Published on *Marco de Desarrollo de la Junta de Andalucía* (http://127.0.0.1/servicios/madeja)

Alfresco

Código: ARQ_ALFRESCO

El contenido de este área ofrece las pautas para el desarrollo de aplicaciones que usen el gestor documental Alfresco.

Código	Título	Tipo	Carácter
PAUT-0023	Opciones de desarrollo con Alfresco	Consejo	

Uso de Alfresco desde Terceras Aplicaciones

Código: ASIALF_UATA

El gestor de contenidos Alfresco contempla un conjunto de casos de uso de negocio, que soporta a través de servicios web de acceso al repositorio remoto para aplicaciones y procesos de negocio.

Código	Título	Tipo	Carácter
PAUT-0029	Configuración de acceso al Repositorio	Directriz	Obligatoria
PAUT-0024	Construccion de una capa de acceso personalizada	Directriz	Recomendada
PAUT-0025	Limitación de registros devueltos	Directriz	Recomendada

Código	Título	Tipo	Carácter
RECU-0041	Ejemplos de uso de servicos web	Ejemplo	Recomendado
RECU-0040	Servicios Web de Alfresco	Manual	Recomendado
RECU-0004	Servicios Web en Alfresco	Ficha	Recomendado
RECU-0043	Tipos de datos para los servicios web	Manual	Recomendado
RECU-0045	Uso de Web Scripts	Manual	Recomendado
RECU-0046	Ejemplos de Web Scripts	Ejemplo	Recomendado
RECU-0005	Web Scripts	Ficha	Recomendado
RECU-0047	API de JavaScripts	Manual	Recomendado
RECU-0048	Scripts de Ejemplo	Ejemplo	Recomendado
RECU-0006	API JavaScript	Ficha	Recomendado

Ejemplos Ampliados de Acceso a Alfresco

Código: EjemplosAlfresco

Se han recopilado ejemplos de acceso al repositorio de Alfresco para facilitar el desarrollo de aplicaciones donde se requiera este tipo de vínculo con el gestor de contenidos. Los ejemplos se han estructurado por lenguajes, funcionalidades y uso avanzado.

Estructuras por Lenguajes

Interfaces	
Java API	Consulta de Documentos
	Gestión de Documentos
	Consulta de Espacios
	Gestión de Espacios
	Búsquedas
	Gestión Integrada de Usuarios
	Categorías y Aspectos
	Reglas y Procedimientos
Java JCR API	Consulta de Documentos
	Gestión de Documentos
	Consulta de Espacios
	Gestión de Espacios

	Búsquedas
	Gestión Integrada de Usuarios
	Categorías y Aspectos
	Reglas y Procedimientos
Java Script	Consulta de Documentos
	Gestión de Documentos
	Consulta de Espacios
	Gestión de Espacios
	Búsquedas
	Gestión Integrada de Usuarios
	Categorías y Aspectos
	Reglas y Procedimientos
Web Script	Consulta de Documentos
	Gestión de Documentos
	Consulta de Espacios
	Gestión de Espacios
	Búsquedas
	Gestión Integrada de Usuarios
	Categorías y Aspectos
	Reglas y Procedimientos

Estructura por Funcionalidades

Uso Básico

Tabla de Funcionalidades		
Documentos	Consulta	Java API
		Java JCR API
		JavaScript
		WebScript
	Gestión	Java API
		Java JCR API
		JavaScript
		WebScript
Espacios	Consulta	Java API
		Java JCR API
		JavaScript
		WebScript
	Gestión	Java API
		Java JRC API
		JavaScript
		WebScript

Uso Avanzado

Tabla de funcionalidades	
Búsquedas	Java API
	Java JCR API
	JavaScript
	WebScript
Gestión Integrada de Usuarios	Java API
	Java JCR API
	JavaScript
	WebScript
Categorías y Aspectos	Java API
	Java JCR API
	JavaScript
	WebScript

Reglas y procedimientos	Java API
	Java JCR API
	JavaScript
	WebScript

Código	Título	Tipo	Carácter
RECU-0056	Consulta de Documentos con Java API	Ejemplo	Recomendado
RECU-0062	Gestion de los documentos con Java API	Ejemplo	Recomendado
RECU-0057	Consulta de Espacios con Java API	Ejemplo	Recomendado
RECU-0058	Gestión de Espacios con Java API	Ejemplo	Recomendado
RECU-0059	Búsquedas con Java API	Ejemplo	Recomendado
RECU-0060	Gestión de Usuarios con Java API	Ejemplo	Recomendado
RECU-0061	Categorías y Aspectos con Java API	Ejemplo	Recomendado
RECU-0063	Reglas y Procedimientos con Java API	Ejemplo	Recomendado
RECU-0064	Consulta de Documentos con Java JCR API	Ejemplo	Recomendado
RECU-0065	Gestión de Documentos con Java JCR API	Ejemplo	Recomendado
RECU-0066	Consulta de Espacios con Java JCR API	Ejemplo	Recomendado
RECU-0067	Gestión de Espacios con Java JCR API	Ejemplo	Recomendado
RECU-0068	Búsquedas con Java JCR API	Ejemplo	Recomendado
RECU-0069	Consulta de Documentos con JavaScript	Ejemplo	Recomendado
RECU-0070	Gestión de Documentos con JavaScript	Ejemplo	Recomendado
RECU-0071	Consulta de Espacios con JavaScript	Ejemplo	Recomendado
RECU-0072	Gestión de Espacios con JavaScript	Ejemplo	Recomendado
RECU-0073	Búsquedas con JavaScript	Ejemplo	Recomendado
RECU-0074	Gestion de Usuarios con JavaScript	Ejemplo	Recomendado
RECU-0075	Categorías y Aspectos con JavaScript	Ejemplo	Recomendado
RECU-0076	Reglas y Procedimientos con JavaScript	Ejemplo	Recomendado
RECU-0077	Consulta de Documentos con WebScript	Ejemplo	Recomendado
RECU-0078	Gestión de Documentos con WebScript	Ejemplo	Recomendado
RECU-0079	Consulta de Espacios con WebScript	Ejemplo	Recomendado
RECU-0080	Gestión de Espacios con WebScript	Ejemplo	Recomendado
RECU-0081	Búsquedas con WebScript	Ejemplo	Recomendado
RECU-0082	Gestion de Usuarios con WebScript	Ejemplo	Recomendado
RECU-0083	Categorías y Aspectos con WebScript	Ejemplo	Recomendado
RECU-0084	Reglas y Procedimientos con WebScript	Ejemplo	Recomendado

Extensión de Alfresco

Código: ASIALF_EA

El gestor de contenidos Alfresco ofrece la posibilidad de ampliar sus funcionalidades y características básicas mediante extensiones de su modelo de contenidos y de sus servicios. Se han establecido directrices respecto a estas extensiones y a su despliegue.

Código	Título	Tipo	Carácter
PAUT-0026	Extension de clases base	Directriz	Obligatoria
PAUT-0027	Metodo de despliegue de extensiones en Alfresco	Directriz	Recomendada
LIBP-0002	Desarrollo del modelo de contenidos	Directriz	Obligatoria

Código	Título	Tipo	Carácter
RECU-0049	Métodos para el despliegue de extensiones de Alfresco	Manual	Recomendado
RECU-0050	Extensiones en el modelo de contenidos	Manual	Recomendado
RECU-0051	Creación de acciones personalizadas	Manual	Recomendado
RECU-0052	Comportamientos	Manual	Recomendado
RECU-0053	Desarrollo de extracción de datos	Manual	Recomendado
RECU-0054	Workflows	Manual	Recomendado

RECU-0055	Nuevos Objetos para motor JavaScript	Manual	Recomendado

Source URL: http://127.0.0.1/servicios/madeja/contenido/subsistemas/arquitectura/alfresco

Opciones de desarrollo con Alfresco

- **Área:** Alfresco
- **Tipo de pauta:** Consejo

Código: PAUT-0023

Alfresco proporciona muchas opciones para interactuar con el repositorio a traves de una aplicación externa. En el momento de escoger una u otras es importante que se analice cual de ellas se ajusta mas a las necesidades del proyecto en el que se este trabajando.

Con el objetivo de ayudar en este analisis se va a ofrecer una breve descripción de cada una de estas opciones indicando sus prinicipales caracteristicas.

Incrustar el Repositorio (Alfresco foundation API)

El repositorio de Alfresco puede ser incrustado dentro de una apliación propia. Esta aproximación nos da acceso total a la API del repositorio de Alfresco. Esta es una API Java local para las comunicaciones con el repositorio. Esta interfaz es la mas nativa para la conectividad con Alfresco. Puede usar también esta interfaz también cuando su aplicación esta ligada con la tecnología de implementación de Alfresco (Spring) y sus interfaces.

A favor

- Interfaz nativa para el repositorio (proporcionara el mejor rendimiento)

En contra

- Arquitectura muy acoplada
- No es una interfaz distribuida
- Interfaz especifca de Alfresco

Direccionalidad URL (URL Adressability)

Una aplicación externa puede realiazar una llamada XMLHttpRequest a Alfresco que especifica una referencia a un nodo (que son accesible via URL) y una referencia a una plantilla FreeMarker en el servidor. Alfresco procesara la plantilla en el contexto del nodoy devolvera el resultado.

A favor

- Facilidad de implementación

En contra

- Utilidad muy especifica
- Escasa potencia

Web Scripts (RESTFUL API, Web Scripts Framework)

Los Web scripts proveen al repositorio de Alfresco con servicios de contenido que son accesibles desde cualquier parte (consultar el repositorio, extraer contenido y alterar el comportamiento del repositorio), expone el repositorio para gestion de documentos y contenidos web, proporciona medios de busqueda personalizados. Cada web script estara ligado a un metodo HTTP y a una URL personalizada. Se puede construir una libreria de URLs que proporcione una API HTTP completa (RESTful or STRESTful) HTTP API. Las Librerias pueden ser instaladas facilmente en cualquier servidor Alfresco. La API Restful de Alfresco esta en construcción y aún carece de muchas de las funcionalidades del sistema.

A favor

- Convierte Alfresco en un servidor HTTP destinado a la gestión de contenidos
- Construye facilmente Servicios Web para la gestion de contenidos accesibles via HTTP
- Accede, gestiona y enlaza facilmete los contenidos a traves de una RESTful API hecha a medida.
- No se necesita amplios conocimientos de Java ni herramientas complejas.

En contra

- Interfaz por defecto con muy poca funcionalidad aunque esta previsto añadirla toda
- La interfaz por defecto es especifica de Alfresco
- Tecnología nueva no accesible antes de la versión 2.1

Servicios Web (Alfresco Web Services API)

Alfresco proporciona acceso remoto a traves de una API de Servicios Web basada en SOAP. La API de Servicios Web esta especificamente diseñada para reducir la cantidad de comunicación con el servidor lo que la hace una manera bastante eficiente de acceso remoto. En algunos casos la Api propia de Alfresco no da un buen rendimiento en cuestión de escalabilidad y algunos clientes optan por escribir sus propios Servicios. Los inconvenientes a la utilización de la API de servicios web de Alfresco es el acoplamiento con la API especifica de Alfresco y los fuertes requerimientos en el lado del cliente.

A favor

- Implementada sobre la API del repositorio de alfresco para obtener el mejor rendimiento.
- Interfaz optimizada para transacciones distribuidas
- Interfaz remota mas compatible

En contra

- La Interfaz es especifica de Alfresco
- Fuertes requerimientos en la parte cliente.

Interfaz JCR de Alfresco (JCR API)

La interfaz JCR de Alfresco JCR muestra el repositorio de Alfresco tras un adaptador de compatibilidad con el estandar JCR lo que permite comunicarse con Alfresco con una aproximación basanda en un estandar. El estandar JCR no cubre toda la funcionalidad que Alfresco así que normalmente se combina con una de las otras aproximaciones. La interfaz JCR de Alfresco es una interfaz local se puede ser adquirida a traves de interaccion Spring (Opcionalmente la configuración spring se puede extender para vincular la interfaz JCR con JNDI).

A favor

- JCR es una especificación creada y soportada por la industria

En contra

- CR es una nueva especificacion y esta todavia en plena evolución
- El adaptador JCR puede que no rinda el mismo nivel que la interfaz nativa de Alfresco.
- La interfaz JCR de Alfresco no es una interfaz distribuida
- Capacidades de Alfresco no cubiertas

Extension JCR RMI de Alfresco

La extension JCR RMI envuelve la implementación Alfresco JCR en una capa remota (mediante el uso de Apache Jackrabbit), proporcionando acceso remoto a la api JCR de Alfresco JCR. La interfaz JCR no ha sido optimizada para un entorno remoto/distribuido. Esto significa que realiza que realiza multitud de pequeñas llamadas a traves de la red lo que puede constituir un problema de rendimiento. RMI se basa en la serializacion de objectos .. algunos tipos de contenidos tienen propiedades nodos que pueden ser dificiles de manejar, convirtiendose en un verdadero problema.

A favor

- RMI proporciona una interfaz de programación nativa facil de usar
- Rapida implementacion remota
- JCR es interfaz basado en un estandar industrial

En contra

- RMI es problematico para en la interacción con firewall.
- RMI es especifico de java
- Puede provocar problemas de rendimiento.
- Problemas de serialización de onjetos complejos.

Configuración de acceso al Repositorio

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Tipo de pauta:** Directriz
- **Carácter de la pauta:** Obligatoria

Código: PAUT-0029

Para poder acceder a Alfresco desde nuestras aplicaciones necesimos definir con que usuario se realizará la conexión. Este usuario, por razones de seguridad, no debe ser nunca el usuario administrador de Alfresco.

Lo ideal es crear un usuario con permisos suficientes para realizar las operaciones que necesita la aplicación montada por encima y usarlo para la conexión. Si no fuera suficiente con permisos que ofrecen los roles que se implementan en Alfresco por defecto es posible que haya que modificar el fichero de configuración de los mismos (permissionDefinitions.xml) para añadir el modelo de permisos que necesitemos.

Es recomendable ademas que las aplicaciones usen siempre un espacio de trabajo propio como espacio raíz para todas sus operaciones. Esto facilita el uso compartido del repositorio por varias aplicaciones.

Con esta configuración de usuarios genericos y espacios propios la compartición de información entre aplicaciones se puede realizar de manera sencilla mediante la concesión de permisos de lectura sobre elementos del propio espacio de trabajo a usuarios genericos de otras apllicaciones.

Construccion de una capa de acceso personalizada

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Tipo de pauta:** Directriz
- **Carácter de la pauta:** Recomendada

Código: PAUT-0024

Para facilitar el uso del repositorio es recomensable crear una capa por encima sobre el cliente de servicios web de Alfresco que nos proporcione una API de fácil manejo y personalizada para nuestra aplicación.

Esta API debería agrupar las operaciones más básicas y permitir así la integración de forma mucho más sencilla.

Lo ideal es que mediante esta API se puedan realizar operaciones atómicas como:

1. Crear, modificar, consultar y eliminar espacios o carpetas
2. Crear, modificar, consultar y eliminar documentos
3. Crear, modificar, consultar y eliminar versiones de documentos
4. Proteger, actualizar y desproteger versiones de documentos
5. Crear, modificar, consultar y eliminar categorías
6. Asignar categorías a documentos y carpetas
7. Crear, modificar, consultar y eliminar usuarios
8. Asignar y revocar permisos a usuarios sobre carpetas y documentos
9. Mover, copiar y enlazar documentos y carpetas

La construcción de esta API nos facilita un medio de asegurar que el ciclo típico de un servicio web (Autenticar, Operar y Cerrar Sesión) es un proceso atomico. Esto es necesario para impedir confusiones de identidad durante la ejecución de varias llamadas simultáneas a un servicio web.

Limitación de registros devueltos

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Tipo de pauta:** Directriz
- **Carácter de la pauta:** Recomendada

Código: PAUT-0025

Cuando se usan Web Services para realizar consultas en Alfresco es recomendable limitar el número de registros que devolverá la petición.

De esta manera se evitará tener que recoger resultado con un tamaño demasiado grande para que se puedan manejar correctamente. Esta limitación se realiza estableciendo la propiedad fetchSize del elemento Query Header y colocando este directamente en la cabecera SOAP:

```
// establecemos el tamaño del fetchSize en el elemento Query Header
int Tamanno = 10;
QueryConfiguration queryCfg = new QueryConfiguration();
queryCfg.setFetchSize(Tamanno);
//Insertamos Query Header en cabecera SOAP
RepositoryServiceSoapBindingStub repositoryService = WebServiceFactory.getRepositoryService();
repositoryService.setHeader(new RepositoryServiceLocator().
                GetServiceName().
                 getNamespaceURI(),
                 "QueryHeader",
                 queryCfg);
//recogemos los primeros resultados
QueryResult resultado = repositoryService.query(STORE, getQuery(sConsulta), true);
//procesamos el primer resultado
String querySession = resultado.getQuerySession();
while (querySession != null) {
   // recogemos los siguientes conjunto de resultados
   resultado = repositoryService.fetchMore(querySession);
   // procesamos los siguientes conjuntos de resultados.
   querySession = resultado.getQuerySession();
}
```

Ejemplos de uso de servicos web

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Carácter del recurso:** Recomendado

Código: RECU-0041
Tipo de recurso: Ejemplo

Descripción

Se van a mostrar ejemplos de uso de servicios web con JAVA, con Ruby, con PHP y con WSS/WSSE

Ejemplos

Ejemplos de uso de servicios web con Java.

Los siguientes ejemplos ilustran un conjunto de sencillos casos de uso de servicios web.

Iniciar sesion.

Antes de que se pueda establecer una comunicación con cualquier servicio web de Alfresco se debe autenticar el usuario actual para iniciar una sesion. Esto se puede hacer usando el metodo startSession que se encuentra en el Authentication Web Services.

Al metodo startSession se le pasa un usauario y un password y, si la autenticacion tiene exito, devolvera un ticket. Este ticket puede ser usado cuando se llamen otros metodos de los servicios web.

```
// Obtener la referencia de Authentication service
   AuthenticationServiceSoapBindingStub authenticationService =
      (AuthenticationServiceSoapBindingStub) new AuthenticationServiceLocator()
      .getAuthenticationService();

   // Iniciar la sesion
   AuthenticationResult result = authenticationService
                              .startSession(userName, password);
   String ticket = result.getTicket();
```

Una vez que todo el trabajo este hecho una llamada al metodo endSession invalidará el ticket.

```
// Finalizar la sesion
   authenticationService.endSession(ticket);
```

Uso del Ticket

Las llamadas a todos los metodos de los servicios web deben tener la información de seguridad WS en la cabecera. Esto asegura el que solo los usuarios autenticados puedan acceder a la API de servicios web.

En los ejemplos la información de Seguridad de Servicios Web es proporcionado como un defínción de despliegue XML:

```
<deployment xmlns='http://xml.apache.org/axis/wsdd/'
              xmlns:java='http://xml.apache.org/axis/wsdd/providers/java'>
     <transport name='http'
               pivot='java:org.apache.axis.transport.http.HTTPSender'/>
     <globalConfiguration >
       <requestFlow >
         <handler type='java:org.apache.ws.axis.security.WSDoAllSender' >
           <parameter name='action' value='UsernameToken'/>
           <parameter name='user' value='ticket'/>
           <parameter name='passwordCallbackClass'
            value='org.alfresco.example.webservice.sample.WebServiceSample1'/>
           <parameter name='passwordType' value='PasswordText'/>
         </handler>
       </requestFlow >
     </globalConfiguration>
   </deployment>
```

La *passwordCallbackClass* es entonces responsable de proporcionar el ticket cuando los metodos de los servicios web son

llamados. Se debe implementa *CallbackHandler* el cual tiene un unico metodo *handle*.

```
/**
     * Implementación de el password callback usado por el protocolo WS Security
     *
     * @ver javax.security.auth.callback.CallbackHandler#handle(javax.security.auth.callback.Callback[])
     */
    public void handle(Callback[] callbacks)
                throws IOException, UnsupportedCallbackException
    {
       for (int i = 0; i < callbacks.length; i++)
       {
          if (callbacks[i] instanceof WSPasswordCallback)
          {
             WSPasswordCallback pc = (WSPasswordCallback)callbacks[i];
             // establecr el password como el ticket del usurario actual
             pc.setPassword(currentTicket);
          }
          else
          {
             throw new UnsupportedCallbackException(callbacks[i],
                      "Unrecognized Callback");
          }
       }
    }
```

Para que la información de despliegue y el *callback handler* sean usados cuando se realiza una llamada a un servicio web, se debe proporcionar a la clase cliente del servicio web esta información cuando es creada. El suguiente trozo de código muestra como crear una referencia la Repository Web Service teniendo esto en cuenta.

```
// Create the respository service, adding the WS security header information
    EngineConfiguration config = new FileProvider(
            new ByteArrayInputStream(
               WebServiceSample1.WS_SECURITY_INFO.getBytes()
               )
            );
    RepositoryServiceLocator repositoryServiceLocator =
                           new RepositoryServiceLocator(config);
    RepositoryServiceSoapBindingStub repositoryService =
    (RepositoryServiceSoapBindingStub)repositoryServiceLocator.getRepositoryService();
```

Ejemplo 1. Autenticacion y obtencion de Stores.

Este ejemplo muestra como autenticar un usuario, proporciona un ejemplo de CallbackHandler y muestra como obtener una lista de stores disponibles.

```
import org.alfresco.webservice.repository.RepositoryServiceSoapBindingStub;
import org.alfresco.webservice.types.Store;
import org.alfresco.webservice.util.AuthenticationUtils;
import org.alfresco.webservice.util.WebServiceFactory;
/**
 * Ejemplo 1.
 *
 * Connectar al repositorio y obtener una lista de todos los stores disponibles en el repositorio.
 *
 * @author Roy Wetherall
 */
public class GetStores extends SamplesBase
{
    /**
     * Connect to the respository and print out the names of the available
     *
     * @param args
```

```
    */
    public static void main(String[] args)
        throws Exception
    {
        // Iniciar la sesion
        AuthenticationUtils.startSession(USERNAME, PASSWORD);

        try
        {
            // Optener respository service
            RepositoryServiceSoapBindingStub repositoryService = WebServiceFactory.getRepositoryService();
```

Ejemplo 2. Busquedas simples

Este ejemplo muestra como ejecutar busquedas simples y navegar por los resultadosusando el Repository Web Service.

```
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import org.alfresco.webservice.repository.QueryResult;
import org.alfresco.webservice.repository.RepositoryFault;
import org.alfresco.webservice.repository.RepositoryServiceSoapBindingStub;
import org.alfresco.webservice.types.NamedValue;
import org.alfresco.webservice.types.Query;
import org.alfresco.webservice.types.Reference;
import org.alfresco.webservice.types.ResultSet;
import org.alfresco.webservice.types.ResultSetRow;
import org.alfresco.webservice.util.AuthenticationUtils;
import org.alfresco.webservice.util.Constants;
import org.alfresco.webservice.util.WebServiceFactory;
/**
 * Ejemplo 2
 * <p>
 * Este ejemplo muestra como ejecutar una busqueda usando el repository web service y como
 * buscar por los nodos padres.
 *
 * @author Roy Wetherall
 */
public class Query1 extends SamplesBase
{
    /**
     * Funcion principal
     */
    public static void main(String[] args)
        throws Exception
```

Ejemplo 3. Manipular contenido

Este ejemplo muestra como crear contenido, leer y actualizar usando la Content Web Service.

```
import java.io.InputStream;
import org.alfresco.webservice.content.Content;
import org.alfresco.webservice.content.ContentServiceSoapBindingStub;
import org.alfresco.webservice.repository.UpdateResult;
import org.alfresco.webservice.types.CML;
import org.alfresco.webservice.types.CMLCreate;
import org.alfresco.webservice.types.ContentFormat;
import org.alfresco.webservice.types.NamedValue;
import org.alfresco.webservice.types.ParentReference;
import org.alfresco.webservice.types.Predicate;
import org.alfresco.webservice.types.Reference;
import org.alfresco.webservice.util.AuthenticationUtils;
import org.alfresco.webservice.util.Constants;
import org.alfresco.webservice.util.ContentUtils;
import org.alfresco.webservice.util.Utils;
```

```
import org.alfresco.webservice.util.WebServiceFactory;
/**
 * Ejemplo 3
 * <p>
 * Est ejemplo de servicio web muestra como se puede añadir nuevo contenido al repositorio y como puede
 * leido y actualizado mediante la API de web service.
 *
 * @author Roy Wetherall
 */
public class ContentReadAndWrite extends SamplesBase
{
    /** cadenas de Contenido usado en el ejemplo */
```

Ejemplo 4. Actualizaciones CML

Este ejemplo muestra como construir y ejecutar consultas CML usando el Repository Web Service.

```
import org.alfresco.webservice.repository.RepositoryServiceSoapBindingStub;
import org.alfresco.webservice.repository.UpdateResult;
import org.alfresco.webservice.types.CML;
import org.alfresco.webservice.types.CMLAddAspect;
import org.alfresco.webservice.types.CMLCreate;
import org.alfresco.webservice.types.NamedValue;
import org.alfresco.webservice.types.ParentReference;
import org.alfresco.webservice.types.Reference;
import org.alfresco.webservice.util.AuthenticationUtils;
import org.alfresco.webservice.util.Constants;
import org.alfresco.webservice.util.Utils;
import org.alfresco.webservice.util.WebServiceFactory;
/**
 * Ejemplo 4
 * <p>
 * Este ejemplo muestra como construir y ejecutar consulatas CML usando el respository web service.
 *
 * @author Roy Wetherall
 */
public class CMLUpdates extends SamplesBase
{
    /**
     * Fuuncion principal
     */
    public static void main(String[] args)
        throws Exception
    {
```

Ejemplo 5. Proteger y desproteger

Este ejemplo muestra como desproteger documentos, protegerlos y ver el historico de vesiones usando el Authoring Web Service.

```
import org.alfresco.webservice.authoring.AuthoringServiceSoapBindingStub;
import org.alfresco.webservice.authoring.CheckoutResult;
import org.alfresco.webservice.content.Content;
import org.alfresco.webservice.content.ContentServiceSoapBindingStub;
import org.alfresco.webservice.repository.RepositoryServiceSoapBindingStub;
import org.alfresco.webservice.types.CML;
import org.alfresco.webservice.types.CMLAddAspect;
import org.alfresco.webservice.types.ContentFormat;
import org.alfresco.webservice.types.NamedValue;
import org.alfresco.webservice.types.Predicate;
import org.alfresco.webservice.types.Reference;
import org.alfresco.webservice.types.Store;
```

```
import org.alfresco.webservice.types.Version;
import org.alfresco.webservice.types.VersionHistory;
import org.alfresco.webservice.util.AuthenticationUtils;
import org.alfresco.webservice.util.Constants;
import org.alfresco.webservice.util.ContentUtils;
import org.alfresco.webservice.util.Utils;
import org.alfresco.webservice.util.WebServiceFactory;
/**
 * Web service sample 5
 * <p>
 * Este ejemplo muestra como desproteger y proteger documentos y como ver el
 * historial de versiones.
 *
 * @author Roy Wetherall
 */
```

Ejemplo 6. Categorias

Este ejemplo muestra como el contenido puede ser mediante sus categorias usando el Classification Web Service.

```
import org.alfresco.webservice.classification.ClassificationServiceSoapBindingStub;
import org.alfresco.webservice.repository.QueryResult;
import org.alfresco.webservice.repository.RepositoryServiceSoapBindingStub;
import org.alfresco.webservice.types.Category;
import org.alfresco.webservice.types.Classification;
import org.alfresco.webservice.types.Query;
import org.alfresco.webservice.util.AuthenticationUtils;
import org.alfresco.webservice.util.Constants;
import org.alfresco.webservice.util.ISO9075;
import org.alfresco.webservice.util.WebServiceFactory;
/**
 * Ejemplo 6
 * <p>
 * Este ejemplo muestra como el contenido puede ser consultado usando categorias
 *
 * @author Roy Wetherall
 */
public class Categories extends SamplesBase
{
    /**
     * Función principal
     */
    public static void main(String[] args)
        throws Exception
    {
        AuthenticationUtils.startSession(USERNAME, PASSWORD);
        try
```

Ejemplo de uso de servicios web con Ruby

La forma mas simple de acceder a Alfresco desde una applicacion ruby usando servicios web es usar Sop4r. La actual implememtación no offrece los estandar WS-Security directamente así que para los ejemplos se ha usado Wss4r que añade esta seguridad.

Iniciar una sesion

```
require 'soap/wsdlDriver'
require "wss4r/rpc/wssdriver"
...
wsdl_url = "file://" + <path-a-su-directorio-wsdl> + "/authentication-service.wsdl"
soapDriver = SOAP::WSDLDriverFactory.new(wsdl_url).create_rpc_driver()
authentication_result = soapDriver.startSession {:username => '<username>', :password => '<password>'}
ticket = authentication_result.ticket
```

```
ticket = authentication_result.ticket
…
```

Como se puede ver, una vez que se tiene configurado el entorno y un wsdl valido unas pocas lineas hacen el resto. El resultado de la uatenticación es un ticket que se tendrá que reusar cada vez que se haga una consulta.

Enviar una consulta

```
require 'soap/wsdlDriver'
require "wss4r/rpc/wssdriver"
require "wss4r/tokenresolver/authenticateuserresolver"
…
wsdl_url = "file://" + <path-a-su-directorio-wsdl>  + "/repository-service.wsdl"
soapDriver = SOAP::WSDLDriverFactory.new(wsdl_url).create_rpc_driver()
# añadir la informacion de autenticacion
resolver = AuthenticateUserResolver.new()
soapDriver.security().add_security_resolver(resolver)
soapDriver.security().add_security_token(UsernameToken.new(user, ticket))
# la consulta
query_result = soapDriver.query({:store => {:scheme => 'workspace', :address => 'SpacesStore'},
                                 :query => {:language => 'lucene', :statement => 'TEXT:"<some text to seach>"'},
                                 :includeMetaData => false})
…
```

Hacer una consulta requiere un poco mas de esfuerzo al necesitar que se incluya el ticket que se generó anteriormente.

Leer the resultados

Obtener el numero de filas devueltas:

```
number_of_results = query_result.queryReturn.resultSet.totalRowCount
```

Ejemplo de uso de servicios web con PHP

```
<?php

require_once "Alfresco/Service/Repository.php";
require_once "Alfresco/Service/Session.php";
require_once "Alfresco/Service/SpacesStore.php";
require_once "Alfresco/Service/Node.php";
require_once "Alfresco/config.php";
require_once('include/database/PearDatabase.php'); //conexion a bbdd
function UpdateDMSContent(){
   global $repositoryUrl;
   global $userName;
   global $password;
   global $adb;
   $repository = new Repository($repositoryUrl);
   $repository = new Repository($repositoryUrl);
   $ticket = null;
   if (isset($_SESSION["ticket"]) == false)
   {
      $ticket = $repository->authenticate($userName, $password);
      $_SESSION["ticket"] = $ticket;
   }
   else
   {
     $ticket = $_SESSION["ticket"];
   }
 try{
   $session = $repository->createSession($ticket);
```

Ejemplo de uso de servicios web con WSS/WSSE

Construccion de una cabeceras WSSE

Construccion de una cabeceras WSSE

```
<process name="alfresco
...
xmlns:ns4="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
xmlns:ns5="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
<variables>
 <variable name="wsseHeader" element="ns4:Security"/>
 <variable name="wstimestamp" element="ns5:Timestamp"/>
</variables>
<sequence name="main">
 <assign name="makeWSSEHeaders">
 <copy>
 <from expression="xp20:current-dateTime()"/>
 <to variable="wstimestamp" query="/ns5:Timestamp/ns5:Created"/>
 </copy>
 <copy>
 <from expression="xp20:current-dateTime()"/>
 <to variable="wstimestamp" query="/ns5:Timestamp/ns5:Expires"/>
 </copy>
 <bpelx:append>
 <bpelx:from variable="wstimestamp" query="/ns5:Timestamp"/>
 <bpelx:to variable="wsseHeader" query="/ns4:Security"/>
 </bpelx:append>
 <bpelx:append>
 <bpelx:from>
 <wsse:UsernameToken xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
 <wsse:Username>admin</wsse:Username>
```

Iniciar una sesion

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSch
 <env:Body>
  <startSession xmlns="http://www.alfresco.org/ws/service/authentication/1.0">
   <username>admin</username>
   <password>admin</password>
  </startSession>
 </env:Body>
</env:Envelope>
```

Llamada al RepositoryService usando las cabeceras WSSE

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSc
 <env:Header>
  <Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <Timestamp xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" ans1:Id=
    <ans1:Created>2007-08-01T12:13:37+01:00</ans1:Created>
    <ans1:Expires>2007-08-01T12:13:37+01:00</ans1:Expires>
   </Timestamp>
   <UsernameToken xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.x
    <Username>admin</Username>
    <Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#Passwo
   </UsernameToken>
  </Security>
 </env:Header>
 <env:Body>
  <query xmlns="http://www.alfresco.org/ws/service/repository/1.0" xmlns:ns3="http://www.alfresco.org/ws/model/co
   <store>
    <ns1:scheme xmlns:ns1="http://www.alfresco.org/ws/model/content/1.0">workspace</ns1:scheme>
    <ns2:address xmlns:ns2="http://www.alfresco.org/ws/model/content/1.0">SpacesStore</ns2:address>
```

```
  </store>
  <query>
   <ns3:language xmlns:ns3="http://www.alfresco.org/ws/model/content/1.0">lucene</ns3:language>
   <ns4:statement xmlns:ns4="http://www.alfresco.org/ws/model/content/1.0">@cm\:name:"Alfresco-Tutorial.pdf"</r
  </query>
  <includeMetaData>true</includeMetaData>
 </query>
</env:Body>
</env:Envelope>
```

Usar el ContentService con una Consulta Lucene con cabeceras WSSE

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSc
 <env:Header>
  <Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <Timestamp xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" ans1:Id=
    <ans1:Created>2007-08-01T12:13:37+01:00</ans1:Created>
    <ans1:Expires>2007-08-01T12:20:37+01:00</ans1:Expires>
   </Timestamp>
   <UsernameToken xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.x
    <Username>admin</Username>
    <Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#Passwo
   </UsernameToken>
  </Security>
 </env:Header>
 <env:Body>
  <read xmlns="http://www.alfresco.org/ws/service/content/1.0">
   <items>
    <store xmlns="http://www.alfresco.org/ws/model/content/1.0">
     <scheme>workspace</scheme>
     <address>SpacesStore</address>
    </store>
    <query xmlns="http://www.alfresco.org/ws/model/content/1.0">
     <language>lucene</language>
     <statement>@cm\:name:"Alfresco-Tutorial.pdf"</statement>
    </query>
   </items>
   <property>{http://www.alfresco.org/model/content/1.0}content</property>
  </read>
```

Usar ContentService con uuid con cabeceras WSSE

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSc
 <env:Header>
  <Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <Timestamp xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" ans1:Id=
    <ans1:Created>2007-08-01T12:00:28+01:00</ans1:Created>
    <ans1:Expires>2007-08-01T12:05:28+01:00</ans1:Expires>
   </Timestamp>
   <UsernameToken xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.x
    <Username>admin</Username>
    <Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#Passwo
   </UsernameToken>
  </Security>
 </env:Header>
 <env:Body>
  <read xmlns="http://www.alfresco.org/ws/service/content/1.0">
   <items>
    <nodes xmlns="http://www.alfresco.org/ws/model/content/1.0">
     <store>
```

```
    <scheme>workspace</scheme>
    <address>SpacesStore</address>
   </store>
   <uuid>635c49e6-0af5-11dc-9bc5-2ddbdefb608b</uuid>
  </nodes>
 </items>
 <property>{http://www.alfresco.org/model/content/1.0}content</property>
</read>
</env:Body>
```

Usar el ContentService con path con cabeceras WSSE

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSc
 <env:Header>
  <Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <Timestamp xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" ans1:Id=
    <ans1:Created>2007-08-01T12:00:28+01:00</ans1:Created>
    <ans1:Expires>2007-08-01T12:05:28+01:00</ans1:Expires>
   </Timestamp>
   <UsernameToken xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.x
    <Username>admin</Username>
    <Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#Passwo
   </UsernameToken>
  </Security>
 </env:Header>
 <env:Body>
  <read xmlns="http://www.alfresco.org/ws/service/content/1.0">
   <items>
    <nodes xmlns="http://www.alfresco.org/ws/model/content/1.0">
     <store>
      <scheme>workspace</scheme>
      <address>SpacesStore</address>
     </store>
     <path>/app:company_home/app:guest_home/cm:Alfresco-Tutorial.pdf</path>
    </nodes>
   </items>
   <property>{http://www.alfresco.org/model/content/1.0}content</property>
  </read>
 </env:Body>
```

Enlaces externos

- Ruby Forge

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/41

Servicios Web de Alfresco

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Carácter del recurso:** Recomendado

Código: RECU-0040
Tipo de recurso: Manual

Descripción

Alfresco dispone de una API de Servicios web disponible para el acceso al repositorio diseñada para facilitar su comprensión y utilización, resultar accesible a tantos lenguajes como sea posible y proporcionar acceso remoto al repositorio.

Evaluación de los Web Services de Alfresco de acuerdo a MADEJA

MADEJA hace las recomendaciones acerca del desarrollo de Servicios Web en las aplicaciones en el subsistema de Arquitectura en el área de integración.

De acuerdo a dichas recomendaciones se puede señalar que los servicios web de Alfresco cumplen con dichas recomendaciones, tal y como se resume a continuación:

- Los WS, son del tipo Contract-first/document literal, lo que se ajusta al escenario ideal marcado en MADEJA.
- El namespace, pej ***http://www.alfresco.org/ws/service/repository/1.0***, es de tipo URL e incluye número de versión. En MADEJA se recomienda el formato URN, pero se puede permitir este formato, dado que es el elegido por el fabricante.
- Al incluir número de versión en el namespace, se cubre el mínimo exigido en MADEJA para el control de versionado en WS.
- El WSDL tiene cierta división estructural (wdsl, xsd), en este aspecto, se cubre lo mínimos recomendados por MADEJA al respecto.
- Por último, para la generación de errores se emplean SOAP FAULT, tal y como se recomienda en MADEJA.

Servicios

La siguiente lista muestra los distintos conjuntos de servicios web que implementa alfresco para dar acceso a las distintas funcionalidades de su repositorio. Todas ellas se encuentran en API de Web Services de Alfresco:

- **Autenticacion (Authentication):** Login y desconexion
- **Repositorio (Repository):** Consulta y Manipulacion del modelo
- **Contenido (Content):** Manipulacion de contenido
- **Autoria (Authoring):** Creacion cloaborativa de contenido
- **Clasificacion (Classification):** Aplicacion de clasificaciones y categorias
- **Control de acceso (Access Control):** Roles, permisos y propiedad
- **Acciones (Action):** Gestión de acciones y reglas
- **Administración (Administration):** Gestión de usuarios, exportacion e importacion
- **Diccionario (Dictionary):** Descripciones de modelos

Tipos de datos

Cada método de los servicios web se basa en los siguientes tipos de datos para la entrada y salida de mensajes. Los tipos de datos estan enfocados hacia un dominio de contenido y, si es necesario, podrían ser utilizados fuera del contexto de Web Services si es necesario. Todos ellos se encuentran en Tipos de datos para los servicios web.

- **Identificadores (Identifiers):** Medios para la identificacion y localizacion de contenido
- **Contenido (Content):** Datos de los contenido
- **Consulta (Query):** Consulatas y conjuntos de resultados
- **Metadatos (Meta Data):** Metadatos de los contenidos (dicionario de datos)
- **Versionado (Versioning):** Historico y graficos de versiones
- **Clasificacion (Classification):** Categorizacion de contenido

Cabeceras Soap

Las siguientes cabeceras SOAP proporcionan control extra sobre los metodos de los Servicios Web:

- **Cabecera de Consulta (QueryHeader):** Proporciona gestion para consultas
- **Cabecera de Localización (LocaleHeader):** Proporciona gestion para requerimientos localización
- **Cabecera de NameSpaces (NamespaceHeader):** Mapeo de los pefijos de los espacios de nombre
- **Perfil UsernameToken (UsernameToken Profile):** Proporciona gestion de la seguridad de la informacion

La definición formal de las cabeceras anteriores se encuentra en el SOAP Header XML Schema

Lenguajes de consulta

El núcleo de cualquier interfaz a un gestor de contenidos es el lenguaje de consulta. Actualmente son soportadas una interfaz Xpath y Lucene pero se está trabajando para implementar un lenguaje de consulta basado en SQL.

- **XPath:** Su propósito es el de consulta. (JCR XPath)
- **Lucene:** Su propósito es de consulta. Estilo Google.
- **Lenguaje de Manipulacion de contenidos(CML):** Su propósito es de actualizacion. Lenguaje de manipulacion de contenidos

Ejemplos

Se va a mostrar ahora algunos codigos de ejemplo de como acceder a la API de servicios web de Alfresco desde diferentes lenguajes de programación. Todos ellos se encuentran en Ejemplos de uso de servicos web:

- Ejemplos de acceso a Servicios Web para Java
- Ejemplos de acceso a Servicios Web para PHP
- Ejemplos de acceso a Servicios Web con Ruby
- Ejemplos de WSS / WSSE (Web Services Security)
- Web Service Samples for .NET

Documentos

oasis-200401-wss-username-token-profile-1.0 (83.56 KB)

Servicios Web en Alfresco

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Carácter del recurso:** Recomendado

Código: RECU-0004
Tipo de recurso: Ficha

Descripción

Alfresco proporciona acceso remoto a traves de una API de Servicios Web basada en SOAP. Un servicio web (en inglés Web service) es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. La API de Servicios Web esta especificamente diseñada para reducir la cantidad de comunicación con el servidor lo que la hace una manera bastante eficiente de acceso remoto.

Recursos

Área: Arquitectura » Arquitectura de Sistemas de Información » Alfresco » Uso de Alfresco desde Terceras Aplicaciones

Código	Título	Tipo	Carácter
RECU-0040	Servicios Web de Alfresco	Manual	Recomendado

Tipos de datos para los servicios web

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Carácter del recurso:** Recomendado

Código: RECU-0043
Tipo de recurso: Manual

Descripción

Los servicios web de Alfresco utilizan los siguientes tipos de datos en los mensajes de entrada y salida. A continuacion veremos el esquema de estos tipos de datos.

Identificadores y Localizadores

Store

EL tipo dato store identifica univocamente un store en el repositorio

Esta compuesto de dos partes:

- xsd:string scheme: indica el tipo de store que esta siendo referenciado siendo el mas comun 'workspace'. Otros valores validos son 'versionStore', 'user', 'search' y 'system'.
- xsd:string address: este es el nombre del store

El ejemplo mas comun es el store usado por la Interfaz web prinicpal, tendrá scheme 'workspace' y address 'SpacesStore'.

Reference

Un tipo de dato reference es una referencia a un nodo individual con un store dado.

Un reference esta compuesta de tres partes:

- xsd:string store: los detalles del store donde pertenece esta referencia.
- xsd:string uuid: el identificador del nodo. Es opcional.
- xsd:string path: el path a un nodo. Es opcional.

Un reference en la cual solo se indica el valor del estore se refiere al nodo raiz de ese store. Un reference en la cual solo se indica el uuid set se refiere al nodo que tiene ese uuid. Si no se encuentra coincidencia se considera error. Un reference en la que solo se indica se refiere al nodo que se obtiene como resultado de la ejecución de la consulta dada por ese path en el store. El lenguaje de consulta esperado es XPath. Por eejemplo para encontrar el nodo company home en el 'SpacesStore' se debería usar el path /app:company_home. Si el resultado de la consulta de path es mas de un nodo se considerá que hay un error. En caso de que ambos,uuid y path, esten indicados la consulta de path se ejecutará en el contexto del nodo referenciado por el uuid.

ParentReference

Una referencia padre se usa para describir como un node se puede asociar a un hijo. Las referencias padres se usuan usualmente cuando una operacion crea un nuevo nodo, por ejemplo crear o mover un contenido.

Una referencia padre se compone de tres partes:

- Reference reference: Una referencia al nodo que es el padre.
- xsd:string associationType: el QName del tipo de asiociación en forma de cadena de caracteres. Por ejemplo:

