>evaluator.doclib.action.viewInExplorer</evaluator>
</action>
```

5. Action groups

Actions are grouped using the `actionGroup` elements. The type of node and also the view currently in use determines the actual group used. The group is calculated by the `calculateActionGroupId()` function in surf-doclist.lib.js and is designed to be overridden if many new and/or altered actions are required.

The action groups defined in a default installation are:

Action Group id	Default Usage
document-browse	Documents on the browse page
document-details	Document on the document details page
folder-browse	Folders on the browse page
folder-details	Folders on the folder details page
document-link-browse	Links to documents on the browse page
document-link-details	Links to folders on the document details page
folder-link-browse	Links to folders on the browse page
folder-link-details	Links to folders on the folder details page

The action groups are located in the `<actionGroups>` config container element with the following structure:

```
<actionGroup id>
    <action />
</actionGroup>
```

where:

`<actionGroup>`	Action group config container element with the following attribute: • `id`: Unique action group id

`<action>`	Action element with the following mandatory attribute: • `id`: Reference to action as defined in `<actions>` config section

Other actions properties are over-ridable here, although it is recommended from a maintenance view to only override "simple" properties like the icon and label. These make it possible to reuse an action with document-biased icon and label to be used for folders.

Example config

```
<actionGroup id="folder-browse">
    <action index="100" id="folder-view-details" />
    <action index="110" id="folder-edit-properties" icon="folder-edit-properties" />
    <label="actions.folder.edit-metadata" />
</actionGroup>
```

6. Custom client extensions

The `DocLibCustom` config section is where dependencies on custom client-side assets can be defined. These are defined in exactly the same way as for custom Forms dependencies.

The client-side dependencies are located in the `<dependencies>` config container element with the following structure:

```
<css src />
<js src />
```

where:

`<css>`	Stylesheet dependencies element with the following attribute: • `src`: Path to the css file, relative to the /res servlet
`<js>`	JavaScript dependencies element with the following attribute: • `src`: Path to the js file, relative to the /res servlet

Other actions properties can be overridden here, although it is recommended from a maintenance point of view to only override "simple" properties like the icon and label. These make it possible to reuse an action with documen-biased icon and label to be used for folders.

Example config

```
<dependencies>
    <cs src="/custom/my-customization.css" />
    <js src="/custom/my-customization.js" />
</dependencies>
```

Parent topic: Extending the Alfresco Share Document Library [19]

Override and extension examples

You configure new evaluators via a **web-extension/custom-slingshot-*-context.xml** file, taking the form of bean definitions.

You can use any of the out-of-the-box evaluators as parents to template from. For example:

```
<bean id="evaluator.doclib.metadata.hasExposure"
      parent="evaluator.doclib.action.propertyNotNull">
   <property name="property" value="exif:exposureTime"/>
</bean>
```

Client-side dependencies are specified in the share-config-custom.xml file using the `DocLibCustom` config section.

```
<config evaluator="string-compare" condition="DocLibCustom">
   <dependencies>
         <js src="/custom/exif.js" />
   </dependencies>
</config>
```

Extra status indicators are configured in the following way via the share-config-custom.xml file.

```
<config evaluator="string-compare" condition="DocumentLibrary">
    <indicator id="my-custom"index="10">
        <evaluator>evaluator.doclib.indicator.myCustomEvaluator</evaluator>
        <labelParam index="0">{jsNode.properties.owner.displayName}</labelParam>
    </indicator>
</config>
```

Custom metadata templates are also specified in the share-config-custom.xml file, in the `DocumentLibrary` config section.

```
<config evaluator="string-compare" condition="DocumentLibrary">
<metadata-templates>
    <!-- Photos -->
    <template id="isPhoto">
       <evaluator>evaluator.doclib.metadata.hasExif</evaluator>
       <line index="10" id="date" view="detailed">{date}{size}</line>
       <line index="20" id="exposure" evaluator="evaluator.doclib.metadata.hasExposure">
         {exposure exif.label.exposure}
       </line>
       <line index="30" id="description" view="detailed">{description}</line>
       <line index="40" id="social" view="detailed">{social}</line>
   </template>
</metadata-templates></config>
```

New actions can be specified within the share-config-custom.xml file as follows.

```
<config evaluator="string-compare" condition="DocLibActions">
   <actions>
      <action id="document-preview-webasset" type="javascript"
              label="actions.wcmqs.preview-webasset">
           <param name="function">onActionPreviewWebAsset</param>
           <evaluator>wcmqs.evaluator.doclib.action.isPreviewable</evaluator>
      </action>
   </actions>
</config>
```

An action may be disabled across the whole application using the following configuration in a share-config-custom.xml file. For example the following config removes the "Upload New Version" action from users.

```
<config evaluator="string-compare" condition="DocLibActions">
   <actions>
      <action id="document-upload-new-version">
          <evaluator>evaluator.doclib.action.disableAction</evaluator>
      </action>
   </actions>
</config>
```

Add an evaluator, used on an out-of-the-box action:

```
<config evaluator="string-compare" condition="DocLibActions">
   <actions>
      <action id="document-publish">
          <evaluator negate="true">
             wcmqs.evaluator.doclib.action.isWebsiteContainerType
          </evaluator>
      </action>
   </actions>
</config>
```

Parent topic: Extending the Alfresco Share Document Library [19]

Client-side template and action extensions

Two global events are available to make it easier to add new metadata template renders and client-side action handlers.

Metadata template renderer

Custom client renderers are registered with the Document Library using the new `registerRenderer` Bubbling event.

Ensure the client-side assets are loaded onto the page using the DocLibCustom / dependencies configuration section.

Using the example to add a new EXIF metadata renderer to produce the output as follows.

Giving the renderer an id of “exposure” also extends the metadata templates using a custom line config:

```
<line index="20" id="exposure">{exposure exif.label.exposure}</line>
```

The JavaScript to register the custom renderer is then simply as follows. Note the event name, the event property names and where the custom renderer id is specified.

```
YAHOO.Bubbling.fire("registerRenderer",
{
  propertyName: "exposure",
  renderer: function exif_renderer(record, label)
  {
  ...
     return html;
  }
  });
```

See EXIF renderer source code [27] for the complete source for this example.

Custom action handler

In a very similar way to metadata renderers, new client-side actions are registered using the **registerAction** Bubbling event message.

```
YAHOO.Bubbling.fire("registerAction",
{
  actionName: "onActionPreviewWebAsset",
  fn: function WCMQS_onActionPreviewWebAsset(record)
  {
  ...
  }
});
```

A full example of this can be found in the modules/wcmquickstart/wcmquickstartsharemodule project within the main Alfresco SVN repository.
Parent topic: Extending the Alfresco Share Document Library [19]

Customizing document library views

Within the document library it is possible to select from a number of views. It is also possible to add custom views to the document library through configuration in the share-documentlibrary-config.xml file.

When browsing content in the document library it is possible to select from a variety of views including:

- Simple
- Detailed
- Gallery
- Filmstrip
- Table
- Audio
- Media

These views are selected from the Options button.

The share-documentlibrary-config.xml file controls what views will be available as options when browsing the DocumentLibrary, My Files, Shared Files, and Repository pages. It is also possible to use a module that provides evaluated configuration to have the options change based on criteria such as site name, site preset, user group, and so on.

It is possible to customize these views, and also add additional view types through configuration in the share-documentlibrary-config.xml file. These views are also present in the My Files, Shared Files and Repository pages.

The views are rendered by view renderers, which have various attributes and also a block of configuration (in JSON) associated with them. For example:

```
<view-renderer id="email" iconClass="table" label="button.view.email" index="50" widget="Alfresco.DocumentListTableViewRenderer">
     <dependencies>
        <js src="components/documentlibrary/documentlist-view-detailed.js" />
        <js src="components/documentlibrary/documentlist-view-table.js" />
        <css src="components/documentlibrary/documentlist-view-table.css" />
     </dependencies>
     <json-config>
        {
           "actions": {
              "show": "true"
```

