Magnolia CMS Integration Guide

Note: this is a DRAFT

Introduction

Note: The following Integration Guide is aimed at OFBiz Developers with a Basic Knowledge of the actual System.

The following approach is meant to implement the Magnolia CMS (http://www.magnolia-cms.com) on top of the OFBIz System. I used the latest Maven Install of Magnolia (3.7) and Trunk Revision 711975 of OFBiz, there is no guerantee that the approach will work on any other Combination of System Versions, but taken from how this is done, it may as well work 😉.

This method works as the following: Magnolia remains a pure content provider and will render the data by its own means - OFBiz will fetch the data and include it to its current context (hence, this approach could be used to render any other content outside the ofbiz system).

Prerequisits

Installation of OFbiz

I recommend downloading and installing the latest trunk

Installation of Magnolia

I used the latest Maven Installation (see Magnolia Maven Install Reference for Installation) and deployed the war file onto the Ofbiz System (remember to add and edit the ofbiz_component file; For further detail on this matter take a look at Add a WAR file)

Integration Steps

Step 1 Add a service definition file to your application

(Skip if a service.xml file is already existent)

Create a new service.xml file in your appication directory (i created it in an extra folder: servicedef) and add the very same to the ofbizcomponent.xml file --> e.g. by adding:

```
<service-resource type="model" loader="main" location="servicedef/services.xml"/>
```

Step 2 Create the Services to collect the Magnolia data

Open the service.xml file and add:

```
[...]
<\!-\- CMS Services \-->


<service name="runCMSQuery" engine="java"
transaction-timeout="72000"
location="org.brandsparadise.cms.Magnolia"
invoke="runCMSQuery" debug="true" validate="true">
<description>Run a query on CMS and return the results</description>
<attribute mode="IN" name="query" optional="false"
type="String" />
<attribute name="queryResult" type="String" mode="OUT" optional="true"/>

</service>

[...]
```

(take a look at any other service.xml file on the exact layout of a service.xml file)

Step 3 Get OFBiz to load a java source

Create a new src folder and load the sources via ofbiz-component.xml:

```
<classpath type="dir" location="src/*" />
```

Step 4 Create the method classes to fetch the content

Create a new package and call it any name you like (I called mine: org.brandsparadise.cms). Add a class to it (i called mine: Magnolia.java) and add something along the line of :

```
package org.brandsparadise.cms;

import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.util.Map;
import java.net.*;
import junit.framework.TestCase;

import org.ofbiz.service.DispatchContext;
import org.ofbiz.service.ServiceUtil;

/*\*
* Base class for OFBiz Test Tools test case implementations.
\*/

public abstract class Magnolia extends TestCase {

public static final String module = Magnolia.class.getName();
public static String cmsUrl = "http://www.google.com";

/*\*
* runs a query on a url and return the results
\*
\*/
public static Map runCMSQuery(DispatchContext dctx, Map context) {

try {
// get Connection
StringBuilder result = new StringBuilder();
byte\[\] buffer = new byte[8192];

InputStream s;

URL url = new URL(cmsUrl + context.get("query").toString());
s = url.openStream();

int size = 0;

do
{                size = s.read(buffer);                if (size != -1)                    result.append(new
String(buffer, 0, size));             }
while (size \!= \-1);
context = ServiceUtil.returnSuccess();
context.put("queryResult", result.toString());

} catch (MalformedURLException e)
{             // TODO Auto-generated catch block             e.printStackTrace();             context =
ServiceUtil.returnError(e.toString());          }
catch (IOException e)
{             // TODO Auto-generated catch block             e.printStackTrace();             context =
ServiceUtil.returnError(e.toString());         }
// Debug.log("Query Result: "+results.toString());
return context;
}

public static void main(String\[\] args)
{         junit.textui.TestRunner.run(Magnolia.class);         System.exit(0);     }
}
```

It is important to know that you need to edit the cmsUrl to your url of choice. I for my part have two instances of magnolia installed at cms and cmspublic (why 2? Well, thats the way Magnolia works - you need to have an Author and a Public instance), so I redirect my calls to "http://localhost:8080/cmspublic".

Step 5 Which further steps do I need to consider?

This setup so far will fetch any content from the url defined at cmsUrl. You will still need to create a groovy script of your own that will pass the given site parameters (defined as query) to the the "runCMSQuery"-service and then display the results. This can be done through a script in a groovy/bsh file similar to the following:

```
private String getContentFromUrl(String contentUrl){
        Map context = UtilMisc.toMap("query",contentUrl);
        Map result = null;

         try {
          result = dispatcher.runSync("runCMSQuery", context);
        } catch (GenericServiceException e) {
     e.printStackTrace();
        }
return (String) result.get("queryResult");
}
```

The code above will, once called, affectively parse a url defined as contentUrl to the service and return the result, so that you can display it on your site. For instance, if you wish to display the content located at http://localhost:8080/cmspublic/main.html in your freemarker template file (.ftl) you will need to call the function getContentFromUrl in a way similar to this at the end of your groovy file and add the results to the context:

```
context.put("cms",getContentFromUrl("main.html"));
```

Left to be Done

The above should fetch and render any content available. Of course, this is only a basic step to what you wish to accomplish. If you actually want to further integrate the system, I recommend to create XML files in Magnolia in which the content is moved between numbered boxes (remember to use the <![CDATA[Placeholder]]> brackets if you wish to include html content in the xml file) and have OFBiz handle these xml data files accordingly...

Though this is a working solution, I am not 100% satisfied with this approach, as Magnolia is left as an independent platform. There is a way to integrate Magnolia even further, but it would require the content to be access through both magnolias' and Ofbiz' filter chains...