```
{http://www.alfresco.org/model/content/1.0}contains
```

- xsd:string childName: el QName de la asiociación en forma de cadena de caracteres. Por ejemplo:

```
{http://www.alfresco.org/model/content/1.0}mydocs
```

Esquema

```
<xsd:simpleType name="Name">
      <xsd:annotation>
       <xsd:documentation>TODO: Define constraints</xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:string">
   <xsd:minLength value="1"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="UUID">
```

```
 <xsd:restriction base="xsd:string">
    <xsd:pattern value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="Path">
      <xsd:annotation>
       <xsd:documentation>TODO: Define constraints</xsd:documentation>
      </xsd:annotation>
 <xsd:restriction base="xsd:string">
  <xsd:minLength value="1"/>
 </xsd:restriction>
</xsd:simpleType>
  <xsd:simpleType name="StoreEnum">
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="workspace"/>
    <xsd:enumeration value="version"/>
    <xsd:enumeration value="search"/>
    <xsd:enumeration value="http"/>
```

Representación del Contenido

ContentFormat

El tipo de dato formato de contenido esta compuesto de dos partes:

- xsd:string mimetype: el tipo mime del contenido
- xsd:string encoding: la codificación de caracteres del contenido

Se usa para encapsular la informacion del formato sobre un nodo de contenido.

NamedValue

El tipo de dato valor nominal contiene el nombre y el valor/valores de un atributo del repositorio de Alfresco:

- xsd:string name: la representacion como cadena de texto del QName
- xsd:boolean isMultiValue: indica cuando se esta tratando con un atributo multi-valor. Si contiene el valor true el atributo *values* es usado.Si este atributo no contiene nugún valor toma el valor por defecto *false*
- xsd:string value: el valor una cadena de caracteres
- xsd:string[] values: un array de cadenas de caracteres con los valores del atributo

El tipo de dato valor nominal se usa normalmente para representar el nombre y el valor de una propiedad en un nodo.

Node

El tipo de dato Nodo representa los detalles de un nodo de acuerdo al repositorio. Incluye el tipo, aspecto y detalle y valores de las propiedades.

El tipo de dato Nodo se compone de cuatro partes:

- Reference reference: la referencia al nodo
- xsd:String type: la representación como cadena de texto del QName del tipo del nodo
- xsd:String aspect: la representación como cadena de texto de los QNames de los aspectos applicados al nodo
- NamedValue[] properties: los nombres y valores de las propertiedades del nodo

Esquema

```
<xsd:complexType name="Node">
    <xsd:sequence>
     <xsd:element name="reference" type="alf:Reference"></xsd:element>
     <xsd:element name="type" type="alf:Name">
      <xsd:annotation>
       <xsd:documentation>TODO: Change type to NodeDescription</xsd:documentation>
      </xsd:annotation>
     </xsd:element>
     <xsd:element name="properties" type="alf:Property" minOccurs="0" maxOccurs="unbounded"></xsd:element>
    </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Content">
    <xsd:complexContent>
```

```
    <xsd:extension base="alf:Node">
     <xsd:sequence>
      <xsd:element name="format" type="alf:ContentFormat"></xsd:element>
      <xsd:element name="length" type="xsd:long"></xsd:element>
     </xsd:sequence>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="ContentFormat">
   <xsd:sequence>
    <xsd:element name="mimetype">
     <xsd:annotation>
      <xsd:documentation>
       TODO: MinOccurs = 0?
```

Consultas

Consulta

El tipo de dato consulta se compone de dos partes:

- xsd:string language: le lenguaje de la consulta, por ejemplo Lucene o XPath.
- xsd:String statement: la declaración de la consulta

El tipo de dato consulta se usa para encapular información sobre la query. …

Esquema

```
<xsd:simpleType name="QueryLanguageEnum">
    <xsd:restriction base="xsd:string">
     <xsd:enumeration value="lucene"/>
     <xsd:enumeration value="xpath"/>
     <xsd:enumeration value="cql"/>
    </xsd:restriction>
   </xsd:simpleType>
   <xsd:complexType name="Query">
    <xsd:sequence>
     <xsd:element name="language" type="alf:QueryLanguageEnum"/>
     <xsd:element name="statement" type="xsd:string"/>
    </xsd:sequence>
   </xsd:complexType>

 <xsd:complexType name="ResultSet">
  <xsd:sequence>
   <xsd:element name="row" type="alf:ResultSetRow" maxOccurs="unbounded" minOccurs="0"></xsd:element>
   <xsd:element name="size" type="xsd:long"></xsd:element>
   <xsd:element name="metaData" type="alf:ResultSetMetaData" maxOccurs="1" minOccurs="0"></xsd:element>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="ResultSetRow">
  <xsd:sequence>
   <xsd:element name="rowIndex" type="xsd:long"></xsd:element>
   <xsd:element name="column" type="alf:Property" maxOccurs="unbounded" minOccurs="0"/>
   <xsd:element name="score" type="xsd:float" maxOccurs="1" minOccurs="0"></xsd:element>
   <xsd:element name="node" maxOccurs="1" minOccurs="0">
```

Metadatos

Esquema

```
<xsd:complexType name="ClassDefinition">
 <xsd:sequence>
```

```
  <xsd:element name="name" type="alf:Name"></xsd:element>
  <xsd:group ref="alf:TextualDescription"></xsd:group>
  <xsd:element name="superClass" type="alf:Name" maxOccurs="1" minOccurs="0"></xsd:element>
  <xsd:element name="isAspect" type="xsd:boolean"></xsd:element>
  <xsd:element name="properties" type="alf:PropertyDefinition"></xsd:element>
  <xsd:element name="associations" type="alf:AssociationDefinition"></xsd:element>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ValueDefinition">
 <xsd:sequence>
  <xsd:element name="name" type="alf:Name"></xsd:element>
  <xsd:group ref="alf:TextualDescription"></xsd:group>
  <xsd:element name="dataType" type="xsd:anyType"></xsd:element>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="PropertyDefinition">
 <xsd:complexContent>
  <xsd:extension base="alf:ValueDefinition">
   <xsd:sequence>
    <xsd:element name="mandatory" type="xsd:boolean" default="false"></xsd:element>
    <xsd:element name="readOnly" type="xsd:boolean" default="false"></xsd:element>
    <xsd:element name="defaultValue" maxOccurs="1" minOccurs="0" type="xsd:anyType"></xsd:element>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
```

Versionado

Esquema

```
<xsd:complexType name="Version">
 <xsd:sequence>
  <xsd:element name="id" type="alf:Reference"></xsd:element>
  <xsd:element name="created" type="xsd:dateTime" maxOccurs="1" minOccurs="0"></xsd:element>
  <xsd:element name="creator" type="xsd:string" maxOccurs="1" minOccurs="0"></xsd:element>
  <xsd:element name="label" type="xsd:string" maxOccurs="1" minOccurs="0"></xsd:element>
  <xsd:element name="major" type="xsd:boolean" default="true"></xsd:element>
  <xsd:element name="commentary" type="alf:Property" maxOccurs="unbounded" minOccurs="0"></xsd:element>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="VersionHistory">
 <xsd:sequence>
  <xsd:element name="versions" type="alf:Version" maxOccurs="unbounded" minOccurs="0"></xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

Clasificacion

Esquema

```
<xsd:complexType name="Category">
 <xsd:sequence>
  <xsd:element name="id" type="alf:Reference"></xsd:element>
  <xsd:group ref="alf:TextualDescription"></xsd:group>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Classification">
 <xsd:complexContent>
  <xsd:extension base="alf:Category">
```

```
  <xsd:extension base="alf:Category">
   <xsd:sequence>
    <xsd:element name="rootCategory" type="alf:Category"></xsd:element>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/43

Uso de Web Scripts

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Carácter del recurso:** Recomendado

Código: RECU-0045
Tipo de recurso: Manual

Descripción

Una de las formas en las que una aplicación externa puede trabajar con el repositorio de Alfresco es a través de la URL addresability. Los objetos del repositorio de Alfresco son accesibles vía URL y ademas se les pueden aplicar plantillas FreeMarker y JavaScript en el servidor. Una aplicación externa podría realizar una llamada XMLHttpRequest a Alfresco que especifique una referencia a un nodo y a una plantilla FreeMarker que devuelva XML o JSON. Alfresco procesara la plantilla en el contexto del nodo especificado y devolverá el resultado.

El framework de Web Scripts es esencialmente una mejora sobre la idea que comenzó con la URL addresability. Esta disponible desde la versión 2.1 y potencialmente es la mejor manera de interactuar con el repositorio.

Se debe pensar en un web script como en un trozo de código que está mapeado en una URL comprensible para una persona (y un motor de búsqueda). Por ejemplo una URL que devuelva los informes de gastos pendientes de aprobar podría ser algo así:

```
/alfresco/service/gastos/pendientes
```

mientras que una URL que devuelve los gastos pendientes de un usuario en concreto podría ser:

```
/alfresco/service/gastos/pendientes/msanchez
```

en la URL anterior se puede leer el componente msanchez como un argumento implícito. También se podría hacer de una forma mas explicita:

```
/alfresco/service/gastos/pendientes?user=msanchez
```

La decisión sobre la estructura de la URL, de como se incluyen los argumentos, así como de lo que va a devolver la llamada (HTML, XML, JSON, JSR-168 Porlet.....) es decisión del desarrollador.

Al Framework de Web Scripts le es fácil seguir el patrón Vista Controlador (MVC). El Controlador será JavaScript en el servidor, un Bean de Java o ambos. Manejará las peticiones, realizará cualquier lógica de negocio que se necesite, alimentará al Modelo con datos y reenviará la petición a la vista. La vista será una plantilla FreeMarker responsable de construir la respuesta en el formato adecuado. El Modelo básicamente es una estructura de datos que se intercambian Vista y Controlador.

La accesibilidad URL al controlador se hace con un descriptor XML que es responsable de declarar el patrón URL, de si el script requiere una transacción o no y de los requisitos de autenticación para el script.

Los formatos de respuesta son mapeados hacia las plantillas FreeMarker usando convenciones de nombres. Así, por ejemplo, la plantilla FreeMarker que devuelve informes en HTML se llamara con una extensión de *html (xxx.html.xxx)* mientras la que lo haga en XML se llamará con una extensión de *xml (xxx.xml.xxx).*

El descriptor, el fichero JavaScript y la plantilla FreeMarker pueden estar en el repositorio o en el sistema de ficheros. Si el Web Script usa un Bean de Java la clase debe estár en el classpath.

Se pueden usar Web Scripts para exponer el contenido del repositorio de Alfresco a traves de una API RESTful para:

- Proveer al repositorio de Alfresco de servicios de contenidos que serán accesibles para todo el mundo.
- Consultar en el repositorio, extraer contenido y alterar su comportamiento.
- Exponer el repositorio para realizar gestión documental o gestión de contenidos web.
- Proveer de facilidades de búsqueda personalizadas.
- Crear ATOM o RSS feeds para los contenidos del repositorio o procesos de negocio.

API de Web Script

Alfresco ofrece por defecto una API con Web Scripts construidos sobre esta plataforma de web scripts. Los principales objetivos de diseño de esta API son la facilidad de comprensión y uso, la minimización de las herramientas necesarias por parte del cliente, soportar el acceso remoto, soportar el acceso a multiples repositorios, mantenerse en linea con los estandares REST (como Atom,OpenID,OpenSearch), facilitar la actualizacion y ampliacion de servicios y cubrir el 100% de los servicios del Repositorio.

Estos objetivos se intentaran alcanzar en proximas versiones del producto. En el siguiente enlace podremos consultar la referencia a la actual ***API Restful de Alfresco?***.

Nuevas Funcionalidades en Alfresco 3.0

- Web Script ahora es un componente independiente capaz de acoger otros Web Scripts en otros niveles del Repositorio de

Alfresco. por ejemplo con Alfresco SURF.

- Ahora URI Templates cumplen con JSR-311 (JAX-RS).
- Posibilidad de que pueda ejecutar un usuario especifico el Web Script.
- Posibilidad de personalizar WebScript con una configuració Web Script especifica.
- Per Web Script Message Bundle.
- Mejora de procesamiento de los formularios.
- Acceso a las cabeceras de Respuesta.
- Procesar las solicitudes realizadas en el Body.
- Content Negotiation.
- Declarative and programmatic Cache Control.
- Capacidad de Categorizar WebScript por familias.
- Specialized Kinds of Web Script.

Crear un WebScript

Uno de las grandes ventajas de los Web Scripts es que permite la construcción con mucha facilidad de una API Restful personlizada que se ajuste a las necesidades de cada problema. El objetivo de esta sección es mostrar una guía con los pasos que son necesarios dar para crear un web script.

Como ejemplo se implementará una búsqueda de texto sobre un blog que devuelva valores HTML y ATOM.

El primer paso es decidir la URL y el metodo HTTP que vamos a utilizar. Todas las URL de los web scripts deben comenzar por /alfresco/service/ , el resto es libre aunque con la restricción de que debe ser único.

```
GET /alfresco/service/sample/blog/buscar?q={terminoDeBusqueda}
```

Las siguientes espacios de nombres URL están reservados por Alfresco:

```
/alfresco/service/sample
/alfresco/service/api
/alfresco/service/ui
/alfresco/service/office
```

Una vez decidida la URL de acceso al web script debemos realizar la implementación. Crear un web script incluye los siguientes pasos:

1. Crear el descriptor XML. Este documento de descripción describe la URL que iniciará la ejecución del script.
2. Crear el script de ejecución si es necesario. Por ejemplo el script puede consultar el Repositorio de Alfresco para construir un conjunto de datos para pintar en la respuesta o, para URLs que quieran modificar el repositorio, el script tendrá que hacerlo. El JavaScript tiene acceso a todos los argumentos de la URL, los servicios de Alfresco y los puntos de entrada del repositorio.
3. Crear las plantillas de respuesta para realizar la presentación del resultado en el formato que se necesite. Las plantillas tendrán acceso a todos los argumentos de la URL, los puntos de entrada comunes del repositorio y cualquier dato construido por la ejecución del script.

Crear el descriptor XML

El descriptor es un documento .xml que se debe almacenar en una de las carpetas de web scripts, o en cualquiera de sus subcarpetas:

```
/Company Home/Data Dictionary/Web Scripts/org/alfresco/sample/busquedaenblog.get.desc.xml
```

Esta será la ruta completa para el web script de búsqueda en blogs. La nomenclatura es importante. Los Web Scripts emplean convenciones sobre la configuración de su definición para reducir la cantidad de escritura requerida. La terminación .desc.xml indica a alfresco que se trata de un documento de descripción de Web Scripts. El nombre completo de la carpeta y el fichero definen:

- Un paquete de org/alfresco/sample.
- Un id de servicio busquedaenblog.
- Un vinculo al método HTTP GET.

El contenido del documento de definición tendrá la siguiente estructura:

```
<webscript>
  <shortname>Busqueda Blog</shortname>
  <description>Encuentra todas las entradas del blog que contengan el termino de busqueda</description>
  <url>/sample/blog/buscar?q={terminoBusqueda}</url>
  <url>/sample/blog/buscar.atom?q={terminoBusqueda}</url>
```

```
  <url>/sample/b/b?q={terminoBusqueda}</url>
  <url>/sample/b/b.atom?q={terminoBusqueda}</url>
  <format default="html">extension</format>
  <authentication>guest</authentication>
  <transaction>required</transaction>
</webscript>
```

El elemento *shortname* es un nombre legible para el web script. El elemento *description* es opcional y sirve para documentar la funcionalidad.

Podemos tener uno o varios elementos *url*. Estos elemento definen las plantillas URL a las cuales el web script esta vinculado. No es necesario registrar las variantes de las plantillas url que indican el formato *(search.atom)* pero no esta de mas hacerlo por motivos de documentación.

El elemento *format* especifica como es declarado el tipo de la respuesta en la url. Los valores validos son:

- *argument:* El tipo de contenido sera declarado con un argumento llamado format en la url. *(/buscar?q=tutorial&format=atom)* .
- *extension:* El tipo de argumento sera declarado usando la extensión de la url. Por ejemplo. *(/buscar.atom?q=tutorial)*.
- *any:* Cualquiera de las dos formas anteriores esta permitida. Este será el valor por defecto si el elemento no es especificado.

Si no se especifica ningún valor de formato en la URL el valor por defecto es tomado del atributo opcional default. Si este valor no ha sido declarado se asumirá por defecto el formato html. En algunos casos se necesita decidir el tipo de contenido en tiempo de ejecución, para estos casos se especifica un formato vacío, por ejemplo con el argumento *format*="".

El elemento opcional *authentication* indica los requerimientos a nivel de autenticación. Los valores validos son:

- *none:* No es requerida ningún tipo de autenticación. Este es el valor por defecto si el elemento no es especificado.
- *guest:* Al menos es requerido autenticarse como invitado para acceder al web script.
- *user:* Al menos es requerido autenticarse como un usuario para acceder al web script.
- *admin:* Es necesario autenticarse como admin para acceder al web script.

El elemento opcional *transaction* indica el nivel requerido de transacción. Los valores validos son:

- *none:* Indica que no se necesita ninguna declarar una transacción.
- *required:*Indica que se necesita una transacción para la ejecución del web script. Se puede heredar una transacción que ya este abierta.
- *requiresnew:* Indica que se requiere una nueva transacción.

Si no se especifica este elemento el valor por defecto es *none* si el valor del elemento *authentication* es también *none*. En cualquier otro caso el valor por defecto es *required*.

En resumen esta descripción suministra a Alfresco toda la información necesaria para vincular el web script con una o mas URLs. También se suministra suficiente información para documentarlo.

Plantillas URL

Una plantilla URL es simplemente un url que contiene tokens que pueden ser sustituidos con valores reales. La sintaxis de una plantilla URL es simple y sencilla. Ejemplos comunes de plantillas URL incluyen:

```
/content/a/b/c/d.txt
  /blog/buscar?q={terminoBusqueda}
  /blog/buscar?q={terminoBusqueda}&n={numResultados}
  /blog/categoria/{categoria}?n={itemsporpagina?}
  /usuario/{usuario}
```

Un token de la forma (?) significa que es opcional. Normalmente, se utilizan tokens para argumentos URL pero también puede ser usado para los segmentos de la ruta. En el tercer ejemplo se pasan dos parámetros al web script. Normalmente un ampersand (&) se utiliza para separar los parámetros, aunque en las plantillas hay que utilizar el literal "&" en su lugar.

Todos las plantillas URL se especifican relativas a :

```
http://<host>:<port>/alfresco/service.
```

Cuando se realiza una petición de URL, Alfresco localiza el web script mas adecuado buscando la mayor coincidencia con una plantilla URL en un documento de descripción. La parte de la plantilla URL que se utiliza para comprobar la coincidencia es la parte estática, esto es desde el comienzo de la plantilla hasta cualquiera de los argumentos o el primer token de ruta. Por ejemplo, en las plantillas URL de antes se usan estas partes para buscar la coincidencia:

```
/content/a/b/c/d.txt (coincidencia exacta)
/blog/search (empieza con la cadena)
/blog/category (empieza con la cadena)
```

```
/user/ (empieza con la cadena)
```

En algunas de las siguientes URLs hay coincidencia con las anteriores y otras no:

```
Hay coincidencia
        /content/a/b/c/d.txt
        /blog/search
        /blog/search?q=tutorial
        /blog/category
        /blog/category/web20
        /user/fred
        No hay coincidencia
        /content/a/b/c/d.txt.plain
        /blog
        /usr/fred
```

Formatos de respuesta

Un web script soporta intrínsicamente un formato de respuesta por defecto. Un formato de respuesta es un nombre corto registrado (atom) que identifica un tipo/subtipo MIME (application/atom+xml).

No obstante es posible para cualquier web script soportar múltiples tipos de respuesta. Cuando este es el caso es necesario determinar que formato se escoge cuando un web script es invocado. Esto se consigue codificando el formato en la URL ya sea como un argumento (format) o como una extensión. Cuando no se especifica de ninguna de estas formas siempre se escoge el formato por defecto.

Los formatos soportados por en el modelo original son:

Nombre Registrado	Tipo Mime
html	text/html
text	text/plain
xml	text/xml
atom	application/atom+xml
rss	application/rss+xml
json	application/json
opensearchdescription	application/opensearchdescription+xml
mediawiki	text/plain (Media Wiki markup)
porlet	text/html (head & tail chopped)

Crear el script de ejecución

Un web script puede opcionalmente ejecutar código JavaScript en algunas invocaciónes de su respectiva URL. El código JavaScript puede realizar consultas o actualizaciones contra el repositorio. Además, el código JavaScript puede construir un modelo de datos que se pasa a la plantilla de respuesta adecuada para su posterior prestación.

Por ejemplo, para el buscador en blogs, se va a crear el siguiente fichero en la misma carpeta que el documento de descripción de web script.

```
busquedaenblog.get.js
```

También se aplican para los scripts las convenciones de nombres. Así los scripts de ejecución han de tener la siguiente estructura:

```
<idDeSevicio>.<metodoHTTP>.js
```

La API JavaScript de Alfresco es completamente accesible desde los web scripts. Sin embargo, hay algunas diferencias en cuanto a la disponibilidad de los objetos raíz: los objetos *document, space y script ~~* no estarán disponibles y los objetos *~~companyhome, person y userhome* solo estará disponible si se esta autenticado.

Por contra los web scripts tienen accesibles algunos objetos raíz adicionales:

- *argsM:* Una matriz asociativa de cualquier parámetro de URL (donde cada llave es un nombre de parámetro y cada valor es una matriz que contiene los valores de los parámetros, aunque sólo uno sea suministrado) - complementa al objeto raíz args. Véase el ejemplo. (disponible a partir de V2.1 Enterprise).
- *url:* Proporciona acceso a la url (o partes de la url) que desencadeno el web script.
- *formdata:* Proporciona acceso a peticiones multipart/form-data permitiendo la subida de ficheros por medio de web scripts. (Disponible a partir de V2.1 Enterprise).
- *model:* Un array asociativo vacío que puede ser rellenado por el JavaScript. Los valores almacenado aquí estarán disponibles como objetos raíz para las plantillas de respuesta.
- *roothome:* El nodo raíz del Repositorio.

- *guest:* Un booleano que indica si el web script se esta ejecutando como "Guest"
- *server:* Un array de meta-datos que describen las propiedades del servidor del Repositorio que aloja el web script.

Para el buscador en blogs se podría escribir el siguiente JavaScript para realizar la búsqueda:

```
var nodes = search.luceneSearch("TEXT:" + args.q);
   model.resultset = nodes;
```

El resultado de la búsqueda se añade al objeto *raíz mode*l y se le da el nombre *resultset*. Las plantillas web script presentadas después de la ejecución de los JavaScript ahora también tienen el objeto raíz llamado *resultset*, que en este caso representa una serie de nodos que coincidan con los criterios de búsqueda.

Por supuesto, un web script también pueden realizar actualizaciones, en cuyo caso, el objeto raíz *model* se puede utilizar para grabar los datos que han sido actualizados, el estado....

Crear una plantilla de respuesta

La fase final de un web script es presentar una respuesta para la petición HTTP. Se pueden proporcionar multitud de formatos de respuesta. Las respuestas son presentadas usando plantillas FreeMarker.

Por ejemplo, para el buscador en blogs, se va a crear el siguiente fichero en la misma carpeta que el documento de descripción de web script.

```
busquedaenblog.get.html.ftl
```

También se aplican para las plantillas las convenciones de nombres. Así los ficheros con las plantillas de respuesta han de tener la siguiente estructura:

```
<idDeSevicio>.<metodoHTTP>.<formato>.ftl
```

donde formato especifica el formato de la plantilla de respuesta y debe ser uno de los que vimos en los apartados anteriores.

La API de Plantillas de Alfresco es completamente accesible desde los web scripts. Sin embargo, hay algunas diferencias en cuanto a la disponibilidad de los objetos raíz: los objetos *document, space y template* no estarán disponibles y los objetos *companyhome, person y userhome* solo estará disponible si se esta autenticado.

Por contra los web scripts tienen accesibles algunos objetos raíz adicionales:

- *argsM:* Una matriz asociativa de cualquier parámetro de URL (donde cada llave es un nombre de parámetro y cada valor es una matriz que contiene los valores de los parámetros, aunque sólo uno sea suministrado) - complementa al objeto raíz *args*. Véase el ejemplo. (disponible a partir de V2.1 Enterprise).
- *guest:* Un booleano que indica si el web script se esta ejecutando como "Guest".
- *date:* Una representación en formato fecha del dia y la hora de la llamada al web script.
- *server:* Un array de meta-datos que describen las propiedades del servidor del Repositorio que aloja el web script.
- *roothome:* El nodo raíz del Repositorio.
- *webscript:* Un array de meta-datos que describen las propiedades del web script.
- *url:* Proporciona acceso a la url (o parts de la Url) que desencadeno el web script.

Ademas, la plantilla también tiene acceso a cualquier objeto raíz creado por el script de ejecución.

Existen también algunos métodos adicionales a los que tienen acceso las plantillas:

- *absurl(url):* Devuelve una representación URL absoluta de la url pasada por parametro. Es útil cuando se están presentando enlaces sin ATOM(o un formato similar)
- *xmldate(date):* Devuelve como resultado una representación formateado según el estandar ISO8601 de la fecha pasada por parámetro. Es útil cuando se representen fechas con XML.
- *scripturl(queryString):* Devuelve la url que referencia al web script. La cadena *queryString* pasada por parametro es añadida a la url. Argumentos del sistema como *format* son añadidos automáticamente también. Este método es particularmente útil para aislarse del entorno en el que está ejecutando el web script. En algunos entornos de ejecución la url puede estar codificada.
- *clienturlfunction(funcName):* Genera una función JavaScript en el cliente que puede generar un URL de vuelta al web script.

Para el buscador de referencias en blogs podemos escribir la siguiente plantilla FreeMarker para presentar una respuesta HTML:

```
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    Blog query: ${args.q}
    <br>
    <table>
      <#list resultset as nodo>
```

```
    <tr>
      <td><img src="/servicios/madeja/%24%7Burl.context%7D%24%7Bnodo.icon16%7D"/>
      <td><a href="/servicios/madeja/%24%7Burl.serviceContext%7D/api/node/content/%24%7Bnodo.nodeRef.storeRe
    </tr>
    </#list>
   </table>
  </body>
</html>
```

Podemos observar como la plantilla hace uso de los objetos raíz estándar(${args.q}), objetos raíz específicos de los web script, por ejemplo ${url.context}, y objetos raíz creados por el script de ejecución, por ejemplo <#list resultset as nodo>.

Respuestas en múltiples formatos

Las respuestas en múltiples formatos son posibles gracias a la creación de ficheros de plantillas adicionales siguiendo la convención de nombres que se vieron antes.

Para el ejemplo de buscador en blogs, podríamos tener una respuesta en formato ATOM añadiendo el siguiente fichero:

```
blogsearch.get.atom.ftl
```

cuyo contenido sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <generator version="${server.version}">Alfresco (${server.edition})</generator>
  <title>Blog query: ${args.q}</title>
  <updated>${xmldate(date)}</updated>
  <icon>${absurl(url.context)}/images/logo/AlfrescoLogo16.ico</icon>
  <#list resultset as nodo>
  <entry>
    <title>${node.name}</title>
    <link rel="alternate" href="/servicios/madeja/%24%7Babsurl%28url.serviceContext%29%7D/api/node/content/%24%
    <icon>${absurl(url.context)}${node.icon16}</icon>
    <id>urn:uuid:${nodo.id}</id>
    <updated>${xmldate(node.properties.modified)}</updated>
    <summary>${nodo.properties.description!""}</summary>
    <author>
      <name>${nodo.properties.creator}</name>
    </author>
  </entry>
  </#list>
</feed>
```

Uniendo esta plantilla a la anteriormente definida y al documento de descripción se realizarán los siguientes mapeos:

```
/alfresco/service/blog/buscar?q=tutorial & ->  plantilla HTML
/alfresco/service/blog/buscar.html?q=tutorial & ->   plantilla HTML
/alfresco/service/blog/buscar?q=tutorial&format=atom & ->  plantilla ATOM
/alfresco/service/blog/buscar.atom?q=tutorial & ->  plantilla ATOM
```

Respuesta de Estado

Los web scripts usan las respuesta asociadas a lo códigos de estado HTTP para informar al cliente de un error, de un evento, del éxito en la operación o pedir al cliente que realice alguna acción de seguimiento de la solicitud.

Las respuestas asociadas a los códigos de estado son fijadas en el script de ejecución mediante el objeto raíz *status*. Se puede añadir un estado al ejemplo de búsqueda en blogs indicando que el termino de búsqueda no ha sido proporcionado en la URL de búsqueda. Se usará el código de estado 400 (Solicitud Incorrecta). Ampliaremos el script de ejecución con el siguiente código:

```
if (args.q is null || args.q.length == 0)
{
   status.code = 400;
   status.message = "Termino de busqueda no indicado.";
   status.redirect = true;
```

```
}
else
{
   // se realiza la busqueda como antes
   var nodes = search.luceneSearch("TEXT:" + args.q);
   model.resultset = nodes;
}
```

La propiedad *redirect* nos permite mostrar una plantilla personalizada de respuesta para un código de estado especifico. Si la propiedad *redirect* es igual a false se usará la plantilla estándar de respuesta adecuada al código de estado. Si la propiedad *redirect* es true, entonces se realiza una búsqueda de una plantilla de respuestas adecuada. La búsqueda se realiza en el siguiente orden:

- Plantilla de Web Script especifica llamada:

<idDeServicio>.<metodoHTTp>.<formato>.<codigo>.ftl.

Localizado en la misma carpeta que el documento de descripción y usado para el código de estado especifico.

```
/org/alfresco/sample/busquedaenblogs.get.html.400.ftl
```

- Plantilla de Web Script especifico llamada:

<idDeServicio>.<metodoHTTp>.<formato>.status.ftl.

Localizado en la misma carpeta que el documento de descripción y usado para cualquier código de estado.

```
/org/alfresco/sample/busquedaenblogs.get.html.status.ftl
```

- Plantilla a nivel de paquete llamada <formato>.<codigo>.ftl.

Se busca primero en el paquete del web script y después a través de la jerarquía de paquetes hasta el raíz.

```
/org/alfresco/sample/html.400.ftl
```

- Plantilla a nivel de paquete llamada <formato>.status.ftl.

Se busca primero en el paquete del web script y después a través de la jerarquía de paquetes hasta el raíz.

```
/org/alfresco/sample/html.status.ftl
```

- Plantilla por defecto llamada <code>.ftl

Se encuentra en el paquete raíz y siempre es en formato .html.

```
/400.ftl
```

- Plantilla por defecto llamada status.ftl

Se encuentra en el paquete raíz (viene en la distribución por defecto) y siempre es en formato .html.

Las plantillas de respuesta de estado tiene acceso a los mismos objetos raíz que las plantillas de respuesta normales de web script con la adición del objeto *status*. La excepción son las plantillas por defecto */<code>.ftl* y */status.ftl* solo tienen acceso a los objetos raíz *url, status, server* y *date*.

Para proporcionar una respuesta de estado personalizado para el código 400 en el web script de búsqueda en un blog se creará el siguiente fichero en el mismo directorio donde está el documento de descripción:

```
busquedaenblog.get.html.400.ftl
```

con el contenido:

```
<html>
  <body>
    ${status.message}
  </body>
</html>
```

Estos ficheros son opcionales, la búsqueda de una planilla adecuada para devolver un error siempre termina encontrando la plantilla *status.fl*. El fichero de recursos */alfresco/messages/webscripts.properties* contiene los nombres y descripciones para todos los códigos de estado HTTP.

Almacenar un web scripts

Se pueden almacenar los web scripts en cualquiera de las siguientes carpetas, se muestran en el orden en que Alfresco busca en ellos los web scripts:

1. Carpeta del repositorio: /Company Home/Data Dictionary/Web Scripts Extensions
2. Carpeta del repositorio: /Company Home/Data Dictionary/Web Scripts
3. Carpeta del classpath: /alfresco/extension/templates/webscript
4. Carpeta del classpath: /alfresco/templates/webscript

En cualquiera de esas carpetas se pueden usar subcarpetas para organizar los web script. Nunca se debe almacenar en /org/alfreco ni en ninguno de sus subcarpetas ya que están reservadas para el uso de Alfresco.

Registrar un web script y listar los web scripts registrados

Los web scripts tanto nuevos como actualizados pueden registrarse sin necesidad de reiniciar el servidor de Alfresco. Esto facilita el uso del repositorio de Alfresco para desarrollar web scripts.

Para registrar un web script:

- En la barra de direcciones del navegador escribir la siguientes URLs:

```
http://<host>:<port>/alfresco/service/
http://<host>:<port>/alfresco/service/index
```

Aparece la home de Web Scripts.

- Hacer click en "*Refresh list of Web Script*". Alfresco registrará cualquier web script que se halla añadido desde la última vez que se pulso el botón.

Desde esta pagina se tiene acceso al listado de todos los web scripts instalados, y además se permite recorrerlos navegando por paquete o por url.

Ejemplos

En la siguiente pagína podremos encontrar una serie de codigos de ejemplo de construcción de web scripts

Recursos

Área: **Arquitectura** » **Arquitectura de Sistemas de Información** » **Alfresco** » **Uso de Alfresco desde Terceras Aplicaciones**

Código	Título	Tipo	Carácter
RECU-0046	Ejemplos de Web Scripts	Ejemplo	Recomendado

Ejemplos de Web Scripts

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Carácter del recurso:** Recomendado

Código: RECU-0046
Tipo de recurso: Ejemplo

Descripción

Para cada ejemplo, se mostrara la siguiente información:

- **URL:** Define el formato de la URL usado para acceder al Script
- **Examples:** Un ejemplo concreto de una URL valida para acceso al Script
- **Documento de Descripción**
- **Script Ejecutable**
- **Plantilla de Respuesta**

Ejemplos

Hola Mundo

Este un ejemplo muy simple de como crear un web script. Aqui se puede encontrar una guia paso a paso de como crear este web script.

URL

```
GET /alfresco/service/sample/hola
```

Ejemplo:

```
http://<host>:<port>/alfresco/service/sample/hola
```

Documento de descripción

```
Fichero: hola.get.desc.xml
```

```
<webscript>
  <shortname>Hola</shortname>
  <description>Saludos educados</description>
  <url>/sample/hola</url>
  <authentication>user</authentication>
</webscript>
```

Plantilla de Respuesta

```
Fichero: hola.get.html.ftl
Hola ${person.properties.userName}.
```

Navegador de Carpetas/RSS Feed

Este ejemplo muestra como implementar un servicio URL que muestra los contenidos de una carpeta en formato HTML o ATOM. La representación HTML permite la navegación de una jerarquia de carpetas.

URL

```
GET /alfresco/service/sample/carpeta/{ruta}
```

Ejemplos:

```
http://<host>:<port>/alfresco/service/sample/carpeta/Company%20Home
http://<host>:<port>/alfresco/service/sample/carpeta/Company%20Home?format=atom
```

Documento de Descripcion

```
Fichero: folder.get.desc.xml
<webscript>
  <shortname>Ejemplo de listado de carpeta</shortname>
  <description>Ejemplo que demuestra el listado de contenidos de carpetas</description>
  <url>/sample/carpeta/{path}</url>
```

```
  <format default="html">argument</format>
  <authentication>guest</authentication>
  <transaction>required</transaction>
</webscript>
```

Script de Ejecucion

```
Fichero: carpeta.get.js
// localiza la carpeta por la ruta
var carpeta = roothome.childByNamePath(url.extension);
if (carpeta == undefined || !carpeta.isContainer)
{
   status.code = 404;
   status.message = "Carpeta " + url.extension + " no encontrada.";
   status.redirect = true;
 }
 model.folder = carpeta;
```

Plantilla de respuesta

```
Fichero: carpeta.get.html.ftl
<html>
  <head>
    <title>${folder.displayPath}/${folder.name}</title>
  </head>
  <body>
    Folder: ${folder.displayPath}/${folder.name}
    <br>
    <table>
     <#if folder.parent.parent?exists>
     <tr>
       <td><td><a href="/servicios/madeja/%24%7Burl.serviceContext%7D/sample/folder%2526lt%3B%40encodepath
     </tr>
     </#if>
     <#list folder.children as child>
     <tr>
       <#if child.isContainer>
         <td>><td><a href="/servicios/madeja/%24%7Burl.serviceContext%7D/sample/carpeta%2526lt%3B%40encod
       <#else>
         <td><td><a href="/servicios/madeja/%24%7Burl.serviceContext%7D/api/node/content/%24%7Bchild.nodeRef
       </#if>
     </tr>
</#list>
    </table>
  </body>
</html>
<#macro encodepath node><#if node.parent?exists><@encodepath node=node.parent/>/${node.name?url}</#if
```

```
Fichero: carpeta.get.atom.ftl
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <generator version="${server.version}">Alfresco (${server.edition})</generator>
  <title>Carpeta: ${folder.displayPath}/${folder.name}</title>
  <updated>${xmldate(date)}</updated>
  <icon>${absurl(url.context)}/images/logo/AlfrescoLogo16.ico</icon>
<#list folder.children as child>
  <entry>
    <title>${child.name}</title>
    <#if child.isContainer>
      <link rel="alternate" href="/servicios/madeja/%24%7Babsurl%28url.serviceContext%29%7D/sample/carpeta%2526lt
    <#else>
```

```
    <link rel="alternate" href="/servicios/madeja/%24%7Babsurl%28url.serviceContext%29%7D/api/node/content/%24%
   </#if>
   <icon>${absurl(url.context)}${child.icon16}</icon>
   <id>urn:uuid:${child.id}</id>
   <updated>${xmldate(child.properties.modified)}</updated>
   <summary>${child.properties.description!""}</summary>
   <author>
    <name>${child.properties.creator}</name>
   </author>
  </entry>
</#list>
</feed>
<#macro encodepath node><#if node.parent?exists><@encodepath node=node.parent/>/${node.name?url}</#if>
```

Lista de Stores WCM

Este ejemplo demuestra como implementar un servicio URL que muestra las lista de todos los stores WCM

URL

```
GET /alfresco/service/sample/avm/stores
```

Ejemplos:

```
http://<host>:<port>/alfresco/service/sample/avm/stores
```

Documento de Descripcion

```
Fichero: avmstores.get.desc.xml
<webscript>
  <shortname>Ejemplo de Stores AVM</shortname>
  <description>Ejemplo que muestra el listado de los stores AVM</description>
  <url>/sample/avm/stores</url>
  <format default="html"/>
  <authentication>admin</authentication>
  <transaction>required</transaction>
</webscript>
```

Plantilla de Respuesta

```
Fichero: avmstores.get.html.ftl
<html>
  <head>
    <title>Stores AVM</title>
  </head>
  <body>
    Stores AVM
    <br>
    <br>
    <table>
     <tr>
     <#list avm.stores as store>
     <tr>
        <td>${store.creator}<td> <td>${store.createdDate?datetime}<td> <td><a href="/servicios/madeja/%24%7B
     </tr>
</#list>
    </table>
  </body>
</html>
```

Navegador de carpetas WCM

Este ejemplo demuestra como implementar un servicio URl que muestre los contenidos de una carpeta WCM en html. Esta muestra permitira la navegación por la categoria de carpetas.

URL

```
GET /alfresco/service/sample/avm/path/{storeid}/{path}
```

Ejemplos:

```
http://<host>:<port>/alfresco/service/sample/avm/path/main--admin/www
```

Documento de Descripcion

```
Fichero: avmbrowse.get.desc.xml
<webscript>
  <shortname>Ejemplo de Navegador AVM</shortname>
  <description>ejemplo que demuestra como listar y navegar a través de las carpetas AVM </description>
  <url>/sample/avm/path/{storeid}/{path}</url>
  <format default="html">argument</format>
  <authentication>admin</authentication>
  <transaction>required</transaction>
</webscript>
```

Script de Ejecucion

```
Fichero: avmbrowse.get.js
script:
{
   // extract avm store id and path
   var fullpath = url.extension.split("/");
   if (fullpath.length == 0)
   {
     status.code = 400;
     status.message = "Id del Store no suministrado.";
     status.redirect = true;
     break script;
   }
   var storeid = fullpath[0];
   var path = (fullpath.length == 1 ? "/" : "/" + fullpath.slice(1).join("/"));

   // locate avm node from path
   var store = avm.lookupStore(storeid);
   if (store == undefined)
   {
     status.code = 404;
     status.message = "Store " + storeid + " no encontrado.";
     status.redirect = true;
     break script;
   }
   var node = avm.lookupNode(storeid + ":" + path);
   if (node == undefined)
   {
     status.code = 404;
```

Plantilla de Respuesta

```
Fichero: avmbrowse.get.html.ftl
<html>
  <head>
    <title>AVM Folder: ${folder.displayPath}/${folder.name}</title>
  </head>
  <body>
    <a href="/servicios/madeja/%24%7Burl.serviceContext%7D/sample/avm/stores">Store AVM</a>: ${store.id}
    <br>
    <br>
    Carpeta AVM: ${folder.displayPath}/${folder.name}
    <br>
```

```
    <br>
    <table>
     <#if folder.parent?exists>
     <tr>
      <td>${folder.parent.properties.creator}<td> <td>${folder.parent.size}<td> <td>${folder.parent.properties.m
     </tr>
     </#if>
<#list folder.children as child>
     <tr>
       <#if child.isContainer>
        <td>${child.properties.creator}<td> <td>${child.size}<td> <td>${child.properties.modified?datetime}<td>
       <#else>
         <td>${child.properties.creator}<td> <td>${child.size}<td> <td>${child.properties.modified?datetime}<td>
       </#if>
     </tr>
</#list>
```

Busqueda tipo Blog

Este ejemplo muestra como implementar un serviocio URL que realice una busqueda completa de tecto cuyos resultados serán mostrados como HTML o ATOM

URL

```
GET /alfresco/service/sample/blog/search?q={searchTerm}
```

Ejemplos:

```
http://<host>:<port>/alfresco/service/sample/blog/search?q=alfresco
http://<host>:<port>/alfresco/service/sample/blog/search.atom?q=alfresco
```

Documento Descripcion

```
Fichero: blogsearch.get.desc.xml
<webscript>
  <shortname>Busqueda</shortname>
  <description>Este ejemplo busca todas la entradas cuyo contenido incluye un termino especificado</description>
  <url>/sample/blog/search?q={searchTerm}</url>
  <url>/sample/blog/search.atom?q={searchTerm}</url>
  <url>/sample/b/s?q={searchTerm}</url>
  <url>/sample/b/s.atom?q={searchTerm}</url>
  <format default="html">extension</format>
  <authentication>guest</authentication>
  <transaction>required</transaction>
</webscript>
```

Script de Ejecucion

```
Fichero: blogsearch.get.js
// comprobar el termino de la busqueda
if (args.q == undefined || args.q.length == 0)
{
   status.code = 400;
   status.message = "Termino de busqueda no proporcionado.";
   status.redirect = true;
}
else
{
   // realizar la busqueda
   var nodes = search.luceneSearch("TEXT:" + args.q);
   model.resultset = nodes;
}
```

NOTA: El script anterior ejecuta una busqueda completa en todo el repositorio. Se podría extender para buscar solo en una carpeta especifica o xon un tipo de contenido especifico.