```
            },
            "indicators": {
               "show": "true"
            },
            "selector": {
               "show": "true"
            },
            "thumbnail": {
               "show": "false"
            },
            "propertyColumns": [
               {
                  "property": "cm:originator",
                  "label": "table.email.label.originator",
                  "link": "true"
               },
               {
                  "property": "cm:subjectline",
                  "label": "table.email.label.subjectline",
                  "link": "true"
               },
               {
                  "property": "cm:sentdate",
                  "label": "table.email.label.sentdate"
               },
               {
                  "property": "cm:addressee",
                  "label": "table.email.label.addressee"
               },
               {
                  "property": "cm:addressees",
                  "label": "table.email.label.addressees"
               },
               {
                  "property": "cm:attachments",
                  "label": "table.email.label.attachments"
               }
            ]
         }
      </json-config>
   </view-renderer>
```

For example the following snippet shows a custom simplified view called "minimalist":

```
   <view-renderer id="minimalist" iconClass="table" label="button.view.minimalist" index="60" widget="Alfresco.DocumentListTableViewRenc
     <dependencies>
       <js src="components/documentlibrary/documentlist-view-simple.js" />
       <js src="components/documentlibrary/documentlist-view-table.js" />
       <css src="components/documentlibrary/documentlist-view-table.css" />
     </dependencies>
     <json-config>
       {
          "actions": {
             "show": "false"
          },
          "indicators": {
             "show": "false"
          },
          "selector": {
             "show": "true"
          },
          "thumbnail": {
             "show": "false"
          },
          "propertyColumns": [
             {
                "property": "cm:name",
                "label": "table.minimalist.label.name",
                "link": "true"
             }
          ]
       }
     </json-config>
   </view-renderer>
```

Note that the value of labels such as `table.minimalist.label.name` are set in properties files, so that multiple translations can be provided.

The minimalist custom view uses the Table View renderer.

There are four columns that are always present in the table, which can be hidden if required:

- actions - the drop-down menu of actions that can be peformed on the document
- indicators - the set of icons that visually communicate information about the document
- selector - the checkbox to use when selecting multiple documents
- thumbnail - the thumbnail-sized preview of the document

All other columns must be defined in the `propertyColumns` array. The property attribute can be set to either a document property, such as `cm:name` or a metadata template renderer such as `size`, `tags` or `date`.
Parent topic: Extending the Alfresco Share Document Library [19]

Reference

Reference material.

- **jsNode reference** [28] `jsNode` is the preferred object to access node properties and aspects via JavaScript on the browser.
- **Predefined Evaluators** [29] The following is a list of the evaluators defined within the core Alfresco Share code, as of the v4.0 release.
- **EXIF renderer source code** [30] The EXIF renderer source code is as follows.

Parent topic: Extending the Alfresco Share Document Library [19]

jsNode reference

`jsNode` is the preferred object to access node properties and aspects via JavaScript on the browser.

When dealing with DataTable records, `record.jsNode` should be available.

Note: It is the responsibility of any code that updates DataTable records to also ensure the `jsNode` property is updated (usually within the AJAX success callback).
To create a `jsNode` instance, use:

```
jsNode = new Alfresco.util.Node(p_node)
```

where `p_node` can either be a JavaScript object or JSON string. In either case, it should be in the format returned by the doclist-v2 data web scripts.

Methods

The jsNode methods are:

`getNode`	Returns original node object. If a JSON string was passed in, this method returns a JavaScript object
`toJSON`	Return the JSON string serialization of the node
`setNodeRef`	Sets a new nodeRef - doesn't requery node properties however. Used solely when generating new page urls
`hasAspect`	Returns true if this node has the given aspect
`hasTag`	Returns true if this node has the given tag applied

Also new to v4.0 is the **slingshot-documentlibrary-context.xml** file containing all bean definitions for web tier evaluators.

Properties

The `jsNode` properties are:

Core node properties	
`nodeRef`	NodeRef
`type`	The node's type in short QName format
`isContainer`	Returns true if the node is a container type
`isLink`	Returns true if the node is a file or folderlink type
`isLocked`	Returns true if the node has been locked by any user
`linkedNode`	If this node is a link, returns a jsNode instance of the linked node
Content nodes	
`contentURL`	Of the format /api/node/content/{nodeRef}/{filename}
`mimetype`	Content mimetype
`size`	Content size in bytes
Properties	

`properties`	All properties are available either via:
	`properties["my:property"]`
	or
	`properties.my_property`
	Note that `cm:` properties are available without the prefix, i.e. "`properties.description`", "`properties.title`"

Aspects	
`aspects`	Array of aspects present on this node. See also `hasAspect()`
Permissions	
`permissions`	The permissions the current user has on this node. The list of permissions is defined in the `applicationScriptUtils` bean configuration.
Tags	
`tags`	Array of tags. See also `hasTag()`
Categories	
`categories`	Returns an array of the format [`categoryName`, `category path`]

Parent topic: Reference [25]

Predefined Evaluators

The following is a list of the evaluators defined within the core Alfresco Share code, as of the v4.0 release.

They are all defined in slingshot-documentlibrary-context.xml and can be reused in customizations as required.

Evaluators

These evaluators may need extra configuration before they can be used and form the basis of all metadata and action evaluators via the bean configuration `parent` attribute.

Evaluator	Properties
`evaluator.doclib.action.hasAspect`	• aspects: List of aspects the node must have
`evaluator.doclib.action.isMimetype`	• mimetypes: The node must match one of the mimetypes
`evaluator.doclib.action.propertyNotNull`	• property: the node property must not be null for this evaluator to return true
`evaluator.doclib.action.chainedMatchAll`	• evaluators: List of evaluators that are run in turn until one returns false or the end is reached
`evaluator.doclib.action.chainedMatchOne`	• evaluators: List of evaluators that are run in turn until one returns true
`evaluator.doclib.action.disableAction`	• Always returns false
`evaluator.doclib.action.sitePreset`	• presets: current site must match one of the listed presets
`evaluator.doclib.action.siteBased`	• Returns true if the current node is located within a Share site and the Site Document Library is being used
`evaluator.doclib.action.containerType`	• types: Current documentLibrary container folder must match one of the listed types
`evaluator.doclib.action.nodeType`	• allow Subtypes: Whether subtypes of the node are also allowed • types: Node must match one of the listed types
`evaluator.doclib.action.value`	This evaluator is described in further detail in the next topic. • accessor: `jsNode` property to be tested • comparator: Bean definition of class implementing the Comparator interface
`evaluator.doclib.action.metadataValue`	This evaluator is described in further detail in the next topic. • accessor: metadata property to be tested • comparator: Bean definition of class implementing the Comparator interface

evaluator.doclib.action.isBrowser	• regex: Regular expression to match against browser userAgent string
evaluator.doclib.action.isPortlet	Returns true if the application is currently deployed within a portlet environment
evaluator.doclib.action.notPortlet	Returns the inverse of isPortlet

Comparators

The evaluator.doclib.action.value and evaluator.doclib.action.metadataValue evaluators use comparator helper beans in order to test a value against certain conditions. The following comparators are available in a standard install.

Evaluator	Properties
org.alfresco.web.evaluator.StringEqualsComparator	• aspects: Value to test string against • caseInsensitive: Defaults to true
org.alfresco.web.evaluator.NullValueComparator	• value: Boolean to indicate if null should be the pass or fail case

Evaluator Instances

The following lists describe each of the evaluators currently defined for the v4.0 release.

Status indicator evaluators

evaluator.doclib.indicator.editing	The current user is editing this node (working copy)
evaluator.doclib.indicator.lockOwner	The current user is the lock owner (original of a working copy pair)
evaluator.doclib.indicator.locked	The node is locked by another user
evaluator.doclib.indicator.googleDocsEditing	The node is being edited via Google Docs
evaluator.doclib.indicator.googleDocsOwner	The current user is editing the node is Google Docs
evaluator.doclib.indicator.googleDocsLocked	Another user is editing the node via Google Docs
evaluator.doclib.indicator.activeWorkflows	The node is involved in one or more active (advanced) workflows
evaluator.doclib.indicator.simpleWorkflow	The node is part of a simple workflow process
evaluator.doclib.indicator.rules	The node has rules applied
evaluator.doclib.indicator.exifMetadata	The node has the EXIF metadata aspect applied
evaluator.doclib.indicator.geographicMetadata	The node has the Geographic aspect applied

Metadata template evaluators

evaluator.doclib.metadata.hasCategories	The node has the classifiable aspect applied
evaluator.doclib.metadata.isWorkingCopy	The node is a working copy

Action evaluators

evaluator.doclib.action.simpleApprove	Uses simpleWorkflowAspect and simpleApproveProperty to check for the simple workflow “Approve” action validity
evaluator.doclib.action.simpleReject	Uses simpleWorkflowAspect and simpleRejectProperty to check for the simple workflow “Reject” action validity
evaluator.doclib.action.locateAction	Checks the current filter is “path”
evaluator.doclib.action.inlineEdit	Uses the inlineEditAspect and inlineEditMimetype evaluators to determine if a content node can be edited inline
evaluator.doclib.action.onlineEdit	Uses onlineEditVtiServer, onlineEditBrowser and onlineEditMimetype evaluators to determine if the “Edit Online” action is valid
evaluator.doclib.action.hasLockableAspect	Used in an inverted state for the “Edit Offline”, “Copy”, “Move” and “Publish” actions
evaluator.doclib.action.siteBased	Enables the site-based permissions dialog. Used in inverted state for the repository-based permissions page action
evaluator.doclib.action.isWorkingCopy	Tests whether a node is a working copy
evaluator.doclib.action.viewInExplorer	Reads the repository-url config value to determine the validity of the “View in Explorer” action

`evaluator.doclib.action.googleDocsEditable`	Enables “Check out to Google Docs” action
`evaluator.doclib.action.googleDocsCheckIn`	Tests for the validity of the “Check in from Google Docs” action
`evaluator.doclib.action.googleDocsView`	Tests whether a node has been checked out to Google Docs
`evaluator.doclib.action.googleMaps`	Checks for the `cm:geographic` aspect
`evaluator.doclib.action.transferred`	Tests for the `trx:transferred` action for the “View in Source Repository” action

Parent topic: Reference [25]

EXIF renderer source code

The EXIF renderer source code is as follows.