Response Templates

Fichero: blogsearch.get.html.ftl

```
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    Busqueda: ${args.q}
    <br>
    <table>
<#list resultset as node>
     <tr>
       <td><img src="/servicios/madeja/%24%7Burl.context%7D%24%7Bnode.icon16%7D"/>
       <td><a href="/servicios/madeja/%24%7Burl.serviceContext%7D/api/node/content/%24%7Bnode.nodeRef.storeRe
     </tr>
</#list>
    </table>
  </body>
</html>
```

Fichero: blogsearch.get.atom.ftl

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <generator version="${server.version}">Alfresco (${server.edition})</generator>
  <title>Blog query: ${args.q}</title>
  <updated>${xmldate(date)}</updated>
  <icon>${absurl(url.context)}/images/logo/AlfrescoLogo16.ico</icon>
<#list resultset as node>
  <entry>
    <title>${node.name}</title>
    <link rel="alternate" href="/servicios/madeja/%24%7Babsurl%28url.serviceContext%29%7D/api/node/content/%24%
    <icon>${absurl(url.context)}${node.icon16}</icon>
    <id>urn:uuid:${node.id}</id>
    <updated>${xmldate(node.properties.modified)}</updated>
    <summary>${node.properties.description!""}</summary>
    <author>
      <name>${node.properties.creator}</name>
    </author>
  </entry>
</#list>
</feed>
```

Plantilla de Estado

Fichero: blogsearch.get.html.404.ftl

```
<html>
  <body>
    ${status.message}
  </body>
</html>
```

File: blogsearch.get.atom.404.ftl

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  ${status.code}
  <codeName>${status.codeName}</codeName>
  <codeDescription>${status.codeDescription}</codeDescription>
  <message>${status.message}</message>
</response>
```

Busqueda por categorias tipo Blog

Este ejemplo muestra como implementar un servicio URL que realiceuna busqueda por categorias cuyos resultados sran mostrados en HTML o ATOM.

URL

```
GET /alfresco/service/sample/blog/category/{category}
```

Ejemplos:

```
http://<host>:<port>/alfresco/service/sample/blog/category/EUROPE
```

Documentos de Descripcion

```
Fichere: categorysearch.get.desc.xml
<webscript>
  <shortname>Busqueda por Categorias</shortname>
  <description>Encuentra todas las entradas marcadas con las categorias indicadas</description>
  <url>/sample/blog/category/{category}</url>
  <format default="html">extension</format>
  <authentication>guest</authentication>
  <transaction>required</transaction>
</webscript>
```

Script de Ejecucion

```
Fichero: categorysearch.get.js
// comproba la existencia de la categoría?
var category = search.luceneSearch("PATH:\"/cm:generalclassifiable//cm:" + url.extension + "\"");
if (category == undefined)
{
   status.code = 404;
   status.message = "Categoria " + url.extension + " no encontrada.";
   status.redirect = true;
}
else
{
   // realizar la busqueda por categorias
   var nodes = search.luceneSearch("PATH:\"/cm:generalclassifiable//cm:" + url.extension + "//member\"");
   model.resultset = nodes;
}
```

NOTA: Al igual que en el ejemplo anterior esta busqueda por todo el repositorio se puede extender para buscar solo en una carpeta concreta o un tipo especifico de contenido.

Plantillas de Respuesta

```
Fichero: categorysearch.get.html.ftl
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    Categoria buscada: ${url.extension}
    <br>
    <table>
<#list resultset as node>
     <tr>
       <td><img src="/servicios/madeja/%24%7Burl.context%7D%24%7Bnode.icon16%7D"/>
       <td><a href="/servicios/madeja/%24%7Burl.serviceContext%7D/api/node/content/%24%7Bnode.nodeRef.storeRe
     </tr>
</#list>
    </table>
  </body>
</html>
```

Fichero: categorysearch.get.atom.ftl

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <generator version="${server.version}">Alfresco (${server.edition})</generator>
  <title>Categoría buscada: ${url.extension}</title>
  <updated>${xmldate(date)}</updated>
  <icon>${absurl(url.context)}/images/logo/AlfrescoLogo16.ico</icon>
<#list resultset as node>
  <entry>
    <title>${node.name}</title>
    <link rel="alternate" href="/servicios/madeja/%24%7Babsurl%28url.serviceContext%29%7D/api/node/content/%24%7
    <icon>${absurl(url.context)}${node.icon16}</icon>
    <id>urn:uuid:${node.id}</id>
    <updated>${xmldate(node.properties.modified)}</updated>
    <summary>${node.properties.description!""}</summary>
    <author>
      <name>${node.properties.creator}</name>
    </author>
  </entry>
</#list>
</feed>
```

Plantilla de Estado

Fichero: categorysearch.get.html.404.ftl

```
<html>
  <body>
    ${status.message}
  </body>
</html>
```

Fichero: categorysearch.get.atom.404.ftl

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <code>${status.code}</code>
  <codeName>${status.codeName}</codeName>
  <codeDescription>${status.codeDescription}</codeDescription>
  <message>${status.message}</message>
</response>
```

Subida de Ficheros

Este ejemplo muestra como implementar un servicio URL para la subida de ficheros dividido en dos web scipts . (funcionalidad disponible a partir de la versión 2.1 Enterprise)

El primero presenta formulario HTML simple con tres campos, uno de los cuales es el fichero. El formulario llama al segundo web script el cual obtiene y crea el nuevo fichero en el repositorio usando los valores insertados en el formulario.

Formulario de Subida

URL

```
GET /sample/upload
```

Documento de descripcion

Fichero: upload.get.desc.xml

```
<webscript>
  <shortname>Ejemplo de formulario de subida de ficheros</shortname>
  <description>Formulario para subir el contenido de un fichero y  los meta-datos al Repositorio</description>
  <url>/sample/upload</url>
  <authentication>user</authentication>
</webscript>
```

Plantilla de Respuesta

File: upload.get.html.ftl

```
<html>
 <head>
  <title>Web Script de Subida</title>
  <link rel="stylesheet" href="/servicios/madeja/%24%7Burl.context%7D/css/main.css" TYPE="text/css">
 </head>
 <body>
  <table>
   <tr>
    <td><img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" /></td
    <td><nobr>Web Script de Subida</nobr></td>
   </tr>
   <tr><td><td>Alfresco ${server.edition} v${server.version}
  </table>
  <p>
  <table>
   <form action="${url.service}" method="post" enctype="multipart/form-data" accept-charset="utf-8">
    <tr><td>Fichero:<td><input type="file" name="file">
    <tr><td>Titulo:<td><input name="title">
    <tr><td>Descripcion:<td><input name="desc">
    <tr><td><td>
    <tr><td><td><input type="submit" name="submit" value="Upload">
   </form>
  </table>
 </body>
</html>
```

Script de Subida

URL

```
POST /sample/upload
```

Documento de Descripcion

Fichero: upload.post.desc.xml

```
<webscript>
  <shortname>Eejmplo de Subida de Ficheros</shortname>
  <description>Inserta en el Repositorio un fichero y meta-datos</description>
  <url>/sample/upload</url>
  <authentication>user</authentication>
</webscript>
```

Script de Ejecucion

Fichero: upload.post.js

```
var filename = null;
var content = null;
var title = "";
var description = "";
// coger los atributos del fichero
for each (field in formdata.fields)
{
  if (field.name == "title")
  {
    title = field.value;
  }
  else if (field.name == "desc")
  {
    description = field.value;
  }
  else if (field.name == "file" && field.isFile)
```

```
  {
    filename = field.filename;
    content = field.content;
  }
}
// asegurarse de que los atributos obligatorios del fichero an sido obtenidos
if (filename == undefined || content == undefined)
{
  status.code = 400;
  status.message = "No se puede localizar el fichero a insertar en el request";
  status.redirect = true;
```

Plantilla de Respuesta

```
Fichero: upload.post.html.ftl
<html>
 <head>
   <title>Ejemplo de subida de ficheros mediante Web Script</title>
   <link rel="stylesheet" href="/servicios/madeja/%24%7Burl.context%7D/css/main.css" TYPE="text/css">
 </head>
 <body>
   <table>
     <tr>
       <td><img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" /></td
       <td><nobr>Ejemplo de subida de ficheros mediante Web Script</nobr></td>
     </tr>
     <tr><td><td>Alfresco ${server.edition} v${server.version}
     <tr><td><td><td>
     <tr><td><td>Subido <a href="/servicios/madeja/%24%7Burl.serviceContext%7D/sample/folder%24%7Bupload.dis
   </table>
 </body>
</html>
```

Manejo de argumentos URL

Este ejemplo muestra como procesar srgumentos URL (unicos o multi-valor).

URL

GET /alfresco/service/sample/args

Ejemplos:

http://<host>:<port>/alfresco/service/args?a=2&a=1&b=3

Documento de Descripcion

```
Fichero: args.get.desc.xml
<webscript>
 <shortname>Ejemplo de Manejo de Argumentos</shortname>
 <description>Muestra acceso a argumentos unicos and multi-valor</description>
 <url>/sample/args</url>
 <authentication>none</authentication>
</webscript>
```

Script de ejecución

```
Fichero: args.get.js
// insertar en el log cada argumento (asumiendo que solo se a suministrado un valor para cada uno)
for (arg in args)
{
  logger.log(arg + "=" + args[arg]);
}
// insertar en el log cada argumento (asumiendo que solo se a suministrado uno o mas valores para cada uno)
```

```
for (arg in argsM)
{
  for each (val in argsM[arg])
  {
     logger.log(arg + "=" + val);
  }
}
```

Plantillas de Respuesta

```
Fichero: args.get.html.ftl
<#list args?keys as arg>
  ${arg}=${args[arg]}
</#list>
<#list argsM?keys as arg>
<#list argsM[arg] as val>
  ${arg}=${val}
</#list>
</#list>
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/46

Web Scripts

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Carácter del recurso:** Recomendado

Código: RECU-0005
Tipo de recurso: Ficha

Descripción

Los Web Scripts son el metodo que proporciona Alfresco para acceder al repositorio a traves de una API estilo REST.

El término REST en la actualidad se usa para describir cualquier interfaz web simple que utiliza XML y HTTP, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios web SOAP. Es posible diseñar interfaces XMLHTTP de acuerdo con el estilo de llamada a procedimiento remoto pero sin usar SOAP. Los sistemas que siguen los principios REST se llaman con frecuencia RESTfull.

Recursos

Área: Arquitectura » Arquitectura de Sistemas de Información » Alfresco » Uso de Alfresco desde Terceras Aplicaciones

Código	Título	Tipo	Carácter
RECU-0045	Uso de Web Scripts	Manual	Recomendado

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/5

API de JavaScripts

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Carácter del recurso:** Recomendado

Código: RECU-0047
Tipo de recurso: Manual

Descripción

La API Javascript de Alfresco permite a los desarrollaores de scripts escribir ficheros compatibles con el estándar JavaScript (ECMA Script) 1.6 para acceder, modificar, y crear objetos del repositorio de Alfresco. Proporciona una API simple, clara y orientada a objetos hacia conceptos bien conocidos de Alfresco como Nodos, Propiedades, Asociaciones y Aspectos. Esta API es similar a la API de Templates con la diferencia de que está permite la modificación y creación de nodos, aspectos y propiedades.

Usando la API los desarrolladores de scripts podrán:

- Encontrar nodos (usando Xpath).
- Recorrer jerarquías de nodos.
- Realizar búsquedas (incluso búsquedas completas Lucene).
- Examinar y modificar el valor de propiedades y aspectos.
- Procesar proyectos web WCM.
- Crear Grupos.

También desde un script se podrán crear nuevos ficheros, espacios o nodos de cualquier tipo, copiar/mover/borrar nodos y crear, modificar o quitar asociaciones entre nodos. Se aplicará todo el sistema de seguridad y permisos ACL de Alfresco.

A partir de la versión 2.1 se ha incluido la capacidad de incluir otros scripts. El desarrollador podrá construir librerías de scripts que se incluiran en tiempo de ejecución para reducir los peligrosos copia/pega entre ficheros de script.

Los ficheros de Script pueden ser guardados en el ClassPath (por ejemplo, en alfresco/extension/myscripts) o en cualquier localización del repositorio.

Los Scripts se pueden ejecutar de dos maneras diferentes. La primera es creando una regla y seleccionando Ejecutar un Script desde la lista de acciones disponibles para la regla. Los Scripts que se muestran en esta lista son cargados desde el espacio */Company Home/Data Dictionary/Scripts* los usuarios con los permisos adecuados podrán crear, añadir o modificar los scripts disponibles.

La segunda forma es usar direccionamiento URL para un estilo REST de ejecución. El cliente web de Alfresco ofrece un "comando" servlet genérico que permite la ejecución directa de scripts a través de Urls. Esta característica permite el acceso directo a los scripts y el resultado de estos scripts se devuelve con una respuesta HTML. Usando este metodo podremos acceder a los scripts en el repositorio incluidos los que erán accesibles desde la acción Ejecutar un Script.

Aquellos Scripts que no estén en el repositorio sino que se encuentren dentro del ClassPath podrán ser importados en otros scripts pero no ejecutados directamente en el cliente web. La sintaxis para este importación es especifica de Alfresco con es una característica estándar de Javascript. Es ademas muy estricta y debe ser seguida exactamente o podría fallar. La directivas de importación deben ser las primeras lineas del fichero, ni comentarios ni código son permitidos por encima. La sintaxis puede ser una de las siguientes:

Importación de un script desde el repositorio mediante un path basado en el nombre:

```
<import resource="/Company Home/Data Dictionary/Scripts/library.js">
```

Importación de un script desde el repositorio mediante una referencia NodeRef:

```
<import resource="workspace://SpacesStore/6f73de1b-d3b4-11db-80cb-112e6c2ea048">
```

Importación de un script desde una localización del Java Classpath:

```
<import resource="classpath:alfresco/extension/myutils.js">
```

El motor de JavaScript es capaz de manejar la importación múltiple del mismo script y las dependencias circulares. Se podrán importar varios scripts desde diferentes localizaciones, uno desde el classpath y otros desde el repositorio por ejemplo.

API Java de ejecución de Scripts.

La API de JavaScript de Alfresco proporciona un rico conjunto de objetos Java que estarán disponibles para la escritura de los Scripts. Por defecto se proporcionan algunos objetos de ámbito "raiz", como el acceso a la carpeta de inicio del usuario, la carpeta de empresa, los proyectos web WCM, busqueda, People API y funcionalidad de login.

Objetos de ambito raiz (Root Scope Objects)

La mayoría de estos objetos por defecto son conocidos como objetos Script Node que abarcan los conceptos comunes del

repositorio de Alfresco como nodos, aspectos, asociaciones y propiedades. También proporcionan acceso a servicios comunes a través de una API orientada a objetos.

Independientemente de la forma en que se este accediendo al motor de script (a través de una regla/accion o mediante URLs) los siguientes objetos están disponibles por defecto en el ambito de ejecución de scripts raiz:

- **companyhome:** El Script Node del Company Home.
- **userhome:** La carpeta de inicio del usuario actual.
- **person:** Script Node que representa en objeto Person del usuarío actual.
- **space:** El Script Node del espacio actual si hay alguno. Hay que tener en cuenta que si el script se ejecuta desde una regla heredad este objeto representará al espacio donde la regla esta definida no al actual.
- **document:** El Script Node que representa el documento actual (si hay alguno).
- **script:** El Script Node que representa al script mismo. No está presenta si el script se carga desde el Java Classpath.
- **args**: Un array asociativo con los parametros URL pasados a través del Servlet de Procesamiento de Scripts, solo estará disponible si así es como se ha invocado el script.
- **search**: Un objeto host que proporciona acceso a resultados Lucene y de Busquedas Guardadas.
- **people**: Un objeto host que proporciona acceso a los grupos y personas de Alfresco.
- **actions:** Un objeto host que proporciona capacidad para invocar Acciones registradas de Alfresco.
- **logger:** Un objeto host que proporciona acceso a facilidades para acceder al log de la aplicación para poder realizar depuración de los scripts.
- **session:** Información relativa a la sesión actual así como al ticket de autenticación actual.
- **classification**: Acceso a los elementos raíces de la API de clasificación.
- **utils:** Acceso a una librería de funciones de ayuda útiles que no proporciona el JavaScript generico.
- **avm:** Acceso a los objetos WCM como a los paths AVM y busquedas con AVM stores y proyectos web.
- **crossRepoCopy:** Soporte para copia cruzada entre repositorios, espacios WCM y espacios de gestión documental.
- **workflow***: Un objeto host que proporciona acceso a los Workflow.

*Disponible a partir de la versión 3.0

El modelo de objetos por defecto se puede acceder directamente desde la raíz en el ámbito del script y seguir el patron estandar de Javascript de estilo de sintaxis notación punto para propiedades, por ejemplo:

```
var name = userhome.properties.name
```

El valor de la variable name se obtiene accediendo al valor name de la propiedad properties.

Si se requiere los modelos de nodos hijo, modelos de asociaciones y modelos y aspectos se construyen dinámicamente. Por ejemplo, se pueden escribir sentencias que recorran colecciones múltiples de nodos como:

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

ScriptNode API

Los objetos companyhome, userhome, document, space y person representan objetos Nodos de Alfresco y proporcionan acceso a al conceptos comunes de Alfresco como son propiedades, aspectos y asociaciones. La siguiente API de Nodos está disponible para usar en los scripts:

- **properties:** Un array asociativo (mapeo) de propiedades para el nodo.

Por ejemplo *userhome.properties.name.* Para dar nombre a las propiedades se usa el Qname del modelo, el cual puede ser escrito en forma de namespace completamente cualificado, la forma corta (*como cm:content*) o solo el nombre de la propiedad si asumimos el namespace del Modelo de Contenido por defecto. Así todos las siguiente sentencias son equivalentes:

```
var name1 = userhome.properties.name
var name2 = userhome.properties["name"];
var name3 = userhome.properties["cm:name"];
var name4 = userhome.properties["{http://www.alfresco.org/model/content/1.0}name"];
```

Una característica muy útil es que las propiedades de tipo d:noderef son automáticamente convertidas en un otro objeto de tipo ScriptNode. Esto significa que los programadores de scripts podrán dinamicamente recorrer la jerarquía de objetos para un nodo. Por ejemplo, si un nodo de documento tiene una propiedad NodeRef llamada "locale" se podrá ejecutar la siguiente sentencia para obtener el nombre del nodo que la propiedad referencia.

```
var locale = document.properties.locale.properties.name;
```

Si lo que se desea es obtener es el valor NodeRef para una propiedad d:noderef el siguiente código lo hará:

```
var localeNoderef = document.properties.locale.nodeRef;
```

Ademas, cualquier propiedad del tipo d:content es convertida en un objeto ScriptContent especial que por si mismos ofrecen un API para acceder o modificar el contenido, el tipo mime, el tamaño y la url de la propiedad. Por ejemplo:

```
var mimetype = document.properties.content.mimetype;
var content = document.properties.content.content;
```

Como la mayoría de los documentos se derivan del Modelo de Contenido por defecto, tipo cm:content, la API esta provista de "atajos" para el contenido, el tipo mime, el tamaño y la url para la propiedad cm:content, por ejemplo:

```
document.content = "some new text content";
  var mimetype = document.mimetype;
  var url = document.url;
```

Las propiedades pueden ser modificadas e incluso pueden ser añadidas nuevas:

- **children:** Un array JavaScript de solo lectura de los nodos hijos (mynode.chlidren[0]). Desde Alfresco 2.1, las estructuras del estandar JavaScript 1,6 for each y for each in se pueden usar para iterar por los resultados de todos los métodos que devuelven listas de datos.
- **assoc:** Un array asociativo (mapeo) de solo lectura de las asociaciones entre pares del nodo (mynode.assoc["cm:discussions"][0]). Cada entrada en el array contiene a su vez un array con los objetos ScriptNode en el otro extremo de la asociación.
- **chlidAssoc**: Un array asociativo (mapeo) de solo lectura de las asociaciones padré/hijo del nodo (mynode.childAssoc["cm:discussions"][0]). Cada entrada en el array contiene a su vez un array con los objetos ScriptNode en el otro extremo de la asociación.
- **aspects:** Un array de solo lectura de los aspectos (como cadenas de Qname completamente cualificados) aplicados al nodo. (Se devuelve como un HashSet de Java).
- **isContainer:** Devuelve Verdadero (True) si el nodo es una carpeta, falso de cualquier otra forma.
- **isDocument**: Devuelve Verdadero (True) si el nodo es un contenido, falso de cualquier otra forma.
- **content**: El contenido del nodo como cadena de texto en la propiedad por defecto cm:content. Este valor puede ser modificado para actualizar el contenido del nodo.
- **url:** La url de solo lectura al stream de contenido para este nodo.
- **downloadUrl:** La url de solo lectura al stream de contenido para este nodo como un objeto adjunto HTTP1.1.
- **mimetype:** El tipo mime con el que está codificado el contenido de la propiedad por defecto cm:content adjunta a este nodo.
- **size:** El tamaño en bytes,de solo lectura, del contenido de la propiedad por defecto cm:content adjunta a este nodo.
- **displayPath:** La ruta para visionado, solo lectura. Se debe tener cuidado cuando se accede al displayPath si el usuario actual no tiene permisos para verlo todo. Un parche ha sido aplicado en la versión 2.1.1E de tal manera que el display puede ser siempre obtenido de manera segura por cualquier usuario.
- **qnamePath:** La ruta (el qname) de solo lectura hacia el nodo. (disponible a partir de la versión 2.1)
- **icon16:** La imagen pequeña para este nodo, solo lectura.
- **icon32:** La imagen pequeña para este nodo, solo lectura.
- **isLocked**: Devuelve Verdadero (True) si el nodo esta bloqueado. Devolverá falso en cualquier otra circunstancia.
- **id:** GUID para el nodo.
- **nodeRef:** NodeRef para el nodo como una cadena de texto.
- **name:** Atajo de acceso a la propiedad cm:name.
- **type:** Qname completamente cualificado del tipo del nodo.
- **parent:** El nodo padre puede ser nulo si este es el nodo raíz. e debe tener cuidado cuando se accede al displayPath si el usuario actual no tiene permisos para verlo. Un parche ha sido aplicado en la versión 2.1.1E de tal manera que el nodo padre siempre puede ser obtenido para realizar una comprobación hasPermission() antes de acceder a él en el script.
- **isCategory:** Devuelve verdadero (true) si el nodo actual es un nodo categoría. Devuelve falso en cualquier otra circunstancia.

Las siguientes funciones de la API de nodos se usan para ayudar a localizar a los nodos hijos usando Xpath o una ruta basada en nombres:

- **Node childByNamePath (string path):** Realiza un xpath basado en la propiedad name de los nodos y devuelve el nodo encontrado en la ruta especificada. La ruta es relativa al nodo actual. Si el nodo referido se encuentra se devuelve, sino se devuelve un nulo. Ejemplo:

```
var testingFolder = userhome.childByNamePath("QA/Performance/Testing");
```

- **Array childrenByXpath (string xpath):** Realiza un consulta de búsqueda basada en xpath y devuelve un array con los nodos encontrados. La búsqueda XPath es relativa al nodo actual. Un array vacío se devolverá si no se hay resultados. Por ejemplo:

```
var nodes = userhome.childrenByXPath("*[@cm:name='Finance Documents']/*");
```

- **Array activeWorkflows*:** Devuelve un Array de todos los workflows activos en los que este incluido el nodo. Si el nodo no esta incluido en el Workflow,se devolvera null. Los elementos de la array devuelta son JscriptWorkflowInstance.
- **boolean isScriptContent(object obj)*:** Devuelve Verdadero si la propiedad del nodo es del tipo ScriptContentData.

*Disponible a partir de la versión 3.0

ScriptNode API avanzada

- **primaryParentAssoc:** Devuelve la instancia ChildAssociationRef de la asociación padre/hijo primaria para el nodo. Este valor esta disponible pero solo se requiere en casos especiales.

API de Modificación y Creación

La mayoría de la API de ScriptNode devuelve valores de solo lectura, sin embargo el verdadero potencial de la API de scripts se puede encontrar en que soporta objetos que se pueden modificar y que proporciona acceso a los servicios del repositorio de Alfresco a través de esa API. El objeto ScriptNode permite modificación de propiedades, inserción de nuevas propiedades y aspectos, creación de nuevos ficheros, carpetas y tipos de nodos personalizados y , por supuesto, actualizar y establecer la secuencia de contenido de texto para un nodo. Ademas es posible realizar potentes funcionalidades como borrar nodos, transformar contenido, ejecutar templates y modificar las asociaciones de un nodo.

- **properties:** El Array de propiedades puede ser modificado para actualizar o añadir nuevas propiedades. Por ejemplo:

```
// cambiar el nombre de este documento
   document.properties.name = "Backup of " + document.properties.name;
// añadir una nueva cadena de propiedades
   document.properties["cm:locale"] = mylocalenode;
// guardar las modificaciones en el nodo.
   document.save();
```

Es importante hacer notar que la llamada al API node.call() es necesaria para hacer persistentes las modificaciones en las propiedades. Todas las otras modificaciones realzadas usando la API (como cambiar contenido o añadir aspectos) tienen efecto inmediato.
También es necesario recordar que los objetos JavaScript son distintos los objetos java nativos del repositorio. Los valores de las propiedades en el repositorio deben ser del tipo de objeto correcto según esta definido en el Diccionario de Datos y expuestos por el modelo de contenidos. Así una propiedad cadena de caracteres espera un Strin de Java y una propiedad multi-valor espera un lista. La API Javascript de Alfresco realizará la conversión de la mayoría de tipos de objetos entre Javascript y Java y viceversa por el usuario, por ejemplo Arrays (para las propiedades multivalor), números, fechas, booleanos, y cadenas de caracteres. Los códigos de conversión han sido mejorados en la versión de Alfresco 2.1 y es capaz de manejar todos las conversiones de tipos comunes y listas recursivas de esos tipos.

- **name:** Una propiedad de ayuda para acceder a name. Es un atajo de properties.name.
- **content:** El contenido de texto de un nodo puede ser modificado estableciendo esta propiedad (mynode.content = mynode.content + "añadiendo un poco de texto"). Esto es muy potente ya que permite cambiar el contenido del nodo desde el script. De cualquier nodo es recomendado que los nodos con contenido binario no sean manipulados de esta forma.
- **ScriptNode createFile(string name):** Crea un nuevo nodo de fichero (cm:content) con el nombre especificado e hijo del nodo actual. El nuevo nodo creado es devuelto como resultado de la función, o una execpción lanzada si esta creación falla. Alfresco pone el tipo mime del fichero de el contenido (con la función createnode no hay tipo mime).

```
var mifichero = userhome.createFile("nuevofichero.txt");
```

- **ScriptNode createFolder(string name):** Crea un nuevo nodo carpeta (cm:folder) con el nombre especificado como hijo del nodo actual. El nuevo nodo creado es devuelto como resultado de la función, o una excepción lanzada si esta creación falla.

```
var micarpeta = userhome.createFolder("Nueva Carpeta");
```

- **ScriptNode createNode(string name, string type):** Crea un nuevo nodo con el nombre especificado del tipo especificado. Este tipo se especifica con su Qname ya se en su forma completa o corta. El nuevo nodo creado es devuelto como resultado de la función, o una excepción lanzada si esta creación falla.

```
var nodo  = userhome.createNode("micarpeta", "cm:folder");
```

- **ScriptNode createNode(string name, string type, string assocType):** Crea un nuevo nodo del tipo especificado como hijo del nodo actual según el tipo de asociación hijo especificada. Este tipo de asociación se especifica con su

Qname ya se en su forma completa o corta.

```
var nodo  = miforo.createNode("Mi Discusion", "fm:forum", fm:discussion");
```

- **ScriptNode createNode(string name, string type, Array properties):** Crea un nuevo nodo del tipo especificado como hijo del nodo actual y con las propiedades especificadas en el parámetro properties. Esta propiedad es un array asociativo con las propiedades que deben ser añadidas al nodo después de la creación (es util cuando se tiene un tipo con propiedades obligatorias que debemos rellenar).
- **ScriptNode createNode(string name, string type, Array properties, string assocType):** Crea un nuevo nodo como un hijo del nodo actual con la asociación hijo que se indica y con las propiedades que se especifican.
- **addNode(ScriptNode node):** Añade un nodo existente como hijo del actual.
- **removeNode(ScriptNode node):** Elimina todas las asociaciones padre/hijo existentes entre dos nodos. El nodo hijo será eliminado en cascada si una de las asociaciones era su asociación hijo primaria,esto es, la cual con la que el nodo fue creado.
- **createAssociation(ScriptNode target, string assocType):** Crea una nueva asociación entre pares del tipo definido por el QName dado con el nodo especificado.
- **removeAssociation(ScriptNode target, string assocType):** Borra una asociación entre pares del tipo definido por el QName dado con el nodo especificado.
- **boolean remove():** Borra el nodo. Devuelve verdadero (true) si tiene éxito y falso (false) sino

```
minodo.remove();
```

- **ScriptNode copy(ScriptNode destination):** Se copia el nodo al destino especificado. Se devolverá la instancia recien copiada del ScriptNode se devolverá si se tiene exito sino se devolverá un nulo. Los hijos del nodo fuente no se copiarán.
- **ScriptNode copy(ScriptNode destination, boolean deepCopy):** Se copia el nodo al destino especificado. Se devolverá la instancia recien copiada del ScriptNode se devolverá si se tiene exito sino se devolverá un nulo. Los hijos del nodo fuente no se copiarán. Si el argumento deepCopy es verdadero (true) todos los nodos hijos se copiarán, sino solo el nodo fuente será copiado.

```
var docCopiado = document.copy(userhome);
```

- **boolean move(ScriptNode destination):** Se mueve el nodo al nuevo destino padre. Devuelve verdadero (true) si se tiene exito, faldo en otro caso.
- **boolean addAspect(string aspect):** Se añade un nuevo aspecto al nodo. Devuelve verdadero (true) si se tiene exito, faldo en otro caso. El parametro es el QName del aspecto que queremos añadir, en forma completa o corta.

```
document.addAspect("cm:translatable");
```

- **boolean addAspect(string aspect, Array properties):** Se añade un nuevo aspecto al nodo. Esta funcionalida permirte que se puedan proporcionar el valor de las propiedades obligatorias al aplicar el nuevo aspecto. El argumento propiedades debe ser un array asociativo de propiedades definidas por QName.

```
var props = new Array(1);
props["cm:template"] = myTemplateNode.nodeRef;
document.addAspect("cm:templatable", props);
```

- **boolean removeAspect(string aspect):** Elimina el un aspecto de un nodo.

ScriptContent API

El ScriptContent API proporciona varias propiedades y funciones relativas a propiedades de nodos del tipo *d:content*, por ejemplo, *document.properties.content.*

- **content:** Un valor de lectura y escritura que representa el contenido como una cadena.
- **write(ScriptContent content):** Copia el contenido desde un ScripContent especifico (disponible desde la versión 2.1).
- **mimetype:** Un valor de cadena de caracteres de lectura y escritura que representa el tipo mime del contenido.
- **encoding:** Un valor de cadena de caracteres de lectura y escritura que representa la codificación del contenido.
- **size:** Un valor de solo lectura que representa el tamaño del contenido.
- **url:** Una cadena de caracteres de solo lectura que representa la url de descarga del contenido.
- **downloadUrl:** Una cadena de caracteres de solo lectura que representa la url de descarga (como adjunto) del contenido.

API de Permisos y Seguridad

Las API de ScriptNode implementa varias funciones y propiedades relacionadas con los permisos en el repositorio. Normalmente siempre se deben comprobar si los permisos sobre un nodo son los apropiados antes de intentar modificarlo.

- **boolean hasPermission(string permission):** Devuelve verdadero si el usuario tiene los permisos especificados

sobre el nodo. Los permisos por defecto se encuentran en org.alfresco.service.cmr.security.PermissionService. Los permisos mas comprobados suelen ser "Read", "Write", "Delete" and "CreateChildren".

- **String[ ] getPermissions():** Devuelve un array de cadenas de caracteres con los permisos asociados a un nodo. Las cadenas devueltas tienen el formato [ALLOWED|DENIED];[USERNAME|GROUPNAME];PERMISSION, por ejemplo ALLOWED;kevinr;Consumer, así pueden ser facilmente separadas en permisos unicos usando como token de separación el caracter ';'.
- **boolean inheritsPermissions():** Devuelve verdadero si el nodo actual hereda sus permisos desde su spacio padre y falso para indicar que los permisos han sido asignados especificamente en para el nodo.
- **void setInheritsPermissions(boolean inherit):** Cuando se pasa por parametro true indica que el nodo debe heredar los permisos de su nodo padre. Cuando se pasa false se rompe la herencia en cadena de permisos.
- **void setPermission(string permission):** Aplica un permiso para TODOS los usuarios al nodo.
- **void setPermission(string permission, string authority):** Aplica un permisio para la authority especificada (e.g. nombre de usuario o grupo) al nodo.
- **void removePermission(string permission):** Quita un permiso del nodo para TODOS los usuarios.
- **void removePermission(string permission, string authority):** Quita un permiso del nodo para la authority especificada (e.g. nombre de usuario or grupo).
- **void setOwner(String userId):** Asigna un propietario al nodo.
- **void takeOwnership():** Toma posesion de un nodo.
- **String getOwner():** Obtiene el propietario del nodo (como un uid).
- **owner:** El propietario del nodo (como un uid).

API de Protección/Desprotección

La API ScriptNode de protección/desprotección implementa métodos para realizar desproteciones, proteciones y cancelar desprotecciones de copias de trabajo. Hay que tener en cuenta que es posible que se desee añadir el aspecto cm:versionable a un nodo antes de desprotegerlo si se desea que se grabe el histórico de versiones cuando se usen estos métodos.

- **ScriptNode checkout():** Realiza una desprotección del nodo devolviendo el nodo de copia de trabajo resultante.
- **ScriptNode checkout(ScriptNode destination):** Realiza la desproteción del nodo especificado por el parámetro destination devolviendo el nodo de copia de trabajo resultante.
- **ScriptNode checkin():** Realiza una operación de proteción sobre un nodo de copia de trabajo. El estado actual de la copia de trabajo es copiada en el nodo original lo que incluirá cualquier contenido actualizado en el nodo de trabajo. Devuelve el nodo original que había sido anteriormente desprotegido. Este método solo puede ser llamado en un nodo de copia de trabajo.
- **ScriptNode checkin(String history):** Realiza una operación de protección sobre un nodo de copia de trabajo aplicando la descripción de la versión definida en el parámetro *history*. Este método solo puede ser llamado en un nodo de copia de trabajo.
- **ScriptNode checkin(String history, boolean majorVersion):** Realiza una operación de protección sobre un nodo de copia de trabajo aplicando la descripción de la versión definida en el parámetro *history* y un incremento o decremento de versión según se requiera. Este método solo puede ser llamado en un nodo de copia de trabajo.
- **ScriptNode cancelCheckout():** Cancela la desproteción de un nodo de trabajo. La copia de trabajo se borrara y cualquier cambio que se hiciese se perdará. Este método solo puede ser llamado en un nodo de copia de trabajo. Cualquier referencia a esta copia de trabajo nodo debe ser desechada. Devuelve el nodo original que había sido anteriormente desprotegido.

API de Versiones*

Los API ScriptNode de versiones implementa funciones y metodos para realizar el mantenimiento y recuperar las versiones de un documento.

- **isVersioned*:** Propiedad de Solo Lectura para determinar si un documento esta versionado.
- **ScriptVersion[ ] versionHistory***: Propiedad de Solo Lectura para listar todas las versiones del documento (de forma descendente por fecha de creación).
- **ScriptVersion getVersion(label)*:** Recupera la versión de un documento identificado por 'label', devuelve null si la etiqueta no existe.
- **ScriptVersion createVersion(history, major)*:** Crea una versión del documento actual.

La API ScriptVersion representa una version especifica de un documento.

- **createdDate*:** Propiedad de Solo Lectura que representa la fecha en la que se ha creado la versión del documento.
- **creator*:** Propiedad de Solo Lectura que representa el nombre del usuario de la persona que ha creado la versión.
- **label*:** Propiedad de Solo Lectura que representa la etiqueta label.
- **type*:** Propiedad de Solo Lectura que representa el tipo de versión ("MAYOR", "MENOR").

- **description*:** Propiedad de Solo Lectura que representa la descripción(historico de comentarios) de la versión.
- **nodeRef*:** Propiedad de Solo Lectura que representa el nodo de referencia del documento que esta versionado.
- **ScriptNode node*:** Propiedad de Solo Lectura que representa el nodo de la version.

*Disponible a partir de la versión 3.0.

API de Transformación

Las siguientes funciones ScriptNode hacen uso de los servicios de transformación de documentos disponibles en Alfresco. El servicio OpenOffice es necesario para algunas de las transformaciones.

- **ScriptNode transformDocument(string mimetype):** Transforma un documento a un nuevo tipo mime. Se hace una copia del documento y se cambia su extensión para encajar con su nuevo tipo, entonces se aplica la transformación. Si la transformación tiene éxito se devuelve el nodo del documento transformado en caso contrario se devuelve null.
- **ScriptNode transformDocument(string mimetype, ScriptNode destination):** Transforma un documento a un nuevo tipo mime . Se hace una copia del documento en la carpeta destino especificada y se cambia su extensión para encajar con su nuevo tipo, entonces se aplica la transformación. Si la transformación tiene éxito se devuelve el nodo del documento transformado en caso contrario se devuelve null.

Las siguientes funciones hacen uso de los servicios de transformación de imágenes disponibles en Alfresco. Se necesita que el componente ImageMagick este correctamente instalado y funcionando. Hay que tener en cuenta que se puede encontrar información detallada de las opciones de ImageMagick mencionadas a continuación en la web de ImageMagick.

- **ScriptNode transformImage(string mimetype):** Transforma una imagen a un nuevo formato. Se hace una copia de la imagen y se cambia su extensión para encajar con su nuevo tipo, entonces se aplica la transformación. Si la transformación tiene éxito se devuelve el nodo de la imagen transformada en caso contrario se devuelve null.
- **ScriptNode transformImage(string mimetype, string options):** Transforma una imagen a un nuevo formato, aplicando las opciones ImageMagick suministradas. Se hace una copia de la imagen y se cambia su extensión para encajar con su nuevo tipo, entonces se aplica la transformación. Si la transformación tiene éxito se devuelve el nodo de la imagen transformada en caso contrario se devuelve null.
- **ScriptNode transformImage(string mimetype, ScriptNode destination):** Transforma una imagen a un nuevo formato. Se hace una copia de la imagen en la carpeta destino especificada y se cambia su extensión para encajar con su nuevo tipo, entonces se aplica la transformación. Si la transformación tiene éxito se devuelve el nodo de la imagen transformada en caso contrario se devuelve null.
- **ScriptNode transformImage(string mimetype, string options, ScriptNode destination):** Transforma una imagen a un nuevo formato, aplicando las opciones ImageMagick suministradas. Se hace una copia de la imagen en la carpeta destino especificada y se cambia su extensión para encajar con su nuevo tipo, entonces se aplica la transformación. Si la transformación tiene éxito se devuelve el nodo de la imagen transformada en caso contrario se devuelve null.

Las siguientes funciones hacen uso de los servicios de procesamiento de plantillas de Freemarker disponible en Alfresco.El resultado de la ejecución de la plantilla es devuleto por cada función como una cadena de caracteres. Hay que tener en cuenta que el nodo se usa como contexto en la plantilla. Si el nodo es un documento será inicializado como contexto del objeto '*document*' para la plantilla y su espacio padre inicializado como contexto del objeto 'space' para la plantilla. Si es una carpeta sera solo inicializado como contexto del objeto 'space' para la plantilla. Una lista de argumentos puede ser también pasada al la plantilla y estará disponible como el objeto '*args*'.

- **string processTemplate(ScriptNode template):** Ejecuta una fichero de plantilla FreeMarker contra un nodo. El nodo será usado como contexto para el objeto 'documen' o 'space' en el modelo de ejecución de plantillas por defecto. El resultado de la ejecución de la plantilla es devuelto como una cadena de texto.
- **string processTemplate(ScriptNode template, Array args):** Ejecuta una fichero de plantilla FreeMarker contra un nodo, se pasan los argumentos suministrados como un array de pares nombre/valor al template. El nodo será usado como contexto para el objeto '*documen*' o 'space' en el modelo de ejecución de plantillas por defecto. El resultado de la ejecución de la plantilla es devuelto como una cadena de texto.
- **string processTemplate(string template):** Ejecuta una plantilla FreeMarker contra un nodo. La plantillas se suministra directamente como una cadena. El nodo será usado como contexto para el objeto '*documen*' o '*space*' en el modelo de ejecución de plantillas por defecto. El resultado de la ejecución de la plantilla es devuelto como una cadena de texto.
- **string processTemplate(string template, Array args):** Ejecuta una plantilla FreeMarker contra un nodo, se pasan los argumentos suministrados como un array de pares nombre/valor al template. La plantillas se suministra directamente como una cadena. El nodo será usado como contexto para el objeto '*documen*' o '*space*' en el modelo de ejecución de plantillas por defecto. El resultado de la ejecución de la plantilla es devuelto como una cadena de texto.

API Thumbnailing*

En Alfresco 3.0 un nuevo Repósitorio y el servicio REST para thumbnailing automático de contenido están disponibles. El objeto ScriptNode ha sido ampliado para permitir al acceso de JavaScript a operaciones CRUD para thumbnailing.

*Disponible a partir de la versión 3.0

API Tagging*

En Alfresco 3.0 un nuevo Repósitorio y el servicio de REST para Web 2.0 contenido está disponible. El objeto ScriptNode ha sido ampliado para permitir al acceso de JavaScript a operaciones CRUD para la marcación.

*Disponible a partir de la versión 3.0

Varias Funciones y Propiedades de la API ScriptNode API

- **boolean hasAspect(string type):** Devuelve true si un aspecto está aplicado al nodo. Por ejemplo: v*ar esPlantilla = document.hasAspect("cm:templatable")*;
- **boolean specializeType(string type):** Especializa el tipo de un nodo. Devuelve *true* si tiene éxito y *false* si no. El nombre del tipo suministrado debe ser un sub-tipo del tipo actual según lo definido en el Diccionario de Datos.

API de Búsqueda

La API de Búsqueda proporciona acceso directo a resultados de búsquedas Lucene a nivel de repositorio y Búsquedas Almacenadas . Es accesible a través del objeto de ámbito raiz '*search*'. Hay que tener en cuenta que las búsquedas locales pueden ser realizadas usando las APIs ScriptNode childByNamePath and childByXPath como se detallo antes.

El objeto de búsqueda es parte del ambito raiz disponible para los escritores de scripts. La API proporciona las siguientes funciones:

- **Array luceneSearch(string query):** Devuelve un array de objetos ScriptNode que son encontrados mediante una búsqueda completa del repositorio de Alfresco, por ejemplo: *var nodos = search.luceneSearch("TEXT:alfresco")*;
- **Array xpathSearch(string xpath):** Devuelve un array de objetos ScriptNode que son encontrados mediante una búsqueda xpath del repositorio de Alfresco
- **Array savedSearch(ScriptNode node):** Devuelve un array de objetos ScriptNode que son encontrados ejecutando la Búsqueda Almacenada referenciada por el objeto *node* pasado como parámetro.
- **Array savedSearch(NodeRef noderef):** Devuelve un array de objetos ScriptNode que son encontrados ejecutando la Búsqueda Almacenada referenciada por la cadena *noderef* suministrada.
- **ScriptNode findNode(NodeRef noderef):** Devuelve un único ScriptNode especificado por el NodeRef para ese nodo. Devuelve null si la búsqueda falla.
- **ScriptNode findNode(string noderef):** Devuelve un único ScriptNode especificado por la forma de cadena del NodeRef para ese nodo. Devuelve null si la búsqueda falla.
- **Array luceneSearch(string store, string query)*:** Devuelve un array de objetos ScripNode que seran el resultado de la búsqueda del texto completo en el repositorio de Alfresco. Por ejem: var nodes = search.luceneSearch("workspace://sitestore", "TEXT:site");
- **Array luceneSearch(string query, string sortColumn, boolean asc)*:** Devuelve un array de ScriptNode satisfaciendo la ordenación en el criterio de búsqueda por una columna especifica y la ordenación (true => orden ascendente, false => orden descendente). Por ejem: var nodes = search.luceneSearch("TEXT:alfresco", "@cm:modified", false);
- **Array luceneSearch(string store, string query, string sortColumn, boolean asc)*:** Devuelve un array de ScriptNode satisfaciendo el criterio de búsqueda y ordenando el store obtenido.
- **Array xpathSearch(string store, string xpath)*:** Devuelve un array de objetos ScriptNode que son encontrados en el repositorio de Alfresco mediante una búsqueda XPath.
- **ScriptNode findNode(string referenceType, string[ ] reference)*:** Ayudante para convertir una solicitud URL de un Web Script a NodeRef. ReferenceType puede ser uno de los nodos, la ruta, avmpath o QName.

*Disponible a partir de la versión 3.0

People API

La People API proporciona acceso a los usuarios y grupos de Alfresco. La API proporciona las siguientes funciones:

- **ScriptNode getPerson(string username):** Devuelve un nodo *(cm:person)* asociado con el nombre de usuario especificado. Devuelve *null* si el usuario no existe.
- **ScriptNode getGroup(string groupname):** Devuelve un nodo *(usr:authorityContainer)* asociado con el nombre de grupo especificado. Devuelve *null* si no existe.
- **deleteGroup(ScriptNode group):** Elimina un Grupo del sistema.
- **ScriptNode createGroup(String groupName):** Crea un nuevo grupo de primer nivel. El parámetro *groupname* es el nombre único del grupo a crear.
- **ScriptNode createGroup(ScriptNode parentGroup, string groupName)**: Crea un nuevo Grupo como hijo del nodo grupo padre especificado. Este nodo puede ser null para crear un grupo de primer nivel.
- **addAuthority(ScriptNode parentGroup, ScriptNode authority):** Añade una autoridad (Usuario o Grupo) al Grupo padre especificado.
- **removeAuthority(ScriptNode parentGroup, ScriptNode authority):** Elimina una autoridad de un Grupo.
- **Array getMembers(ScriptNode group):** Devuelve un Array de nodos people que pertenencen al grupo especificado (incluyendo sub-grupos)

- **Array getMembers(ScriptNode group, boolean recurse):** Devuelve un Array de nodos people que pertenecen al grupo especificado. Los nodos de los sub-grupos solo se devuelven si se especifica como true el parámetro recurse.
- **Array getContainerGroups(ScriptNode person):** Obtiene el grupo que contiene la autoridad especificada.
- **ScriptNode createPerson(String username)*:** Crea una persona (cm:person) con el nombre de usuario dado. Devuelve el nodo de la persona creada o null si el usuario ya existe.
- **boolean isAdmin(ScriptNode person)*:** Devuelve true si el usuario usuario especificado tiene Derechos de Administrador.

*Disponible a partir de la versión 3.0

Actions API

Un objeto de nivel raíz '*actions*' se proporciona para permitir la invocación de Acciones de Alfresco registradas en el Repositorio.

- **registered:** Devuelve un array de cadenas de caracteres que representan los nombres de todas las Acciones registradas.
- **ScriptAction create(String name):** Devuelve la acción para un nombre dado. Si el nombre de acción no esta registrado se devuelve un null.

ScriptAction API

Un ScriptAction representa una Acción Alfresco registrada con el Repositorio.

- **name:** Devuelve el nombre de la acción.
- **parameters:** Un array asociativo de los parámetros para la accion. Por ejemplo: *mail.parameters.subject*. Cada parámetro es identificado por una cadena de caracteres. Se podrá añadir nuevos parámetros o modificar los existentes.
- **execute(ScriptNode node):** Ejecuta la acción contra el nodo especificado. Esta acción (y sus parametros) puede se reutilizada contra muchos nodos invocando repetidamente la acción *execute*. Entre invocaciones, los parámetros de la acción pueden cambiar. Un ejemplo de ejecución de la acción "*mail*" sería:

```
// crear la accion mail
  var mail = actions.create("mail");
  mail.parameters.to = "davidc@alfresco.com";
  mail.parameters.subject = "Hola desde JavaScript";
  mail.parameters.from = "davidc@alfresco.com";
  mail.parameters.template = root.childByNamePath("Company Home/Data Dictionary/Email Templates/notify_user_email.
  mail.parameters.text = "insertamos algo de texto, por si no se encuentra la plantilla";
  // ejecutar la accion contra un documento
  mail.execute(doc);
```

API de Logging

Un objeto '*logger*' de nivel raíz proporciona funcionalidad para ayudar en la depuración de los scripts.

- **boolean isLoggingEnabled():** Devuelve true si el logging esta habilitado. Para habilitar el logging la categoría Log4J de *log4j.logger.org.alfresco.repo.jscript* debe ser fijada como DEBUG. Esto se debe hacer en el fichero *log4j.properties* (TomCat) o el fichero log4j.xml (JBoss) en el servidor Alfresco.
- **void log(string):** Inserta en el fichero de log la cadena pasada por parámetro.

API de Sesion

Un objeto '*session*' de nivel raíz proporciona acceso al ticket de sesión del usuario actualmente logado como una cadena de caracteres.

- **ticket:** Obtiene el actual ticket de autenticación.
- **string getTicket():** Obtiene el actual ticket de autenticación.

API de Clasificacion

La API se divide en dos partes: la manipulación de de las clasificaciones y la manipulación de las categorías que contienen. Se proporciona un objeto de nivel raiz '*classification*' para devolver los nodos de categorías. Los objetos CategoryNode devueltos por las funciones son extensiones del modelo estandar ScriptNode de JavaScript incluyendo la manipulación de las categorías.

- **CategoryNode[ ] getAllCategoryNodes(String aspect):** Obtiene un array de todos los nodos categoria de una clasificación dada.
- **string[ ] getAllClassificationAspects():** Obtiene un array de todos los QNames (en formato prefijo:nombreLocal).
- **CategoryNode createRootCategory(string aspect, string name):** Crea un nuevo nodo raiz en una clasificación dada.
- **CategoryNode[ ] getRootCategories(string aspect):** Obtiene un array todos los nodos raíz para una clasificación

dada.

La API del objeto CategoryNode

- **boolean isCategory:** Soportado por todos los tipos de nodo. Es *true* si es un nodo de categoría y *false* si es cualquier otro tipo de nodo.
- **CategoryNode[ ] categoryMembers:** Obtiene un array de todos los miembros de esta categoría a cualquier nivel de profundidad.
- **CategoryNode[ ] subCategories:** Obtiene un array de todas las subcategorías de esta categoría a cualquier nivel de profundidad.
- **CategoryNode[ ] membersAndSubCategories:** Obtiene un array de todos los miembros y las subcategorías de esta categoría a cualquier nivel de profundidad.
- **CategoryNode[ ] immediateCategoryMembers:** Obtiene un array con todos los miembros directos de esta categoría (solo miembros directos no los que sean miembros a través de subcategorías).
- **CategoryNode[ ] immediateSubCategories:** Obtiene un array con todos las subcategorías directas de esta categoría (solo subcategorías directas no las que lo sean a través de subcategorías).
- **CategoryNode immediateMembersAndSubCategories**: Obtiene un array con todos los miembros y las subcategorías directas de esta categoría (solo miembros y subcategorías directos no las que lo sean a través de subcategorías).
- **CategoryNode createSubCategory(String name):** Crear una nueva subcategoría a partir del actual nodo de categoría.
- **removeCategory():** Borrar el actual nodo de categoría.

API de AVM

La API de Maquina de Versionado de Alfresco (Alfresco Versioning Machine - AVM) proporciona acceso a los almacenes de Gestion de Contenido Web (WCM) y sus ficheros y carpetas asociados. Un proyecto WCM esta dividido en "*stores*" tales como el directorio fuente (*Staging Store*) y varios directorios de desarrollo (*User Sandbox*) ademas de los nodos hijos de estos.

- **avm.stores:** Devuelve un Array con todos los objetos store en la AVM.
- **avm.lookupStore(storeid):** Función que devuelve el objeto store qdado un id específico.
- **avm.lookupStoreRoot(storeid):** Función que devuelve el nodo raiz dado un nodo específico.
- **avm.lookupNode(path):** Función que devuelve un único nodo AVM dada la ruta completa hacia el nodo incluyendo el store.
- **avm.webappsFolderPath:** Devuelve la ruta hacia la carpeta de la aplicación web AVM para el *store*.

API de los objetos AVM Store

Los objetos Store devueltos por los métodos anteriores tienen la siguiente API adicional:

- **store.id:** ID interna del *store*.
- **store.name:** Nombre del *store*.
- **store.creator:** Usuario que creo el *store*.
- **store.createdDate:** Fecha de creación del *store*.
- **store.lookupRoot:** Devuelve el nodo raíz del *store*.
- **store.lookupNode(path):** Función que devuelve un nodo AVM con el *store* dada la ruta relativa de este (relativa al directorio raíz de la aplicación web AVM).
- **store.luceneSearch(query):** Función que ejecuta una búsqueda Lucene en el store y devuelve como resultado un Array de nodos AVM.

API de los objetos Node

Los objetos *AVM* node devueltos por las funciones anteriores extienden el objeto ScriptNode tal y como se detalla a continuación. Tiene ademas la siguiente API adicional.

- **node.version:** Versión del nodo.
- **node.path:** Ruta AVM Totalmente Cualificada al nodo.
- **node.parentPath:** Ruta AVM Totalmente Cualificada al padre del nodo.
- **node.isDirectory:** Devuelve *true* si este nodo AVM es un directorio.
- **node.isFile:** Devuelve true si este nodo AVM es un fichero.
- **node.isLocked:** Devuelve *true* si el nodo esta actualmente bloqueado.
- **node.isLockOwner:** Devuleve *true* si el nodo esta bloqueado y el usuario actual es el que lo ha bloqueado(*lock owner*).
- **node.hasLockAccess:** Devuelve *true* si este usuario puede realizar operaciones sobre el nodo cuando esta bloqueado. Será *true* si el el ítem esta desbloqueado o esta bloqueado y el usuario actual es el *lock owner* o esta bloqueado y el

usuario actual tiene un rol de *Content Manager* en el proyecto web asociado.

- **node.rename(nombre):** Renombra el nodo (esta es una operación especial en la AVM, no puede ser realizada simplemente cambiando el valor de la propiedad *cm:name*).

Copia cruzada de Repositorios

Se proporciona el objeto de nivel raíz 'crossRepoCopyroot' para habilitar la copia de nodos entre los espacios de gestión documental (ADM) y los espacio WCM (AVM).

- **ScriptNode copy(ScriptNode fuente, ScriptNode destino, String nombre):** Copia el nodo fuente al directorio destino especificado. La fuente y el destino pueden estar en cualquier store de los repositorios tal como un *AVM Store* o el *SpaceStore*. Cuando se copia entre *stores* de diferentes tipos el aspecto y las propiedades del nodo permanecerán intactos pero el tipo de nodo podría ser bajado de categoría y todas las asociaciones se perderán.

Funciones de Utilidades

Se proporciona un objeto de nivel raíz 'utils' como una librería de funciones de ayuda que no están en el JavaScript genérico.

- **string pad(s, length):** Rellena una cadena con ceros por la izquierda hasta una longitud dada y devuelve la nueva cadena.
- **ScriptNode getNodeFromString(noderef)*:** Devuelve un *ScriptNode*, Representa el *NodeRef* suministrado mediante una cadena. Nota,no se comprueba que el nodo exista en el repositorio.
- **boolean toBoolean(string)*:** devuelve un objeto booleano del valor de la cadena.

*Disponible a partir de la versión 3.0

Recursos

Área: Arquitectura » Arquitectura de Sistemas de Información » Alfresco » Uso de Alfresco desde Terceras Aplicaciones

Código	Título	Tipo	Carácter
RECU-0048	Scripts de Ejemplo	Ejemplo	Recomendado

Scripts de Ejemplo

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Carácter del recurso:** Recomendado

Código: RECU-0048
Tipo de recurso: Ejemplo

Descripción

Para facilitar la programación de scripts de acceso a Alfresco, se proponen varios ejemplos que resuelven diferentes casos de negocio que se presentan habitualmente en la construcción de aplicaciones con Alfresco.

Ejemplos

Crear una copia de seguridad de un documento

Se crea una copia de seguridad de un documento y se añade a un espacio:

```
// buscamos la carpeta backup - la creamos si no existe
var carpetaBackup = space.childByNamePath("Backup");
if (carpetaBackup == null && space.hasPermission("CreateChildren"))
{
  // crear la carpeta por primera vez
  carpetaBackup = space.createFolder("Backup");
}
if (carpetaBackup != null && carpetaBackup.hasPermission("CreateChildren"))
{
  // copiar el documento en la carpeta de backup
  var copia = document.copy(carpetaBackup);
  if (copia != null)
  {
    // cabiar el nombre para así saber que es una copia de seguridad
    copia.name = "Copia de seguridad de " + copy.name;
    copia.save();
  }
}
```

Crear copia de seguridad de un documento y realizar log de las propiedades del mismo

Se crea un backup de un documento y se realiza un log de las propiedades del mismo en un fichero de texto:

```
// buscamos la carpeta backup - la creamos si no existe
var carpetaBackup = space.childByNamePath("Backup");
if (carpetaBackup == null && space.hasPermission("CreateChildren"))
{
  // crear la carpeta por primera vez
  carpetaBackup = space.createFolder("Backup");
}
if (carpetaBackup != null && carpetaBackup .hasPermission("CreateChildren"))
{
  // copiar el documento en la carpeta de log
  var copia = document.copy(carpetaBackup );
  if (copia != null)
  {
    // cabiar el nombre para así saber que es una copia de seguridad
    var nombreCopia = "Copia de seguridad de " + copia.name;
    copia.name = nombreCopia ;
    copia.save();
  }

  // record the time of the backup to a log file
  var logFile = carpetaBackup .childByNamePath("backuplog.txt");
  if (logFile == null)
```

```
  {
    logFile = carpetaBackup .createFile("backuplog.txt");
  }
  if (logFile != null)
  {
    logFile.content += "Fichero: " + nombreCopia +
```

Añadir linea de Copyright a un fichero

Añade una linea de copyright de contenido a ficheros HTML y de texto plano:

```
if (document.hasPermission("Write"))
{
  if (document.mimetype == "text/plain")
  {
    document.content += "\r\n\r\nCopyright (C) 2008";
  }
  else if (document.mimetype == "text/html")
  {
    document.content += "<br><br><small>Copyright © 2008</small>";
  }
}
```

Añadir Aspectos

Se añaden varios aspectos a un documento:

```
var propiedades = new Array(1);
propiedades["cm:template"] = document.nodeRef;
document.addAspect("cm:templatable", propiedades);
propiedades = new Array(1);
propiedades["cm:lockIsDeep"] = true;
document.addAspect("cm:lockable", propiedades);
propiedades = new Array(1);
propiedades["cm:hits"] = 1;
document.addAspect("cm:countable", propiedades);
```

Buscar documentos usando una búsqueda Lucene

Buscar todos los documentos que contengan el texto 'Alfresco' usando una búsqueda Lucene y grabar los resultados en un fichero de log.

```
// realizar log de los documentos que contienen la palabra 'Alfresco' en un fichero.
var logFile = userhome.childByNamePath("alf docs.txt");
if (logFile == null)
{
  logFile = userhome.createFile("alf docs.txt");
}
if (logFile != null)
{
  // execute a lucene search across the repo for the text 'alfresco'
  // ejecuta una busqueda Lucenea través del repositorio para el texto 'alfresco'
  var documentos = search.luceneSearch("TEXT:alfresco");
  var log = "";
  for (var i=0; i<documentos.length; i++)
  {
    log += "Nombre: " + documentos[i].name + "\tRuta: " + documentos[i].displayPath + "\r\n";
  }
  logFile.content += log;
}
```

Devolver un valor como resultado

Se devuelve un valor como resultado de la ejecución del script. Esto es util para scripts que son procesados mediante URIs usando el Servlet de Ejecució de Scripts ya que los resultados son devueltos como respuestas HTML por el servlet.

```
function resultado()
{
   return "El nombre de mi espacio raiz es: " + userhome.name;
}
result();
```

La siguiente solución también devolverá un valor si se coloca al final de un script:

```
// aqui el script here
// ...
var resultado = "algunos resultados...";
result;
```

Crear Documento, Hacerlo versionable, Modificarlo

En este scripts se van ha realizar varias manipulaciones de un contenido. Los pasos que e van a seguir son: crear un documento, harcerlo versionable, desprotegerlo, modificar el contenido de la copia de trabajo, protegerlo de nuevo y repetir el proceso pero añadiendo un descripción para el histórico y un incremento del numero de versión al protegerlo:

```
// crear un fichero y hacerlo versionable
var documento = userhome.createFile("checkmeout.txt");
documento.addAspect("cm:versionable");
documento.content = "texto original";
// desprotegerlo y cambiar el contenido de la copia de trabajo
var copiadeTrabajo= documento.checkout();
copiadeTrabajo.content = "Actualizando texto 1";
// Protegerlo
documento = copiadeTrabajo.checkin();
// Repetimos el proceso
copiadeTrabajo= documento.checkout();
copiadeTrabajo.content = "Actualizando texto 2";
// Lo protegemos de nuevo, pero nota para el historico de versiones y un incremento en la versión
documento = copiadeTrabajo.checkin("nota para el historico", true);
```

Cambiar el Tipo Mime de un documento

Cambiar el tipo mime de un documento después de establecer el contenido

```
var fichero = userhome.createFile("ficherodetest.html");
fichero.content = "texto <b>HTML</b> aqui";
fichero.mimetype = "text/html";
```

Crear un Documento y Transformarlo

Crear un documento y convertirlo en nuevos formatos usando la API de transformaciones:

```
// crear un documento de texto plano y convertirlo a PDF, el fichero generado en el mismo espacio que el original
var documento1 = userhome.createFile("transforma_me1.txt");
documento1.mimetype = "text/plain";
documento1.content = "Esto es texto plano";
var transformado1 = documento1.transformDocument("application/pdf");
// crear un documento HTML and convertirlo en texto plano, en nuevo fichero será generado en el espacio companyhom
var documento2 = userhome.createFile("transforma_me2.html");
documento2.mimetype = "text/html";
documento2.content = "Esto es un <b>documento</b> <font color=blue><i>HTML</i>!</font>";
var transformado2 = documento2.transformDocument("text/plain", companyhome);
// crear un documento HTML and convertirlo  en un archivo swf flash, el fichero generado se debe crear en el espacio co
var documento3 = userhome.createFile("transforma_me3.html");
documento3 .mimetype = "text/html";
documento3 .content = "Esto es un <b>documento</b> <font color=blue><i>HTML</i>!</font>";
var transformado3 = documento3.transformDocument("application/x-shockwave-flash", companyhome);
```

Convertir un documento de imagen en otros formatos:

```
// convertir a formato GIF y colocarlo en el espacio companyhome
```

```
var imagenGif = document.transformImage("image/gif", companyhome);
// convertir una imagen a formato JPG y redimensionarla a tamaño de thumbnail
var thumbImage = document.transformImage("image/jpeg", "-resize 120");
```

Ejecutar una plantilla Freemarker

Ejecuta una plantilla del repositorio en el nodo Document actual:

```
var plantilla = companyhome.childByNamePath("/Data Dictionary/Presentation Templates/doc_info.ftl");
if (plantilla != null)
{
   var resultado = document.processTemplate(plantilla);
   // escribir los resultados en la consola de log
   logger.log(resultado);
}
```

Se puede también construir una plantilla FreeMarker directamente en el script, también construir una lista de argumentos para la plantilla. El resultado de la plantilla es grabado en un nuevo nodo:

```
var plantilla = "<h3>El nombre del Documento es ${document.name}</h3>" +
                "El argumento ID: ${args['id']}";
var argumentos = new Array()
argumentos ["id"] = "01234-56789";
var resultado = document.processTemplate(plantilla, argumentos);
// grabar el contenido del resultado de la plantilla a un nuevo nodo en el espacio raiz del usuario
var ficheroSalida = userhome.createFile("output.txt");
ficheroSalida .content = resultado;
```

Mostrar el actual conjunto de permisos en el espacio del usuario

```
var permisos = userhome.permissions;
var resultado = "";
for (var i=0; i<permisos.length; i++)
{
   resultado += permisos[i] + "<br>";
}
resultado;
```

Incluir otros dos javascripts, uno desde el classpath y otro desde el repositorio

```
<import resource="classpath:alfresco/extension/misutilidades.js">
<import resource="/Company Home/Data Dictionary/Scripts/mislibrerias.js">
// a partir de aqui el script
// ...
```

Crear diferentes tipos de nodos hijos

Se crearán nodos hijo de varias clases por ejemplo a través de una asociación hija especificada por el nombre y con propiedades por defecto.

```
var nodo1 = userhome.createNode("creado test1.txt", "cm:content");
nodo1.content = "contenido nodo1";
var nodo2 = userhome.createNode(null, "sys:base");
var propiedades = new Array();
propiedades ["cm:name"] = "creado test3.txt";
var nodo3 = userhome.createNode(null, "cm:content", propiedades);
propiedades ["cm:name"] = "nodo nombre4.txt";
propiedades ["cm:title"] = "nodo titulo4";
var nodo4 = userhome.createNode(null, "cm:content", propiedades , "cm:contains");
var resultado = "nodos creados correctamente";
resultado;
```

Itererar por valores usando for in y for each in

```
// ejemplo de iteración por indices de array usando 'for .. in'
```

```
var salida1 = "";
for (i in userhome.children)
{
  salida1 += userhome.children[i].name + "<br>";
}
// ejemplo de iteración por indices de array usando 'for each .. in'
var salida2 = "";
for each (n in userhome.children)
{
  salida2 += n.name + "<br>";
}
var salida = salida1 + "<br><br>" + salida2;
salida;
```

Creación y eliminacion de asociaciones

Ejemplo de creacion y eliminación asociaciones:

```
var nodoOrigen= userhome.createNode("ficherotestassocs.txt", "cm:content");
nodoOrigen.content = "el texto original";
nodoOrigen.addAspect("cm:transformable");
var nodoDestino = userhome.createNode("traduccion1.txt", "cm:content");
nodoDestino.content = "la tranduccion del texto original";
nodoOrigen.createAssociation(nodoDestino, "cm:formats");
var nodoTemp = userhome.createNode("borrar.txt", "cm:content");
nodoTemp.content = "borrar";
nodoOrigen.createAssociation(nodoTemp, "cm:formats");
//obtener los nodos objetivo de la asociación 'cm:formats'
var translations = nodoOrigen.assocs["cm:formats"];
nodoOrigen.removeAssociation(tempNode, "cm:formats");
nodoTemp.remove();
```

Usar la API AVM para Procesar un Store de un Proyecto Web

Ejemplo de uso de la API AVM para procesar un store - el nombre del store se pasa en los argumentos de la URL

```
if (args["store"] == null)
{
  logger.log("ERROR: Argumento 'store' no especificado.");
}
else
{
  main();
}
function main()
{
  var nodoStoreRaiz = avm.lookupStoreRoot(args["store"]);
  if (nodoStoreRaiz != null)
  {
    var ruta = nodoStoreRaiz.path + "/ROOT/admin/index.html";
    var nodo = avm.lookupNode(ruta);
    if (nodo == null)
    {
      return "ERROR: Imposible encontrar ruta: " + ruta;
    }

    var store = avm.lookupStore(args["store"]);
    if (store == null)
    {
      return "ERROR: Imposible encontrar store: " + args["store"];
    }
    var nodoRaiz = store.lookupRoot();
```

```
    if (nodoRaiz == null)
    {
```

Descubrir que acciones del repositorio están disponibles

```
function resultado()
{
   return (actions.registered);
}
resultado();
```

La salida será algo parecido a:

```
[transform-image, mail, copy-to-web-project, extract-metadata, counter, check-in, simple-workflow, script, transform, rem
```

Descubrir todos los formularios XML de un tipo especifico y devolver el contenido XML

```
var store = avm.lookupStore("test");
var resultados = store.luceneSearch("wca\\:parentformname:Comunicado de Prensa");
var comunicadosprensa = new XML();
comunicadosprensa = <comunicadosprensa></comunicadoprensa>
for (var i=0, len=resultados.length; i<len; ++i){
    var str = new String(resultados[i].content);
    str = str.substr(38, str.length - 1 );    // Nota: se asume codificación UTF-8 para todos los documentos XML
    var comunicadoprensa = new XML(str);
    comunicadosprensa.comunicadoprensa = comunicadoprensa ;
}
model.comunicadosprensa = comunicadosprensa.toXMLString();
```

Se podrá devover el XML usando:

```
${comunicadosprensa}
```

API JavaScript

- **Área:** Uso de Alfresco desde Terceras Aplicaciones
- **Carácter del recurso:** Recomendado

Código: RECU-0006
Tipo de recurso: Ficha

Descripción

La API Javascript de Alfresco permite a los desarrolladores de scripts escribir ficheros compatibles con el estándar JavaScript (ECMA Script) 1.6 para acceder, modificar, y crear objetos del repositorio de Alfresco. Proporciona una API simple, clara y orientada a objetos hacia conceptos bien conocidos de Alfresco como Nodos, Propiedades, Asociaciones y Aspectos.

Consulta de Documentos con Java API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0056
Tipo de recurso: Ejemplo

Descripción

Obtenemos el servicio de Registro,necesario para poder realizar cualquier acción.

```
ApplicationContext ctx = ApplicationContextHelper.getApplicationContext();
        final ServiceRegistry serviceRegistry = (ServiceRegistry) ctx.getBean(ServiceRegistry.SERVICE_REGISTRY);
```

Obtenemos los servicios necesarios para poder realizar la consulta de documentos

```
AuthenticationService authenticationService = serviceRegistry.getAuthenticationService();
SearchService searchService = serviceRegistry.getSearchService();
        NodeService serviceNode=serviceRegistry.getNodeService();
```

El siguiente paso sera el de identificarnos en el sistema

```
authenticationService.authenticate("admin", "admin".toCharArray());
```

Realizamos la búsqueda mediante lucene

```
StoreRef storeRef = new StoreRef(StoreRef.PROTOCOL_WORKSPACE, "SpacesStore");
        ResultSet resultSet = searchService.query(storeRef, SearchService.LANGUAGE_LUCENE, "PATH:\"/app:company_hor
```

Guardamos el resultado de la búsqueda en una lista.

```
List<NodeRef> spaces = resultSet.getNodeRefs();
```

Mostramos los documentos encontrados

```
QName QNamedoc = QName.createQName("{http://www.alfresco.org/model/content/1.0}name");

        for(int i=0;i<spaces.size();i++)
        {
        NodeRef space=(NodeRef) spaces.get(i);
        String nombre=serviceNode.getProperty(space, QNamedoc).toString();
        System.out.println("Nombre:"+nombre);
        }
```

Gestion de los documentos con Java API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0062
Tipo de recurso: Ejemplo

Descripción

Obtenemos el servicio de Registro,necesario para poder realizar cualquier acción.

```
ApplicationContext ctx = ApplicationContextHelper.getApplicationContext();
        final ServiceRegistry serviceRegistry = (ServiceRegistry) ctx.getBean(ServiceRegistry.SERVICE_REGISTRY);
```

El siguiente paso sera el de identificarnos en el sistema

```
        AuthenticationService authenticationService = serviceRegistry.getAuthenticationService();
        authenticationService.authenticate("admin", "admin".toCharArray());
```

Obtenemos el nodo Company Home,que sera en el cual realizaremos las pruebas

```
SearchService searchService = serviceRegistry.getSearchService();
        StoreRef storeRef = new StoreRef(StoreRef.PROTOCOL_WORKSPACE, "SpacesStore");
        ResultSet resultSet = searchService.query(storeRef, SearchService.LANGUAGE_LUCENE, "PATH:\"/app:company_hor
        NodeRef companyHome = resultSet.getNodeRef(0);
```

Creamos el mapa de propiedades y establecemos las propiedades para el documento

```
String name = "sample API JAVA (" + System.currentTimeMillis() + ").txt";
        Map<QName, Serializable> contentProps = new HashMap<QName, Serializable>();
        contentProps.put(ContentModel.PROP_NAME, name);
```

Creamos el documento

```
NodeService nodeService = serviceRegistry.getNodeService();
        ChildAssociationRef association = nodeService.createNode(companyHome,
                ContentModel.ASSOC_CONTAINS,
                QName.createQName(NamespaceService.CONTENT_MODEL_PREFIX, name),
                ContentModel.TYPE_CONTENT,
                contentProps);
        NodeRef content = association.getChildRef();
```

Añadimos el aspecto Titled (para la Visualización en el cliente Web)

```
Map<QName, Serializable> titledProps = new HashMap<QName, Serializable>();
        titledProps.put(ContentModel.PROP_TITLE, name);
        titledProps.put(ContentModel.PROP_DESCRIPTION, name);
        nodeService.addAspect(content, ContentModel.ASPECT_TITLED, titledProps);
```

Insertamos contenido en el documento

```
ContentService contentService = serviceRegistry.getContentService();
        ContentWriter writer = contentService.getWriter(content, ContentModel.PROP_CONTENT, true);
        writer.setMimetype(MimetypeMap.MIMETYPE_TEXT_PLAIN);
        writer.setEncoding("UTF-8");
        String text = "Texto de ejemplo para el documento sample API JAVA";
        writer.putContent(text);



        System.out.println("Finalización creación Documento");
```

Consulta de Espacios con Java API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0057
Tipo de recurso: Ejemplo

Descripción

Obtenemos el servicio de Registro,necesario para poder realizar cualquier acción.

```
ApplicationContext ctx = ApplicationContextHelper.getApplicationContext();
        final ServiceRegistry serviceRegistry = (ServiceRegistry) ctx.getBean(ServiceRegistry.SERVICE_REGISTRY);
```

Obtenemos los servicios necesarios para poder realizar la consulta de documentos

```
AuthenticationService authenticationService = serviceRegistry.getAuthenticationService();
SearchService searchService = serviceRegistry.getSearchService();
        NodeService serviceNode=serviceRegistry.getNodeService();
```

El siguiente paso sera el de identificarnos en el sistema

```
authenticationService.authenticate("admin", "admin".toCharArray());
```

Realizamos la búsqueda mediante lucene

```
StoreRef storeRef = new StoreRef(StoreRef.PROTOCOL_WORKSPACE, "SpacesStore");
        ResultSet resultSet = searchService.query(storeRef, SearchService.LANGUAGE_LUCENE, "PATH:\"/app:company_hor
```

Guardamos el resultado de la búsqueda en una lista.

```
List<NodeRef> spaces = resultSet.getNodeRefs();
```

Mostramos los espacios encontrados

```
QName QNamedoc = QName.createQName("{http://www.alfresco.org/model/content/1.0}name");

        for(int i=0;i<spaces.size();i++)
        {
        NodeRef space=(NodeRef) spaces.get(i);
        String nombre=serviceNode.getProperty(space, QNamedoc).toString();
        System.out.println("Nombre:"+nombre);
        }
```

Gestión de Espacios con Java API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0058
Tipo de recurso: Ejemplo

Descripción

Obtenemos el servicio de Registro,necesario para poder realizar cualquier acción. Declaramos los servicios necesarios.

```
ApplicationContext ctx = ApplicationContextHelper.getApplicationContext();
final ServiceRegistry serviceRegistry = (ServiceRegistry) ctx.getBean(ServiceRegistry.SERVICE_REGISTRY);
AuthenticationService authenticationService = serviceRegistry.getAuthenticationService();
SearchService searchService = serviceRegistry.getSearchService();
NodeService serviceNode = serviceRegistry.getNodeService();
```

El siguiente paso sera el de identificarnos en el sistema

```
authenticationService.authenticate("admin", "admin".toCharArray());
```

Obtenemos el nodo Company Home,que sera en el cual realizaremos las pruebas

```
StoreRef storeRef = new StoreRef(StoreRef.PROTOCOL_WORKSPACE, "SpacesStore");
ResultSet resultSet = searchService.query(storeRef, SearchService.LANGUAGE_LUCENE, "PATH:\"/app:company_home\"")
NodeRef companyHome = resultSet.getNodeRef(0);
```

Creamos una variable para el nombre y un mapa de propiedades para el archivo

```
String name = "sample2";
Map<QName, Serializable> contentProps = new HashMap<QName, Serializable>();
contentProps.put(ContentModel.PROP_NAME, name);
```

Creamos un nuevo nodo (espacio) en el nodo Company Home

```
ChildAssociationRef association = serviceNode.createNode(companyHome, ContentModel.ASSOC_CONTAINS,
QName.createQName(NamespaceService.CONTENT_MODEL_PREFIX, name), ContentModel.TYPE_FOLDER, contentProps);
NodeRef space = association.getChildRef();
```

Borramos el espacio creado

```
serviceNode.deleteNode(space);
```

Búsquedas con Java API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0059
Tipo de recurso: Ejemplo

Descripción

Búsqueda con Lucene

Obtenemos el servicio de Registro,necesario para poder realizar cualquier acción. Declaramos los servicios necesarios.

```
ApplicationContext ctx = ApplicationContextHelper.getApplicationContext();
final ServiceRegistry serviceRegistry = (ServiceRegistry) ctx.getBean(ServiceRegistry.SERVICE_REGISTRY);
```

Obtenemos los servicios necesarios para poder realizar la búsqueda

```
TransactionService transactionService = serviceRegistry.getTransactionService();
AuthenticationService authenticationService = serviceRegistry.getAuthenticationService();
SearchService searchService = serviceRegistry.getSearchService();
NodeService serviceNode=serviceRegistry.getNodeService();
```

El siguiente paso sera el de identificarnos en el sistema

```
authenticationService.authenticate("admin", "admin".toCharArray());
```

Realizamos la búsqueda y obtenemos los resultados

```
StoreRef storeRef = new StoreRef(StoreRef.PROTOCOL_WORKSPACE, "SpacesStore");
	ResultSet resultSet = searchService.query(storeRef, SearchService.LANGUAGE_LUCENE, "PATH:\"/app:company_hor

	QName QNamedoc = QName.createQName("{http://www.alfresco.org/model/content/1.0}name");


	List<NodeRef> spaces = resultSet.getNodeRefs();
```

Mostramos el resultado de la búsqueda

```
for(int i=0;i<spaces.size();i++)
	{
	NodeRef space=(NodeRef) spaces.get(i);
	String nombre=serviceNode.getProperty(space, QNamedoc).toString();
	System.out.println("Nombre:"+nombre);
	}
```

Búsqueda con XPath

Obtenemos el servicio de Registro,necesario para poder realizar cualquier acción. Declaramos los servicios necesarios.

```
ApplicationContext ctx = ApplicationContextHelper.getApplicationContext();
final ServiceRegistry serviceRegistry = (ServiceRegistry) ctx.getBean(ServiceRegistry.SERVICE_REGISTRY);
```

Obtenemos los servicios necesarios para poder realizar la búsqueda

```
TransactionService transactionService = serviceRegistry.getTransactionService();
AuthenticationService authenticationService = serviceRegistry.getAuthenticationService();
SearchService searchService = serviceRegistry.getSearchService();
NodeService serviceNode=serviceRegistry.getNodeService();
```

El siguiente paso sera el de identificarnos en el sistema

```
authenticationService.authenticate("admin", "admin".toCharArray());
```

Realizamos la búsqueda y obtenemos los resultados

```
StoreRef storeRef = new StoreRef(StoreRef.PROTOCOL_WORKSPACE, "SpacesStore");
```

```
        ResultSet resultSet = searchService.query(storeRef, SearchService.LANGUAGE_XPATH, "/*/*/.");

        List<NodeRef> spaces = resultSet.getNodeRefs();
        QName QNamedoc = QName.createQName("{http://www.alfresco.org/model/content/1.0}name");
```

Mostramos el resultado de la búsqueda

```
for(int i=0;i<spaces.size();i++)
        {
        NodeRef space=(NodeRef) spaces.get(i);
        String nombre=serviceNode.getProperty(space, QNamedoc).toString();
        System.out.println("Nombre:"+nombre);
        }
```

Gestión de Usuarios con Java API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0060
Tipo de recurso: Ejemplo

Descripción

Gestión de usuarios

Obtenemos el objeto ServiceRegistry el cual nos permitira acceder a los servicios necesarios.

```
ApplicationContext ctx = ApplicationContextHelper.getApplicationContext();
final ServiceRegistry serviceRegistry = (ServiceRegistry) ctx.getBean(ServiceRegistry.SERVICE_REGISTRY);
```

Declaramos los servicios necesarios para poder realizar las funciones necesarias

```
AuthenticationService authenticationService = serviceRegistry.getAuthenticationService();
SearchService searchService = serviceRegistry.getSearchService();
NodeService serviceNode = serviceRegistry.getNodeService();
PersonService servicePerson = serviceRegistry.getPersonService();
```

El siguiente paso sera identificarnos en el sistema

```
authenticationService.authenticate("admin", "admin".toCharArray());
```

Buscamos el nodo principal para crear un espacio para el nuevo usuario

```
StoreRef storeRef = new StoreRef(StoreRef.PROTOCOL_WORKSPACE, "SpacesStore");
        ResultSet resultSet = searchService.query(storeRef, SearchService.LANGUAGE_LUCENE, "PATH:\"/app:company_hor
        NodeRef usersStorageSpace = resultSet.getNodeRef(0);
```

Establecemos el nombre del usuario y creamos el mapa de propiedades del usuario.

```
String userName = "usuario";
Map<QName, Serializable> contentProps = new HashMap<QName, Serializable>();
contentProps.put(ContentModel.PROP_NAME, userName);
```

Creamos el espacio para el nuevo usuario

```
ChildAssociationRef association = serviceNode.createNode(usersStorageSpace, ContentModel.ASSOC_CONTAINS,
 QName.createQName(NamespaceService.CONTENT_MODEL_PREFIX, userName), ContentModel.TYPE_FOLDER, contentPr
```

Establecemos los tipos de propiedades para el mapa de propiedades del usuario.

```
NodeRef userSpace = association.getChildRef();
        Map<QName, Serializable> titledProps = new HashMap<QName, Serializable>();
        titledProps.put(ContentModel.PROP_TITLE, userName);
        titledProps.put(ContentModel.PROP_DESCRIPTION, userName);
        serviceNode.addAspect(userSpace, ContentModel.ASPECT_TITLED, titledProps);
```

Establecemos las propiedades para el usuario

```
HashMap<QName, Serializable> properties = new HashMap<QName, Serializable>();
        properties.put(ContentModel.PROP_USERNAME, userName);
        properties.put(ContentModel.PROP_HOMEFOLDER, userSpace);
        properties.put(ContentModel.PROP_PASSWORD, "password");
        properties.put(ContentModel.PROP_FIRSTNAME, "nombre");
        properties.put(ContentModel.PROP_LASTNAME, "apellido");
        properties.put(ContentModel.PROP_EMAIL, "user@user.es");
        properties.put(ContentModel.PROP_ORGID, "orgid");
```

Creamos al usuario

```
servicePerson.createPerson(properties);
```

Borramos el espacio creado para el usuario

```
serviceNode.deleteNode(userSpace);
```

Borramos a la persona

```
servicePerson.deletePerson(userName);
```

Gestión de grupos

Obtenemos el objeto ServiceRegistry el cual nos permitira acceder a los servicios necesarios.

```
ApplicationContext ctx = ApplicationContextHelper.getApplicationContext();
final ServiceRegistry serviceRegistry = (ServiceRegistry) ctx.getBean(ServiceRegistry.SERVICE_REGISTRY);
```

Declaramos los servicios necesarios para poder realizar las funciones necesarias

```
AuthenticationService authenticationService = serviceRegistry.getAuthenticationService();
        AuthorityService servicioaut= serviceRegistry.getAuthorityService();
```

Nos identificamos

```
authenticationService.authenticate("admin", "admin".toCharArray());
```

Establecemos el nombre del grupo

```
String groupName="grupo_prueba";
```

Comprobamos si existe el grupo

```
String actualName = serviceRegistry.getAuthorityService().getName(AuthorityType.GROUP, groupName);
        if (servicioaut.authorityExists(actualName) == false)
        {
           servicioaut.createAuthority(AuthorityType.GROUP, null, groupName);
        }
        else
        {
           System.out.println("El grupo ya existe");
        }
```

Añadimos el usuario al grupo

```
servicioaut.addAuthority(actualName, "usuario");
```

Obtenemos los grupos en los que esta el usuario logueado

```
Set<String> a= servicioaut.getAuthorities();
        for(int i=0;i<a.size();i++)
        {
            System.out.println("El Grupo es:"+a.toArray()[i].toString());
        }
```

Se obtienen los grupos en los que se encuentra un usuario especifico

```
Set<String> d= servicioaut.getAuthoritiesForUser("usuario");
        System.out.println("tamaño de los grupos: "+d.size());
        for(int i=0;i<d.size();i++)
        {
            System.out.println("El Grupo es: "+d.toArray()[i].toString());
        }
```

Eliminamos al usuario del grupo

```
servicioaut.removeAuthority(actualName, "usuario");
```

Eliminamos el grupo

```
servicioaut.deleteAuthority(actualName);
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/60

Categorías y Aspectos con Java API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0061
Tipo de recurso: Ejemplo

Descripción

Obtenemos el objeto ServiceRegistry el cual nos permitira acceder a los servicios necesarios.

```
ApplicationContext ctx = ApplicationContextHelper.getApplicationContext();
        final ServiceRegistry serviceRegistry = (ServiceRegistry) ctx.getBean(ServiceRegistry.SERVICE_REGISTRY);
```

Declaramos los servicios necesarios

```
AuthenticationService authenticationService = serviceRegistry.getAuthenticationService();
        CategoryService serviceCategory=serviceRegistry.getCategoryService();
        SearchService searchService = serviceRegistry.getSearchService();
        NodeService serviceNode= serviceRegistry.getNodeService();
```

El siguiente paso sera identificarnos en el sistema

```
authenticationService.authenticate("admin", "admin".toCharArray());
```

Declaramos los qnames necesarios

```
QName QNamecategories = QName.createQName("{http://www.alfresco.org/model/content/1.0}categories");
        QName QNamedoc = QName.createQName("{http://www.alfresco.org/model/content/1.0}name");
        QName QNameAspect = QName.createQName("{http://www.alfresco.org/model/content/1.0}generalclassifiable");
```

Declaramos el storeref con el cual trabajeremos

```
StoreRef storeRef = new StoreRef(StoreRef.PROTOCOL_WORKSPACE, "SpacesStore");
```

Buscamos la ruta general de las categorias

```
ResultSet resultSet = searchService.query(storeRef, SearchService.LANGUAGE_LUCENE, "PATH:\"/cm:generalclassifiable\"
        NodeRef usersStorageSpace = resultSet.getNodeRef(0);
```

Creamos la categoria

```
NodeRef categoria= serviceCategory.createCategory(usersStorageSpace, "pruebadecategoria");
```

Obtenemos un documento con el que trabajaremos

```
String query="PATH:\"/app:company_home/*\" AND TYPE:\"cm:content\"  ";
 ResultSet resultSet2 = searchService.query(storeRef, SearchService.LANGUAGE_LUCENE, query);
 List<NodeRef> spaces = resultSet2.getNodeRefs();
 NodeRef doc=null;
        for(int i=0;i<spaces.size();i++)
        {
            String cadena=serviceNode.getProperty(resultSet2.getRow(i).getNodeRef(), QNamedoc).toString();
            if(cadena.equals( "prueba.txt" ))
            {
                System.out.println("entra en el archivo");
                doc=(NodeRef) spaces.get(i);
            }
        }
```

Si el documento existe empezamos a trabajar sobre el.

```
if(doc!=null)
        {
```

Añadimos el aspecto al documento

```
if(!serviceNode.hasAspect(doc, QNameAspect))
        {
        serviceNode.addAspect(doc,QNameAspect  , null);
        }
```

Buscamos la categoria y obtenemos el NodeRef correspondiente

```
String query2="PATH:\"/cm:generalclassifiable//cm:pruebadecategoria\"";
        ResultSet resultSet3 = searchService.query(storeRef, SearchService.LANGUAGE_LUCENE,query2 );
        List<NodeRef> Categorias = resultSet3.getNodeRefs();
        NodeRef categoriaobtenida=Categorias.get(0);
```

Añadimos la categoria al documento

```
ArrayList<NodeRef>categorias1=(ArrayList<NodeRef>) serviceNode.getProperty(doc, QNamecategories);
categorias1.add(categoriaobtenida);
serviceNode.setProperty(doc, ContentModel.PROP_CATEGORIES, categorias1);
```

Eliminamos la categoria del documento

```
ArrayList<NodeRef>categorias2=(ArrayList<NodeRef>) serviceNode.getProperty(doc, QNamecategories);
    for(int i=0;i<categorias2.size();i++)
    {
        if(categorias2.get(i).getId().equals(categoriaobtenida.getId()))
        {
   categorias2.remove(i);
        }
    }
    serviceNode.setProperty(doc, ContentModel.PROP_CATEGORIES, categorias2);
```

Borramos la categoria

```
serviceCategory.deleteCategory(categoriaobtenida) ;
```

Finalizamos la estructura del if

```
}
else
{
    System.out.println("el documento es nulo");
}
```

Eliminamos el aspecto del documento

```
serviceNode.removeAspect(doc, QNameAspect);
```

Reglas y Procedimientos con Java API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0063
Tipo de recurso: Ejemplo

Descripción

Obtenemos el objeto ServiceRegistry el cual nos permitira acceder a los servicios necesarios.

```
ApplicationContext ctx = ApplicationContextHelper.getApplicationContext();
        final ServiceRegistry serviceRegistry = (ServiceRegistry) ctx.getBean(ServiceRegistry.SERVICE_REGISTRY);
```

Declaramos los servicios necesarios para poder realizar las funciones necesarias

```
ActionService serviceAction = serviceRegistry.getActionService();
        SearchService serviceSearch = serviceRegistry.getSearchService();
        AuthenticationService authenticationService = serviceRegistry.getAuthenticationService();
```

El siguiente paso sera identificarnos en el sistema

```
authenticationService.authenticate("admin", "admin".toCharArray());
```

Declaramos el storeref con el cual trabajaremos

```
StoreRef storeRef = new StoreRef(StoreRef.PROTOCOL_WORKSPACE, "SpacesStore");
```

Obtenemos la lista de definiciones de acciones

```
List<ActionDefinition> ruleActions = serviceAction.getActionDefinitions(companyhome);
        ArrayList<SelectItem> actions = new ArrayList<SelectItem>();
        for (ActionDefinition ruleActionDef : ruleActions)
        {
           String title = ruleActionDef.getTitle();
           if (title == null || title.length() == 0)
           {
              title = ruleActionDef.getName();
           }
           actions.add(new SelectItem(ruleActionDef.getName(), title));
        }
```

Vamos a escoger la acción de enviar un email , para esto seleccionaremos el indice 0 del ArrayList para escoger la Definición de la Acción y con esta definición creamos la acción .

```
Action a = serviceAction.createAction(actions.get(0).getValue().toString());
```

Creamos el mapa de parametros y lo rellenamos

```
Map<String, Serializable> repoActionParams = new HashMap<String, Serializable>();
        repoActionParams.put("to", "correo@dominio.com");
        repoActionParams.put("subject", "Envio de Correo desde Java API");
        repoActionParams.put("from", "correo@dominio.com");
        repoActionParams.put("text", "Este texto sera el contenido del email");
```

Insertamos el mapa de parametros a la acción

```
a.setParameterValues(repoActionParams);
```

Realizamos la ejecución de la acción

```
serviceAction.executeAction(a, companyhome);
```

Consulta de Documentos con Java JCR API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0064
Tipo de recurso: Ejemplo

Descripción

Obtenemos el Bean necesario para poder acceder al respositorio mediante JCR.

```
ApplicationContext context = new ClassPathXmlApplicationContext("classpath:alfresco/application-context.xml");
        Repository repository = (Repository)context.getBean("JCR.Repository");
```

El siguiente paso sera el de identificarnos en el sistema

```
Session sesion = repository.login(new SimpleCredentials("admin", "admin".toCharArray()));
```

Obtenemos los nodos que existan en el Company Home , comprobaremos si es un documento y mostraremos su nombre .

```
Node root;
    try {
    root = sesion.getRootNode();
    Node companyHome = root.getNode("app:Company_Home");
    NodeIterator entries = companyHome.getNodes();
    while (entries.hasNext())
            {
               Node entry = (Node) entries.nextNode();
                  if(entry.isNodeType("cm:content"))
                  {
                  System.out.println("Nombre:" +entry.getProperty("cm:name").getString());
                  }
               }

            }
        catch (RepositoryException e)
        {
    e.printStackTrace();
    }
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/64

Gestión de Documentos con Java JCR API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0065
Tipo de recurso: Ejemplo

Descripción

Obtenemos el Bean necesario para poder acceder al respositorio mediante JCR.