```
/*******************************************************************************
                                          EXIF
                                        EXTENSION
*******************************************************************************/
(function()
{
  /**
   * Alfresco Slingshot aliases
   */
   var $html = Alfresco.util.encodeHTML,
   $isValueSet = Alfresco.util.isValueSet;

   if (Alfresco.DocumentList)
   {
        YAHOO.Bubbling.fire("registerRenderer",
        {
           propertyName: "exposure",
           renderer: function exif_renderer(record, label)
           {
                 var jsNode = record.jsNode,
                 properties = jsNode.properties,
                 html = "";

              var expTime = properties["exif:exposureTime"] || 0,
                 exifObj =
                {
                      exposureFraction: expTime > 0 ? "1/" + Math.ceil(1/expTime) : expTime,
                      fNumber: properties["exif:fNumber"] || 0,
                      isoSpeedRatings: properties["exif:isoSpeedRatings"] || 0
                 };

              html = '<span class="item">' + label + '<b>' +
YAHOO.lang.substitute(this.msg("exif.metadata.exposure"), exifObj) + '</b></span>';
              return html;
           }
        });
   }
})();
```

Parent topic: Reference [25]

Share themes

When you run Alfresco Share, the look and feel is set by a default theme. This section describes how to select one of the alternative themes available in Share, and also how to create and use your own themes for corporate branding.

Share themes consist of a directory containing a CSS and images files, and they can be located in the theme directory (<TOMCAT_HOME>/webapps/share/WEB-INF/classes/alfresco/site-data/themes). The default theme is called default.xml.

The following themes are available in Share:

- Blue theme (default)
- Yellow theme
- Green theme
- High contrast black
- Google Docs theme

The default theme, which comprises the CSS and image assets used across all pages, displays in a new Alfresco installation.

- **Selecting themes** [31] Only an Administrator user can select the Share theme. Any change to the theme will affect all users of the Alfresco instance from the next time that they log in or from a browser refresh.
- **Creating a new theme** [32] Additional themes may be defined by creating a new theme directory containing the necessary files, as well as the corresponding XML file, whose name must match that of the theme's directory.
- **Editing a theme** [33]

Parent topic: Customizing Alfresco Share [1]

Selecting themes

Only an Administrator user can select the Share theme. Any change to the theme will affect all users of the Alfresco instance from the next time that they log in or from a browser refresh.
The installation wizards install the default theme and the sample alternative themes. The available themes are installed in the <configRootShare>/classes/alfresco/site-data/themes directory.

1. On the toolbar, expand the More menu and click Application in the Tools list.
   The Options page appears.
2. Select the required theme from the menu:
   - Green Theme
   - Yellow Theme
   - High Contrast Theme
   - Default Theme
   - Google Docs Theme
3. Click Apply.

The new theme displays in Share. The new theme persists across sessions.
Parent topic: Share themes [11]

Creating a new theme

Additional themes may be defined by creating a new theme directory containing the necessary files, as well as the corresponding XML file, whose name must match that of the theme's directory.

1. Make a new directory within the /themes directory.
   Note: Do not include spaces in the directory name.
2. Copy the contents of an existing theme directory to the new theme directory.
   For example, copy the `greenTheme` directory.
3. Open the following files:
   a. base.css
   b. ie6.css
   c. ie7.css
   d. presentation.css
   e. yui/assets/skin.css
4. Specify the new theme by searching for `.yui-skin-greenTheme` and replacing with `.yui-skin-XXX` where `XXX` is the name of the new theme directory.
5. Save the files.

The new theme is then available in the list of themes on the Application tool.
Parent topic: Share themes [11]

Editing a theme

A theme consists of some CSS files, an image directory, and a directory for assets for YUI. To create a new look, change the presentation.css file and, if required, replace or add images to the /images directory.

1. Open the presentation.css file.
2. Locate the properties at the end of the presentation.css file.
3. Edit the following four properties:
   a. `color`
   b. `background`
   c. `background-color`
   d. `border`
   Any change to these properties will change the theme.

```
/ Theme colors /
.theme-color-1,
a.theme-color-1,
a.theme-color-1:visited,
a.theme-color-1:hover
{
   color: #6CA5CE;
}

.theme-color-2,
a.theme-color-2,
a.theme-color-2:visited,
a.theme-color-2:hover
{
   color: #038603;
}

.theme-color-3,
a.theme-color-3,
a.theme-color-3:visited,
a.theme-color-3:hover
{
   color: #C7DBEB;
```

```
}

.theme-color-4,
a.theme-color-4,
a.theme-color-4:visited,
a.theme-color-4:hover
{
   color: #0D3B57;
}

/ Theme background colors /
 .theme-bg-color-1,
div.theme-bg-color-1
{
   background-color: #6CA5CE;
}

 .theme-bg-color-2,
div.theme-bg-color-2
{
   background-color: #fffbdd;
}

 .theme-bg-color-3,
div.theme-bg-color-3
{
   background-color: #DEE8ED;
}

 .theme-bg-color-4,
div.theme-bg-color-4
{
   background-color: #EBEFF1;
}

.theme-bg-color-5,
div.theme-bg-color-5
{
   background-color: #2B6EB5;
}

 .theme-bg-1
{
   / background-image: url(images/navbar-bg.png); /
}

 .theme-bg-2
{
   / background-image: url(images/navbar-bg-2.png); /
}

 / Theme border type/colors /
 .theme-border-1
{
   border-color: #457f63;
   border-style: dotted;
}