```
ApplicationContext context = new ClassPathXmlApplicationContext("classpath:alfresco/application-context.xml");
        Repository repository = (Repository)context.getBean("JCR.Repository");
```

El siguiente paso sera el de identificarnos en el sistema

```
Session sesion = repository.login(new SimpleCredentials("admin", "admin".toCharArray()));
```

Obtenemos el nodo Company_Home

```
Node root;
        try {
                root = sesion.getRootNode();
                Node companyHome = root.getNode("app:Company_Home");
```

Creamos el documento

```
companyHome.addNode("cm:archivo.txt", "cm:content");
```

Obtenemos el documento creado

```
Node doc = root.getNode("app:company_home/cm:archivo.txt");
```

Establecemos propiedades al documento

```
doc.setProperty("cm:title", "archivotitulo");
                doc.setProperty("cm:name", "archivonombre.txt");
                doc.setProperty("cm:content", "esto es el contenido del documento ");
```

Mostramos las propiedades para confirmar su correcta edición

```
System.out.println("titulo:"+doc.getProperty("cm:title").getString());
                System.out.println("nombre:"+doc.getProperty("cm:name").getString());
                System.out.println("contenido:"+doc.getProperty("cm:content").getString());
```

Eliminamos el documento

```
doc.remove();
```

Guardamos la sesion

```
sesion.save();
```

Finalizamos el Try Catch

```
} catch (RepositoryException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            }
            catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
```

Consulta de Espacios con Java JCR API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0066
Tipo de recurso: Ejemplo

Descripción

Obtenemos el Bean necesario para poder acceder al respositorio mediante JCR.

```
ApplicationContext context = new ClassPathXmlApplicationContext("classpath:alfresco/application-context.xml");
        Repository repository = (Repository)context.getBean("JCR.Repository");
```

El siguiente paso sera el de identificarnos en el sistema

```
Session sesion = repository.login(new SimpleCredentials("admin", "admin".toCharArray()));
```

Obtenemos los nodos que existan en el Company Home , comprobaremos si es un espacio y mostraremos su nombre.

```
Node root;
    try {
    root = sesion.getRootNode();
    Node companyHome = root.getNode("app:Company_Home");
    NodeIterator entries = companyHome.getNodes();
    while (entries.hasNext())
            {
               Node entry = (Node) entries.nextNode();
                  if(entry.isNodeType("cm:folder"))
                  {
                  System.out.println("Nombre:" +entry.getProperty("cm:name").getString());
                  }
               }

            }
        catch (RepositoryException e)
        {
    e.printStackTrace();
    }
```

Gestión de Espacios con Java JCR API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0067
Tipo de recurso: Ejemplo

Descripción

Obtenemos El bean necesario para poder trabajar con el respositorio

```
ApplicationContext context = new ClassPathXmlApplicationContext("classpath:alfresco/application-context.xml");
Repository repository = (Repository)context.getBean("JCR.Repository");
```

El siguiente paso sera el de identificarnos en el sistema

```
Session sesion = repository.login(new SimpleCredentials("admin", "admin".toCharArray()));
```

Obtenemos el nodo Company_Home

```
Node root;
        try {
            root = sesion.getRootNode();
Node companyHome = root.getNode("app:Company_Home");
```

Creamos el directorio prueba,establecemos el titulo, lo recuperamos y lo mostramos.

```
Node prueba=companyHome.addNode("app:prueba", "cm:folder");
                prueba.setProperty("cm:title","TituloPrueba");

                String titulo=prueba.getProperty("cm:title").getString();

                System.out.println("titulo Obtenido:"+titulo);
```

Borramos el espacio

```
prueba.remove();
```

Guardamos la sesion

```
sesion.save();
```

Finalizamos el Try Catch

```
} catch (RepositoryException e) {
        e.printStackTrace();
        }
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/67

Búsquedas con Java JCR API

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0068
Tipo de recurso: Ejemplo

Descripción

Búsqueda con Lucene

La búsqueda con Lucene no se implementa en la API de JCR

Búsqueda con Xpath

Obtenemos el Bean necesario para poder acceder al respositorio mediante JCR

```
ApplicationContext context = new ClassPathXmlApplicationContext("classpath:alfresco/application-context.xml");
        Repository repository = (Repository)context.getBean("JCR.Repository");
```

El siguiente paso sera el de identificarnos en el sistema

```
Session sesion = repository.login(new SimpleCredentials("admin", "admin".toCharArray()));
```

Generamos la consulta en formato Xpath

```
try {
        Workspace workspace = sesion.getWorkspace();
        QueryManager queryManager = workspace.getQueryManager();
        Query query = queryManager.createQuery("//app:company_home/*", Query.XPATH);
```

Realizamos la búsqueda

```
QueryResult result = query.execute();
```

Obtenemos los resultados de la búsqueda

```
NodeIterator it = result.getNodes();
```

Mostramos los resultados

```
while (it.hasNext())
            {

            Node n = it.nextNode();
            System.out.println("Nombre: "+n.getProperty("cm:name").getString());

            }
```

Finalizamos en try

```
} catch (InvalidQueryException e) {
            e.printStackTrace();
        } catch (RepositoryException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

Consulta de Documentos con JavaScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0069
Tipo de recurso: Ejemplo

Descripción

Iniciamos las variables necesarias

```
var nombre="listado"; //nombre del archivo
var doc="";
var fechaformateada="";
var log = "";
var name = space.properties.name;
var fecha=new Date ();
var query="";
var resultset=null;
fechaformateada="."+fecha.getDate()+"-"+(fecha.getMonth()+1)+"-"+fecha.getFullYear()+"_"+fecha.getHours()+"."+f
```

Realizamos la búsqueda mediante lucene

```
query="PATH:\"/";
query+=  space.qnamePath;
query+=  "//*\" ";
query+=  "AND TYPE:\"cm:content\" ";
resultset = search.luceneSearch( query);
```

Creamos el contenido del archivo para poder observar el resultado posteriormente en el archivo Log

```
for (var i=0; i< resultset.length; i++)
{      log += "Nombre: " + resultset[i].name + " \r\n " ; }
```

Creamos el archivo de log

```
doc = space.createFile(nombre+fechaformateada+".txt");
doc.content += "Listado de Documentos en el Espacio : "+ space.properties.name + " \r\n\r\n ";
doc.content +=  log+" \r\n\r\n ";
doc.content +="Archivo creado el:" +fecha;
```

Gestión de Documentos con JavaScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0070
Tipo de recurso: Ejemplo

Descripción

Declaramos la variable salida que nos permitira seguir un control sobre las acciones que se realizan(log),y la variable nombrefichero sera el nombre del documento que se va a crear.

```
var salida="";
var nombrefichero = "documentoprueba";
```

Creamos un espacio de pruebas en el directorio Company Home

```
space.createFolder("PruebadeDocumentos");

    var dir1= space.childByNamePath("PruebadeDocumentos");

    salida+="Creacion de espacios correcta \r\n";
```

Creamos un nuevo documento

```
var doc= dir1.createFile(nombrefichero+".txt");
    doc.addAspect("cm:versionable");

    salida+="Creacion del documento correcta \r\n";
```

Insertamos contenido en el documento

```
doc.content="texto original\r\n";

    //Hacemos el checkout al documento
    var doccopy=doc.checkout();

    //actualizamos el documento
    doccopy.content+="actualizacion 1";

    //Hacemos el checkin al documento
    doc= doccopy.checkin();
```

Mostramos el contenido del documento para comprobar su correcta edición

```
salida+="Mostramos el contenido del documento :\r\n";
    salida+=doc.content;
    salida+="\r\n";
```

Se le cambian los metadatos

```
doc.name="nuevonombre.txt";
    doc.properties.author="Jose";
    doc.save();
    salida+="se ha cambiado correctamente el nombre al documento\r\n";
```

Borramos el documento

```
if(doc.remove())
    {
    salida+="Se la eliminado correctamente el documento\r\n";
    }
    else
    {
    salida+="Se ha producido un fallo en la eliminacion del documento\r\n";
```

```
    }
```

Creamos el archivo de log

```
doc = dir1.createFile("log.txt");
    doc.content += "LOG \r\n\r\n";
    doc.content +=  salida+"\r\n\r\n";
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/70

Consulta de Espacios con JavaScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0071
Tipo de recurso: Ejemplo

Descripción

Iniciamos las variables necesarias

```
var nombre="listado"; //nombre del archivo
var doc="";
var fechaformateada="";
var log = "";
var name = space.properties.name;
var fecha=new Date ();
var query="";
var resultset=null;
fechaformateada="."+fecha.getDate()+"-"+(fecha.getMonth()+1)+"-"+fecha.getFullYear()+"_"+fecha.getHours()+"."+f
```

Realizamos la búsqueda mediante lucene

```
query="PATH:\"/";
query+=  space.qnamePath;
query+=  "//*\" ";
query+=  "AND TYPE:\"cm:folder\" ";
resultset = search.luceneSearch( query);
```

Creamos el contenido del archivo para poder observar el resultado posteriormente en el archivo Log

```
for (var i=0; i< resultset.length; i++)
{      log += "Nombre: " + resultset[i].name + " \r\n " ; }
```

Creamos el archivo de log

```
doc = space.createFile(nombre+fechaformateada+".txt");
doc.content += "Listado de Documentos en el Espacio : "+ space.properties.name + " \r\n\r\n ";
doc.content +=  log+" \r\n\r\n ";
doc.content +="Archivo creado el:" +fecha;
```

Gestión de Espacios con JavaScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0072
Tipo de recurso: Ejemplo

Descripción

Código correspondiente al archivo OperacionesEspacios.js

Iniciamos las variables necesarias

```
var salida="";
var nombredirectorio = "directorio";
```

Creamos Dos espacios con los cuales trabajaremos

```
space.createFolder(nombredirectorio);
    space.createFolder(nombredirectorio+"2");
    var dir1= space.childByNamePath(nombredirectorio);
    var dir2= space.childByNamePath(nombredirectorio+"2");
    salida+="Creación de espacios correcta \r\n";
```

Movemos un espacio dentro de otro

```
if(dir2.move(dir1))
    {
    salida+="espacio movido correctamente\r\n";
    }
    else
    {
    salida+="fallo en el movimiento del directorio\n\r\n";
    }
```

Renombramos el directorio creado

```
dir1.properties.name="TEST";
        dir1.save();
        salida+="Se ha producido correctamente el cambio de nombre del directorio\r\n";
```

Le asignamos unos nuevos permisos al espacio, el usuario "usuario" obtendra el permiso de Consumer sobre el espacio creado

```
dir1.setInheritsPermissions(false);
    dir1.removePermission("Consumer","GROUP_GROUP_EVERYONE");
    dir1.setPermission("Consumer","usuario");
    salida+="Se le han otorgado correctamente los permisos al usuario \"usuario\" \r\n";
```

Eliminamos el directorio creado

```
if(dir1.remove())
{
salida+="Se la eliminado correctamente el directorio\r\n";
}
else
{
salida+="Se ha producido un fallo en la eliminación del directorio\r\n";
}
```

Creamos el archivo de log

```
doc = dir1.createFile("log.txt");
    doc.content += "LOG \r\n\r\n";
    doc.content +=  salida+"\r\n\r\n";
```

Código correspondiente al archivo OperacionesEspaciosPermisos.js

Para comprobar que el directorio tiene los permisos que se le han otorgado el siguiente código comprueba si el usuario tiene permisos de lectura y escritura, (Recuerda iniciar una sesión con el usuario que se le haya otorgado al espacio).

```
var salida="";
var nombreespacio = "TEST";
var dir1= space.childByNamePath(nombreespacio);


        if(dir1.hasPermission("Read"))
        {
        salida+="Si tiene permisos de lectura\r\n";
        }
        else
        {
        salida+="No tiene permisos de lectura\r\n";
        }
        if(dir1.hasPermission("Write"))
        {
        salida+="Si tiene permisos de Escritura\r\n";
        }
        else
        {
        salida+="No tiene permisos de Escritura\r\n";
        }
```

Creamos el archivo de log

```
doc = dir1.createFile("log_permisos.txt");
    doc.content += "LOG \r\n\r\n";
    doc.content +=  salida+"\r\n\r\n";
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/72

Búsquedas con JavaScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0073
Tipo de recurso: Ejemplo

Descripción

Iniciamos las variables necesarias

```
var salida="";
```

Búsqueda con Lucene

Establecemos la consulta para realizar la búsqueda.

```
var consultaLucene1="TEXT:alfresco";
var consultaLucene2="PATH:\"//app:company_home/*\"";
var consultaLucene3="TYPE:\"cm:content\"";
var consultaLucene4="TYPE:\"cm:folder\"";
```

Realizamos la búsqueda

```
var nodeslucene = search.luceneSearch(consultaLucene2+" AND "+consultaLucene3);
```

Mostramos los resultados de la búsqueda

```
for(var i=0;i<nodeslucene.length;i++)
{
salida+="Documento:"+nodeslucene[i].name+"\r\n";
}
```

Creamos el archivo de Log en el directorio pruebas

```
var dir1= space.childByNamePath("pruebas");
    doc = dir1.createFile("log.txt");
    doc.content += "LOG \r\n\r\n";
    doc.content +=  salida+"\r\n\r\n";
```

Búsqueda con Xpath

Establecemos la consulta para realizar la búsqueda.

```
var consultaXpath1="//app:company_home//@";
var consultaXpath2="//app:company_home//.";
```

Realizamos la búsqueda

```
var nodesxpath = search.xpathSearch(consultaXpath1);
```

Mostramos los resultados de la búsqueda

```
for(var i=0;i<nodesxpath.length;i++)
{
salida+="Documento:"+nodesxpath[i].name+"\r\n";
}
```

Creamos el archivo de Log

```
var dir1= space.childByNamePath("pruebas");
    doc = dir1.createFile("log.txt");
    doc.content += "LOG \r\n\r\n";
    doc.content +=  salida+"\r\n\r\n";
```

Gestion de Usuarios con JavaScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0074
Tipo de recurso: Ejemplo

Descripción

La creación de un usuario en JavaScript no esta disponible hasta la versión 3 de Alfresco

```
var salida="";
    var nombregrupo="grupo_prueba";
    var groupPrefix="GROUP_";
```

Creamos un grupo

```
people.createGroup(nombregrupo);
```

Movemos un usuario al grupo

```
var user="admin";
var lista=0;
var srcGrpNode=people.getGroup(groupPrefix+nombregrupo);
var authority = people.getPerson(user);


    if(authority)
    {
       people.addAuthority(srcGrpNode,authority);
       salida+="1.-a?adido usuario: \" "+user+" \" al grupo "+ nombregrupo+"<br>";
    }
```

Obtenemos usuarios de un grupo especificado

```
var grupoObtenido = people.getGroup(groupPrefix+nombregrupo);
    lista = people.getMembers(grupoObtenido);

    salida+="2.-el taman/o de la lista es :"+lista.length+"<br>";
```

Eliminamos al usuario del grupo

```
people.removeAuthority(srcGrpNode,authority);
```

Borramos el grupo

```
people.deleteGroup(srcGrpNode);
```

Categorías y Aspectos con JavaScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0075
Tipo de recurso: Ejemplo

Descripción

Iniciamos las variables necesarias

```
var salida="";
var espacio= "\r\n";
var nomFile="prueba.txt";
var documento = space.childByNamePath(nomFile);
```

Obtenemos el documento con el cual vamos a trabajar.

```
var query="PATH:\"//app:company_home/*\"";
    query= query + "AND TYPE:\"cm:content\" ";
    var nodes = search.luceneSearch(query);
    for(var i=0;i<nodes.length;i++)
    {

        if(nodes[i].name==nomFile)
        {
        var doc=nodes[i];
        }

    }
```

Creamos la categoria

```
classification.createRootCategory("cm:generalclassifiable","categoriatest");
    salida+="Categoria creada"+espacio;
```

Añadimos el aspecto al documento para que se pueda clasificar mediante categorias

```
doc.addAspect("cm:generalclassifiable");
    doc.save();
```

Obtenemos las categorias del documento

```
var catNodeRef = search.luceneSearch("PATH:\"/cm:generalclassifiable//cm:categoriatest\"")[0]  ;
    var cats = doc.properties["cm:categories"];

    if(cats==null)
    {
    cats=new Array();
    }

    salida+="el numero de categorias que tiene el documento son :"+cats.length+espacio;
```

Añadimos la categoria al documento respetando las que ya posee

```
var newCats = new Array();
    for (var i=0; i < cats.length; i++)
    {
         newCats[i] = cats[i];
    }
    newCats.push(catNodeRef);
```

Guardamos las categorias en el documento

```
doc.properties["cm:categories"] = newCats;
```

```
    doc.save();
```

Quitamos la categoria al documento

```
var catsActuales = doc.properties["cm:categories"];
    var newCatsActuales = new Array();

    if(catsActuales.length<=1)
    {
    doc.properties["cm:categories"]=newCatsActuales;
    }
    else
    {

    for (var i=0; i < catsActuales.length; i++)
    {
            if(catsActuales[i].properties.name!=catNodeRef.properties.name)
       {
         newCatsActuales.push(catsActuales[i]);
       }
    }

    doc.properties["cm:categories"] = newCatsActuales;




    }
```

Guardamos los cambios en el documento

```
doc.save();
```

Quitamos el aspecto al documento

```
doc.removeAspect("cm:generalclassifiable");
    doc.save();
```

Para la eliminación de la categoria de un documento ,primero se debe de obtener la categoria en cuestión y despues comprobar si el documento la tiene,en caso afirmativo se eliminara.

```
var categorias = classification.getAllCategoryNodes("cm:generalclassifiable");
    for(var i=0;i<categorias.length;i++)
    {
     if(categorias[i].name=="categoriatest")
     {
            categorias[i].removeCategory();
     }
    }
    salida+="La categoria ha sido eliminada correctamente"+espacio;
```

Creamos el archivo de log

```
var dir1= space.childByNamePath("pruebas");
doc = dir1.createFile("log.txt");
doc.content += "LOG \r\n\r\n";
doc.content +=  salida+"\r\n\r\n";
```

Reglas y Procedimientos con JavaScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0076
Tipo de recurso: Ejemplo

Descripción

Creamos la accion correspondiente para poder enviar un email

```
var mail = actions.create("mail");
```

Establecemos los parametros necesarios de la acción mail.

```
mail.parameters.to = "direccion@dominio.com";
mail.parameters.subject = "Envio de Correo desde JavaScript";
mail.parameters.from = "direccion@dominio.com";
mail.parameters.text = "texto del mensaje";
```

Ejecutamos la acción

```
mail.execute(document);
```

Consulta de Documentos con WebScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0077
Tipo de recurso: Ejemplo

Descripción

Código correspondiente al archivo listado.get.desc.xml

```
<webscript>
  <shortname>Listado</shortname>
  <description>Listado de documentos en Alfresco</description>
  <url>/sample/listadodocumentos/</url>
  <format default="html">argument</format>
  <authentication>user</authentication>
  <transaction>required</transaction>
</webscript>
```

Código correspondiente al archivo listado.get.html.ftl

```
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    Directorio Actual: ${url.extension}
    <br>
    <br>
    <table>
        <tr>
            <td>
            <td>Nombre</td>
            <td>Tipo </td>
        </tr>

        <#list resultset as node>
        <tr>
            <td><img src="/servicios/madeja/%24%7Burl.context%7D%24%7Bnode.icon16%7D"/></td>
            <td>${node.name}</td>
            <td>${node.type} </td>
        </tr>
        </#list>
    </table>
  </body>
</html>
```

Código correspondiente al archivo listado.get.js

En el caso de que no se introduzca correctamente la url lo controlaremos con el siguiente código.

```
if (url.extension == undefined || url.extension.length == 0)
{
   status.code = 400;
   status.message = "Search term has not been provided.";
   status.redirect = true;
}
```

Realizamos la búsqueda mediante Lucene

```
var folder = roothome.childByNamePath(url.extension);
var query="PATH:\"/";
query= query + folder.qnamePath;
query= query + "//*\" ";
```

```
query= query + "AND TYPE:\"cm:content\" ";
var a= folder.qnamePath;
var nodes = search.luceneSearch( query);
model.folder=folder;
model.resultset = nodes;
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/77

Gestión de Documentos con WebScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0078
Tipo de recurso: Ejemplo

Descripción

Archivo OperacionesDocumento.get.desc.xml

```
<webscript>
  <shortname>OperacionesDocumento</shortname>
  <description>Operaciones que se puede realizar con un documento en Alfresco</description>
  <url>/sample/OperacionesDocumento?nombre={nom}</url>
  <format default="html">argument</format>
  <authentication>guest</authentication>
</webscript>
```

Archivo OperacionesDocumento.get.html.ftl

```
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    ${salida}
  </body>
</html>
```

Archivo OperacionesDocumento.get.js

Declaramos la variable salida ,que nos servirá para posteriormente tener un log con los resultado de las operaciones

```
model.salida="";
```

comprobamos que se pase correctamente el parámetro nombre

```
if ((args.nombre) && (args.nombre!=""))
{
    var parametro_entrada = args.nombre;
}
else
{
    model.salida += "Error en el paso de argumentos\n<br>";
}
```

Obtenemos el nodo Company Home, en el cual realizaremos la operaciones necesarias.

```
var folder = roothome.childByNamePath("Company Home");
model.salida=" parametro: "+parametro_entrada+"<br>";
```

Se comprueba que la variable folder es valida.

```
if (folder == undefined || !folder.isContainer)
{
   status.code = 404;
   status.message = "Folder " + url.extension + " not found.";
   status.redirect = true;
}
else
{
```

Creamos un espacio de pruebas

```
folder.createFolder("PruebadeDocumentos");
```

```
    var dir1= folder.childByNamePath("PruebadeDocumentos");

    model.salida+="Creación de espacios correcta <br>";
```

Creamos el nuevo documento

```
var doc= dir1.createFile(parametro_entrada+".txt");
    doc.addAspect("cm:versionable");

    model.salida+="Creación del documento correcta <br>";
```

Insertamos informacion en el documento

```
doc.content="texto original\r\n";

    //Hacemos el checkout al documento
    var doccopy=doc.checkout();

    //actualizamos el documento
    doccopy.content+="actualizacion 1";

    //Hacemos el checkin al documento
    doc= doccopy.checkin();
```

Mostramos el contenido para comprobar que la edición se ha realizado correctamente

```
model.salida+="Mostramos el contenido del documento :<br>";
    model.salida+=doc.content;
    model.salida+="<br>";
```

Cambiamos los metadatos del documento

```
doc.name="nuevonombre.txt";
    doc.properties.author="Jose";
    doc.save();
    model.salida+="se ha cambiado correctamente el nombre al documento";
```

Borramos el documento

```
if(doc.remove())
    {
    model.salida+="Se la eliminado correctamente el documento<br>";
    }
    else
    {
    model.salida+="Se ha producido un fallo en la eliminación del documento<br>";
    }
}
```

Consulta de Espacios con WebScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0079
Tipo de recurso: Ejemplo

Descripción

Código correspondiente al archivo listado.get.desc.xml

```
<webscript>
  <shortname>OperacionesEspacio</shortname>
  <description>Operaciones que se puede realizar con un espacio/directorio en Alfresco</description>
  <url>/sample/OperacionesEspacio?nombre={nom}</url>
  <format default="html">argument</format>
  <authentication>user</authentication>
  <transaction>required</transaction>
</webscript>
```

Código correspondiente al archivo listado.get.html.ftl

```
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    Directorio Actual: ${url.extension}
    <br>
    <br>
    <table>
        <tr>
            <td>
            <td>Nombre</td>
            <td>Tipo </td>
        </tr>

        <#list resultset as node>
        <tr>
            <td><img src="/servicios/madeja/%24%7Burl.context%7D%24%7Bnode.icon16%7D"/></td>
            <td>${node.name}</td>
            <td>${node.type} </td>
        </tr>
        </#list>
    </table>
  </body>
</html>
```

Código correspondiente al archivo listado.get.js

En el caso de que no se introduzca correctamente la url lo controlaremos con el siguiente código.

```
if (url.extension == undefined || url.extension.length == 0)
{
   status.code = 400;
   status.message = "Search term has not been provided.";
   status.redirect = true;
}
var folder = roothome.childByNamePath(url.extension);
```

Realizamos la búsqueda de los espacios

```
var query="PATH:\"/";
query= query + folder.qnamePath;
query= query + "//*\" ";
```

```
query= query + "AND TYPE:\"cm:folder\" ";
var a= folder.qnamePath;
var nodes = search.luceneSearch( query);
model.folder=folder;
model.resultset = nodes;
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/79

Gestión de Espacios con WebScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0080
Tipo de recurso: Ejemplo

Descripción

Código correspondiente al archivo OperacionesEspacio.get.desc.xml

```
<webscript>
  <shortname>OperacionesEspacio</shortname>
  <description>Operaciones que se puede realizar con un espacio/directorio en Alfresco</description>
  <url>/sample/OperacionesEspacio?nombre={nom}</url>
  <format default="html">argument</format>
  <authentication>guest</authentication>
</webscript>
```

Código correspondiente al archivo OperacionesEspacio.get.html.ftl

```
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    ${salida}
  </body>
</html>
```

Código correspondiente al archivo OperacionesEspacio.get.js

Declaramos la variable salida ,que nos servirá para posteriormente tener un log con los resultado de las operaciones

```
model.salida="";
```

Comprobamos que el parametro recibido sea valido.

```
if ((args.nombre) && (args.nombre!=""))
{
    var parametro_entrada = args.nombre;
}
else
{
    model.salida += "Error en el paso de argumentos\n<br>";
}
```

Obtenemos el nodo Company Home ,que sera donde realizaremos las pruebas.

```
var folder = roothome.childByNamePath("Company Home");
model.salida=" parametro: "+parametro_entrada+"<br>";

if (folder == undefined || !folder.isContainer)
{
   status.code = 404;
   status.message = "Folder " + url.extension + " not found.";
   status.redirect = true;
}
else
{
```

Creamos Dos espacios con los cuales realizaremos las pruebas

```
folder.createFolder(parametro_entrada);
    folder.createFolder(parametro_entrada+"2");
    var dir1= folder.childByNamePath(parametro_entrada);
```

```
    var dir2= folder.childByNamePath(parametro_entrada+"2");
    model.salida+="Creación de espacios correcta <br>";
```

Movemos un espacio dentro de otro

```
if(dir2.move(dir1))
    {
    model.salida+="espacio movido correctamente<br>";
    }
    else
    {
    model.salida+="fallo en el movimiento del directorio\n<br>";
    }
```

Renombramos el directorio creado

```
dir1.properties.name="TEST";
        dir1.save();
        model.salida+="Se ha producido correctamente el cambio de nombre del directorio<br>";
```

Le asignamos unos nuevos permisos al espacio, el usuario "Jose" obtendra el permiso de Consumer sobre el espacio creado

```
dir1.setInheritsPermissions(false);
    dir1.removePermission("Consumer","GROUP_GROUP_EVERYONE");
    dir1.setPermission("Consumer","jose");
    model.salida+="Se le han otorgado correctamente los permisos al usuario \"jose\" <br>";
}
```

Eliminamos el directorio creado

```
if(dir1.remove())
{
model.salida+="Se la eliminado correctamente el directorio<br>";
}
else
{
model.salida+="Se ha producido un fallo en la eliminación del directorio<br>";
}
```

Permisos

Para comprobar que el directorio tiene los permisos que se le han otorgado el siguiente código comprueba si el usuario tiene permisos de lectura y escritura, (Recuerda iniciar una sesión con el usuario que se le haya otorgado al espacio).

Código correspondiente al archivo permisos.get.desc.xml

```
<webscript>
  <shortname>Permisos</shortname>
  <description>WebScript para comprobar los permisos de un directorio en Alfresco</description>
  <url>/sample/permisos?nombre={nom}</url>
  <format default="html">argument</format>
  <authentication>guest</authentication>
</webscript>
```

Código correspondiente al archivo permisos.get.html.ftl

```
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    ${salida}
  </body>
</html>
```

Código correspondiente al archivo permisos.get.js

Iniciamos la variable salida.

```
model.salida="";
```

Comprobamos que el parametro nombre sea valido y no este vacío,

```
if ((args.nombre) && (args.nombre!=""))
{
    var parametro_entrada = args.nombre;
}
else
{
    model.salida += "Error en el paso de argumentos\n<br>";
}
```

Obtenemos el nodo Company Home

```
var folder = roothome.childByNamePath("Company Home");
var dir1= folder.childByNamePath(parametro_entrada);
if (folder == undefined || !folder.isContainer || dir1==null)
{
   status.code = 404;
   status.message = "Directorio " + url.extension + " no encontrado.";
   status.redirect = true;
}
else
{
```

Comprobamos si el usuario tiene permisos de escritura y lectura sobre el espacio indicado por parametro.

```
model.salida+="El espacio : \""+folder.qnamePath +"/" +parametro_entrada +"\" es un espacio valido <br>";

        if(dir1.hasPermission("Read"))
        {
        model.salida+="Si tiene permisos de lectura\n<br>";
        }
        else
        {
        model.salida+="No tiene permisos de lectura\n<br>";
        }
        if(dir1.hasPermission("Write"))
        {
        model.salida+="Si tiene permisos de Escritura\n<br>";
        }
        else
        {
        model.salida+="No tiene permisos de Escritura\n<br>";
        }
}
```

Búsquedas con WebScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0081
Tipo de recurso: Ejemplo

Descripción

Código correspondiente al archivo Busquedas.get.desc.xml

```
<webscript>
  <shortname>Busquedas</shortname>
  <description>Busquedas en Alfresco</description>
  <url>/sample/Busquedas?nombre={nom}</url>
  <format default="html">argument</format>
  <authentication>guest</authentication>
</webscript>
```

Código correspondiente al archivo Busquedas.get.html.ftl

```
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    ${salida}
  </body>
</html>
```

Código correspondiente al archivo Busquedas.get.get.js

Iniciamos las variables

```
model.salida="";
var folder = roothome.childByNamePath("Company Home");
```

Búsqueda con Lucene

Establecemos la consulta para realizar la búsqueda.

```
var consultaLucene1="TEXT:alfresco";
var consultaLucene2="PATH:\"//app:company_home/*\"";
var consultaLucene3="TYPE:\"cm:content\"";
var consultaLucene4="TYPE:\"cm:folder\"";
```

Realizamos la búsqueda

```
var nodeslucene = search.luceneSearch(consultaLucene2+" AND "+consultaLucene3);
```

Mostramos los resultados de la búsqueda

```
for(var i=0;i<nodeslucene.length;i++)
{
model.salida+="Documento:"+nodeslucene[i].name+"<br>";
}
```

Búsqueda con Xpath

Establecemos la consulta para realizar la búsqueda.

```
var consultaXpath1="//app:company_home//@";
var consultaXpath2="//app:company_home//.";
```

Realizamos la búsqueda.

```
var nodesxpath = search.xpathSearch(consultaXpath1);
```

Mostramos los resultados de la búsqueda

```
for(var i=0;i<nodesxpath.length;i++)
{
model.salida+="Documento:"+nodesxpath[i].name+"<br>";
}
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/81

Gestion de Usuarios con WebScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0082
Tipo de recurso: Ejemplo

Descripción

Código correspondiente al archivo GestionUsuarios.get.desc.xml

```
<webscript>
  <shortname>GestionUsuarios</shortname>
  <description>Operaciones que se puede realizar con un usuarios en Alfresco</description>
  <url>/sample/GestionUsuarios</url>
  <format default="html">argument</format>
  <authentication>guest</authentication>
</webscript>
```

Código correspondiente al archivo GestionUsuarios.get.html.ftl

```
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    ${salida}
  </body>
</html>
```

Código correspondiente al archivo GestionUsuarios.get.js

Declaramos las variables necesarias

```
model.salida="";
var folder = roothome.childByNamePath("Company Home");
var nombregrupo="grupo_prueba";
var groupPrefix="GROUP_";
```

Creamos el grupo

```
people.createGroup(nombregrupo);
```

Movemos un usuario a un grupo espeficico

```
var user="admin";
    var lista=0;
    var srcGrpNode=people.getGroup(groupPrefix+nombregrupo);
    var authority = people.getPerson(user);


    if(authority)
    {
       people.addAuthority(srcGrpNode,authority);
       model.salida+="1.-añadido usuario: \" "+user+" \" al grupo "+ nombregrupo+"<br>";
    }
```

Obteniendo usuarios de un grupo especifico

```
var grupoObtenido = people.getGroup(groupPrefix+nombregrupo);
    lista = people.getMembers(grupoObtenido);

    for(var i=0;i<lista.length;i++)
    {
    model.salida+="usuario "+i+ "es :"+(lista[i]).name+"<br>";
    }
```

Eliminamos al usuario del grupo especifico

```
people.removeAuthority(srcGrpNode,authority);
```

Borramos al grupo

```
people.deleteGroup(srcGrpNode);
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/82

Categorías y Aspectos con WebScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0083
Tipo de recurso: Ejemplo

Descripción

Código correspondiente al archivo GestionCategorias.get.desc.xml

```
<webscript>
  <shortname>GestionCategorias</shortname>
  <description>Operaciones que se puede realizar con categorias en Alfresco</description>
  <url>/sample/GestionCategorias</url>
  <format default="html">argument</format>
  <authentication>guest</authentication>
</webscript>
```

Código correspondiente al archivo GestionCategorias.get.html.ftl

```
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    ${salida}
  </body>
</html>
```

Código correspondiente al archivo GestionCategorias.get.js

Inicializamos las variables necesarias

```
model.salida="";
var espacio="<br>";
var nomFile="prueba.txt";
var folder = roothome.childByNamePath("Company Home");
var documento = roothome.childByNamePath(nomFile);
```

Obtenemos el documento con el cual vamos a trabajar

```
var query="PATH:\"/";
    query= query + folder.qnamePath;
    query= query + "//*\" ";
    query= query + "AND TYPE:\"cm:content\" ";
    var a= folder.qnamePath;
    var nodes = search.luceneSearch( query);
    for(var i=0;i<nodes.length;i++)
    {

        if(nodes[i].name==nomFile)
        {
        var doc=nodes[i];
        }

    }
```

Creamos la categoria

```
classification.createRootCategory("cm:generalclassifiable","categoriatest");
    model.salida+="Categoria creada"+espacio;
```

Añadimos el aspecto al documento para que se pueda clasificar mediante categorías

```
doc.addAspect("cm:generalclassifiable");
```

```
    doc.save();
```

Obtenemos las categorías del documento

```
var catNodeRef = search.luceneSearch("PATH:\"/cm:generalclassifiable//cm:categoriatest\"")[0]  ;
    var cats = doc.properties["cm:categories"];



    if(cats==null)
    {
    cats=new Array();
    }

    model.salida+="el numero de categorias que tiene el documento son :"+cats.length+espacio;
```

Añadimos la categoría a las propiedades del documento

```
var newCats = new Array();
    for (var i=0; i < cats.length; i++)
    {
          newCats[i] = cats[i];
    }

    newCats.push(catNodeRef);
```

Guardamos las categorías en el documento

```
doc.properties["cm:categories"] = newCats;
        doc.save();
```

Quitamos la categoria al documento

```
var catsActuales = doc.properties["cm:categories"];
    var newCatsActuales = new Array();

    model.salida+="el documento tiene "+catsActuales.length+"categorias"+espacio;

    if(catsActuales.length<=1)
    {
    doc.properties["cm:categories"]=newCatsActuales;
    }
    else
    {

        for (var i=0; i < catsActuales.length; i++)
        {
            if(catsActuales[i].properties.name!=catNodeRef.properties.name)
            {
              newCatsActuales.push(catsActuales[i]);
            }
        }
        doc.properties["cm:categories"] = newCatsActuales;

    }
```

Guardamos los cambios en el documento

```
doc.save();
```

Quitamos el aspecto del documento

```
doc.removeAspect("cm:generalclassifiable");
    doc.save();
```

Eliminamos la categoría si exite

```
var categorias = classification.getAllCategoryNodes("cm:generalclassifiable");
     model.salida+="el numero de categorias es :"+categorias.length+espacio;
    for(var i=0;i<categorias.length;i++)
    {
     if(categorias[i].name=="categoriatest")
     {
          categorias[i].removeCategory();
     }
    }
    model.salida+="La categoria ha sido eliminada correctamente"+espacio;
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/83

Reglas y Procedimientos con WebScript

- **Área:** Ejemplos Ampliados de Acceso a Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0084
Tipo de recurso: Ejemplo

Descripción

Código correspondiente al archivo reglas.get.desc.xml

```
<webscript>
    <shortname>Reglas</shortname>
    <description>Accion que envia un Email</description>
    <url>/email</url>
    <format default="text">any</format>
    <authentication>admin</authentication>
    <transaction>required</transaction>
</webscript>
```

Código correspondiente al archivo reglas.get.html.ftl

```
<html>
  <body>
    <img src="/servicios/madeja/%24%7Burl.context%7D/images/logo/AlfrescoLogo32.png" alt="Alfresco" />
    ${salida}
  </body>
</html>
```

Código correspondiente al archivo reglas.get.desc.xml

Iniciamos la variable salida, y la variable NodeRef que sera el nodo principal donde realizaremos las pruebas necesarias.

```
model.salida="";
var NodeRef=roothome.childByNamePath("Company Home");
```

Creamos la acción necesaria para poder enviar un email

```
var mail = actions.create("mail");
```

Introducimos los parametros correspondientes para la acción "mail"

```
mail.parameters.to = "correo@dominio.com";
  mail.parameters.subject = "Envio de Correo desde Webscript";
  mail.parameters.from = "correo@dominio.com";
  mail.parameters.template = roothome.childByNamePath("Company Home/Data Dictionary/Email Templates/notify_user_
  mail.parameters.text = "texto alternativo ,en el caso de no encontrar la plantilla";
```

Ejecutamos la acción

```
mail.execute(NodeRef);
  model.salida+="se ha enviado el correo correctamente";
```

Extension de clases base

- **Área:** Extensión de Alfresco
- **Tipo de pauta:** Directriz
- **Carácter de la pauta:** Obligatoria

Código: PAUT-0026

Para cambiar una funcionalidad implementada en el núcleo(core) de Alfresco la recomendación es nunca sobrescribir la clase que la implementa sino crear una nueva que herede de ésta y sobrecargar los métodos que sean necesarios. Despues a traves de la configuración de Spring haremos que se use nuestra clase a la hora de ejecutar la funcionalidad que hemos sobreescrito.

Como ejemplo vamos a ver como cambiar la autenticación de Alfresco para que ésta se haga de forma automatica en vez de contra la base de datos de Alfresco. Para ello vamos a crear una nueva clase AutenticationAutomaticCustom que extenderá la clase de Alfresco AbstractAuthenticationComponent y sobrecargaremos el método que nos interesa en este caso authenticateImpl().

El código será algo parecido a:

```
public class AutenticacionAutomoticCustom extends AbstractAuthenticationComponent
{
  private static final Log logger = LogFactory.getLog(AutenticacionAutomoticCustom.class);

  /**
  * Implementación de un método de autenticación personalizado y automático
  */
  protected void authenticateImpl(String userName, char[] password) throws AuthenticationException
  {
    try
    {
      logger.debug(" ------> Begin authentication process <------");
     /*Implementación del metodo de Autenticación Automático */
       .
       .
       .
    }
    catch (NamingException e)
    {
      logger.error("Usuario no valido");
      throw new AuthenticationException("Usuario no válido ");
    }
    finally
    {
       .
       .
       .
      logger.debug(" ------> Fin proceso de autenticación ------");
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/pauta/26

Metodo de despliegue de extensiones en Alfresco

- **Área:** Extensión de Alfresco
- **Tipo de pauta:** Directriz
- **Carácter de la pauta:** Recomendada

Código: PAUT-0027

Para un entorno de producción la recomendación es distribuir las extensiones que se desarrollen como modulos AMP instalables en un fichero WAR de Alfresco. Este metodo y el uso de la Herramienta de Gestion de Modulos (MMT) permite controlar los modulos instalados y su versión asi como realizar mas facilmente labores de actualización.

Antes de instalar un modulo amp se recomienda realizar una ejecución del MMT con la opción -preview lo que proporcionará un listado de las modificaciones que se van a realizar. Es necesario tener especial cuidado cuando se va a actualizar un fichero que ya existe. Normalmente debe evitarse sobreescribir un archivo, aunque a veces es inevitable. El MMT realiza una copia de seguridad del fichero actualizado y lo guarda dentro del war para que pueda ser restaurado en caso de actualización del modulo. El problema puede ocurrir cuando varios modulos instalados modifican el mismo fichero. En estos casos si hemos de volver a un estado anterior se tendrá que hacer una recuperación manual.

En entornos de desarrollo donde deben realizarse continuas modificaciones y por lo tanto pequeños despliegues de una extensión, o de parte de ella, es quiza preferible usar alguno de los otros metodos de despliegue que son mas sencillos de realizar.

Si se esta usando Tomcat como servidor de aplicaciones para Alfresco se debería borrar (o mover) los directorios temporales (work y temp) antes de reiniciar Alfresco cuando lo hemos detenido para la instalación de una extensión ya que puede que no funcione correctamente por su causa.

Desarrollo del modelo de contenidos

- **Área:** Extensión de Alfresco
- **Tipo de pauta:** Directriz
- **Carácter de la pauta:** Obligatoria

Código: LIBP-0002

Cuando se esté desarrollando un modelo de contenidos no se debe cambiar el modelo original de Alfresco sino extenderlo con tipos personalizados. Los modelos de contenidos personalizados deben tener definido su propio espacio de nombres para evitar así conflictos entre extensiones distintas del modelo original.

Pautas

Título	Carácter
Definir un tipo de contenido raiz	Obligatoria
Dimensionamiento del modelo	Obligatoria
Uso de aspectos	Obligatoria
Segmentar los espacios	Obligatoria
Implementaciones de clases	Obligatoria

Definir un tipo de contenido raiz

Se debe definir un tipo de contenido raíz del cual hereden todos los demás tipos de nuestro modelo si es que los requerimientos obligan a tener diferentes tipos o solo porque esto proporciona a los gestores de contenidos de un tipo *catch-all* para usar cuando ningún otro tipo pueda ser usado. Se puede usar como tipo base el cm:content.

Volver al índice

Dimensionamiento del modelo

Se debe dimensionar bien el modelo tanto en la profundidad del mismo como en los tipos y propiedades que incluimos en él. Ampliar un modelo será fácil pero reducirlo no. Reducir los modelos puede hacer que se caiga en errores y hacer inaccesibles algunos contenidos ya que su el tipo o sus propiedades no coincidan con la definición que de ellos se hace en el modelo,Se debe tener un plan para realizar cambios en el modelo.

No existe ninguna regla que indique cuál debe ser el nivel máximo de profundidad recomendable para los modelos, pero alcanzar un nivel alto conllevará un rendimiento pobre de la aplicación. Cuando en modelo alcance varios niveles de profundidad por debajo de cm:content se debería realizar una prueba del modelo con una cantidad realista de datos, software y hardware para asegurar que no se esté creando un problema de rendimiento.

Volver al índice

Uso de aspectos

Existen algunos recursos de definición que no deben olvidarse mientras se esta realizando un modelo personalizado. Así si en el modelo dos o más tipos tienen propiedades comunes, y si esas propiedades están siendo utilizadas para describir alguna característica común de alto nivel para todos los tipos, es mejor modelarlas como un aspecto. Puede que merezca la pena definir contenidos que no tengan propiedades ni asociaciones para diferenciar contenidos de otros tipos a efectos de búsquedas o para definir algún comportamiento especial solamente aplicable a instancias de este tipo. Las carpetas son tipos que por tanto pueden ser extendidos como todo lo demás en modelo. No se deben desarrolla extensiones del modelo para realizar clasificaciones de tipos si esta se puede realizar usando categorias.

Volver al índice

Segmentar los espacios

Si el modelo creado es muy extenso se debe segmentar el modelo en múltiples espacios de nombres y múltiples ficheros XML para hacer mas facil su gestión. En cualquier caso los nombres de estos últimos deben ser descriptivos. No se deben tener ficheros llamados *miModelo.xml* o *customModel.xml*.

Volver al índice

Implementaciones de clases

Por último para facilitar el desarrollo, ya que frecuente referirse al *Qname* de estos elementos, es recomendable implementar una clase java que corresponda con cada modelo de contenido que definamos. En cada clase Java se definen constantes que correspondan a los espacios de nombres del modelo, los nombres de los tipos, de las propiedades, de los aspectos, etc.

Volver al índice

Métodos para el despliegue de extensiones de Alfresco

- **Área:** Extensión de Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0049
Tipo de recurso: Manual

Descripción

Extensión básica de Alfresco

Existen varias maneras para realizar un despliegue de una extensión para Alfresco, el método concreto depende de la complejidad del desarrollo y del servidor de aplicaciones que utilicemos. Así, por ejemplo, si la personalización consiste solamente en cambios en fichero de configuración o propiedades bastaría con colocarlos en el classpath del servidor de aplicaciones.

Extensión de Alfresco usando librerías JAR

Las clases java que implementan las nuevas funcionalidades que se desarrollan para Alfresco se deben empaquetar en librerías JAR que se deben poner en el classpath de servidor de aplicaciones. Estas librerías se situarán en el directorio WEB_INF/lib de la aplicación web.

La recomendación es que todo lo que se modifique y/o cree en Alfresco debe empaquetarse en librerías. Los ficheros de configuración que van a sobrescribir los ya existentes en Alfresco se empaquetarán en el directorio META-INF del propio JAR, incluso los que añadan funcionalidades o cualquier cosa al Alfresco original.

En versiones anteriores esto era un problema ya que si se tenían en distintas librerías JAR ficheros con el mismo nombre empaquetados en la misma ruta el Classloader solo era capaz de cargar el primero que encontraba lo que provocaba comportamientos impredecibles. Sin embargo a partir de la versión de Alfresco 2.0 este problema se ha solucionado y el sistema es capaz de cargar todas las extensiones que introduzcamos en los paquetes JAR.

Las modificaciones del tipo nuevas acciones, diálogos, wizards, idiomas que se definen originalmente en web-client-config.xml también se podrían definir en el fichero web-client-config-custom.xml bajo la ruta alfresco/extension empaquetado en el JAR. Así lo podemos ver en el fichero de configuración alfresco/web-client-applicattion-context.xml.

```
<bean id="webClientConfigSource" class="org.alfresco.config.source.UrlConfigSource">
  <constructor-arg>
    <list>
      <value>classpath:alfresco/web-client-config.xml</value>
      <value>classpath:alfresco/web-client-config-dialogs.xml</value>
      <value>classpath:alfresco/web-client-config-wizards.xml</value>
      <value>classpath:alfresco/web-client-config-properties.xml</value>
      <value>classpath:alfresco/web-client-config-navigation.xml</value>
      <value>classpath:alfresco/web-client-config-wcm.xml</value>
      <value>classpath:alfresco/web-client-config-actions.xml</value>
      <value>classpath:alfresco/web-client-config-forum-actions.xml</value>
      <value>classpath:alfresco/web-client-config-wcm-actions.xml</value>
      <value>classpath:alfresco/web-client-config-workflow-actions.xml</value>
      <value>classpath:alfresco/extension/web-client-config-custom.xml</value>
      <value>jar:*!/META-INF/web-client-config-custom.xml</value>
    </list>
  </constructor-arg>
```

Si necesitamos definir algún bean de Spring nuevo para inyectar nuestra funcionalidad, no habrá más que crear un fichero xxx-context.xml y definir ahí los beans necesarios, este fichero XML debe ir empaquetado bajo la ruta alfresco/extension, que es accesible por el sistema tal y como se indica en el bean definido en application-context.xml:

```
<beans>
  <import resource="classpath:alfresco/cache-context.xml" />
  .
  .
  .
  <import resource="classpath*:alfresco/extension/*-context.xml"/>
  <import resource="classpath*:alfresco/extension/dev-context.xml" />
</beans>
```

Extensiones del Cliente Web

La implantación de las modificaciones que se hagan en el cliente web, modificaciones sobre las interfaces de usuario, depende de lo que se quiera personalizar. La implementación de JSF procesa cualquier fichero de configuración disponible siguiendo las siguientes reglas:

1. Primero busca todos los recursos llamados META_INF/faces_config.xml. (ej: Ficheros empaquetados en las librerías JAR).
2. Después comprueba todos los ficheros especificados por el parámetro javax.faces.CONFIG_FILES definido en el web.xml de la aplicación web.
3. Por último procesará, si existe, el fichero de configuración /WEB_INF/faces_config.xml.

Para la reescritura de reglas de navegación JSF se sigue la política de quedarse con la primera definición que encuentre sin embargo para la reescritura de beans de respaldo se usará la última definición encontrada.

Si lo que se quiere es sobrescribir una regla de navegación tendremos que crear un fichero de definición faces_config.xml y empaquetarlo en un paquete .jar. Si por el contrario se quiere sobrescribir un bean el procedimiento es crear un fichero faces_config_custom.xml y copiarlo al directorio WEB_INF de la aplicación web de Alfresco, ya que si leemos el web.xml de Alfresco nos encontramos que:

```
<web-app>
  …
  <context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/faces-config-app.xml,
      [?]/WEB-INF/faces-config-custom.xml,[?]
      /WEB-INF/faces-config-enterprise.xml
    </param-value>
  </context-param>
  …
</web-app>
```

En la siguiente tabla vemos una pequeña guía de la configuración JSF:

Extensión	Fichero y Localización
Sobrescribir reglas de navegación	faces-config.xml en META-INF
Sobrescribir beans de respaldo	faces-config-custom.xml en WEB-INF
Nuevas reglas de navegación	Cualquiera de los dos
Nuevos beans de respaldo	Cualquiera de los dos

Extensiones fuera de librerías JAR

Muchas extensiones también implicarán el desarrollo de ficheros JSP personalizados. Estos ficheros no se pueden colocar dentro de ficheros JAR. Del mismo modo otro tipo de contenidos web como imágenes, ficheros JavaScript u hojas de estilo que, aunque pueden ser extraídos de un JAR usando weblets2, se quiere tener en el sistema de ficheros.

Los JSP personalizados deben colocarse en el directorio jsp/extension de la aplicación web, las imágenes en el directorio images/extension, hojas de estilo en jsp/extension y ficheros JavaScript en scripts/extension.

La forma más sencilla de empaquetar una extensión es construir la estructura de directorios que necesitamos colocar los distintos ficheros donde necesitemos y comprimirlo todo en un fichero ZIP. El fichero tendría una estructura parecida a esta:

```
css
  > extension
    > estilos-personalizados.css
> images
  > extension
    > incono-usuario1.jpg
> jsp
  > extension
    > login-personalizado.jsp
> WEB_INF
  > lib
    > login-personalizado.jar
      >alfresco
        > extension
          > web-client-config-custom.xml
```

```
        > META_INF
          > faces-config.xml
          > MANIFEST.MF
        > es
          > intecna
            > ejemplos
              > LoginBeanPersonlizado.class
> scripts
```

Para hacer el despliegue bastaría con parar el servidor descomprimir el fichero en el raíz de la aplicación Alfresco e iniciar de nuevo el servidor. Este método de despliegue solo es recomendable durante la fase de desarrollo o con extensiones muy pequeñas. Si la implantación se ha hecho por este sistema se vuelve a desplegar el Alfresco WAR en el servidor de aplicaciones web la extensión se borrará.

Por eso para implantaciones en sistemas de producción se recomienda hacer el despliegue mediante ficheros de extensión AMP.

Extensión con Módulos AMP

Siempre es recomendable el uso de módulos de extensión AMP (Alfresco Module Package) para las nuevas funcionalidades. Los módulos de extensión AMP son ficheros que contienen una colección de código, XML, imágenes, CSS, etc… que extienden la funcionalidad proporcionada por el repositorio de Alfresco.

Estos contenidos están distribuidos dentro de una estructura como la siguiente:

```
/
  |
  |- /config
  |
  |- /lib
  |
  |- /licenses
  |
  |- /web
    |
    |- /jsp
    |
    |- /css
    |
    |- /images
    |
    |- /scripts
  |
  |- module.properties
  |
  |- file-mapping.properties
```

Esta no es una estructura obligatoria, puede faltar cualquier directorio o estar vacío. Cada uno de ellos será mapeado dentro de un Alfresco WAR usando la Herramienta de Manejo de Módulos (MMT):

- **/config:** El contenido será mapeado dentro del directorio /WEB-INF/classes en el fichero WAR. En este directorio se situarán los ficheros que definirán la configuración Spring y la del Interfaz de Usuario. En este directorio debe estar también el fichero module-contex.xml con la configuración Spring del modulo.
- **/lib:** El contenido será mapeado en el directorio en /WEB-INF/lib. Deberá contener los ficheros JAR de nuestro modulo.
- **/licenses:** Si el modulo usa algún librería que necesite de una licencia, estas deberán estar situadas en este directorio.
- **/web/jsp:** El contenido será mapeado con el directorio /jsp del fichero WAR. El directorio debe contener todos los JSP nuevos o modificados del módulo.
- **/web/css:** El contenido será mapeado con el directorio /css del fichero WAR. El directorio debe contener las hojas de estilo de nuestro módulo.
- **/web/images:** El contenido será mapeado en el directorio /images del WAR. Contendrá las imágenes que use nuestro módulo.
- **/web/scripts:** Los contenidos serán mapeados en el directorio /scripts del WAR. Los ficheros con código JavaScript que use el módulo deberán ser puestos aquí.

Si no se indica lo contrario se creará una copia del WAR antes de procesar el fichero AMP. Cualquier estructura de carpetas que

se encuentre en estos directorios será mapeada de manera idéntica dentro de los directorios del WAR. Si alguno de los ficheros que estamos mapeando se encuentran ya en el WAR será sobrescrito.

El fichero **module.properties** debe estar obligatoriamente en el AMP. Contiene los meta-datos del módulo, lo más importante son el id y versión del módulo que estamos empaquetando.

El fichero **file-mapping.properties** no es obligatorio, nos servirá para cambiar el mapeo por defecto y usar uno personalizado. Para ello daremos una serie de pares llave valor con los nombres de los directorios del AMP y su destino en el WAR.

Herramienta de Gestion de Modulos (MMT)

La Herramiente de gestión de modulos (MMT) ayuda ha manejar los modulos AMP instalados en un fichero Alfresco WAR estandar. La herramienta de gestión de modulos soporta: instalación de modulos AMP incluyendo actualizaciones a versiones mas recientes, habilitación y deshabilitación de modulos instalados, desinstalación de modulos instalados y listado de modulos actualmente instalados.

Los Modulos AMP son empaquetados e instalados como ficheros AMP. Un Fichero AMP hace referencia a un modulo y versión especifica. Durante el proceso de instalación, estos datos son tomados en consideración.

Ejecución

Desde la v2.1 de Alfresco el mmt se distribuye separadamente como un jar ejecutable. Se puede descargar desde el área de descargas de Alfresco (alfresco-mmt-2.1.jar). Es compatible con ficheros war de versión 2.0 o superior. La sintaxis de ejecución es:

```
java -jar alfresco-mmt-2.1.jar [args]
```

Comandos MMT

Para acceder a las diferentes funcionalidades que ofrece em mmt deberemos usar en la llamda uno de los siguientes comandos.

- **install:** Para la instalación de módulos usaremos el siguiente comando:

```
uso: install <LocalizaciondelAMP> <LocalizaciondelWAR> [options]
opciones validas:
  -verbose   : habilitar la salida
  -directory : indica que la localizacion del fichero amp especificado es un
               directorio. Todos los ficheros amp encontrados en el directorio
               y sus subdirectorios serán instalados.
  -force     : fuerza la instalación del AMP independientemente de la versión
               que ya este instalada
  -preview   : vista preliminar de la instalación del AMP sin modificar el WAR
  -nobackup : indica que no se debe hacer backup del WAR
```

Este comando instala los ficheros encontrados en el fichero AMP dentro del Alfresco WAR, actualizándolo si una versión antigua ya está instalada. Al realizar esta actualización los ficheros instalados por la versión antigua serán borrados del war y reemplazados por los de la nueva.Si la versión del modulo es igual o menor que la versión que ya está instalada la instalación se cancelará, a menos que se especifique la opción -force. Antes de que un Modulo AMP sea instalado en un WAR se realiza una copia del original y se guarda en el mismo directorio. Especificando la opción -nobackup se evita esta copia.

Así la sintaxis completa para la instalación de un modulo será:

```
java -jar alfresco-mmt-2.1.jar install <FicAMP> <FicWAR> -force
```

- **list:**

```
uso: list <FicWAR>
```

Lista los detalles sobre todos los modulos actualmente instalados en el fichero WAR especificado.

- **disable:** (Este comando no esta disponible en el realease 2.1)

```
uso: disable <moduleId> <FicWAR>
```

- **enable:** (Este comando no esta disponible en el realease 2.1)

```
usage: enable <moduleId> <FicWAR>
```

- **uninstall:** (Este comando no esta disponible en el realease 2.1)

```
uso: uninstall<moduleId> <FicWAR>
```

Source URL: http://127.0.0.1/servicios/madeja/contenido/recurso/49

Extensiones en el modelo de contenidos

- **Área:** Extensión de Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0050
Tipo de recurso: Manual

Descripción

El primer paso en el proceso de diseño de un sistema personalizado es crear el modelo de contenido que necesitamos para describir la información que se guardará en el repositorio. Vamos a ver qué elementos tendremos que usar y como extenderlos.

Conceptos básicos del modelado

El modelo de contenido es una parte importante de Alfresco, sin ella este sería poco más que un sistema de ficheros. Veamos alguna de la información básica que el modelo de contenido proporciona a Alfresco:

- Los tipos de datos fundamentales y como esos datos serán persistentes en la base de datos. Sin el modelo de contenido Alfresco no conocería la diferencia entre una cadena y una fecha.
- Los tipos de datos de alto nivel como *content* y *folder* así como los tipos de contenido personalizados como Contrato.
- Los aspectos originales de Alfresco como *auditable* o *clasiflicable* y los aspectos personalizado como *puntuable* o *comentable*
- Propiedades(o metadatos) propios a cada tipo de contenido.
- Restricciones de cada propiedad, como que los valores de cierta propiedad deban coincidir con cierto patrón o que deban de provenir de una lista de posibles valores.
- Como indexar los contenidos para búsquedas.
- Relaciones entre los tipos de contenidos.

Los modelos de contenido de Alfresco están construidos a partir de un conjunto de conceptos que debemos conocer: los Tipos, las Propiedades, los tipos de Propiedades, las Restricciones, las Asociaciones y los Aspectos.

Tipos

Se podrían equiparar a los tipos o las clases en el mundo de la orientación a objetos. Pueden ser usados como objetos del modelo de negocio. Tienen propiedades y pueden heredarlas de sus padres. Importantes ejemplos de los tipos definidos en origen en Alfresco son *Contenido* o *Persona*. Los tipos personalizados solo están limitados por la imaginación y los requerimientos de negocio, por ejemplo *Informe de Ventas, Historial Medico*.

Los tipos, las propiedades, las restricciones, las asociaciones y los aspectos tienen nombre que se hacen únicos en el repositorio usando un espacio de nombres específico para cada modelo. Los espacios de nombres tienen una abreviatura que siempre se colocará delante de cualquier tipo definido como parte del modelo al que ese espacio de nombres está asignado. El uso de espacios de nombres ayuda a prevenir colisiones de nombres cuando se comparten modelos entre repositorios.

Propiedades

Las propiedades son metadatos asociados con un tipo particular. Por ejemplo un Informe de gastos podría incluir propiedades como *Nombre de empleado, Fecha de envio, Projecto, Cliente, Numero de Informe de Gastos, Cantidad total* o *Moneda*. También podría tener una propiedad de contenido donde se guardara el informe de venta real (quizá sea un documento en pdf o una hoja de cálculo excel).

Tipos de propiedades

Describen los tipos fundamentales de datos que el repositorio utilizara para almacenar las propiedades (cadenas, fechas, enteros, flotantes, booleanos).

Restricciones

Las restricciones pueden ser definidas para restringir los valores que Alfresco puede almacenar en una propiedad. Existen cuatro tipos de restricciones disponibles:

- REGEX: Se usa para asegurar que el valor de una propiedad concuerda con un patrón definido por una expresión regular.
- LIST: Se usa para definir una lista de valores validos para una propiedad.
- MINMAX: Se usa para definir un rango numérico para los valores de una propiedad.
- LENGHT: Establece una restricción de longitud para una cadena.

Las restricciones pueden ser definidas una vez y reutilizadas a lo largo del modelo. Por ejemplo el modelo original de Alfresco tiene definida una restricción llamada ?cm:filename? que define una restricción de expresión regular para identificar un nombre de fichero. Si se necesita en nuestro modelo una restricción de este tipo no hace falta volverla a definir solo hacer referencia a esta.

Asociaciones

Con este elemento podemos definir relaciones entre los tipos. Podríamos querer que en nuestro modelo además de un *Informe de Gastos* se pudieran representar cada una de los gastos que contenga de manera individual. Así además del tipo *Informe de Gastos* tendríamos el tipo *Gastos* y usando asociaciones podríamos describir a Alfresco la relación entre un *Informe de Gastos* y una o más *Gastos*.

Existen dos tipos de Asociaciones: Asociaciones entre pares (Assotiations) y Asociaciones padre-hijo (Child Assotiations). Las asociaciones entre pares definen la relación entre dos objetos en la que no existe subordinación entre uno y otro. Las asociaciones padre-hijo, por otro lado, se usan cunado el objetivo de la asociación (el hijo) no debe existir cuando se borra la fuente (el padre). Funciona como la eliminación en cascada en las bases de datos relacionales, si eliminamos al padre eliminamos al hijo.

Como ejemplo podemos ver la asociación original de Alfresco cm:contains que define la asociación padre-hijo entre las carpetas (*cm:folder)* y todos los otros objetos (instancias de *sys:base* y sus tipos hijos). Si tenemos por ejemplo un carpeta llamada *Recursos Humanos* tendrá una asociación *cm:contains* entre ella y sus inmediatos hijos, que podrían ser tipos personalizados como Currículum o Evaluación de Rendimiento.

Aspectos

Antes de hablar de los aspectos es necesario que revisemos como actúa la herencia y la implicación que tiene en el modelo de contenido. Supongamos que utilizamos Alfresco para gestionar contenidos que van a ser mostrados en un portal web y solo un subconjunto del contenido en el repositorio sera subido a ella. Ademas, cuando el contenido se suba necesitamos guardar una serie de información, por ejemplo la fecha y hora de subida.

Usando los conceptos que se han visto hasta ahora solo tenemos dos opciones. Podríamos definir un tipo raiz con una propiedad *fecha_de_publicacion* para que todo el resto de tipos la hereden y así estuviera disponible para todo el mundo o, por otra parte, podríamos poner la propiedad de fecha de publicación solo en los tipos de contenido que sepamos que van a ser publicados en el portal.

Las dos presentan bastantes problemas. En la primera vamos a terminar arrastrando en todos y cada uno de los contenidos una propiedad que probablemente nunca se vaya a utilizar, lo que generara problemas de mantenimiento y rendimiento. La segunda no es mucho mejor. Para empezar estamos asumiendo que se sabe a ciencia cierta cuales son los tipos de contenidos que van a ser publicados por adelantado. Ademas corremos el riesgo de que el mismo tipo de metadato se defina de forma distinta en distintos tipos de contenido. Por último no proporciona un sistema fácil para encapsular el comportamiento o la lógica de negocio que queramos unir a la fecha de publicación.

Hay una tercera opción que es el uso de los aspectos. Los aspectos permiten ampliar cualquier tipo de contenido de nuestro modelo (o incluso a instancias concretas) con propiedades y asociaciones con solo ser adjuntadas a estos donde y cuando se necesiten.

En el ejemplo del portal web podríamos definir un aspecto *Visible en portal* con una propiedad fecha de publicación. Este aspecto podrá ser añadido a cualquier contenido que se necesite mostrar en el portal independientemente del tipo.

Son un concepto fundamental para el modelado de contenido en Alfresco. Los aspectos permiten añadir funcionalidad a tipos de contenidos ya existentes. Pueden tener propiedades que se unen a los de los tipos de contenidos cuando se aplican a ellos. También comportamientos y flujos de trabajo pueden ser ligados a los aspectos.

Comportamientos personalizados

Es posible que se necesite tener asociados a los aspectos o tipos personalizados ciertos comportamientos o cierta lógica de negocio. Así por ejemplo cada vez que se compruebe un *Informe de Gastos* es posible que se quiera recalcular el total iterando por todos los *Gastos* que tenga asociados. Una opción puede ser incorporar esta lógica dentro de reglas o acciones en el cliente web de Alfresco o un aplicación personalizada. Sin embargo existen algunos comportamientos son tan fundamentales para un tipo o aspecto que deben ser asociados directamente a ellos.

Creación de un modelo personalizado

Ya conocemos los conceptos básicos de los modelos de contenido y algunas recomendaciones que deberemos tener en cuenta a la hora de realizar cualquier extensión. Veamos ahora que debemos hacer para crear nuestro modelo personalizado.

Creación de un nuevo modelo

Se necesita crear un fichero Xml y situarlo en un directorio accesible a través del classpath:

- Tomcat: TOMCAT_HOME/shared/classes/alfresco/extension.
- JBoss : JBOSS_HOME/server/default/conf/alfresco/extension.

En el fichero debemos proporcionar la descripción del modelo. Las descripciones XML de modelos de contenido deben tener el formato representado en el esquema XML del diccionario de datos2. Las partes mas importantes de esta definición serán el nombre y el namespace.

La definición del modelo se inicia con los elementos description, author, published y version que nos sirven para presentar el contenido del mismo. Estos elementos no son obligatorios según el esquema pero es conveniente rellenarlos para que al leer el documento tengamos una primera idea de lo que vamos a ver.

Lo que si es obligatorio es la definición del namespace para nuestro modelo, así como su prefijo. Esto lo haremos con la elemento namespaces. Un modelo puede definir el numero de espacios que desee.

En el siguiente trozo de código encontramos la definición XML de un modelo de ejemplo:

```
<model name="mi:mimodelo" xmlns="http://www.alfresco.org/model/dictionary/1.0">
  <description>Ejemplo de Modelo personalizado</description>
  <author>Junta de Andalucia</author>
  <published>2007-06-03</published>
  <version>1.0</version>
  <!-- Los imports (referencias a otros modelos)  iran aqui -->
  <namespaces>
    <!-- Define un Namespace para mis definiciones-->
    <namespace uri="mi.nuevo.modelo" prefix="mi"/>
  </namespaces>
  <!-- Las definiciones de Tipos y Aspectos iran aqui -->
</model>
```

Creación de un nuevo tipo de contenido

Un modelo puede referirse a otros para usar los elementos definidos en ellos. Para hacer esto basta con importar sus namespaces.

Lo primero que debemos hacer para definir nuestro tipo de contenido es declarar todos los modelos a cuyas definiciones se quiera referir. Para hacerlo se deben añadir el elemento imports, justo antes del elemento namespace.

Con el siguiente ejemplo ejemplo estaríamos permitiendo que se puedan usar los elementos definidos en el Modelo de Contenidos de Alfresco y en el modelo de diccionario (tipos fundamentales):

```
...
    <imports>
      <!-- Importar definiciones del Diccionario de Alfresco -->
      <import uri="http://www.alfresco.org/model/dictionary/1.0" prefix="d"/>
      <!-- Importar  definiciones del Modelo de Contenido de Alfresco   -->
      <import uri="http://www.alfresco.org/model/content/1.0" prefix="cm"/>
    </imports>
    <namespaces>
      ...
    <!-- Definiciones de tipos y aspectos aqui -->
  </model>
```

El siguiente paso es escribir la definición de nuestro nuevo tipo de contenido usando el elemento type. Podemos definir tantos tipos como queramos para eso las definiciones de tipos serán una secuencia dentro del elemento types.

```
...
  <!-- Definiciones de Tipos y  Aspectos  aqui -->
  <types>
    <type name="mi:sop">
      <title>Procedimiento Estandar</title>
      <parent>cm:content</parent>
      <properties>
        <property name="mi:fechaPubicacion">
          <type>d:datetime</type>
        </property>
        <property name="mi:autorizadoPor">
          <type>d:text</type>
        </property>
      </properties>
      <mandatory-aspects>
        <aspect>cm:auditable</aspect>
      </mandatory-aspects>
    </type>
  </types>
</model>
```

El elemento type tiene un atributo obligatorio llamado name que representará el nombre con el que se identificará el tipo en nuestro modelo.

La definición comenzara con una descripción del tipo compuesta por los elementos title y description, es recomendable poner al menos el primero.

Tras la descripción usaremos el elemento parent para definir de que va a heredar nuestro tipo. Se recomienda siempre crear un tipo raíz para todo nuestro modelo o usar cm:content como tipo raiz, como se hace ene el ejemplo.

Ahora debemos definir las propiedades de nuestro tipo, si es que las tiene. Para ello usaremos una secuencia de elementos property dentro de un elemento properties. Para cada una de ellas se deberá definir al menos el atributo name que nos da el nombre con el que nos referiremos a esta propiedad en nuestro modelo y el elemento type, que nos va a indicar el tipo de dato básico del que es la propiedad. Ademas para cada una de las propiedades se puede indicar si será obligatoria (elemento mandatory), si tiene un valor por defecto (elemento default), si su valor puede ser alterado, (elemento protected), si puede tener múltiples valores (elemento multiple).

También es posible describir como se indexa cada propiedad usando el elemento index.* Usando el atributo Enabled podemos hacer que la propiedad no tenga entrada en los indices. Por defecto una propiedad son indexadas atómicamente (ej. sincronizada con la inserción del contenido en el repositorio), el valor de la propiedad no es almacenado en el indice y la propiedad es convertida en tokens3 antes de ser indexada. Dentro de index podemos incluir otros elementos para cambiar esta definición. Con el elemento atomic se selecciona si la indexación será atomica o se realizara en segundo plano. El elemento stored si el valor de la propiedad se guardará en el indice y podrá ser obtenido a través de la API de consultas Lucene de bajo nivel. Por último el elemento tokenised definirá si el valor de cadena de la propiedad sera convertido en tokens o no antes de ser indexado. El tipo de división en tokens depende del tipo de propiedad que sea.

Como se verá mas adelante con mas detalle también es posible para cada propiedad hacer referencia a una restricción definida en nuestro modelo o crear una nueva.

Con esto se terminaría la definición de propiedades. Lo siguiente que debemos incorporar a la definición de un tipo son las asociaciones. La definición de asociaciones se verá mas adelante con mas detalle.

También es posible sobrescribir la obligatoriedad (mandatory) y el valor por defecto (default) de una propiedad heredada usando el elemento overrides.

Para finalizar con la creación de nuestro se puede hacer uso del elemento mandatory-aspect, con él podemos definir una lista de aspectos que serán aplicados automáticamente cuando se creá un contenido del tipo.

Definición de asociaciones

Las asociaciones deben declararse dentro de la definición del tipo después de la de las propiedades o el modelo no se iniciara correctamente. Para hacerlo usaremos el elemento associations.

Dentro de este elemento pondremos todas las asociaciones entre pares y padre e hijo que pueda tener nuestro tipo. En el caso de las asociaciones entre pares usaremos el elemento association:

```
<associations>
        <association name="cm:subscribedBy">
          <source>
            <mandatory>false</mandatory>
            <many>true</many>
          </source>
          <target>
            <class>cm:person</class>
            <mandatory>false</mandatory>
            <many>true</many>
          </target>
        </association>
</associations>
```

La definición puede comenzar con una descripción textual de la asociación, usando los elementos title y description, aunque no es obligatorio.

Los dos siguientes elementos source y target nos van a servir para definir el rol del origen y el destino de la asociación respectivamente. Se podrá definir la cardinalidad de la asociación usando los elementos mandatory y many y seleccionar la clase (tipo de contenido o aspecto) del destino, con el elemento class que solo puede estar presente en la definición del objetivo.

Para las asociaciones padre-hijo usaremos el elemento child-association. Las características de padre e hijo se definirán de la misma forma que en las asociaciones de pares (el padre es el origen/source y el hijo el destino/target). Cada hijo recibirá un nombre dentro del padre y es posible definir si este nombre será único o no dentro de él (duplicate). Se puede navegar fácilmente por las asociaciones padre-hijo usando los lenguajes de consulta Xpath y Lucene.

Podemos suponer que en un modelo hemos definido dos asociaciones padre-hijo distintas. Al insertar los datos en el repositorio hemos seleccionado un mismo contenido como hijo en dos instancias de las distintas asociaciones. En este caso solo la primera que se ha definido (la primaria) implicará comportamientos en cascada (borrado o copiado).

```
<type name="cm:folder">
        <title>Folder</title>
        <parent>cm:cmobject</parent>
```

```
      <associations>
        <child-association name="cm:contains">
          <source>
            <mandatory>false</mandatory>
            <many>false</many>
          </source>
          <target>
            <class>sys:base</class>
            <mandatory>false</mandatory>
            <many>true</many>
          </target>
          <duplicate>false</duplicate>
        </child-association>
      </associations>
</type>
```

Definición de Restricciones

Las propiedades en el modelo pueden tener asociadas restricciones que se definirán como parte del modelo. Para hacerlo se usa la etiqueta constrains justo antes de iniciar la declaración de tipos:

```
<model name="mi:mimodelo" xmlns="http://www.alfresco.org/model/dictionary/1.0">
…
  <imports>….  </imports>
  <namespaces>…  </namespaces>
  <constraints>
    <constraint name="mi:regex1" type="REGEX">
      <parameter name="expression"><value>[A-Z]*</value></parameter>
      <parameter name="requiresMatch"><value>false</value></parameter>
    </constraint>
    <constraint name="mi:stringLength1" type="LENGTH">
      <parameter name="minLength"><value>0</value></parameter>
      <parameter name="maxLength"><value>256</value></parameter>
    </constraint>
    <constraint name="mi:minMax1" type="MINMAX">
      <parameter name="minValue"><value>0</value></parameter>
      <parameter name="maxValue"><value>256</value></parameter>
    </constraint>
    <constraint name="mi:list1" type="LIST">
      <parameter name="allowedValues">
        <list>
          <value>ABC</value>
          <value>DEF</value>
        </list>
      </parameter>
      <parameter name="caseSensitive"><value>true</value></parameter>
    </constraint>
  </constraints>
```

Para cada restricción que queramos definir escribiremos un elemento constraint, que tienen dos argumentos name con el nombre de la restricción y type con el tipo de restricción que queremos definir. Se usará el elemento parameter para indicar las características de la restricción.

- Tipo REGEX: Usaremos dos elementos parameter para configurarlo, una de nombre expresion donde definiremos la expresión regular con la que vamos a comparar y requiresMatch con el que indicaremos si el valor debe encajar con la expresión o por el contrario lo que se busca es que no lo haga.
- Tipo MINMAX: Debemos definir los valores mínimo y máximo del rango usando los elementos parameter de nombre minValue y maxValue.
- Tipo LENGTH: Debemos definir los valores mínimo y máximo para la longitud de la cadena usando los elementos parameter de nombre minLength y maxLength.
- Tipo LIST: Debemos dar la lista de valores permitidos con elemento parameter de nombre allowedValues e indicar si la comparación sera o no sensible a mayusculas, elemento parameter de nombre caseSensitive.

Estas restricciones pueden ser referenciadas en la definición de las propiedades:

```
<property name="mi:prop1">
          <type>d:text</type>
          <protected>true</protected>
          <default></default>
          <constraints>
            <constraint ref="mi:regex1"/>
            <constraint ref="mi:stringLength1"/>
          </constraints>
        </property>
```

También sería posible definir las restricciones dentro de la definición de la propiedad pero esto haría que no se pudieran reutilizar en el resto del modelo, por eso se recomienda hacer la definición por separado:

```
<property name="mi:prop1">
          <type>d:text</type>
          <protected>true</protected>
          <default></default>
          <constraints>
            <constraint ref="mi:regex1"/>
            <constraint type="LENGTH">
              <parameter name="minLength"><value>0</value></parameter>
              <parameter name="maxLength"><value>128</value></parameter>
            </constraint>
          </constraints>
</property>
```

La implementación de las restricciones originales está en el paquete org.alfresco.repo.-dictionary.constraint y son:

- REGEX org.alfresco.repo.dictionary.constraint.RegexConstraint
- LENGTH org.alfresco.repo.dictionary.constraint.StringLengthConstraint
- MINMAX org.alfresco.repo.dictionary.constraint.NumericRangeConstraint
- LIST org.alfresco.repo.dictionary.constraint.ListOfValuesConstraint

Los elementos parameter se corresponden con métodos de asignación de las clases. Cuando definimos un tipo derivado y sobrescribimos una propiedad, la restricción que este asociada a esa propiedad también se puede sobrescribir.

Se pueden añadir restricciones adicionales ademas de las que están precompiladas. Para hacerlo solo se necesita implementar la interfaz org.alfresco.service.cmr.dictionary.-Constraint. Normalmente las restricciones deben extender la clase org.alfresco.repo.- dictionary.constraint.AbstractConstraint.

Para definir una de estas restricciones adicionales dentro del modelo, solo tenemos que poner el nombre de clase que hemos implementado completo en el atributo type. Veamos por ejemplo de definición de una restricción que solo permite números múltiplos de uno dado:

```
<constraint name="mi:FactoresDe" type="es.soporte.intecna.constraints.MultiplosDeConstraint">
      <parameter name="multiploDe"><value>10</value></parameter>
   </constraint>
```

Creación de un nuevo aspecto

El siguiente paso de la creación de nuestro modelo es definir nuestro aspectos personalizados. La definición de aspectos es muy similar a la de tipos, solo que en vez de los elementos types y type usaremos los elementos aspects y aspect y que no es posible usar el elemento mandatory-aspect, todo el resto de consideracionos son iguales.

Veamos un ejemplo de definición de aspecto:

```
</types>
<aspects>
    <aspect name="mi:imagen">
      <title>Image Classification</title>
      <properties>
        <property name="mi:ancho">
          <type>d:int</type>
        </property>
        <property name="mi:alto">
          <type>d:int</type>
```

```
        </property>
        <property name="mi:resolucion">
           <type>d:int</type>
         </property>
      </properties>
   </aspect>
</aspects>
```

Registro del modelo

Ahora que se tienen una definición Xml de un modelo personalizado el siguiente objetivo es registrarlo en el Repositorio. Cuando el repositorio se inicia el Diccionario de Datos lee todos los modelos de contenido registrados y los compila para ser usados en tiempo de ejecución. Cuando esto ocurre el Repositorio puede empezar a manejar contenido como es descrito por esos modelos.

El Diccionario de datos lee en el arranque del sistema usando un componente llamado Dictionary Bootstrap, el cual es informado de los modelos que tienen que ser cargados por la siguiente configuración de Spring:

```
<bean id="misModelos" parent="dictionaryModelBootstrap">
      <property name="models">
         <list>
            <value>[path del fichero con la definición del modelo]</value>
         </list>
      </property>
   </bean>
```

Todos los modelos originales de Alfresco son registrados en un bean llamado dictionaryBootstrap en el fichero de configuración core-service-contex.xml.

Para registrar los modelos personalizados se ha de crear un fichero Xml para definir una configuración de Spring con una estructurá como esta:

```
<beans default-lazy-init="false" default-autowire="no" default-dependency-check="none">
   <bean id="extension.dictionaryBootstrap" parent="dictionaryModelBootstrap" depends-on="dictionaryBootstrap">
      <property   name="models">
         <list>
            <value>my/modeloPersonalizado1.xml</value>
            <value>my/modeloPersonalizado2.xml</value>
   ........
         </list>
      </property>
   </bean>
</beans>
```

indicando el path de los fichero con la definición de los modelos que se han realizado. Este fichero debe ser nombrado xxx-context.xml (xxx puede ser la cadena que se quiera) y ser situado en el directorio de extensiones de Alfresco (alfresco/extension).

La recomendación es usar ficheros de módulos AMP para hacer esta inserción.

Tenemos dos medios básicos para comprobar que la semántica y la sintaxis del modelo son correctas:

- Hacer los cambios en el proyecto y reiniciar el servidor. Al iniciarse el servidor los modelos son leídos y validados. Si existe algún error se informa de ello y deben ser arreglados para que el servidor se inicie correctamente, así que editar, desplegar y volver a iniciar el servidor.
- Ejecutar la aplicación Java org.alfresco.repo.dictionary.TestModel. Esta comprobación incluye una pequeña inicialización que es muy rápida en su ejecución lo que permite que los ciclos de edición y comprobación sean también mas rápidos. Sintaxis: org.alfresco.repo.dictionary.TestModel [localizacion del modelo]. Ejemplo:

```
org.alfresco.repo.dictionary.TestModel alfresco/extension/exampleModel.xm
   Testing dictionary model definitions...
      alfresco/model/dictionaryModel.xml
      alfresco/model/systemModel.xml
       alfresco/model/contentModel.xml
      alfresco/model/applicationModel.xml
      alfresco/extension/exampleModel.xml
    Models are valid.
```

Mostrar un modelo personalizado en el Cliente Web

La interfaz de usuario del cliente web de Alfresco puede ser configurada para mostrar los tipos y aspectos personalizados. El interfaz de usuario esta configurado en el fichero web-client-config.xml que esta compuesto por numerosos elementos config. Cada uno de ellos tienen un evaluator(evaluador) y un condition(condicion). Extender el cliente web es solo cuestión de saber que pareja evaluator/condition debemos usar.

Para hacer la personalización del cliente web se debe crear un fichero web-client-config-custom-xml, en el directorio de extensión, con un elemento raíz llamado alfresco-config y añadir los elementos config apropiados como sus hijos.

Se necesitará mostrar el modelo de contenidos personalizados en los siguientes lugares:

- **Hoja de propiedades:** Cuando un usuario mira los detalles de un documento almacenado como un tipo personalizado o que tiene asociado un aspecto personalizado en la hoja de propiedades deberán aparecer todas las propiedades personalizadas de los mismos.

Para añadir propiedades a la hoja debemos usar el evaluador aspect-name para los aspectos y node-type para contenidos.

```
<config evaluator="node-type" condition="mi:sop">
    <property-sheet>
      <show-property name="name" show-in-edit-mode="false" />
      <show-property name="mimetype" display-label-id="mimetype"
                converter="org.alfresco.faces.MimeTypeConverter"
                show-in-edit-mode="false" />
      <show-property name="title" show-in-edit-mode="false" />
      <show-property name="description" show-in-edit-mode="false" />
      <show-property name="size" display-label-id="size"
                converter="org.alfresco.faces.ByteSizeConverter"
                show-in-edit-mode="false" />
      <show-property name="mi:fechaPublicación" />
      <show-association name="mi:signOff" />
      <show-property name="mi:autorizadoPor"/>
      <show-child-association name="mi:pasoProceso" />
    </property-sheet>
  </config>
  <config evaluator="aspect-name" condition="mi:Imagen">
    <property-sheet>
      <show-property name="mi:alto" display-label-id="altura"/>
      <show-property name="mi:ancho" display-label="anchura"/>
      <show-property name="mi:resolucion"/>
    </property-sheet>
  </config>
```

Como se puede ver en el ejemplo se deben usar los elementos show-property, show-association y show-child-association para indicar los elementos del modelo personalizado que queremos mostrar. Existen tres formas para especificar la etiqueta que se muestra para un propiedad en la hoja. Si no especificamos nada la etiqueta mostrada se tomara del elemento title que para la propiedad fue definido en el modelo. Otra opción es sobrescribir este usando el atributo display-label del elemento show-property. Sin embargo una buena practica es usar externalizar las cadenas y usar el atributo display-label-id para especificar la llave de búsqueda en ficheros de recursos, usando el modelo de internacionalizacion. Las propiedades que se definen para mostrarse en la hoja de propiedades aparecen automáticamente cuando se selecciona el modo de edición. Para evitar que una variable aparezca en la vista de edición de propiedades se ha de añadir en el elemento show-property el atributo show-in-edit-mode con un valor false. También es posible realizar la configuración contraria que la propiedad solo se muestre en el modo de edición. Para ello usaremos el atributo show-in-edit-mode con un valor false.

- **Crear/Añadir contenido:** Cuando un usuario accede al asistente de creación o inserción de contenido los tipos personalizados deben ser una opción en la lista de tipos disponibles.

Para añadir tipos de contenido a la lista de tipos disponibles en los diálogos de de crear y añadir contenidos se debe usar el evaluador *string-compare* y la condición *Content Wizards*.