 .theme-border-2
{
   border-color: #2B6EB5;
}
```

4. Locate the `YUI Theme Changes` section.
   This section allows changes to the YUI components.
5. Edit the properties in this section to change the theme.

Parent topic: Share themes [11]

Forms

Alfresco Share presents data view and entry forms throughout its user interface, which are built on the Surf framework. This framework provides a convention for implementing forms.

Both DM (Document Management) and WCM (Web Content Management) forms use the same services, meaning that Alfresco uses only one configuration syntax and one set of UI controls for forms.

- **Use of forms in Share** [34] Forms are used in the View Metadata and Edit Metadata pages within Share.
- **Forms architecture** [35] The forms engine consists of four major parts:
- **Forms event sequence** [36] When a request is made to a page containing the form component, the following sequence of events occurs.
- **Configuring forms** [37] The default forms configuration is specified in the <configRootShare>/classes/alfresco/share-form-config.xml file. This file contains all the default controls and constraint handlers for the Alfresco content model and the form configuration for the `cm:content` and `cm:folder` types. This file also contains an example of configuring the `cm:content` type.
- **Customizing forms controls** [38] One of the most common customization is to add new controls. A control is the label for a field and the interface that the user interacts with for setting the value of the field.
- **Customizing the validation handler** [39] A validation handler is a small JavaScript function that gets called by the forms runtime when a field value

needs to be validated.

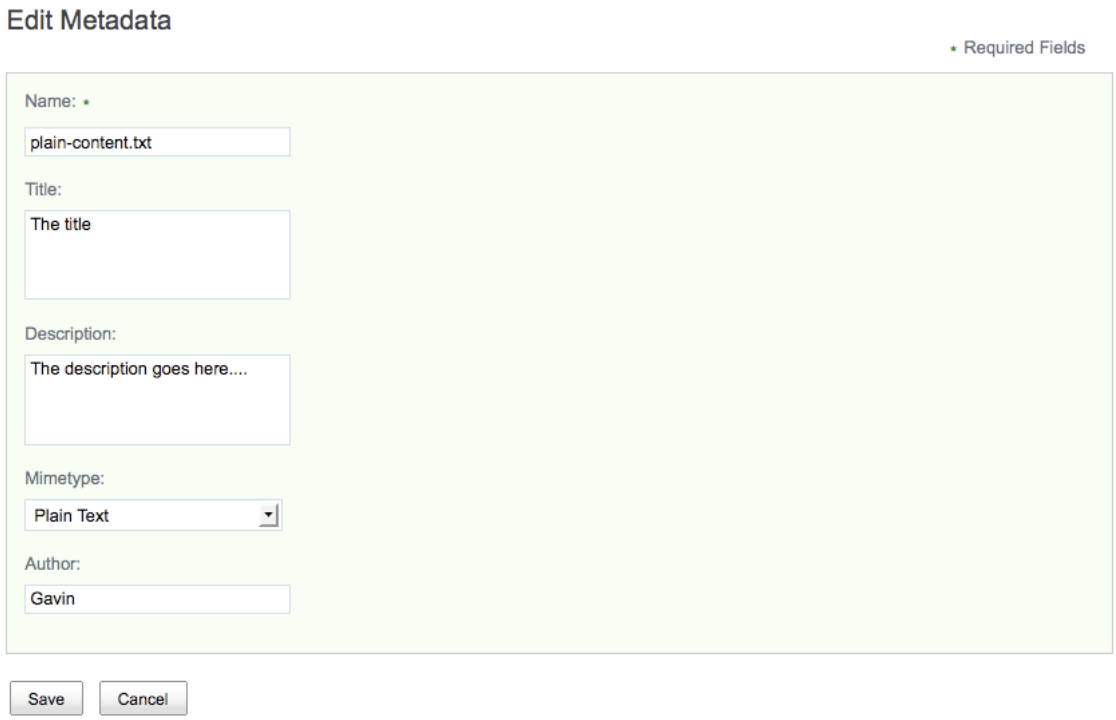
- **Displaying type metadata** [40] The configuration to define the fields for the `cm:content` type exists in the system file called web-framework-config-commons.xml. You can configure the type metadata in the share-config-custom.xml file in <web-extension>. It is also possible to deploy custom configurations via JARs or AMPs.
- **Displaying aspect metadata** [41] Add the properties and associations defined on aspects by adding them to the list of fields to show for a type. The aspects that appear can be defined on a type by type basis, and you can control the ordering of the fields.
- **Configuring a form control** [42] Most of the built in controls have parameters, which allow some basic customization.
- **Grouping fields** [43] For longer forms, you can group fields together in logical grouped or nested sections.
- **Changing the default set label** [44] Fields that do not specify a set belong to the implicit `default` set. They are rendered together, by default, but without any visual grouping.
- **Providing a custom form control** [45] If none of the out-of-the-box controls are sufficient, you can add new controls and reference them. Controls are Freemarker template snippets, therefore, they contain only the HTML markup required to represent the control. The templates need to be stored in the site-webscripts directory, which will usually be in the application server shared classpath.
- **Changing the field label position** [46] By default, forms are rendered with the field labels positioned above the input control.
- **Providing a custom form template** [47] The default template that renders the form UI generates one column of fields. There are scenarios where more control may be required over the layout. To enable these scenarios, it is possible to replace the default template with a custom template. A different template can be provided for each form mode.

Parent topic: Customizing Alfresco Share [1]

Use of forms in Share

Forms are used in the View Metadata and Edit Metadata pages within Share.

The following screen shot shows the form component on the Edit Metadata page.



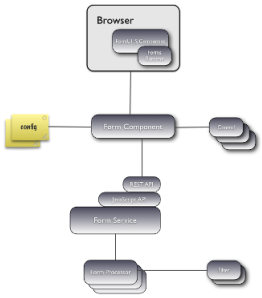
The content of the form is completely driven from configuration custom types, custom aspects. Their properties and associations can be displayed within Share.
Parent topic: Forms [12]

Forms architecture

The forms engine consists of four major parts:

- Form service
- Form component
- Form configuration
- JavaScript formUI component, which includes the forms runtime

The following diagram shows a high-level architecture diagram of the forms engine.



The forms runtime is responsible for the execution of a form. It manages all input, validation (client or call-back), events, submission, and it consists of a small, lightweight JavaScript library. An unobtrusive JavaScript pattern is used, where behavior is added to the HTML form elements when the page loads. The forms runtime provides the following capabilities:

- Mandatory property handling
- Validation (enforceable at submission, as the user types or when a field loses focus), which includes:
  - Regular expression matching
  - String length
  - Email address
  - Is number
  - Numeric range
  - Date range
  - List of values
- Repeating fields (for handling multi-valued properties)

Parent topic: Forms [12]

Forms event sequence

When a request is made to a page containing the form component, the following sequence of events occurs.

1. The form component looks up the form configuration for the item being requested.
2. The form component retrieves the list of fields to display (if any) and requests a form definition for those fields and the item from the form service via its REST API.
3. The form service looks for a form processor that can process the kind of item.
4. The form processor is asked to generate a form definition.
5. The form processor executes any registered filters before and after the main processing.
6. The REST API takes the result from the form processor/form service and constructs the form definition JSON response.
7. The form component receives the result from the form service and combines it with the form configuration to produce the form UI model.
8. The form component Freemarker template iterates around the fields and includes the relevant controls.
9. The form component Freemarker template instantiates the FormUI JavaScript component.
10. The FormUI JavaScript instantiates the forms runtime and registers all validation handlers.

For a description of the available form controls, refer to Forms reference [48]

At this point, the form is ready for the user to interact. When the user interacts with the form, the forms runtime constantly checks the validation rules enabling and disabling the Submit button appropriately. When the user submits the form, the following sequence of events occurs.

1. The browser submits the form by calling the REST API.
2. The form service looks for a form processor that can process the kind of item.
3. The form processor is asked to persist the form data.
4. The form processor executes any registered filters before and after the main processing.
5. The REST API takes the result from the form processor/form service and constructs the JSON response.
6. The browser receives the response and processes it appropriately.

Parent topic: Forms [12]

Configuring forms

The default forms configuration is specified in the <configRootShare>/classes/alfresco/share-form-config.xml file. This file contains all the default controls and constraint handlers for the Alfresco content model and the form configuration for the `cm:content` and `cm:folder` types. This file also contains an example of configuring the `cm:content` type.

You should apply all your forms customizations to a custom configuration file. To configure forms for the Share application, use the custom configuration file named share-config-custom.xml.

1. Open the <web-extension>/share-config-custom.xml file.
2. Modify the forms configuration settings using the XML configuration syntax.
3. Save the file without the .sample extension.

Parent topic: Forms [12]

Customizing forms controls

One of the most common customization is to add new controls. A control is the label for a field and the interface that the user interacts with for setting the value of the field.
A control is a Freemarker template snippet that includes the markup to define the control. The model for the template includes the field and form being generated, represented by a `field` and `form` object, respectively. The following snippet shows the structure of the `field` object, using the `cm:name` property as an example:

```
{
   kind : "field",
   id : "prop_cm_name",
   configName : "cm:name",
   name : "prop_cm_name",
   dataType : "d:text",
   type : "property",
   label : "Name",
   description : "Name",
   mandatory : true
   disabled : false,
   repeating : false,
   dataKeyName : "prop_cm_name",
   value : "plain-content.txt",
   control:
   {
      params: {},
      template : "controls/textfield.ftl"
   }
}
```

Although the `id` property provides a unique identifier for the field, it is only scoped to the current form. If there are multiple forms on the page containing the same field, this identifier will not be unique. The model property `fieldHtmlId` should be used as the identifier for the control, as this is guaranteed to be unique for the page.

The state of the `disabled` property must always be adhered to when implementing controls as this is driven from the field definition returned from the FormService and from the `read-only` attribute in the form configuration. If the `disabled` property is set to true, the control should never allow the value to be edited.

The control is also responsible for rendering an appropriate UI representation for the mode the form is currently in. The form mode can be retrieved from the `form.mode` property. A pattern used by most the out-of-the-box controls is shown below.

```
<#if form.mode == "view">
   // view representation goes here...
<#else>
   // edit and create representation goes here...
</#if>
```

The final rule for controls is that they must supply the field current value in a DOM element that has a `value` property and the `id` property set to the value of `fieldHtmlId` Freemarker variable.

For advanced controls, that is, association, date, period, and so on, this usually requires a hidden `form` field.
Parent topic: Forms [12]

Customizing the validation handler

A validation handler is a small JavaScript function that gets called by the forms runtime when a field value needs to be validated.
The interface for a validation handler is shown below.

```
/**
 * Validation handler for a field.
 *
 * @param field {object} The element representing the field the validation is for
 * @param args {object} Object containing arguments for the handler
 * @param event {object} The event that caused this handler to be called, maybe null
 * @param form {object} The forms runtime class instance the field is being managed by
 * @param silent {boolean} Determines whether the user should be informed upon failure
 * @param message {string} Message to display when validation fails, maybe null
 * @static
 */
function handler-name(field, args, event, form, silent, message)
```

The definition of the built in "mandatory" validation handler is shown below.

```
Alfresco.forms.validation.mandatory = function mandatory(field, args, event, form, silent, message)
```

The `field` parameter is usually the HTML DOM element representing the field's value, which is normally an HTML input DOM element, so that the value property can be accessed. The structure of the `args` parameter is dependent on the handler being implemented. By default, these will be the parameters of the constraint defined on the field.

The handler is responsible for taking the value from the field and uses the `args` parameter to calculate whether the current value is valid or not, returning `true` if it is valid and `false` if it is not.
Parent topic: Forms [12]

Displaying type metadata

The configuration to define the fields for the `cm:content` type exists in the system file called web-framework-config-commons.xml. You can configure the type metadata in the share-config-custom.xml file in <web-extension>. It is also possible to deploy custom configurations via JARs or AMPs.
The following snippet shows the forms definition in the share-config-custom.xml file.

```
<config evaluator="node-type" condition="cm:content">
   <forms>
      <form>
         <field-visibility>
            <show id="cm:name" />
            <show id="cm:title" force="true" />
            <show id="cm:description" force="true" />
            <show id="mimetype" />
            <show id="cm:author" force="true" />
            <show id="size" for-mode="view" />
            <show id="cm:creator" for-mode="view" />
            <show id="cm:created" for-mode="view" />
            <show id="cm:modifier" for-mode="view" />
            <show id="cm:modified" for-mode="view" />
         </field-visibility>
      </form>
   </forms>
</config>
```

The configuration defines that the `cm:name` property is visible in all modes, whereas the `cm:creator`, `cm:created`, `cm:modifier`, and `cm:modified` properties are visible in view mode only.

The `mimetype` and `size` properties are known as transient properties because they do not exist as properties in the model. These properties are formed from the `cm:content` property. The `NodeFormProcessor` knows about these properties and generates a field definition to represent them so that they will appear in the forms.

The `force` attribute ensures that the `NodeFormProcessor` searches the entire Alfresco content model for the property or association definition before returning anything.

1. Open the <web-extension>/share-config-custom.xml file.
2. Enter the configuration for custom types.
   The following example configuration shows the `my:text`, `my:dateTime` and `my:association` properties being configured for the custom `my:example` type.

   ```
   <config evaluator="node-type" condition="my:example">
      <forms>
         <form>
            <field-visibility>
               <show id="my:text" />
               <show id="my:dateTime" />
               <show id="my:association" />
            </field-visibility>
         </form>
      </forms>
   </config>
   ```

3. Add more fields to the default configuration.
   The following example shows how to show the node's `DBID` property for all `cm:content` nodes.

   ```
   <config evaluator="node-type" condition="cm:content">
      <forms>
        <form>
           <appearance>
              <field id="cm:description">
                 <control>
                    <control-param name="rows">20</value>
                    <control-param name="columns">20</value>
                 </control>
              </field>
           </appearance>
        </form>
     </forms>
   </config>
   ```

   Note: The full prefix version of the type is required in the `condition` attribute.
   The `force` attribute forces the `NodeFormProcessor` to search the entire Alfresco content model for the property or association definition before returning anything.
4. Save your file.

Parent topic: Forms [12]

Displaying aspect metadata

Add the properties and associations defined on aspects by adding them to the list of fields to show for a type. The aspects that appear can be defined on a type by type basis, and you can control the ordering of the fields.

1. Open the <web-extension>\share-config-custom.xml file.
   Note: While you can configure the aspect metadata by directly editing the share-config-custom.xml file in <web-extension>. It is also possible to deploy custom configurations via JARs or AMPs.
2. Enter the configuration for custom types.
   The following example configuration shows the `cm:from` and `cm:to` properties for the `cm:effectivity` aspect.

```
<config evaluator="node-type" condition="cm:content">
   <forms>
      <form>
         <field-visibility>
            <show id="cm:name" />
            <show id="cm:title" force="true" />
            <show id="cm:description" force="true" />
            <show id="mimetype" />
            <show id="cm:author" force="true" />
            <show id="size" for-mode="view" />
            <show id="cm:creator" for-mode="view" />
            <show id="cm:created" for-mode="view" />
            <show id="cm:modifier" for-mode="view" />
            <show id="cm:modified" for-mode="view" />

            <!-- cm:effectivity aspect -->
            <show id="cm:from"/>
            <show id="cm:to"/>
         </field-visibility>
      </form>
   </forms>
</config>
```

3. Add custom aspects to the default configuration by overriding the configuration.
   The following example shows how to add the fields of an example aspect to all forms for the `cm:content` type.

```
<config evaluator="node-type" condition="cm:content">
   <forms>
      <form>
         <field-visibility>
            <!-- fields from my example aspect -->
            <show id="my:aspectProperty" />
            <show id="my:aspectAssociation" />
         </field-visibility>
      </form>
   </forms>
</config>
```

   This will apply the aspects to all `cm:content` nodes.
4. Save the file.
5. It is also possible to have the fields appear for any type of node to which the aspect is applied. For example, you may wish to display the `my:aspectProperty` and `my:aspectAssociation` fields for any type of node to which the `my:customAspect` is applied:

```
<config evaluator="aspect" condition="my:customAspect">
   <forms>
      <form>
         <field-visibility>
            <!-- fields from my example aspect -->
            <show id="my:aspectProperty" />
            <show id="my:aspectAssociation" />
         </field-visibility>
      </form>
   </forms>
</config>
```

Parent topic: Forms [12]

Configuring a form control

Most of the built in controls have parameters, which allow some basic customization.

1. Open the <web-extension>\share-config-custom.xml file.
2. Change the number of rows and columns used for the `textarea` control that the `cm:description` field uses by default.

```
<config evaluator="node-type" condition="cm:content">
   <forms>
      <form>
         <appearance>
            <field id="cm:description">
               <control>
                  <control-param name="rows">20</value>
                  <control-param name="columns">80</value>
```

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

3. If all `textarea` controls in the application need to have these settings, configure the control in the `default-controls` element. For example:

```
<config evaluator="node-type" condition="cm:content">
   <forms>
      <default-controls>
         <type name="d:mltext">
            <control-param name="rows">20</control-param>
            <control-param name="columns">80</control-param>
         </type>
      </default-controls>
   </forms>
</config>
```

4. Save the file.

Parent topic: Forms [12]

Grouping fields

For longer forms, you can group fields together in logical grouped or nested sections.

1. Open the <web-extension>\share-config-custom.xml file.
2. Enter the configuration for custom types.
   The following example configuration shows how to group some fields from an imaginary custom `my:example` type.

```
<config evaluator="model-type" condition="my:example">
   <forms>
      <form>
         <field-visibility>
            <show id="cm:name" />
            <show id="my:text" />
            <show id="my:mltext" />
            <show id="my:boolean" />
            <show id="my:int" />
            <show id="my:long" />
            <show id="my:double" />
            <show id="my:float" />
            <show id="my:status" />
            <show id="my:restricted-string" />
            <show id="my:date" />
            <show id="my:dateTime" />
         </field-visibility>
         <appearance>
            <set id="text" appearance="fieldset" label="Text Fields" />
            <set id="number" appearance="panel" label="Number Fields" />
            <set id="date" appearance="fieldset" label="Date Fields" />

            <field id="cm:name" set="text" />
            <field id="my:text" set="text" />
            <field id="my:mltext" set="text" />
            <field id="my:boolean" set="text" />

            <field id="my:int" set="number" />
            <field id="my:long" set="number" />
            <field id="my:double" set="number" />
            <field id="my:float" set="number" />

            <field id="my:date" set="date" />
            <field id="my:dateTime" set="date" />
         </appearance>
      </form>
   </forms>
</config>
```

   Nested sets are also supported. Use the `parent` attribute in the `set` element. The following example configuration shows the fields of the `my:example` type in a nested set.

```
<config evaluator="model-type" condition="my:example">
   <forms>
      <form>
         <field-visibility>
            <show id="cm:name" />
            <show id="my:text" />
            <show id="my:mltext" />
            <show id="my:boolean" />
            <show id="my:int" />
            <show id="my:long" />
            <show id="my:double" />
            <show id="my:float" />
         </field-visibility>
         <appearance>
            <set id="builtin" appearance="fieldset" label="Built In" />
```

```
            <set id="custom" appearance="fieldset" label="Custom Data" />
            <set id="text" parent="custom" appearance="panel" label="Text" />
            <set id="number" parent="custom" appearance="panel" label="Numbers" />

            <field id="cm:name" set="builtin" />

            <field id="my:text" set="text" />
            <field id="my:mltext" set="text" />
            <field id="my:boolean" set="text" />

            <field id="my:int" set="number" />
            <field id="my:long" set="number" />
            <field id="my:double" set="number" />
            <field id="my:float" set="number" />
         </appearance>
      </form>
   </forms>
</config>
```

3. Save the file.

Parent topic: Forms [12]

Changing the default set label

Fields that do not specify a set belong to the implicit `default` set. They are rendered together, by default, but without any visual grouping.

1. Open the <web-extension>\share-config-custom.xml file.
2. Enter the configurations for the set label.
   The appearance of the default set can be controlled in the same way as other sets, for example, using an identifier of an empty string.

   ```
   <set id="" appearance="panel" />
   ```

   This will render a panel around all the fields with a label of Default.
   To specify a different label, add the `label` attribute. For example, the following label will be General.

   ```
   <set id="" appearance="panel" label="General" />
   ```

   You can also use a message bundle key.

   ```
   <set id="" appearance="panel" label-id="form.set.general" />
   ```

3. Save the file.

Parent topic: Forms [12]

Providing a custom form control

If none of the out-of-the-box controls are sufficient, you can add new controls and reference them. Controls are Freemarker template snippets, therefore, they contain only the HTML markup required to represent the control. The templates need to be stored in the site-webscripts directory, which will usually be in the application server shared classpath.

- The following example configuration shows a very simple custom text field control that always displays with a green background, white text, and 700 pixels wide. For a production system, use a CSS class; however, this example shows a hard coded style.

  ```
  <div class="form-field">
     <#if form.mode == "view">
        <div class="viewmode-field">
           <span class="viewmode-label">${field.label?html}:</span>
           <span class="viewmode-value">${field.value?html}</span>
        </div>
     <#else>
        <label for="${fieldHtmlId}">${field.label?html}:<#if field.mandatory><span class="mandatory-indicator">*</span></#if></label>
        <input id="${fieldHtmlId}" type="text" name="${field.name}" value="${field.value}"
                     style="background-color: green; color: white; width: 700px;" <#if field.disabled>disabled="true"</#if> />
     </#if>
  </div>
  ```

- The following example configuration shows this control being used for the `cm:name` property, with a file name of my-textfield.ftl.

  ```
  <config evaluator="node-type" condition="cm:content">
     <forms>
        <form>
           <appearance>
              <field id="cm:name">
                 <control template="/my-textfield.ftl" />
              </field>
           </appearance>
        </form>
     </forms>
  </config>
  ```

Parent topic: Forms [12]

Changing the field label position

By default, forms are rendered with the field labels positioned above the input control.

To change this layout, provide a custom CSS to override the default style rules. Control dependencies can be provided via custom configuration.

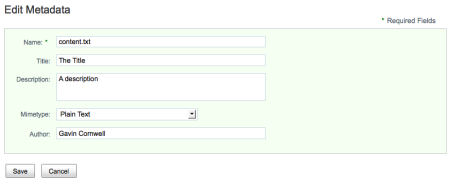
1. Add the custom CSS in the custom-label-layout.css file, located in the /custom/forms directory within the web application.
2. Add the following configuration:

```
<config>
   <forms>
      <dependencies>
         <css src="/custom/forms/custom-label-layout.css" />
      </dependencies>
   </forms>
</config>
```

3. To move the labels to the left of the input control, the following CSS should be present in the custom-label-layout.css file:

```
.form-container label
{
   display: inline;
   float: left;
   text-align: right;
   width: 6em;
   margin-right: 1em;
   margin-top: 0.4em;
}
```

4. Save the file.

   The result of this customization is shown as: 

Parent topic: Forms [12]

Providing a custom form template

The default template that renders the form UI generates one column of fields. There are scenarios where more control may be required over the layout. To enable these scenarios, it is possible to replace the default template with a custom template. A different template can be provided for each form mode.

Store the custom templates in the site-webscripts directory, which is usually be in the application server shared classpath.

1. The example below shows the Edit form for the standard `cm:content` type being configured to render with two columns of fields.

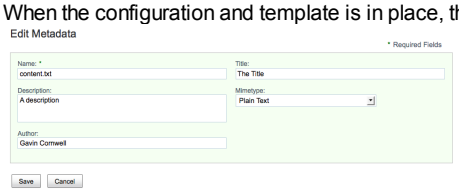
```
<config evaluator="node-type" condition="cm:content">
   <forms>
      <form>
            <edit-form template="/2-column-edit-form.ftl" />
      </form>
   </forms>
</config>
```

   The example template 2-column-edit-form.ftl is available in the distribution in the samples folder.
   The following example shows the contents of the 2-column-edit-form.ftl file. It uses some of the Freemarker macros available in form.lib.ftl but supplies its own `renderSetWithColumns` macro to render the HTML required to create the grid using the YUI grid CSS capabilities.