```
<config evaluator="string-compare" condition="Content Wizards">
    <content-types>
      <type name="mi:sop" [display-label="SOP"] [display-label-id="sop"] />
    </content-types>
  </config>
```

Los atributos opcionales display-label y display-label-id nos servirán para especificar la etiqueta que aparecerá en la lista de selección de la misma forma que para las etiquetas de propiedades y asociaciones.

- **Criterio en reglas, acción de especialización de tipo y acción de añadir aspecto:** Cuando un usuario configura una regla en un espacio y usa el tipo de contenido o un aspecto como criterio o ejecuta una acción relativa a tipos o aspectos es posible que se quiera tener disponibles los personalizados para ser seleccionados.

Todas estas personalizaciones se deben realizar todo en el mismo elemento config. Usaremos el evaluador *string-compare* y la condición *Action Wizards*.

```
<config evaluator="stringcompare" condition="Action Wizards">
    <aspects>
       <aspect name="mi:imagen"/>
    </aspects>
  <subtypes>
     <type name="mi:sop" />
  </subtypes>
  <specialisetypes>
     <type name="mi:sop" />
  </specialisetypes>
</config>
```

El elemento aspects define la lista de aspectos que se muestran cuando se configura la acción Añadir Aspecto. El elemento subtypes sirve para listar los tipos que se mostrarán en el desplegable cuando se esta configurando el tipo de contenido para la regla. El elemento specialise-types se utiliza para listar los tipos disponibles para la acción de especialización de tipo.

- **Búsqueda avanzada:** Cuando un usuario utiliza la búsqueda avanzada debe ser capaz de restringir los resultados a instancias de tipos personalizados y/o a valores para las propiedades de estos.

Vamos a utilizar el elemento config con el evaluador *string-compare* y la condición *Advanced Search*.

```
<config evaluator="string-compare" condition="Advanced Search">
  <advanced-search>
    <content-types>
      <type name="mi:sop" />
    </content-types>
    <folder-types>
    </folder-types>
    <custom-properties>
      <meta-data type="mi:sop" property="mi:authorisedBy" />
      <meta-data aspect="mi:imagen" property="mi:resolucion" />
    </custom-properties>
  </advanced-search>
</config>
```

Podremos configurar la búsqueda por cualquier tipo que herede de cm:content o cm:folder. Para ello añadiremos elementos hijos types al elemento content-types. La etiqueta que se verá en la lista desplegable será obtenida buscando el titulo del tipo en la definición del modelo.

Podremos añadir a la búsqueda avanzada cualquier propiedad validad del modelo. Para ello usaremos el elemento hijo meta-data dentro de la sección custom-properties. Cada elemento hijo añade una propiedad a la búsqueda avanzada, aparecerán en el orden en el que están definidas en el fichero de configuración. El atributo type debe ser el tipo del modelo que contiene la propiedad y el atributo property especifica la propiedad a usar. En vez de un tipo también es posible usar aspectos para ello usaremos el atributo aspect en lugar del atributo type.

La etiqueta asociada se cogerá también de la definición del modelo en el repositorio, aunque también esta disponible el atributo display-label-id para usar externalización de cadenas .

El interfaz de usuario se encargará de pintar el control adecuado según el tipo de la propiedad que queramos incluir en la búsqueda avanzada.

Categorías y clasificaciones

Las categorías sirven para realizar una clasificación de los elementos (espacios o contenidos) aparte de la propia que se establece por la organización del repositorio con el árbol de espacios. Las categorías permiten añadir un meta-dato más al contenido con independencia de su tipo. Esto permitirá clasificar el contenido de forma global y hacer búsquedas más restrictivas y con mejores resultados.

El uso de la navegación por categorías es recomendado cuando los usuarios no tienen una idea concreta de lo que estan buscando, por ello la navegación por categorías les permite ir filtrando la documentación que quieren recuperar. Ejemplo, los que buscan ocasionalmente documentación en un sitio web.

Clasificaciones

El repositorio de Alfresco es capaz de manejar múltiples clasificaciones, o jerarquías de clasificación, aunque las pantallas de Gestión de Categorías del interfaz de usuario solo muestre una de ellas, la que este asociada con el aspecto cm:generalclassifiable.

Un nodo puede ser clasificado de muchas maneras, cada una de las cuales es aplicada añadiendole un aspecto. Cada aspecto de clasificación debe ser definido en el diccionario de datos y debe tener en su definición una propiedad del tipo d:category. Si esta propiedad soporta valores múltiples se podrán aplicar al nodo muchas categorías de la clasificación. Si es una propiedad que solo admite un valor solo podrá tener una categoría asociada.

Para asignar un nodo a una categoría particular en una clasificación se deberá:

1. Asegurar que el nodo tiene el aspecto de clasificación apropiado aplicado.
2. Obtener los valores node ref para la categoría o categorías que se quiere aplicar.
3. Asignar los valores a la propiedad adecuada en el aspecto.

Si se borra una categoría, la categoría invalida se filtra de las propiedades del nodo cuando se leen. Cuando el nodo es actualizado tienen lugar la eliminación de la categoría marcada como invalida. Esto se hace para evitar que se re-indexen todos los nodos de una categoría cuando sea borrada.

Alfresco usa por defecto el aspecto cm:generalclassifiable definido en contentModel.xml para la clasificación. Cuando una nueva instalación es arrancado por primera vez las categorías del fichero categories.xml son inicializadas y cargadas como la jerarquía de categorías.

Clasificaciones, categorías y nodos con categorías pueden ser encontrados a través de consultas(Lucene) o mediante la API CategoryService.

Extender cm:generalclassifiable

Podemos extender la jerarquía por defecto creando nuevas categorías raíz y subcategorías a partir de ellas. Para hacerlo podemos usar la administración de categorías del interfaz de usuario, servicios web o la API CategoryServices.

El interfaz de usuario tratar las categorías raíces de cm:generalclassifiable como clasificaciones y pone toda categorización como una propiedad que permite múltiples valores.

Usando CategoryServices se puede realizar de extensión de la siguiente forma:

```
// Para crear una categoría raíz:
    NodeRef newRootCat = categoryService.createRootCategory(
          spacesStore,
          ContentModel.ASPECT_GEN_CLASSIFIABLE,
         "newRootCat");
    //Para crear una categoría:
    NodeRef newCategory = categoryService.createCategory(newRootCat, "newCategory");
```

Clasificar un nodo

Para poder clasificar un nodo lo primero que ha de hacerse es determinar si tiene aplicado el aspecto adecuado (ejemplo:cm:generalClassifiable). En caso de que no lo esté se añadirá. Después añadiremos o reemplazaremos cualquier categoría existente asignando a la propiedad adecuado la referencia a la categoria o un conjunto de referencias a categorías, dependiendo si puede tener uno o varios valores.

Usando el servicio de nodos se puede realizar esta acción como en este ejemplo:

```
NodeRef targetNode;
NodeRef categoryNode;
…
targetNode = …
…
categoryNode = …
…
// Replace any existing aspects
ArrayList<NodeRef> categories = new ArrayList<NodeRef>(1);
categories.add(categoryNode);
if(!nodeService.hasAspect(targetNode, ContentModel.ASPECT_GEN_CLASSIFIABLE)
{
   HashMap<QName, Serializable> props = new HashMap<QName, Serializable>();
   props.put(ContentModel.PROP_CATEGORIES, categories);
   nodeService.addAspect(targetNode, ContentModel.ASPECT_GEN_CLASSIFIABLE, props);
}
else
{
```

```
    nodeService.setProperty(targetNode, ContentModel.PROP_CATEGORIES, categories);
}
```

Añadir una clasificación propia

Lo primero que hay que hacer es añadir un nuevo aspecto al diccionario de modelos que tenga una propiedad del tipo d:category. En este caso se permite que un nodo sea relacionado con varias categorías (multiple-> true):

```
...
 </types>

 <aspects>
 ...
   <aspect name="mi:aspectoDeClasificacion">
     <title>Mi Clasificacion</title>
     <parent>cm:classifiable</parent>
     <properties>
        <property name="mi:categorias">
          <title>Categorias</title>
          <type>d:category</type>
          <mandatory>false</mandatory>
          <multiple>true</multiple>
          <index enabled="true">
            <atomic>true</atomic>
            <stored>true</stored>
            <tokenised>true</tokenised>
          </index>
        </property>
     </properties>
   </aspect>
 </aspects>
```

No se podrán crear categorías para esta clasificación a través del interfaz de usuario, ya que solo soporta cm:generalclassifiable. Tampoco en la búsqueda avanzada podríamos seleccionar categorías de esta clasificación.

Es posible también extender el tipo cm:category pero la administración de estos tipos no será soportada por el interfaz de usuario, los servicios web o los métodos de utilidades de CategoryService.

Por convenio un aspecto usado para categorización tiene definida solo una propiedad. Se podrían definir mas pero deberían ser propiedades adicionales usadas para la misma clasificación. Por ejemplo el uso de una categoría *primaria* y *secundaria* para la misma clasificación. Esto y la posibilidad de que las propiedades de categoría tengan uno o muchos valores cubren la mayoría de los casos de uso.

Creación de acciones personalizadas

- **Área:** Extensión de Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0051
Tipo de recurso: Manual

Descripción

Una acción es algo que el usuario puede hacer con un contenido. Son unidades de trabajo que pueden ser configuradas en tiempo de ejecución, ejem. check-in, check-out, cortar, pegar o borrar.

El repositorio de Alfresco esta provisto de muchos tipos de acciones que se pueden utilizar. En un espacio podemos encontrar acciones en cada contenido o listadas en el combo de mas acciones. Cuando vemos las propiedades de un contenido podemos ver acciones a la derecha de la página. También se pueden invocar acciones como parte de un flujo de trabajo simple.

Pero ademas es posible añadir tipos de acciones personalizados.

Una acción es algo que el usuario puede hacer con un contenido. Son unidades de trabajo que pueden ser configuradas en tiempo de ejecución, ejem. check-in, check-out, cortar, pegar o borrar.
El repositorio de Alfresco esta provisto de muchos tipos de acciones que se pueden utilizar. En un espacio podemos encontrar acciones en cada contenido o listadas en el combo de mas acciones. Cuando vemos las propiedades de un contenido podemos ver acciones a la derecha de la página. También se pueden invocar acciones como parte de un flujo de trabajo simple.

Pero ademas es posible añadir tipos de acciones personalizados.

Como construir acciones personalizadas

Una acción en su nivel mas básico consiste en una clase Action Executer y la declaración de su bean asociado. Normalmente también necesitemos un fichero de recursos para implementar la internacionalización y para personalizar la interfaz de usuario tendremos que crear paginas JSP para la inserción de valores para los parametros y un manejador que las relacione con la acción.

Antes de empezar una buena practica es buscar código dentro de Alfresco que sea similar y que se pueda aprovechar para codificar la acción. Seguir los mismos patrones que se utiliza en los fuentes de Alfresco para implementar nuestras personalizaciones hara mas fácil de mantener y compartir nuestro código con otros o incluso contribuir al proyecto Alfresco.

Para ilustrar los pasos a seguir para la implementación se va a usar como ejemplo la construcción de una acción personalizada que aplique un aspecto personalizado taggable que permitirá añadir una propiedad llamada tags (propiedad de texto multiple) al nodo y que ademas permita introducir algún tag por defecto.

Escribir un Action Executer

El Action Executer contiene la implementación de la acción. Es donde debemos poner el código que va a hacer el trabajo. Un Action Executer debe implementar la interfaz org.alfresco.repo.action.executer.ActionExecuter:

```
public interface ActionExecuter
{
   /**
    * Obtener la definición de la accion
    *
    * @return  the action definition
    */
   public ActionDefinition getActionDefinition();

   /**
    * Ejecutar el action executer
    *
    * @param action    la accion
    * @param actionedUponNodeRef      la referencia al nodo donde se ejecuta la accion
    */
   public void execute(Action action, NodeRef actionedUponNodeRef);
}
```

En esta definición podemos ver dos métodos. El primero nos servirá para obtener un objeto de definición de la acción. De este objeto podremos obtener detalles como el nombre de la acción o los parámetros. El segundo proporciona el método de ejecución, tendremos que proporcionarle la instancia de la acción y el nodo sobre el que realizar el trabajo.

Esta interfaz puede ser extendida directamente pero, como para que el servicio de acciones sea consciente de que la acción debe estar disponible para el cliente es necesario hacer mas cosas que solo implementar estos dos métodos, normalmente es preferible extender la clase ActionExecuterAbstractBase que se encuentra en el mismo paquete, la cual nos proporcionas mas facilidades para la creación de la acción.

Lo primero que tenemos que hacer pues es crear una nueva clase que extienda de la anterior:

```
public class TagActionExecuter  extends ActionExecuterAbstractBase
{
   /**
    * @ver org.alfresco.repo.action.executer.ActionExecuterAbstractBase#executeImpl(Action, NodeRef)
    */
   @Override
   public void executeImpl(Action action, NodeRef actionedUponNodeRef)
   {
      // TODO Rellenar con la implementación de la acción
   }
   /**
    * @see org.alfresco.repo.action.ParameterizedItemAbstractBase#addParameterDefinitions(java.util.List)
    */
   @Override
   protected void addParameterDefinitions(List<ParameterDefinition> paramList)
   {
      // TODO rellenar con la definición de parametros.
   }
}
```

Antes de empezar la implementación de la acción necesitamos los detalles del aspecto que queremos aplicar y una instancia del servicio de nodos con la que poder hacerlo.

Como ya hemos comentado los Action Executers serán configurados mediante una bean de Spring lo que significa que obtener cualquiera de los servicios del repositorio es cuestión simplemente de añadir un metodo setter que será usado por la configuración de Spring para inyectar el servicio requerido.

```
public class TagActionExecuter extends ActionExecuterAbstractBase
{
   /*** El servicio de nodos  */
   private NodeService nodeService;

    /*** Fijar el servicio de nodos
    * @param nodeService  El servicio de nodos
    */
   public void setNodeService(NodeService nodeService)
   {
      this.nodeService = nodeService;
   }
}
```

La configuración Spring que inyectará en el Action Executer el servicio de nodos se verá mas adelante.

El siguiente paso es parametrizar el Action Executer para así permitir aplicar algunos tags por defecto. La creación de una acción se basa en una definición que puede contener parámetros así las distintas instancias de la acción pueden tener valores distintos para ellas.

En este ejemplo queremos una acción a la que podamos proporcionar el nombre del tag a aplicar por defecto. Esto se especificará añadiendo una definición de parámetro en el método addParametersDefinitions que se vio antes.

```
public class TagActionExecuter extends ActionExecuterAbstractBase
{
   public static final String PARAM_TAG_NAME = "param-tags";
   /*** @ver org.alfresco.repo.action.ParameterizedItemAbstractBase#addParameterDefinitions(java.util.List) */
   @Override
   protected void addParameterDefinitions(List<ParameterDefinition> paramList)
   {
      // Añadir definiciones para la lista de parametros
      paramList.add(
```

```
        new ParameterDefinitionImpl(                                    // Crea una nueva definicion de parametro para la lista
          PARAM_TAG_NAME,                                              // El nombre usado para identificar al parametro
          DataTypeDefinition.TEXT                                       // El tipo del parametro
          true,                                                                 // Indica si el parametro es obligatorio
          getParamDisplayLabel(PARAM_TAG_NAME)));      // La etiqueta que se muestra para el parametro
  }
}
```

Cuando el método *getActionDefinition* sea llamado desde la clase base, se devolverá una definición que contenga una definición correcta de los parámetros. De esto se encarga la ParametizedItemAbstractBase que está en la jerarquía de herencia.

Así por último se puede rellenar la implementación del action executer:

```
protected void executeImpl(Action action, NodeRef actionedUponNodeRef)
{
  if (this.nodeService.exists(actionedUponNodeRef) == true)
  {
    // Añadir aspecto si este no está ya presente en el nodo
    QName tagAspect = QName.createQName("extension.tags", "taggable");
    if (this.nodeService.hasAspect(actionedUponNodeRef, tagAspect) == false)
    {
      this.nodeService.addAspect(actionedUponNodeRef, tagAspect, null);
    }

    // crear la lista de tags
    String tags = (String)action.getParameterValue("tags");
    List<String> tagsList = new ArrayList<String>();
    if (tags != null && tags.length() > 0)
    {
      StringTokenizer tokenizer = new StringTokenizer(tags, ",");
      while (tokenizer.hasMoreTokens())
      {
        tagsList.add(tokenizer.nextToken().trim());
      }
    }
    // fijar los tags a la propiedad
    QName tagsProp = QName.createQName("extension.tags", "tags");
    this.nodeService.setProperty(actionedUponNodeRef, tagsProp, (Serializable)tagsList);
  }
}
```

Configuración en Spring

Para hacer que la acción personalizada este disponible debemos declarar como un bean de Spring. Para ello crearemos un fichero en alfresco/extension siguiendo la nomenclatura que definimos para las extensiones de ficheros de configuración de Spring, por ejemplo add-aspect-action-context.xml, y añadiremos la definición:

```
# Accion personalizada para añadir aspecto

  <bean id="tag" class="org.alfresco.sample.TagActionExecuter" parent="action-executer">
    <property name="nodeService">
      <ref bean="nodeService" />
    </property>
  </bean>
```

Como se vio antes en esta configuración podemos ver como se inyecta el servicio de nodos usando el método que definimos.

Para impedir que la interfaz de usuario permita seleccionar esta acción para ser ejecutada antes de que se le halla añadido los recursos necesarios para hacerlo podemos configurar la acción para ser privada. Esto significará que se podrá usar en el repositorio aunque el web cliente no la mostrará entre los tipos de acciones disponibles.

```
<bean id="tag" class="org.alfresco.sample.TagActionExecuter" parent="action-executer">
    <property name="nodeService">
```

```
      <ref bean="nodeService" />
    </property>
    <property name="publicAction">
      <value>false</value>
    </property>
  </bean>
```

Aplicación del modelo de localización

Toda acción tiene asociado un titulo y una descripción. Además cada parámetro tiene un titulo para mostrar. Todas estas cadenas deben ser obtenidas desde ficheros de recursos I18N para asegurar que el repositorio permanece internacionalizado.

La clase base ActionExecuterAbstractBase busca, en alguno de los ficheros de recursos cargados, las entradas .title y .description.

Ademas debemos usar el método *getParamDisplayLabel* para obtener el mensaje id para el parámetro. Buscara el los ficheros de recursos un id con la siguiente estructura ..display-label. Este mensaje se usará cuando se cree la definición del parámetro como se vio antes.

Así crearemos un fichero tag-action-messages.properties dentro del directorio alfresco/extension y añadiremos las siguientes lineas para Internacionalizar la acción personalizada:

```
# Mensajes para la acción personalizada
  tag.title=Añadir aspecto a item
  tag.description=Añadirá un aspecto al item seleccionado.
  tag.param_tags.display-label =El nombre del aspecto a aplicar al item.
```

Este fichero debe ser registrado a través de Spring para que los mensajes puedan ser cargados así que se tendrá que añadir en nuestro fichero de extensión de Spring un nuevo bean para hacerlo, por ejemplo:

```
<bean id="tag-action-messages" class="org.alfresco.i18n.ResourceBundleBootstrapComponent">
  <property name="resourceBundles">
    <list>
      <value>alfresco.extension.tag-action-messages</value>
    </list>
  </property>
</bean>
```

Crear una página JSP

A partir de ahora se va a mostrar que es lo que se necesita personalizar en el cliente web para poder preguntar a los usuarios por los valores de los parámetros en una acción personalizada.

Aunque los asistentes para acciones han sido integrados en el nuevo Framework de Asistentes los JSPs que recogen parametros para acciones y condiciones son aun JSPs en toda regla como si no fueran mostrados como el contenedor de asistentes. Esto significa que se debe tener la estructura completa de una página en el JSP de la acción, la forma mas fácil de hacerlo es copiando una existente.

Si se coge como base la página jsp/actions/add-features.jsp solo tendremos que sustituir la lista desplegable por un campo de texto y cambiar algunas etiquetas. Así se copia esta pagina en el directorio jsp/extension y se re-nombra como tag.jsp. Una vez que se hallan hecho los cambio necesarios la pagina tendrá este aspecto:

```
<r:page titleId="title_action_tag">
<f:view>

  <%-- load a bundle of properties with I18N strings --%>
  <f:loadBundle basename="alfresco.messages.webclient" var="msg"/>
  <f:loadBundle basename="alfresco.extension.webclient" var="customMsg"/>

  <h:form acceptCharset="UTF-8" id="tag-action">
  ……
  <tr>
    <td><nobr><h:outputText value="#{customMsg.tags}:"/></nobr></td>
    <td width="95%">
      <h:inputText value="#{WizardManager.bean.actionProperties.tags}" size="50" maxlength="1024" />
    </td>
  </tr>
  ……
```

Implementación del manejador de la acción

Para integrar la accion dentro de los asistentes basados en reglas (Ejecutar accion, Crear regla....) necesitamos implementar un manejador. Esta clase, que normalmente extiende de org.alfresco.web.bean.action.handlers.BaseActionHandler, será responsable de conducir al asistente a la pagina de recogida de parámetros y guiar los parámetros entre el asistente y el repositorio.

Si la página que que recoge los parámetros requiere alguna inicialización por defecto se podrá sobrescribir el método setupUIDefaults(), en este ejemplo no se necesitará.

Se creará una clase Java llamada TagActionHandler con los siguientes metodos:

- El método *getJSPPath()*, que debe devolver el camino a nuestro JSP (/jsp/extension/tag.jsp).
- El método *prepareForSave()*, que coloca los tags que introduce el usuario en la propiedades del repositorio cuyo mapeo a sido pasado por parametro.
- El método *prepareForEdit()*, que por el contrario coge los tags guardados en la acción y colocarlas en las propiedades mapeadas para el asistente.
- El método *generateSummary()*, que se usa para generar una cadena de resumen para la acción, normalmente se incluyen los parámetros introducidos por el usuario.

El código de fichero TagActionHandler.java debe ser parecido a esto:

```
public class TagActionHandler extends BaseActionHandler
{
   public static final String PROP_TAGS = "tags";

   public String getJSPPath()
   {
      return "/jsp/extension/tag.jsp";
   }
   public void prepareForSave(Map<String, Serializable> actionProps,  Map<String, Serializable> repoProps)
   {
      repoProps.put(TagActionExecuter.PARAM_TAGS, (String)actionProps.get(PROP_TAGS));
   }
   public void prepareForEdit(Map<String, Serializable> actionProps,  Map<String, Serializable> repoProps)
   {
      actionProps.put(PROP_TAGS, (String)repoProps.get(TagActionExecuter.PARAM_TAGS));
   }
   public String generateSummary(FacesContext context, IWizardBean wizard, Map<String, Serializable> actionProps)
   {
      String tags = (String)actionProps.get(PROP_TAGS);
      if (tags == null)
      {
         tags = "";
      }

      return MessageFormat.format(Application.getMessage(context, "add_tags"),
            new Object[] {tags});
   }
}
```

Para generar el resumen usamos una nueva cadena id, *add_tags*. Tendremos que definirla en nuestro fichero webclient.properties personalizado así:

```
add_tags=Add tags "{0}"
```

El último paso consiste en registrar el nuevo manejador. Esto lo haremos en el fichero de configuración web-client-config-custom.xml añadiendo un nuevo elemento config con la condición "*Action Wizards*":

```
<config evaluator="string-compare" condition="Action Wizards">
   <action-handlers>
      <handler name="tag" class="org.alfresco.sample.TagActionHandler" />
   </action-handlers>
</config>
```

Comportamientos

- **Área:** Extensión de Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0052
Tipo de recurso: Manual

Descripción

Desarrollo de comportamientos personalizados

Como ejemplo de como desarrollar comportamientos personalizados vamos a definir una funcionalidad para que los usuarios puedan asignar una puntuación a un contenido y mostrar la media de todas las puntuaciones insertadas para este dentro del modelo personalizado que definimos en el apartado dedicado a las extensiones en el modelo de contenidos:

```
uri: "mi.nuevo.modelo"
   prefijo: "mi"
```

Si se quiere guardar las puntuaciones en Alfresco, una manera de hacerlo podría ser crear un tipo personalizado puntuacion (*rating*) que podrá estar relacionado con los contenidos a través de una asociación hija.

```
<type name="mi:rating">
    <title>Someco Rating</title>
    <parent>sys:base</parent>
    <properties>
        <property name="mi:puntuacion">
            <type>d:int</type>
            <mandatory>true</mandatory>
        </property>
        <property name="mi:explicacion">
            <type>d:text</type>
            <mandatory>true</mandatory>
        </property>
     </properties>
</type>
```

Se creará también un aspecto puntuable (*rateable*). Este aspecto contendrá una propiedad para guardar la media de puntuaciones y ademas definirá la relación hijo para las puntuaciones relacionadas con un contenido. Usar un aspecto hace que podamos añadir funcionalidad de puntuación a cualquier pieza de contenido del repositorio.

```
<aspect name="mi:rateable">
   <title>Puntuable</title>
   <properties>
      <property name="mi:puntuacionMedia">
         <type>d:double</type>
         <mandatory>false</mandatory>
      </property>
   </properties>
   <associations>
      <childassociation name="mi:puntuaciones">
         <title>Puntuacion</title>
         <source>
            <mandatory>false</mandatory>
            <many>true</many>
         </source>
         <target>
            <class>mi:rating</class>
            <mandatory>false</mandatory>
            <many>true</many>
         </target>
      </childassociation>
   </associations>
</aspect>
```

Esto en cuanto al modelo de datos, el siguiente paso es decidir como se va a ejecutar la lógica que calcule la media después de que se reciba cada puntuación. Una forma de hacerlo podría ser crear una acción que sea llamada por una regla. Cada vez que una puntuación es añadida a una carpeta, la regla desencadenara la acción que actualiza las medias. Esta opción tiene el problema de que cada vez que se quiera usar las puntuaciones de usuario se necesita configurar la regla en el espacio. También se podría implementar como una acción programada que buscará todos los contenidos con el aspecto puntuable y calcule su media, pero se perdería la posibilidad de conocer las puntuaciones en tiempo real.

La opción perfecta será implementar un comportamiento personalizado. Escribiremos la lógica necesaria para el calculo de las medias y los asociaremos a los eventos adecuados del tipo de contenido puntuación, que serán la creación y la eliminación del mismos que es cuando la puntuación afectará a la media.

Implementación usando Java

Escribir el comportamiento personalizado

Se implementará el comportamiento personalizado como una clase Java que implementará las interfaces que corresponden a los eventos con los que queremos enlazar: NodeServicePolicies.OnDeleteNodePolicy y NodeServicePolicies.OnCreateNodePolicy.

La declarción de la clase quedaría como sigue:

```
public class Rating implements NodeServicePolicies.OnDeleteNodePolicy, NodeServicePolicies.OnCreateNodePolicy {
```

La clase tendrá dos dependencias que debe manejar Spring. La primera es el NodeService que será usado en la lógica del calculo de la media y el otro será el PolicyComponent el cual será usado para ligar los comportamientos con los eventos:

```
// Dependencias
  private NodeService nodeService;
  private PolicyComponent policyComponent;
  // Comportamientos
  private Behaviour onCreateNode;
  private Behaviour onDeleteNode;
```

Es momento de crear un método init que se encargue de hacer saber a Alfresco de que el comportamiento debe ser ligado a un evento(policy). El metodo init será llamado cuando Spring cargue el bean.

```
public void init() {
    // Create comportamientos
    this.onCreateNode = new JavaBehaviour(this,"onCreateNode",NotificationFrequency.TRANSACTION_COMMIT);
    this.onDeleteNode = new JavaBehaviour(this,"onDeleteNode",NotificationFrequency.TRANSACTION_COMMIT);
    // Ligar comportamientos a eventos
    this.policyComponent.bindClassBehaviour(Qname.createQName(NamespaceService.ALFRESCO_URI,"onCreateNode"),
                                Qname.createQName("mi.nuevo.modelo","rating"),this.onCreateNode);
    this.policyComponent.bindClassBehaviour(QName.createQName(NamespaceService.ALFRESCO_URI,"onDeleteNode"),
                                    Qname.createQName("mi.nuevo.modelo","rating"),this.onDeleteNode);
}
```

Se puede decidir cuando se invocará el comportamiento especificando el valor apropiado para *NotificationFrequency*. Ademas de *TRANSACTION_COMMIT* las otras opciones son *FIRST_EVENT y EVERY_EVENT*.

Tras la creación de los comportamientos llamamos la método *bindClassBehaviour* para establecer la relación. En este caso estamos ligando el Qname del comportamiento al Qname del tipo "*rating*" y diciendo a Alfresco que llame a los comportamientos onCreateNode y onDeleteNode de nuestro bean.

Para asociaciones (*bindAssociationBehaviour*) y para propiedades (*bindPropertyBehaviour*) existen métodos adicionales para establecer esta relación que se tendrán que usar dependiendo del tipo de evento con el que estemos intentando realizar la ligadura.

El siguiente paso es escribir los metodos requeridos por las interfaces de los eventos. Con independencia de que se este creando una puntuación o se este destruyendo debemos realizar de nuevo el calculo de la media así que en ambos casos lo único que se hará sera llamar a un método (*calcularMedia()*) que implementará la lógica del calculo pasando la referencia al nodo como parámetro.

```
public void onCreateNode(ChildAssociationRef childAssocRef) {
        calcularMedia(childAssocRef);
    }

    public void onDeleteNode(ChildAssociationRef childAssocRef, boolean isNodeArchived) {
        calcularMedia(childAssocRef);
```

```
    }
```

El método *calcularMedia()* obtendrá la referencia del padre del nodo que ha provocado el evento, a este le solicitará la lista de sus hijos y realizará una iteración sobre ellos calculando la media. Por último guardará el valor calculado en la propiedad media del contenido.

```
public void calcularMedia(ChildAssociationRef childAssocRef) {
    // obtener el nodo padre
    NodeRef padreRef = childAssocRef.getParentRef();
    // comprobamos el padre para asegurarnos que tiene el aspecto correcto
    if (nodeService.hasAspect(padreRef,Qname.createQName("mi.nuevo.modelo","rateable"))) {
        // obtener los nodos hijos del padre
        List<ChildAssociationRef> hijos = nodeService.getChildAssocs(parentRef);

        // iterar a traves de los hijos para obtener el total
        Double media = 0d;
        int total = 0;
        for (ChildAssociationRef hijo : hijos) {
            int puntuacion = (Integer)nodeService.getProperty(
                    hijo.getChildRef(),
                    Qname.createQName("mi.nuevo.modelo","puntuacion"));
            total += puntuacion;
        }

        // calcular la media
        media = total / (hijos.size() / 1.0d);

        // guardar la media en el nodo padre
        nodeService.setProperty(padreRef,
                QName.createQName("mi.nuevo.modelo","puntuacionMedia"),
                media);
    }
    return;
}
```

Para terminar solo quedaría establecer los métodos *getter* y *setter* para el *Node Services* y el *Policy Component*.

Configuar el bean de Spring

El último paso es configurar la clase del comportamiento como un bean de Spring. La configuración de este bean puede ponerse en cualquier fichero de configuración como por ejemplo el fichero de registro del modelo. Si tenemos varios comportamientos personalizados se podrían poner todos en un fichero de configuración propio. Para el comportamiento que se esta desarrollando como ejemplo la configuración sería:

```
<bean id="ratingBehavior" class="es.intecna.sad.behavior.Rating" initmethod="init">
    <property name="nodeService">
        <ref bean="nodeService" />
    </property>
    <property name="policyComponent">
        <ref bean="policyComponent" />
    </property>
</bean>
```

Implementación usando JavaScript

Escribir el comportamiento personalizado

Realmente se van a escribir tres scripts. Los scripts *onCreateRating.js* y *onDeleteRating.js*, que se ligaran respectivamente con los eventos *onCreateNode* y *onDeleteNode*, y el script *rating.js*, que contendrá la lógica del calculo de la media y tendra que ser importada por los otros dos usando la etiqueta .

Para hacer el despliegue de esta implementación hay dos opciones, desplegarlos como parte de la aplicación web o cargar los scripts en Alfresco. El primero tiene el inconveniente de que si queremos hacer algún cambio tendremos que volver a desplegar. En el segundo caso los scripts serán accesibles para otros usuarios de Alfresco. En este ejemplo se asumirá que se usa la primera opción así que deberemos desplegar dentro del directorio scripts un fichero *onCreateRating.js*:

```
<import resource="classpath:alfresco/extension/scripts/rating.js">
```

```
  var hayFallo = false;
  // Comprobar el objeto comportamiento que debía haber sido pasado
  if (behaviour == null) {
   logger.log("El objeto comportamiento no ha sido establecido.");
   hayFallo = true;
  }
  // Comprobar el nombre del comportamiento
  if (behaviour.name == null && behaviour.name != "onCreateNode") {
   logger.log("El objeto comportamiento no ha sido establecido adecuadamente.");
   hayFallo = true;
  } else {
   logger.log("Nombre del comportamiento: " + behaviour.name);
  }
  // Comprobar los argumentos
  if (behaviour.args == null) {
   logger.log("Los argumentos no han sido establecidos.");
   hayFallo = true;
  } else {
   if (behaviour.args.length == 1) {
       var assocHijo = behaviour.args[0];
       logger.log("Llamando al calculo de media");
       calcularMedia(assocHijo);
   } else {
       logger.log("El numero de argumentos es incorrecto.");
       hayFallo = true;
   }
  }
```

El código para el *onDeleteRating.js* es equivalente al anterior con la excepción del nombre del comportamiento y del número de argumentos esperados:

```
<import resource="classpath:alfresco/extension/scripts/rating.js">
  var hayFallo = false;
  // Comprobar el objeto comportamiento que debía haber sido pasado
  if (behaviour == null) {
   logger.log("El objeto comportamiento no ha sido establecido.");
   hayFallo = true;
  }
  // Comprobar el nombre del comportamiento
  if (behaviour.name == null && behaviour.name != "onDeleteNode") {
   logger.log("El objeto comportamiento no ha sido establecido adecuadamente.");
   hayFallo = true;
  } else {
   logger.log("Nombre del comportamiento: " + behaviour.name);
  }
  // Comprobar los argumentos
  if (behaviour.args == null) {
   logger.log("Los argumentos no han sido establecidos.");
   hayFallo = true;
  } else {
   if (behaviour.args.length == 2) {
       var assocHijo = behaviour.args[0];
       logger.log("Llamando al calculo de media");
       calcularMedia(assocHijo);
   } else {
       logger.log("El numero de argumentos es incorrecto.");
       hayFallo = true;
   }
  }
```

Para implementar el tercer script se usará la misma lógica que se uso en la implementación con Java pero siguiendo la sintaxis de la API de JavaScripts de Alfresco. Se creará el fichero *rating.js* con la siguiente estructura:

```
function calcularMedia(childAssocRef) {
    var padreRef = childAssocRef.parent;
    //ccomprobar el padré para aseguramos de que tieneel aspecto correcto
    if (padreRef.hasAspect("{mi.nuevo.modelo}rateable")) {
        // obtener los nodos hijos del padre e iterar a traves de los hijos para obtener el total y la media
        var hijos = padreRef.children;
        var media = 0.0;
        var total = 0;
        if (hijos != null && hijos.length > 0) {
            for (i in hijos) {
                var hijo = hijos[i];
                var puntuacion = hijo.properties["{mi.nuevo.modelo}puntuacion"];
                total += puntuacion;
            }
            media = total / hijos.length;
        }
        // guardar la media en el nodo padre
        padreRef.properties["{mi.nuevo.modelo}puntuacionMedia"] = media;
        padreRef.save();
    }
    return;
}
```

Configurar el bean de Spring

En la implementación en Java se usaba el método init de la clase para realizar la ligadura de de los eventos con los comportamientos a traves de llamadas a los metodos de PolicyComponent. En la aproximación mediante JavaScript una vez escritos los scripts se necesita realizar una configuración de Spring para asociar estos a los eventos adecuados *onCreateNode* y *onDeleteNode*.

Así en un fichero de configuración de la aplicación introduciremos la siguiente configuración:

```
<bean id="onCreateRatingNode"
    class="org.alfresco.repo.policy.registration.ClassPolicyRegistration"
    parent="policyRegistration">
    <property name="policyName">
        <value>{http://www.alfresco.org}onCreateNode</value>
    </property>
    <property name="className">
        <value>{mi.nuevo.modelo}rating</value>
    </property>
    <property name="behaviour">
        <bean class="org.alfresco.repo.jscript.ScriptBehaviour" parent="scriptBehaviour">
            <property name="location">
                <bean class="org.alfresco.repo.jscript.ClasspathScriptLocation">
                    <constructorarg>
                  <value>alfresco/extension/scripts/onCreateRating.js</value>
                    </constructorarg>
                </bean>
            </property>
        </bean>
    </property>
</bean>
<bean id="onDeleteRatingNode"
    class="org.alfresco.repo.policy.registration.ClassPolicyRegistration"
    parent="policyRegistration">
    <property name="policyName">
        <value>{http://www.alfresco.org}onDeleteNode</value>
    </property>
    <property name="className">
```

Desarrollo de extracción de datos

- **Área:** Extensión de Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0053
Tipo de recurso: Manual

Descripción

Un fichero tiene asociada una serie de información particular que define las principales caracteristicas del mismo. Esta información esta contenida en el propio fichero y es conocida como metadatos. Por ejemplo el título, el autor, la fecha de creación o el nombre son metadatos por defecto para cualquier tipo de fichero.

Obviamente cada tipo de fichero puede tener asociados distintos tipos de metadatos (por ejemplo, un fichero MP3 tiene tiene atributos como el nombre del album, el titulo de la canción, un artista, un genero y así). Estos meta-datos son muy diferentes en un fichero Word. Cuando un reproductor de MP3 carga un fichero, trata de extreaer todos los meta-datos conocidos para mostrarselos al usuario. Alfresco realiza esta labor de una manera muy similar. Cuando un fichero es subido al repositorio, un extractor de metadatos - si esta asociado explicitamente - trata de realizar la extracción y cargar valores a propiedades asociadas con los meta-datos. Las properties serían visibles entonces en Alfresco a traves del navegador web.

Ademas una vez que el metadato asociado al fichero a sido extraido, el usuario podra ejecutar busquedas avanzadas usando ese metadato: cada metadato puede ser usado como un filtro para ejecutar consultas contra el repositorio.

Alfresco proporciona unos extractores por defecto (para pdf, office, html, mp3, opendocument, openoffice, mail)pero ademas permite realizar un mapeo de metadatos en un modelo personalizado desde un documento que se agregue o sea actualizado en el repositorio. Vamos a ver dos ejemplos de como configurar alfresco para que realice esta labor.

Ejemplos

Solución simple

Lo primero es crear una regla sobre el espacio donde vamos a crear los nuevos contenidos y sobre los tipos de objetos del ámbito que vayamos a utilizar, para que al crear nuevo contenido aplique la acción *Extract common metadata fields from content*. Esto es fundamental, y necesario para todas las soluciones, ya que es lo que permite que se utilice una extracción distinta a la estándar. Sería deseable restringir la aplicación de esta regla solo a los contenidos del tipo correspondiente a nuestro modelo personalizado. Ampliando la funcionalidad de esta regla y mediante el uso de script también podremos ampliar y desarrollar la incorporación de metadatos sin necesidad de ningún otro código específico.

A continuación, para una solución sencilla bastaría con añadir los campos a sustituir en custom-metadata-extrators-context.xml (si estamos trabajando con Tomcat en /shared/classes/alfresco/extensión/), como en el ejemplo siguiente.

Para este ejemplo tendremos las siguientes consideraciones:

- http://www.sample.com/model/content/1.0 sería el namespace de nuestro modelo personalizado.
- Para cada metadato que queramos incluir, añadiríamos una propiedad del tipo

sc:test1

Donde test1 sería el metadato del documento OpenOffice y sc:test1 sería el metadato en nuestro modelo personalizado.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd"[]>
<!-- Este ejemplo muestra como añadir en nuestro modelo, campos específicos -->
<beans>
  <bean id="extracter.OpenDocument" class="org.alfresco.repo.content.metadata.OpenDocumentMetadataExtracter" p
    <property name="inheritDefaultMapping">
      <value>true</value>
    </property>
    <property name="mappingProperties">
        <props>
            <prop key="namespace.prefix.sc">http://www.sample.com/model/content/1.0</prop>
            <prop key="test1">sc:test1</prop>
            <prop key="test2">sc:test2</prop>
        </props>
     </property>
  </bean>
</beans>
```

Si el fichero custom-metadata-extrators-context.xml ya existe, bastará con añadir una nueva sección . La modificación (o

creación) de este fichero también será necesaria en todo caso.

En el caso del ejemplo **sc:test1** se corresponde con un metadato asociado a un aspecto, que a su vez es el aspecto por defecto ("mandatory") para el tipo de documentos que define nuestro modelo. Esto es importante, ya que al añadir el contenido e indicar su tipo, para que el mecanismo funcione, debe tener los metadatos adecuados.

Una vez realizados los cambios en el fichero, será necesario reiniciar el servidor para que se cargue la nueva configuración.

A continuación bastaría con utilizar la opción de "Añadir contenido" e indicar el tipo correspondiente a nuestro modelo personalizado. Al mostrarse las opciones de propiedades de contenido, si al crear el modelo hemos definido las opciones de visualización, podremos comprobar que los metadatos del documento de origen se asignan automáticamente a los metadatos del nuevo contenido de nuestro modelo personalizado, basándonos en las definiciones que hemos realizado.

Si ya necesitamos algo más específico o con una funcionalidad más personalizada habría que desarrollar una clase que herede del extractor de Open Document y modifique el contenido de manera específica, como se detalla en el siguiente punto.

Solución con extractor personalizado

Esta solución sería un mecanismo ampliado, para realizar una manipulación o un procesamiento de los metadatos a incorporar. También puede ser necesario cuando los metadatos en el origen se encuentren comprimidos o sea necesario extraerlos de una manera particular, todos los metadatos del documento OpenOffice en un solo metadatos de nuestro contenido por ejemplo.

Sigue siendo necesaria la creación de una regla y del fichero custom-metadata-extrators-context.xml como se ha explicado en el punto anterior (si trabajamos con TOMCAT en /shared/classes/Alfresco/extensión/).

La clase Java a realizar será una subclase de AbstractMappingMetadataExtracter del paquete org.alfresco.content.metadata, que es la clase abstracta donde están definidas las funcionalidades básicas de un extractor personalizado.