```
<#import "/org/alfresco/components/form/form.lib.ftl" as formLib />

<#if error?exists>
   <div class="error">${error}</div>
<#elseif form?exists>

   <#assign formId=args.htmlid + "-form">
   <#assign formUI><#if args.formUI??>${args.formUI}<#else>true</#if></#assign>

   <#if formUI == "true">
      <@formLib.renderFormsRuntime formId=formId />
   </#if>
```

```
   <div id="${formId}-container" class="form-container">

      <#if form.showCaption?exists && form.showCaption>
         <div id="${formId}-caption" class="caption"><span class="mandatory-indicator">*</span>${msg("form.required.fields")}</div>
      </#if>

      <#if form.mode != "view">
         <form id="${formId}" method="${form.method}" accept-charset="utf-8" enctype="${form.enctype}" action="${form.submissionUrl}
      </#if>

      <div id="${formId}-fields" class="form-fields">
        <#list form.items as item>
            <#if item.kind == "set">
               <@renderSetWithColumns set=item />
            <#else>
               <@formLib.renderField field=item />
            </#if>
         </#list>
      </div>

      <#if form.mode != "view">
         <@formLib.renderFormButtons formId=formId />
         </form>
      </#if>

   </div>
</#if>

<#macro renderSetWithColumns set>
   <#if set.appearance?exists>
      <#if set.appearance == "fieldset">
         <fieldset><legend>${set.label}</legend>
      <#elseif set.appearance == "panel">
         <div class="form-panel">
            <div class="form-panel-heading">${set.label}</div>
            <div class="form-panel-body">
      </#if>
   </#if>

   <#list set.children as item>
      <#if item.kind == "set">
         <@renderSetWithColumns set=item />
      <#else>
         <#if (item_index % 2) == 0>
         <div class="yui-g"><div class="yui-u first">
         <#else>
         <div class="yui-u">
         </#if>
         <@formLib.renderField field=item />
         </div>
         <#if ((item_index % 2) != 0) || !item_has_next></div></#if>
      </#if>
   </#list>

   <#if set.appearance?exists>
      <#if set.appearance == "fieldset">
         </fieldset>
      <#elseif set.appearance == "panel">
            </div>
         </div>
      </#if>
   </#if>
</#macro>
```

2. When the configuration and template is in place, the Edit Metadata page for a `cm:content` type in Share has the following appearance.



Parent topic: Forms [12]

Customizing Alfresco Explorer

Alfresco Explorer provides a web-based user interface providing document management, collaboration, and administration capabilities.

There are a number of ways of customizing Alfresco Explorer using simple configuration or more programmatic extension.

- **Customizing Alfresco Explorer configuration items** [49] There are several different ways that you can customize the Explorer configuration items. The Explorer configuration file is called web-client-config-custom.xml.
- **Alfresco Explorer configuration settings** [50] This topic describes some of the configuration settings that can be used to modify the behavior of Alfresco Explorer.

Parent topic: Customizing Explorer and Share [13]

Customizing Alfresco Explorer configuration items

There are several different ways that you can customize the Explorer configuration items. The Explorer configuration file is called web-client-config-custom.xml.

- Modify the Explorer configuration file in the <extension> directory.
  1. Open the <extension>\web-client-config-custom.xml file.
  2. Uncomment any <config> items that you want to enable.
  3. Save the file.
- Modify the Explorer configuration file in the repository.
  1. Browse to the following space: Company Home > Data Dictionary > Web Client Extensions.
  2. Open the web-client-config-custom.xml file.
  3. Uncomment any <config> items that you want to enable.
  4. Save the file.
     Use this method to edit a file directly in the repository and then to reload it using the Explorer. This means that you do not have to restart the server and applies versions to the configuration changes.
- Modify the Explorer configurations within an AMP file.
  1. Open the module-context.xml file.
  2. Add the following bean definition:

```
<bean id="myModule_configBootstrap" class="org.alfresco.web.config.WebClientConfigBootstrap" init-method="init">
     <property name="configs">
       <list>
          <value>classpath:alfresco/module/myModuleId/my-web-client-custom.xml</value>
       </list>
     </property>
  </bean>
```

  3. Save the file.

Parent topic: Customizing Alfresco Explorer [2]

Alfresco Explorer configuration settings

This topic describes some of the configuration settings that can be used to modify the behavior of Alfresco Explorer.
The default settings for Explorer are defined in the <configRoot>\classes\alfresco\web-client-config.xml file. To override the default settings and activate the Explorer configurations, add your settings to the <extension>\web-client-config-custom.xml file.

1. Open the <extension>\web-client-config-custom.xml.sample file.
2. Add your preferred settings to configure Alfresco Explorer.
   For example, you can set the following:

   Property Description <initial-password>admin</initial-password> Sets the initial password for the Alfresco Administrator user (admin) to a password of admin. <user-group-admin>true</user-group-admin> Specifies whether to allow user group administration by an admin user. You can set this to false if you only ever use external user and group control, such as synchronized LDAP. <allow-user-config>true</allow-user-config> Specifies whether to allow users to modify their personal settings in the user console. Set to false to prevent users from changing their passwords or configuring their person settings. <zero-byte-file-upload>false</zero-byte-file-upload> Set to false to prevent empty (zero byte) files from being uploaded. Set to true to be able to upload empty (zero byte) files. <breadcrumb-mode>path</breadcrumb-mode> Sets the default path view or a location view. By default, the breadcrumbs in Explorer show the history based on where you have visited. This will show the full path to a space rather than a visit history. <edit-link-type>http</edit-link-type> Sets the edit link type to use for online editing. The default is inline editable. The options are http or webdav. Due to heightened security in recent browser versions, it is not advised to use CIFS for online editing. CIFS is not a supported value in the <edit-link-type> setting. <tasks-completed-max-results>100</tasks-completed-max-results> Sets the limit for the number of completed tasks to display. <default-home-space-path>/app:company_home/app:user_homes</default-home-space-path> Specifies the path starting point when creating or finding home folders for new users. <home-space-permission>Consumer</home-space-permission> Specifies the default permissions to apply to a new users home space when it is first created. This permission is for other users attempting to access that home space. Set to Consumer or an empty value to indicate a private hidden space. For the allowed values, see org.alfresco.service.cmr.security.PermissionService. <initial-location>myalfresco</initial-location> Specifies the default location to display when the browse screen is first shown. This value can be myalfresco, userhome, companyhome, or guesthome. <allow-guest-config>false</allow-guest-config> Set to true allow the Guest user to configure the start location preferences. <clipboard-status-visible>true</clipboard-status-visible> Specified that a status message displays when an item is added to the clipboard. <paste-all-and-clear>true</paste-all-and-clear> Specified that the paste all action clears the clipboard. <cifs-url-suffix>.alfresco.org</cifs-url-suffix> Specifies the domain suffix that is appended to the CIFS URL host name. The default is .alfresco.org.

Property	Description
`<initial-password>admin</initial-password>`	Sets the initial password for the Alfresco Administrator user (`admin`) to a password of `admin`.
`<user-group-admin>true</user-group-admin>`	Specifies whether to allow user group administration by an admin user. You can set this to false if you only ever use external user and group control, such as synchronized LDAP.
`<allow-user-config>true</allow-user-config>`	Specifies whether to allow users to modify their personal settings in the user console. Set to false to prevent users from changing their passwords or configuring their person settings.
`<zero-byte-file-upload>false</zero-byte-file-upload>`	Set to false to prevent empty (zero byte) files from being uploaded. Set to true to be able to upload empty (zero byte) files.
`<breadcrumb-mode>path</breadcrumb-mode>`	Sets the default `path` view or a `location` view. By default, the breadcrumbs in Explorer show the history based on where you have visited. This will show the full path to a space rather than

	a visit history.
`<edit-link-type>http</edit-link-type>`	Sets the edit link type to use for online editing. The default is inline editable. The options are `http` or `webdav`. Note: Due to heightened security in recent browser versions, it is not advised to use CIFS for online editing. CIFS is not a supported value in the `<edit-link-type>` setting.
`<tasks-completed-max-results>100</tasks-completed-max-results>`	Sets the limit for the number of completed tasks to display.
`<default-home-space-path>/app:company_home/app:user_homes</default-home-space-path>`	Specifies the path starting point when creating or finding home folders for new users.
`<home-space-permission>Consumer</home-space-permission>`	Specifies the default permissions to apply to a new users home space when it is first created. This permission is for other users attempting to access that home space. Set to `Consumer` or an empty value to indicate a private hidden space. For the allowed values, see `org.alfresco.service.cmr.security.PermissionService`.
`<initial-location>myalfresco</initial-location>`	Specifies the default location to display when the browse screen is first shown. This value can be `myalfresco`, `userhome`, `companyhome`, or `guesthome`.
`<allow-guest-config>false</allow-guest-config>`	Set to true allow the Guest user to configure the start location preferences.
`<clipboard-status-visible>true</clipboard-status-visible>`	Specified that a status message displays when an item is added to the clipboard.
`<paste-all-and-clear>true</paste-all-and-clear>`	Specified that the paste all action clears the clipboard.
`<cifs-url-suffix>.alfresco.org</cifs-url-suffix>`	Specifies the domain suffix that is appended to the CIFS URL host name. The default is `.alfresco.org`.

Other settings that you can modify include the language selection, search controls, and the minimum length of user names, passwords, and group names. These settings are shown in the following tables.

Language setting	Description
`<language-select>true</language-select>`	Sets the language selection from the login window. Set to false to select the language from the client browser locale and the language drop-down is not displayed.
`<languages>`	Shows the list of available language files that are displayed in the login window. Add or remove language entries `<language locale="XX_YY">LangName</language>`. For example: `<language locale="ja_JP">Japanese</language>`.

Search settings	Description
`<search-minimum>3</search-minimum>`	Specifies the minimum number of characters required for a valid search string.
`<search-and-terms>false</search-and-terms>`	Set to true to enable `AND` text terms for simple/advanced search.
`<search-max-results>500</search-max-results>`	Specified the limit for search results. Set to -1 for unlimited results.
`<selectors-search-max-results>500</selectors-search-max-results>`	Specifies the limit for search results within selectors. Set to -1 for unlimited results.
`<invite-users-max-results>500</invite-users-max-results>`	Specifies the limit for search results within the invite users wizard. Set to -1 for unlimited results.

Minimum length settings	Description
`<username-min-length>2</username-min-length>`	Specifies the minimum length for username.
`<password-min-length>3</password-min-length>`	Specifies the minimum length for password.
`<group-name-min-length>3</group-name-min-length>`	Specifies the minimum length for group name.

3. Save the <extension>\web-client-config-custom.xml.sample file without the .sample extension.

Parent topic: Customizing Alfresco Explorer [2]

Source URL: http://docs.alfresco.com/4.2/concepts/ch-customize.html

Links:

[1] http://docs.alfresco.com/../concepts/share-customizing-intro.html
[2] http://docs.alfresco.com/../concepts/dev-explorer.html
[3] http://docs.alfresco.com/../concepts/dev-for-developers.html
[4] http://docs.alfresco.com/dev-extensions-share.html
[5] http://docs.alfresco.com/../concepts/share-configuration-files.html
[6] http://docs.alfresco.com/../tasks/share-customizing-custom-config-file.html
[7] http://docs.alfresco.com/../tasks/share-change-port.html
[8] http://docs.alfresco.com/../concepts/share-policies.html
[9] http://docs.alfresco.com/../tasks/usernametypes-mix-config.html
[10] http://docs.alfresco.com/../concepts/share-repodoclib.html
[11] http://docs.alfresco.com/../concepts/themes-intro.html
[12] http://docs.alfresco.com/../concepts/forms-intro.html
[13] http://docs.alfresco.com/../concepts/ch-customize.html
[14] http://docs.alfresco.com/../concepts/csfr-policy.html
[15] http://docs.alfresco.com/../concepts/iframe-policy.html
[16] http://docs.alfresco.com/../concepts/security-policy.html
[17] http://docs.alfresco.com/../tasks/share-repodoclib-config.html
[18] http://docs.alfresco.com/../tasks/share-repodoclib-hide.html
[19] http://docs.alfresco.com/../concepts/Share-Doclib-Extend-Intro.html
[20] http://docs.alfresco.com/../concepts/doclib-repository-tier.html
[21] http://docs.alfresco.com/../concepts/doclib-web-tier.html
[22] http://docs.alfresco.com/../concepts/doclib-override-extension-examples.html
[23] http://docs.alfresco.com/../concepts/doclib-client-side-template-and-action-extensions.html
[24] http://docs.alfresco.com/../concepts/share-customizing-document-library-views.html
[25] http://docs.alfresco.com/../concepts/doclib-reference.html
[26] http://docs.alfresco.com/doclib-jsNode-reference.html
[27] http://docs.alfresco.com/EXIF-renderer-source-code.html%23Share-doclib-EXIF-renderer-source-code
[28] http://docs.alfresco.com/../concepts/doclib-jsNode-reference.html
[29] http://docs.alfresco.com/../concepts/doclib-predefined-evaluators-reference.html
[30] http://docs.alfresco.com/../concepts/EXIF-renderer-source-code.html
[31] http://docs.alfresco.com/../tasks/themes-select.html
[32] http://docs.alfresco.com/../tasks/themes-create.html
[33] http://docs.alfresco.com/../tasks/themes-edit.html
[34] http://docs.alfresco.com/../concepts/forms-use.html
[35] http://docs.alfresco.com/../concepts/forms-mechanism.html
[36] http://docs.alfresco.com/../concepts/forms-evensequ.html
[37] http://docs.alfresco.com/../tasks/forms-config.html
[38] http://docs.alfresco.com/../tasks/forms-controls-custom.html
[39] http://docs.alfresco.com/../tasks/forms-valhandler.html
[40] http://docs.alfresco.com/../tasks/forms-type-display.html
[41] http://docs.alfresco.com/../tasks/forms-aspect-display.html
[42] http://docs.alfresco.com/../tasks/forms-formcontrol-config.html
[43] http://docs.alfresco.com/../tasks/forms-grouping-fields.html
[44] http://docs.alfresco.com/../tasks/forms-setlabel-change.html
[45] http://docs.alfresco.com/../tasks/forms-custom-formcontrol.html
[46] http://docs.alfresco.com/../tasks/forms-fieldlable-change.html
[47] http://docs.alfresco.com/../tasks/forms-custom-formtemplate.html
[48] http://docs.alfresco.com/forms-reference.html
[49] http://docs.alfresco.com/../tasks/webclient-customize.html
[50] http://docs.alfresco.com/../tasks/explorer-config-settings.html