Veamos la clase CustomOpenOfficeMetadataExtracter como un ejemplo de la clase a construir. En esta clase se han sobrecargado los distintos métodos necesarios:

```
package org.alfresco.module.CustomMetadata;
import java.io.IOException;
import java.io.InputStream;
import java.io.Serializable;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Hashtable;
import java.util.Map;
import java.util.Set;
import org.alfresco.repo.content.*;
import org.alfresco.repo.content.metadata.AbstractMappingMetadataExtracter;
import org.alfresco.service.cmr.repository.ContentReader;
import org.alfresco.service.namespace.QName;
import com.catcode.odf.ODFMetaFileAnalyzer;
import com.catcode.odf.OpenDocumentMetadata;
public class CustomOpenOfficeMetadataExtracter extends
        AbstractMappingMetadataExtracter {
    //private static final Log logger = LogFactory.getLog(CustomOpenOfficeMetadataExtracter.class);

    private static final String KEY_CREATION_DATE = "creationDate";
    private static final String KEY_CREATOR = "creator";
    private static final String KEY_DATE = "date";
    private static final String KEY_DESCRIPTION = "description";
    private static final String KEY_GENERATOR = "generator";
    private static final String KEY_INITIAL_CREATOR = "initialCreator";
    private static final String KEY_KEYWORD = "keyword";
    private static final String KEY_LANGUAGE = "language";
    private static final String KEY_PRINT_DATE = "printDate";
```

Junto con esta clase podríamos desarrollar también un fichero de propiedades (CustomOpenOfficeMetadataExtracter.properties) que nos permitirá definir las equivalencias de metadatos directamente. En nuestro caso, un ejemplo sería:

```
#
# CustomOpenOfficeMetadataExtracter - default mapping
#
# author: Intecna
# Namespaces
```

```
namespace.prefix.sc=http://www.someco.com/model/content/1.0
namespace.prefix.cm=http://www.alfresco.org/model/content/1.0
# Mappings
creationDate=cm:created
creator=cm:author
date=
description=cm:description
generator=
initialCreator=
keyword=
language=
printDate=
printedBy=
subject=
title=cm:title
test1=sc:test1
test2=sc:test2
```

En este caso, si incluimos el fichero de propiedades, nuestro fichero custom-metadata-extrators-context.xml , podría simplificarse quedando como:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd"[]>
<!-- Este ejemplo muestra como añadir en nuestro modelo, campos específicos -->
<beans>
   <bean id="extracter.OpenDocument"  class="org.alfresco.module.CustomMetadata.CustomOpenOfficeMetadataExtra
      <property name="inheritDefaultMapping">
    <value>true</value>
 </property>
      <property name="mappingProperties">
         <bean   class="org.springframework.beans.factory.config.PropertiesFactoryBean">
           <property name="location">
              <valueasspath>cl:alfresco/extension/ CustomOpenOfficeMetadataExtracter.properties</value>
           </property>
         </bean>
      </property>
   </bean>
```

En el caso de que no hayamos definido un fichero de propiedades, podremos mantener custom-metadata-extrators-context.xml como en el punto 1.1, con la salvedad de hacer referencia a nuestra nueva clase:

```
<bean id="extracter.OpenDocument" class="org.alfresco.module.CustomMetadata.CustomOpenOfficeMetadataExtracte
```

A continuación y una vez realizadas las tareas y actualizados todos los ficheros reiniciaremos el servidor para que la nueva configuración sea cargada.

Workflows

- **Área:** Extensión de Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0054
Tipo de recurso: Manual

Descripción

Workflows en Alfresco

En Alfresco existen dos opciones para implementar workflows. Para workflows simples usuarios no expertos pueden usar la funcionalidad de los Workflows Basicos integrada. Para necesidades mas complejas Alfresco tiene integrado un motor Jboss jBPM que provee al sistema de capacidades para Workflows avanzados. En el siguiente cuadro podemos ver a alto nivel una comparativa entre los dos tipos de workflows en la que podremos apreciar claramente sus diferencias.

Workflows basicos de Alfresco

- Son configurables a traves del cliente web por usuarios finales
- Aprovecha las reglas, los espacios y las acciones
- Solo puede manejar procesos con pasos simples, hacia adelante o hacia atrás.
- No soporta decisiones, divisiones, uniones o flujos paralelos.
- No se mantiene metadatos o estados sobre el proceso.

Workflows avanzados de Alfresco

- Son definidos por analistas de negocio o desarrollodores a través de un plug-in grafico de Eclipse o escribiendo XML.
- Aprovecha la potencia del motor de workflows Jboss jBPM integrado en Alfresco.
- Se puede modelar cualquier proceso de negocio incluyendo decisiones, divisiones, uniones, flujos paralelos, sub-procesos, estados de espera y temporizadores.
- Puede incluir lógica de negocio escrita en Java o JavaScript, con acceso a la API de Alfresco.
- Mantiene estado y variables (metadatos) sobre el proceso mismo.

Workflows en Alfresco

Worksflows básicos

Los workflows básicos usan los espacios y un modelo de "paso adelante o paso atras" para implementar procesos en serie. Cuando un contenido es dejado en una carpeta se dispara una regla que le asocia "un paso adelante" o "un paso hacia atrás" o ambos. Estos pasos normalmente están ligados con acciones de Alfresco como "Mover el contenido a un espacios especifico" o "Fijar propiedad". Los usuarios pueden entonces seleccionar en el paso adecuado para cada contenido individual.

Un ejemplo de uso este tipo de flujos sería un proceso simple de revisión y aprobación, en el cual un contenido es subido a Alfresco, despues revisado y por último aprobado o rechazado. Una forma de implementar esto sería con tres espacios en las cuales tendremos reglas que apliquen workflos básicos. En la primera carpeta que se llamará "Borradores" tendremos un workflow que tendrá un solo paso etiquetado como "Enviar". Su acción movera el contenido a la segunda carpeta que se llamará "En revisión" y enviará un correo al grupo de personas encargadas de esta tarea. Esta carpeta a su vez tendrá un Workflow con dos pasos llamados Rechazar y Aprobar. La acción relacionada con estos pasos moverá los contenidos respectivamente a la carpeta "Borradores" o a la tercera carpeta llamada "Aprobados". Los workflows básicos son utiles pero muy limitados en cuanto a la complejidad de los procesos de negocio que pueden manejar.

Workflows avanzados

Un workflow avanzado de Alfresco se compone de 4 partes. La primera de ellas será la implementación del Workflow ya para acometerla necesitaremos un motor BPM. Las otras tres afectan al cliente de Alfresco y definen la interfaz del Workflow.

- **Definición del Proceso:** La definición del proceso describe los estados (pasos) y transiciones (elecciones) de un flujo de trabajo. Un paso puede ser de humanos o de sistema. Los pasos de humanos crean y asignan tareas a usuarios mientras que los pasos de sistema realizan algún tipo de operación sobre el repositorio. Ambos son descritos e implementados en la definición del proceso. Alfresco tiene integrado el motor de procesos Jboss jBPM que proporciona el lenguaje de Definición de Proceso jPDL. jPDL se presenta en uno de los siguientes formatos:

1. Archivo de proceso JBoss jBPM (fichero .par).
2. Documento jPDL xml JBoss jBPM (fichero .xml).

- **Modelo de Tareas:** El modelo de tareas proporciona la descripción de cada una de las tareas humanas en el flujo de trabajo. La descripción se utiliza para guiar al dialogo de la interfaz de usuario para mostrar y administrar la tarea. El mecanismo para describir las tareas del Workflow es similar al usado para definir los tipos de objeto a guardar, editar y ver. Cada descripción de tarea consiste en:

1. Nombre y titulo.

2. Propiedades y Asociaciones (por ejemplo la información adjunta a la tarea)

- **Configuración del cliente web:** La configuración del cliente web especifica la presentación de las tareas a los usuarios a través del cliente web. EL mecanismo de configuración para controlar los diálogos que tiene Alfresco se extiende también a los diálogos de las tareas de Worflows. La configuración permite:

1. Controlar que propiedades de las Tareas son visualizadas.
2. Controlar que propiedades de las Tareas son obligatorias y de solo lectura.
3. Controlar como cada propiedad de la Tarea es dibujada en el diálogo.
4. Presentar un diálogo personalizado para cada tipo de Tarea del Workflow.

- **Fichero de recursos (opcional):** Un fichero de recursos de Workflow proporciona todos los mensajes que se muestran en el interfaz de usuario durante la gestión del flujo de trabajo (Titulos de Tareas, nombres de Propiedades., nombres de las opciones ..).

Alfresco soporta el modelo de internacionalización en todo su cliente web incluidos los Workflows, de está forma la misma configuración que se usa en el cliente web para los ficheros de recursos se extiende a los Workflows.

Creación de un Workflow avanzado personalizado

Para ilustrar la creación de un flujo de trabajo avanzado se va a como ejemplo la creación de un sencillo proceso de tres pasos:

1. El usuario A envia una tarea "ad-hoc" al usuario B.
2. El usuario B realiza la tarea y la finaliza.
3. El usuario A es informado de la finalización.

Para hacer las cosas un poco más interesante vamos a permitir a un usuario especificar en el envio:

- La descripción de lo que se necesita hacer.
- Fecha de vencimiento (opcional).
- Prioridad.
- Si se le notificará o no la finalización mediante un correo electronico.

Paso 1: Crear la definición del proceso

Como se dijo antes existén dos formas de crear la definición del proceso. La primera es escribiendo a mano un documento xml jPDL. La segunda es ayudados por un diseñador que nos genere el documento xml o un archivo de proceso.

Desarrollo a mano del fichero xml jPDL

Usando cualquier editor de texto se escribe el documento. En el cuadro siguiente podemos ver la estructura básica de un workflow, en este caso la Tarea ad-hoc. Siempre debe haber una etiqueta *<star-state>* y una etiqueta *<end-state>*. El atributo name del elemento *<process-definition>* y del los *<task-node>* son importantes porque el resto de módulos de la definición del workflow los referenciarán para referirse al proceso y a las tareas. La etiqueta *swimlane* se utiliza para definir "roles" en el flujo de trabajo. Las tareas se asocian con un rol:

```
<process-definition xmlns="urn:jbpm.org:jpdl-3.1" name="wf:adhoc">
  <swimlane name="initiator"/>
  <start-state name="start">
    <task name="wf:submitAdhocTask" swimlane="initiator"/>
    <transition name="" to="adhoc"/>
  </start-state>
  <swimlane name="assignee"/>

  <task-node name="adhoc">
    <task name="wf:adhocTask" swimlane="assignee"/>
    <transition name="" to="completed"/>
  </task-node>

  <task-node name="completed">
    <task name="wf:completedTask" swimlane="initiator"/>
    <transition name="" to="end"/>
  </task-node>

  <end-state name="end"/>

</process-definition>
```

Desarrollo de un fichero de proceso usando un Diseñador de Procesos

Una manera alternativa para desarrollar el documento jBPM es usando el Diseñador de Procesos JBoss jBPM. Este diseñador es

un plugin para el IDE Eclipse así que para instalarlo necesitaremos tener e instalar:

- Eclipse SDK 3.3 desde *eclipse.org*
- JBoss jBPM 3.2 desde *labs.jboss.com*
- Jboss jBPM Process Designer Plugin 3.0.13 desde *labs.jboss.com.*

El Diseñador de Procesos JBoss jBPM es un editor grafico que sirve para desarrollar workflows. Es posible cambiar entre vistas de Diagrama y XML permitiendo desarrollos sincronizados de xml jPDL bien mediante la elaboración manual del xml como generandolo a través de un diagrama.

Ademas de ayudarnos en el diseño el Diseñador de Procesos JBoss jBPM nos permite guardar la definición del proceso a un archivo de proceso o desplegarlo directamente en un servidor existente. Alfresco esta preparado para permitir este despliegue automatico. Los pasos a seguir son:

1. Asegurarse de que el servidor esta levantado y activo.
2. Desde la pestaña Despliegue (Deployment) introducir:
   a. Nombre del servidor. Nombre de la maquina donde Alfresco está instalado.
   b. Puerto del servidor. Numero de puerto asignado a Alfresco.
   c. Server Deployer = /alfresco/jbpm/deployprocess
3. Pulsar en Test Connection.
4. Si todo es correcto pulsar en Deploy Process Archive.

Si el despliegue es exitoso, la definición de proceso ya esta disponible para su uso en Alfresco. Hay que tener en cuenta que no es necesario reiniciar el servidor para activar la nueva definición desplegada lo que permite realizar un proceso rápido de prueba y error mientras se desarrollan nuevas definiciones de proceso en el Diseñador de Procesos JBoss jBPM.

Desplegar manualmente un fichero xml jPDL o un Archivo de Proceso

Las definiciones de procesos pueden ser configuradas en Alfresco de tal manera que son desplegadas en él cada vez que se inicia.

El bean de Spring *workflowDeployer* permite realizar el despliegue de las Definiciones de Proceso. Puede ser usado conjuntamente con el mecanismo de configuración extensiones para desplegar workflows personalizados.

```
<bean id="myworkflows.workflowBootstrap" parent="workflowDeployer">
  <property name="workflowDefinitions">
    <list>
      <props>
        <prop key="engineId">jbpm</prop>
        <prop key="location">alfresco/workflow/adhoc_processdefinition.xml</prop>
        <prop key="mimetype">text/xml</prop>
      </props>
    </list>
  </property>
</bean>
```

Este bean acepta una lista de workflows para desplegar. Cada descripción consiste en las siguientes propiedades:

- **engineId:** El motor de procesos integrado en el que vamos a desplegar. Desde la versión 1.4 debe ser *jbpm*.
- **location:** El ubicación dentro del classpath del fichero de definición. La recomendación es seguir el estandar del mecanismo de configuración de extensiones la ubicación debe ser algo parecido a:

alfresco/extension/workflow/xxx_processdefinition.xml.

- **mimetype:** El formato del fichero de definición del proceso. Debe ser *text/xml* para ficheros jPDL XML o *application/zip* para ficheros de Archivo de Procesos jBPM. Por defecto será el segundo.
- **redeploy:** Fuerza el despliegue de de la definición del proceso si este ya esta desplegado. Esto es útil durante el proceso de desarrollo y prueba de un flujo. Cada vez que se inicia el servidor Alfresco la definición del proceso es desplegada en una nueva versión, las anteriores versiones se conservan para permitir a los workflows que se estén activos continuar su ejecución. En entornos de producción es recomendable que esté con valor *false*, que es el valor por defecto.

Paso 2: Crear el Modelo de Tareas

Es posible asociar una descripcionde Tarea a cada Tarea en la definición de proceso (elemento en jDPL). La descripción especifica la información que se puede adjuntar a una Tarea, por ejemplo las propiedades (nombre y tipo de datos) y asociaciones (nombres y tipos de objeto asociados). Un usuario puede ver y editar esta información en el dialogo de Tareas con el Cliente Web de Alfresco.

El Modelo de Tareas es expresado como un Modelo de Contenidos. Los pasos a seguir son:

1. Crear un nuevo modelo de contenido para la Definición de Proceso.

2. Crear un Tipo por cada Tarea.
3. Para cada Tipo, describe las Propiedades y Asociaciones (informacion) requerida por la tarea.

Modelo de Tareas Comunes

Alfresco proporciona un Modelo de Contenido pre-registrado para describir los atributos mas comunes para todas la Tareas: Id de la tarea, fecha de inicio, fecha de vencimiento, fecha de fin, prioridad, estado, resultados (p.e. que elección ha sido tomada al final), paquete de flujo de trabajo (p.e. el contenido enviado por el workflow), contexto (p.e. El espacio de Alfresco en el cual se inicia el Workflow)...

Esta información, entre otras, es definida en el modelo '*bpm:businessprocessmodel*' . Este modelo esta localizado en el fichero de configuración *alfresco/model/bpmModel.xml*. Dentro de este modelo las dos definiciones de tipos mas importantes son '*bpm:workflowTask*' y '*bpm:startTask*'. Las definiciones de tareas de workflows personalizadas deben derivar (explicita o implicitamente) de uno de estos dos tipos.

Este modelo de Tareas basico puede ser referenciado en los modelos personalizados usando el elemento import:

```
<imports>
    <import uri="http://www.alfresco.org/model/bpm/1.0" prefix="bpm"/>
  </imports>
```

La Star Task (tarea Inicial)

Existe una tarea especia conocida como Star Task la cual se asigna al Iniciador del workflow. Se usa para recoger información (p.e. los parámetros del workflow) requeridos para el proceso del flujo de datos. En el cliente de Alfresco, la Star Task se presenta como un sola pagina en el Asistente de Inicio de un Workflow que se lanza al iniciar un flujo de trabajo. La pagina (como todo el resto de diálogos de las Tareas) se guía por por la Definición de Tarea para presentarse.

En la Definición del Proceso (e.j. JPDL), la Star Task es la que esta definida con el elemento *<star-state>*. La descripción de la tarea Star Task debe derivar de '*bpm:startTask*'.

A veces en está tarea inicial se deben recoger los participantes (personas, grupos) que van a jugar un papel dentro del flujo. Para permitir esto se proporcionan los siguientes aspectos pre-definidos para que puedan ser adjuntados a la definición de la tarea inicial:

- **bpm:assign** recoge una sola persona que jugara un papel.
- **bpm:assignees** recoge una o mas personas que jugaran un papel.
- **bpm:groupAssignee** recoge un solo grupo que jugara un papel. (desde V2.0)
- **bpm:groupAssignees** recoge uno o mas grupos que jugaran un papel. (desde V2.0)

Se pueden usar aspectos personalizados para recoger cualquier numero arbitrario de nombres o grupos.

Acciones de los paquetes de flujo de trabajo

Los Workflows de Alfresco generalmente trabajan con contenido del Repositorio. Los paquetes de flujos de trabajo se usan para guardar el contenido que se envía a través del flujo. El contenido que esta dentro del grupo puede ser visto, editado o eliminado. También se puede añadir contenido al paquete. Aunque no todas las operaciones pueden ser aplicadas a todas las Tareas. El cliente web de Alfresco soporta la noción de "Grupo de Acciones". Cada tarea tiene asignado un "Grupo de Acciones" que define que opciones del interfaz de usuario están disponibles sobre el paquete de Workflows mientras en el cliente web se desarrolla esa tarea. Esto se consigue usando la propiedad default del elemento property dentro del modelo de Tarea:

```
<property name="bpm:packageActionGroup">
  <type>d:text</type>
  <default>add_package_item_actions</default>
</property>
```

En la distribución de Alfresco están disponibles los siguientes "Grupo de Acciones":

- **read_package_item_actions:** Permite ver los items del paquete.
- **edit_package_item_actions:** Permite modificaciones (edit, checkout, ...) .
- **edit_and_remove_package_item_actions above:** Permite eliminación de items.
- **remove_package_item_actions:** Permite eliminación (pero no modificacion).
- **add_package_item_actions:** Permite añadir items al paquete.

Están todos definidos en el fichero de configuración *web-client-config-workflow-actions.xml* el cual proporciona la definitiva lista de acciones para cada Grupo. Existe soporte para la construcción de "Grupos de Acciones" personalizados.

Flujo de Trabajo Adhoc

Para el ejemplo de las tareas del Workflows Adhoc, se podría definir el siguiente modelo de contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<model name="wf:workflowmodel" xmlns="http://www.alfresco.org/model/dictionary/1.0">
```

```
<imports>
  <import uri="http://www.alfresco.org/model/dictionary/1.0" prefix="d"/>
  <import uri="http://www.alfresco.org/model/bpm/1.0" prefix="bpm"/>
</imports>
<namespaces>
  <namespace uri="http://www.alfresco.org/model/workflow/1.0" prefix="wf"/>
</namespaces>
<types>
  <type name="wf:submitAdhocTask">
    <parent>bpm:startTask</parent>

    <properties>
      <property name="wf:notifyMe">
        <type>d:boolean</type>
        <default>false</default>
      </property>
    </properties>
    <mandatory-aspects>
      <aspect>bpm:assignee</aspect>
    </mandatory-aspects>
  </type>
  <type name="wf:adhocTask">
    <parent>bpm:workflowTask</parent>
  </type>
  <type name="wf:completedAdhocTask">
    <parent>bpm:workflowTask</parent>
```

Se puede usar para describir Tareas todo el poder expresivo del Modelo de Contenidos. Por ejemplo, pueden ser asignados tipos de datos, restricciones y valores por defecto a las propiedades. Los Diálogos de las Tareas asociados se guiaran por esta definición. También se usa para encapsular los requisitos de información común.

El nombre del tipo en el modelo de contenido debe corresponder con el nombre de la tarea en la definición del proceso.

El modelo anterior define:

- Una tarea inicial llamada *wf:submitAdhocTask*.

La tarea define las propiedades que deben ser recogidas cuando el flujo comienza. En este caso, las propiedades son la descripción del workflow, la prioridad y la fecha de vencimiento (todas heredadas desde *bpm:starTask*) y *wf:notifyMe*. Esta tarea también incluye el aspecto *bpm:assignee* que permite la recogida de un asignatario (una sola persona) in el dialogo de la tarea inicial.

- Las tareas del flujo *wf:adhocTask* y *wf:completedAdhocTask*.

Estas tareas no definen nada extra a lo que ya heredan de la tarea básica *wf:workflowTask*.

Desplegar el Modelo de Tarea

Para desplegar un modelo de tarea podemos usar el bean "Workflow Deployer" que vimos antes.

```
<bean id="myworkflows.workflowBootstrap" parent="workflowDeployer">
  <property name="workflowDefinitions">
    ...
  </property>
  <property name="models">
    <list>
      <-- Task Model associated with above process definition -->
      <value>alfresco/workflow/adhocModel.xml</value>
    </list>
  </property>
</bean>
```

El Modelo de Tareas es solo un modelo de contenidos así que como alternativa podemos hacer el despliegue como cualquier otro modelo personalizado, aunque la recomendación es usar el "Workflow Deployer":

```
<bean id="adhocWorkflow.dictionaryBootstrap" parent="dictionaryModelBootstrap" depends-on="dictionaryBootstrap">
  <property name="models">
    <list>
      <value>alfresco/model/adhocTaskModel.xml</value>
```

```
    </list>
  </property>
</bean>
```

Definiciones de workflows avanzados por defecto

La definición de los workflows "Review & Approve" y "Ad-hoc Task" proporcionados por defecto están en el directorio de configuración de Alfresco alfresco/workflow.

- *review_processdefinition.xml* - Definicion de Proceso para "*Review & Approve*"
- *adhoc_processdefinition.xml* - Definicion de Proceso para "*Adhoc Task*"
- *workflow-messages.properties* - Fichero de recursos para ambos
- *workflowModel.xml* - Modelo de Tareas para ambos.

Ambos están configurados en el fichero de inicialización *alfresco/bootstrap-context.xml* en un bean llamado *workflowBootstrap*.

Paso 3: Añadir comportamiento a la Definición de Procesos

Una vez especificados los requerimientos de datos, es momento de volver a la definición del proceso y mejorarla añadiendo comportamientos (p.e. describir el flujo de datos, la asignación de tareas y las acciones del repositorio que realizar). Alfresco soporta todas las características de jDPL

Todos los procesos de datos (incluidos los especificados por el modelo de tarea) son guardados y expuestos como actions scripts de jBPM. Para manipular la tareas usando scripts se debe entender primero como se representa en jBPM.

Las propiedades estandar de una tarea se mapean como sigue:

bpm:taskId <--> taskInstance.id

bpm:description <--> taskInstance.description

bpm:startDate <--> taskInstance.start

bpm:dueDate <--> taskInstance.dueDate

bpm:completionDate <--> taskInstance.end

bpm:priority <--> taskInstance.priority

bpm:comment <--> taskInstance.comments.get(0).message

cm:created <--> taskInstance.create

cm:owner <--> taskInstance.actorId

Todo el resto de propiedades y asociaciones son mapeadas como variables jBPM usando la siguiente convención de nombres:

1. Modelo de Tarea - : e.j. bpm:assignee, wf:notifyMe
2. Variable de Tarea - _ e.j: bpm_assignee, wf_notifyMe

Si la variable no existe ya, la propiedad tiene el valor por defecto especificado en el modelo (o nulo si no existe valor por defecto).

Por defecto, cuando una variable de tarea es convertida en una variable de procesos y viceversa, el mismo nombre es usado. Por ejemplo, La descripición de la Start Task de Adhok especifica la propiedad *wf:notifyMe*. Cuando la Start Task de Adhoc finaliza, la variable *wf_notifyMe* es traspasada al proceso y como variable del proceso también se llamara *wf_notifyMe*.

Sin embargo, JBoss jBPM proporciona un mecanismo para controlar manualmente el mapeo entre variables de tarea y variables de proceso. Este mecanismo se llama task controller y permite especificar variables para ser mapeadas y especificar que nombres usar, e.j. mapear la variable de tarea *wf_notifyMe* a la variable de proceso *notify_via_email*.

```
<task>
    <controller>
      <variable name="notify_via_email" access="write" mapped-name="wf_notifyMe"/>
    </controller>
  </task>
```

El tipo de cada variable de tarea es deifinido por la descripicón de la misma (los tipos de datos de las respectivas propiedades o en caso de las asociaciones una colección. Muchos tipos de datos de Alfresco están convenientemente mapeados en jBPM(*d:text a string, d:boolean a boolean*). De cualquier forma hay un tipo especial de soporte para el tipo *d:noderef* y las asociaciones que requiere mas atención.

Las referencias a nodos y a asociaciones del Repositorio de Alfresco son representadas como un tipo especial el objeto Alfresco Node en JBoss jBPM. Un objeto Alfresco Node es una vista orientada a objecto de un item del Repositorio de Alfresco. Proporciona funcionalidades para recuperar y fijar propiedades y recorrer asociaciones así como metodos para realizar checkout/in, transformaciones etc. Es realmente potente cuando se combina con the capacidades de scripting del Beanshell y los Javascript de Alfresco disponibles en la definición de procesos. Con esto es posible controlar el flujo basandonos en metadatos del repositorio y también realizar automatismos comunes del repositorio usando scripts como parte del workflow.

Las siguientes variables de proceso están disponibles en todos los Workflows:

- **initiator:** Un Nodo del Repositorio (tipo cm:person) que representa la persona que inicializa el workflow
- **initiatorhome:** Un Nodo del Repositorio (tipo cm:space) que representa el home folder de la persona que inicia el workflow
- **companyhome:** Un Nodo del Repository (tipo cm:space) que representa el espacio company home.
- **cancelled:** Un flag que indica cuando el workflow ha sido cancelado.

Las siguientes variables de Proceso estarán también disponibles una vez que la Start Task haya finalizado:

- **bpm_workflowDescription:** La descripcion para este workflow particular (como se define en le start task base)
- **bpm_workflowDueDate:** La fecha de vencimiento para todo workflow (como se define en le start task base)
- **bpm_workflowPriority:** La prioridad para todo el workflow (como se define en le start task base)
- **bpm_package:** Un Nodo de Repositorio (tipo bpm:workflowPackage) que representa el paquete Workflow que contine los contenidos que son enviados por el workflow
- **bpm_context:** Un Nodo del Repositorio (tipo cm:folder) que representa el espacio de Alfresco en el cual se inicio el workflow.

Asignación de tareas

El componente de identificación de JBoss jBPM no esta aún integrado en la gestión de Usuarios de Alfresco y por eso no puede se usado para la asignación de tareas y el swimlane. En cualquier caso, la asignacion se puede hacer usando el atributo *actor-id* de los elemento *swimlanes* and *tasks.*

Por ejemplo, en nuestro Workflow de ejemplo, la Tarea *Submit Adhoc* define una asociacion (a *cm:person*) that representa el *Asiggnee.* El iniciador selecciona una persona en el dialogo de la tarea *Submit Adhoc* el cual es guardado como una variable en la Tarea (llamada *bpm_assignee*). Cuando la tarea finalice, la variable assignee se integrará en el proceso.La variable de proceso llamada "*bpm_assignee*" is ahora accesible en a traves de script jPDL y expresiones de asignacion.

Cabe recordar que las referencias a People de Alfresco son representadaa como objetos Node y por tanto pueden ser examinadas para obtener cualquier propiedad de la persona incluyendo su nombre o dirección de correo electrónico. La siguiente asignación swimlane lo muestra:

```
<swimlane name="assignee">
  <assignment actor-id="#{bpm_assignee.properties['cm:userName']}"/>
</swimlane>
```

Acciones y ejecución de Scripts

Los Scripts y las Acciones de jPDL permiten invocar logica de negocio durante la ejecución de un workflow. La logica de negocio puede ser expesada en Java, jPDL Beanshell o JavaScript de Alfresco. Los tres lenguajes tienen acceso a variables de Tareas y de Procesos.

La sintaxis para incluir Alfresco JavaScript en un Workflow es:

```
<action class="org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">
  <script>
    // some alfresco javascript
  </script>
</action>
```

o

```
<action class="org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">
  <script>
    <expression>
      // some alfresco javascript
    </expression>
    <variable name="passedInValue" access="read"/>
    <variable name="passedOutValue" access="write"/>
  </script>
</action>
```

Los JavaScript Alfresco tienen acceso a todas las variables de proceso y tarea (si procede), a menos que el elemento sea proporcionado, in cuyo casi, las variables se pueden pasar de dentro y fuera del script.

Así, en el Workflow que usamos como ejemplo, un correo electronico se envía usando código Javascript (los items en negrita son referencias a variables de proceso:

```
<transition name="" to="completed">
  <action class="org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">
```

```
    <script>
      if (wf_notifyMe)
      {
        var mail = actions.create("mail");
        mail.parameters.to = initiator.properties["cm:email"];
        mail.parameters.subject = "Adhoc Task " + bpm_workflowDescription;
        mail.parameters.from = bpm_assignee.properties["cm:email"];
        mail.parameters.text = "It's done";
        mail.execute(bpm_package);
      }
    </script>
  </action>
</transition>
```

Se puede navegar y realizar operaciones sobre las variables que representan Nodos del Repositorio Nodes usando la API de Nodos. Esta API esta disponible tanto para los Javascript de Alfresco como para los jPDL Beanshell.

Tambien,para JavaScript de Workflow de Alfresco,están disponibles los siguientes objetos "root":

- A partir de las versiones 1.4 Enterprise y 2.0: objeto person (Nodo (tipo cm:person) que representa la persona que ejecuta los script), objeto userhome (Nodo (tipo cm:folder) que representa el espacio raíz de la persona que ejecuta los script.
- Para las versiones 1.4 Community y Enterprise: objeto search (Un objeto host que proorciona acceso a Lucene y resultados de busquedas almacenadas), objeto actions (Un objeto host que proporciona para la invocación de Acciones de Alfresco registradas) , objeto logger (Un objeto host que proporciona acceso a utilidades de la consola de log para depuración de scripts), objeto session (Información relativa la ticket de autenticacion actual), objeto classification (Acceso a los elementos raiz de la API de clasificación),objeto utils (Acceso a una librería de utilidades)

Bifurcacion For Each

En algunos escenarios, es necesario dar soporte a bifurcaciones donde el numero de caminos se conocerá solo en tiempo de ejecución. Por ejemplo, un proceso de revisión paralela donde la tarea de revisión es destinada a 'N' personas.

Para dar soporte a esta clase de escenarios, se proporciona una acción "ForEachFork":

```
<node name="startreview">
  <action class="org.alfresco.repo.workflow.jbpm.ForEachFork">
    <foreach>collection</foreach>
    <var>variable</var>
  </action>
</node>
```

En este ejemplo *collection* será una expresión Javascript con la foma #expresion que devuelva una colección de cualquier tipo. Para cada entrada de la colección se bifurcará un camino. Variable, por su parte, será el nombre de la variable a crear en cada bifurcación que tiene la entrada de la colección. Como cualquier bifurcación simlpe jPDL, un ForEachFork debe ser acompañado con un Join. Una revisión paralela simple puede ser descrita como sigue:

```
<task-node name="review">
  <task name="wf:reviewTask">
    <assignment class="org.alfresco.repo.workflow.jbpm.AlfrescoAssignment">
      <actor>#{reviewer}</actor>
    </assignment>
  </task>
  <transition name="reject" to="endreview" />
  <transition name="approve" to="endreview">
</task-node>
<join name="endreview">
  <transition to="isapproved" />
</join>
```

Temporizadores

Un temporizador de workflow proporciona la habilidad para lanzar una determinada acción o transición dentro del flujo en un momento concreto del futuro (absoluto o relativo después de la entrada de un nodo en un workflow). Los temporizadores son útiles para implementar procedimientos escalados (ej. un usuario A no ha completado una tarea B, por lo tanto notificamos al usuario C) y tareas retrasadas (ej. subir un contenido a una pagina en una semana).

Un temporizador puede ser asociado con cualquier nodo en una definición de workflow, aunque son particularmente útiles en los nodos de Tarea.

Este es un de como asociar un temporizador a una tareas:

```
<task-node name="submitpending">
  <task name="wcmwf:submitpendingTask" swimlane="initiator">
    <timer duedate="#{wcmwf_launchDate}" transition="launch" >
      <action class="org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">
        <script>
          logger.log("WCM Submission submitted at " + wcmwf_launchDate);
        </script>
      </action>
    </timer>
  </task>
  <transition name="cancel" to="submitcancelled" />
  <transition name="launch" to="submitted" />
</task-node>
```

El atributo *duedate* puede ser tiempo relativo jPDL(ej. "3 days") o objeto de fecha absoluta. El atributo *duedate* puede ser fijado desde un variable de workflow (ej. #wcmwf_launchDate). El temporizador puede tener una acción asociada que se ejecute cuando es disparado. El temporizador puede tener una transacción asociada que se siga cuando este sea disparado. Si se sale explicitamente de la tarea antes de que el temporizador sea lanzado, este será destruido.

Flujo de Trabajo Adhoc

Esta es la versión mejorada de nuestro ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="urn:jbpm.org:jpdl-3.1" name="wf:adhoc">
  <swimlane name="initiator"/>
  <start-state name="start">
    <task name="wf:submitAdhocTask" swimlane="initiator"/>
    <transition name="" to="adhoc"/>
  </start-state>
  <swimlane name="assignee">
    <assignment actor-id="#{bpm_assignee.properties['cm:userName']}"/>
  </swimlane>
  <task-node name="adhoc">
    <task name="wf:adhocTask" swimlane="assignee">
      <event type="task-create">
        <script>
          if (bpm_workflowDueDate != void) taskInstance.dueDate = bpm_workflowDueDate;
          if (bpm_workflowPriority != void) taskInstance.priority = bpm_workflowPriority;
        </script>
      </event>
    </task>
    <transition name="" to="completed">
      <action class="org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">
        <script>
          if (wf_notifyMe)
          {
            var mail = actions.create("mail");
            mail.parameters.to = initiator.properties["cm:email"];
            mail.parameters.subject = "Adhoc Task " + bpm_workflowDescription;
            mail.parameters.from = bpm_assignee.properties["cm:email"];
```

Veamos las mejoras que se han realizado. El swimlane '*assignee*' se ha mapeado con la variable de proceso *bpm_assignee*. Se ha usado un evento *task-create* para inicializar los valores de fecha de vencimiento y prioridad de la tarea. Se usa el evento de transición completa de la tarea adhoc para llamar a un javascript de Alfresco que envíe un email. Se usa el evento *procces-end* para guardar en el log el final de la tarea.

Crear un Fichero de Recursos para el workflow

Para internacionalizar la interacción con los workflows es necesario proporcionar un Fichero de Recursos en el que se encuentren las etiquetas para la interfaz de usuario de cada uno de los textos que allí se muestren.

Se usa el mismo modelo de localización de Alfresco para proporcionar las etiquetas que necesitamos.

Otras etiquetas de workflow son soportadas con las siguientes keys.:

- <workflow_prefix>_<workflow_name>.workflow.[title|description]
- <workflow_prefix>_<workflow_name>.node.<node_name>.[title|description]
- <workflow_prefix>_<workflow_name>.node.<node_name>.transition.<transition_name>.[title|description]
- <workflow_prefix>_<workflow_name>.task.<task_prefix>_<task_name>.[title|description]

donde:

- <workflow_prefix> es el prefijo del namespace del modelo de workflow
- <workflow_name> es el nombre del workflow
- <node_name> es el nombre de un nodo del workflow
- <transition_name> es el nombre de un nodo de transición en el workflow
- <task_prefix> es prefijo del namespace de la tarea
- <task_name> es el nombre de la tarea
- <transition_name> es el nombre de la transición del workflow.

Para el ejemplo el Fichero de Recursos sería algo parecido a:

```
#
# Adhoc Task Workflow
#
wf_adhoc.workflow.title=Adhoc Task
wf_adhoc.workflow.description=Assign task to colleague
# Adhoc Task Definitions
wf_workflowmodel.type.wf_submitAdhocTask.title=Submit Adhoc Task
wf_workflowmodel.type.wf_submitAdhocTask.description=Allocate task to colleague
wf_workflowmodel.property.wf_adhocDescription.title=Task Description
wf_workflowmodel.property.wf_adhocDescription.description=Description of what needs to be achieved
wf_workflowmodel.property.wf_adhocDueDate.description=Task Due Date
wf_workflowmodel.property.wf_adhocPriority.title=Task Priority
wf_workflowmodel.property.wf_notifyMe.title=Notify Me
wf_workflowmodel.property.wf_notifyMe.description=Notify me when task is complete
wf_workflowmodel.association.wf_assignee.title=Assignee
wf_workflowmodel.association.wf_assignee.description=Who's doing the task
wf_workflowmodel.type.wf_adhocTask.title=Adhoc Task
wf_workflowmodel.type.wf_adhocTask.description=Adhoc Task allocated by colleague
wf_workflowmodel.type.wf_completedAdhocTask.title=Adhoc Task Completed
wf_workflowmodel.type.wf_completedAdhocTask.description=Adhoc Task Completed
```

Los recursos de Internacionalización son registrados usando el bean "Workflow Deployer":

```
<bean id="myworkflows.workflowBootstrap" parent="workflowDeployer">
  <property name="workflowDefinitions">
    <list>
      ...
    </list>
  </property>
  <property name="models">
    <list>
      ...
    </list>
  </property>
  <property name="labels">
    <list>
      <-- Resource Bundle associated with above process definition -->
      <value>alfresco/workflow/adhoc-messages</value>
    </list>
  </property>
</bean>
```

Alternativamente se pueden registrar con el modelo de Tareas:

```
<bean id="adhocWorkflow.dictionaryBootstrap" parent="dictionaryModelBootstrap" depends-on="dictionaryBootstrap">
  <property name="models">
```

```
    <list>
      ...
    </list>
  </property>
  <property name="labels">
    <list>
      <value>alfresco/messages/adhoc-messages</value>
    </list>
  </property>
</bean>
```

Configuración de los diálogos de tareas en el Web Client

Los Diálogos de Tareas de Workflow se configuran usando el mecanismo de configuración del Cliente Web de Alfresco existente. Como las tareas son definidas usando el Diccionario de Datos las mismas técnicas de configuración se aplican aquí. Por tanto,para obtener la información requerida pata la *wf:submitAdhocTask* definida en el ejemplo se usará la siguiente configuración:

```
<config evaluator="node-type" condition="wf:submitAdhocTask" replace="true">
   <property-sheet>
      <separator name="sep1" display-label-id="general" component-generator="HeaderSeparatorGenerator" />
      <show-property name="bpm:workflowDescription" component-generator="TextAreaGenerator" />
      <show-property name="bpm:workflowPriority" />
      <show-property name="bpm:workflowDueDate" />
      <show-property name="wf:notifyMe" />
      <separator name="sep2" display-label-id="users_and_roles" component-generator="HeaderSeparatorGenerator" />
      <show-association name="bpm:assignee" />
   </property-sheet>
</config>
```

El Asistente de Inicio de Workflow usa la configuración anterior para mostrarlos controles adecuados para recoger la información del usuario. El Dialogo de Gestión de Tareas usas el mismo enfoque para mostrar los datos que necesita recoger. Así para la wf:adhocTask se utiliza la siguiente configuración:

```
<config evaluator="node-type" condition="wf:adhocTask" replace="true">
   <property-sheet>
      <separator name="sep1" display-label-id="general" component-generator="HeaderSeparatorGenerator" />
      <show-property name="bpm:taskId" />
      <show-property name="bpm:workflowDescription" component-generator="TextAreaGenerator" />
      <show-property name="bpm:status" />
      <show-property name="bpm:dueDate" />
      <show-property name="bpm:priority" />
   </property-sheet>
</config>
```

Si el Dialogo de Gestión de Tareas estandar no es suficiente puede ser sustituido por uno personalizado.

Resumen

En resumen los paso necesarios para la creación de un Flujo de Trabajo Avanzado personalizado serán:

1. Modelar el proceso usando el Diseñador de Procesos jBPM o a mano escribiendo un fichero xml. Centrarse en describir el proceso correctamente.
2. Definir el modelo de contenido del workflow.
3. Añadir lógica usando expresiones beanshell, JavaScript o Java.
4. Crear/actualizar un fichero de propiedades especifico del workflow para externalizar las cadenas en el modelo del workflow y en la definición del proceso.
5. Actualizar el fichero web-client-config-custom.xml para permitir que el cliente web muestre las tareas del workflow.
6. Desplegar la definición del proceso. (Usando configuraciones Spring o desde la pestaña de despliegue de Eclipse).

Nuevos Objetos para motor JavaScript

- **Área:** Extensión de Alfresco
- **Carácter del recurso:** Recomendado

Código: RECU-0055
Tipo de recurso: Manual

Descripción

Es posible crear y añadir Script API's personalizados implementados en Java y accesible como objetos raíz en JavaScript. Este proporciona un punto de integración para extensiones de Alfresco que proporcionar API'S a medida.

Definición del Objeto

Extender la Clase

El primer paso sera realizar una clase en nuestro proyecto que extienda de BaseProcessorExtension, en la cual definiremos los métodos públicos para que sean accesibles mediante el objeto que creemos en JavaScript.

```
public final class test extends BaseProcessorExtension
{....}
```

Definición de objeto ServiceRegistry

Es necesario implementar este objeto dado que es el que nos proporcionara cualquier servicio que se requiera .

```
private ServiceRegistry serviceRegistry;

public void setServiceRegistry(ServiceRegistry services)
   {
      this.serviceRegistry = services;
   }
```

Definición del método público

Definiremos un metodo público el cual sera accesible desde el objeto creado en JavaScript,el cual realizara las operaciones que se definan en el mismo. Este método nos devolverá una array de String en la cual estarán los documentos existentes en el directorio Company Home

```
public String[] ObtenerDocumentos() throws Exception
{......}
```

Definir Bean de Configuración

El siguiente paso sera realizar un archivo de configuración xml en el cual se definirá el bean correspondiente para poder agregar el objeto al motor de JavaScript

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE beans PUBLIC '-//SPRING//DTD BEAN//EN' 'http://www.springframework.org/dtd/spring-beans.dtd'>
<beans>
<bean id="IDENTIFICADORDELBEEN" parent="baseJavaScriptExtension" class="<Package>.<Nombre de la clase>">
          <property name="extensionName">
             <value><NOMBREOBJETO></value>
          </property>
          <property name="serviceRegistry">
             <ref bean="ServiceRegistry"/>
          </property>

   </bean>
</beans>
```

En nuestro caso y siguiendo el ejemplo se definiría de la siguiente manera:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE beans PUBLIC '-//SPRING//DTD BEAN//EN' 'http://www.springframework.org/dtd/spring-beans.dtd'>
<beans>
<bean id="Documentos" parent="baseJavaScriptExtension" class="es.alfresco.test">
          <property name="extensionName">
```

```
            <value>test</value>
        </property>
        <property name="serviceRegistry">
            <ref bean="ServiceRegistry"/>
        </property>


    </bean>
</beans>
```

Generar libreria Jar

Se realizara la generación de la libreria para despues realizar la instalación en alfresco.

Acceso al nuevo objeto mediante JavaScript

Instalación en Alfresco

Para que el objeto sea accesible debemos de copiar la clase generada con en el directorio :

```
<Maquina Local>\<Directorio Apache>\<Directorio Tomcat>\webapps\alfresco\WEB-INF\lib
```

Acceder al objeto mediante JavaScript

Para acceder a las clases públicas definidas en nuestra libreria lo realizaremos mediante el objeto definido, bastara con llamar al nombre del objeto seguido de un punto y el metodo requerido.

```
nombreobjeto.metodo();
```

En nuestro caso y siguiendo el ejemplo se realizaria la llamada así:

```
test.ObtenerDocumentos();
```
