Single Instance Deployment

Your Rating: ☆☆☆☆☆ Results: ★★★☆☆ 123 rates

For some installations the authoring/public default way of running Magnolia may not be possible. This could for instance happen when trying to add Magnolia onto an already existing site with lots of functionality that will continue to be developed in parallel to running the static content pages off the CMS. Duplicating the site structure with css styles would result in a maintenance headache in terms of versioning and keeping webapps in sync.

Luckily Magnolia is flexible enough to work also in a single instance but there are some considerations and adaptations that can smooth it out.

Our setup is:

- Maven standard site (src/main/webapp, src/main/java, src/main/resoures, pom.xml)
- We use jetty:run to test run it.
- Plenty of existing URL-space that needs to keep working.

Modifying pom.xml.

We need to include the magnolia jars in our webapp. Sadly magnolia jars are not available through the central repo - so we need to add magnolia's repo. The list of included jars is what we wanted - another installation may require less or others.

```
  <repositories>
    <repository>
      <id>magnolia</id>
      <name>Magnolia</name>
      <url>http://repo.magnolia.info/m2/</url>
    </repository>
  </repositories>
...
  <properties>
    <magnolia-version>3.6.3</magnolia-version>
  </properties>
...
  <dependencies>
    <!-- Templating is how you format/display content in Magnolia -->
    <dependency>
      <groupId>info.magnolia</groupId>
      <artifactId>magnolia-module-templating</artifactId>
      <version>${magnolia-version}</version>
    </dependency>
    <!-- Core is the JCR-integration amongst other fundamentals -->
    <dependency>
      <groupId>info.magnolia</groupId>
      <artifactId>magnolia-core</artifactId>
      <version>${magnolia-version}</version>
    </dependency>
    <!-- Admin interface is what shows up under
    .magnolia/pages/adminCentral.html. Note that it depends on -gui
    for components that make up the UI. -->
    <dependency>
      <groupId>info.magnolia</groupId>
      <artifactId>magnolia-module-admininterface</artifactId>
      <version>${magnolia-version}</version>
    </dependency>
    <!-- Caching of content being dragged out of Magnolia -->
    <dependency>
      <groupId>info.magnolia</groupId>
      <artifactId>magnolia-module-cache</artifactId>
      <version>${magnolia-version}</version>
    </dependency>
    <!-- The plugin that exports/imports content from authoring to
    public instance. Although we're not using this, we need it for the
    page activation functionality. -->
    <dependency>
      <groupId>info.magnolia</groupId>
      <artifactId>magnolia-module-exchange-simple</artifactId>
      <version>${magnolia-version}</version>
    </dependency>
    <!-- The fckeditor is the rich text editor. -->
    <dependency>
      <groupId>info.magnolia</groupId>
      <artifactId>magnolia-module-fckeditor</artifactId>
      <version>${magnolia-version}</version>
    </dependency>
    <!-- The cms: taglib is what we mostly use to drag out content of
    Magnolia. -->
    <dependency>
      <groupId>info.magnolia</groupId>
      <artifactId>magnolia-taglib-cms</artifactId>
      <version>${magnolia-version}</version>
    </dependency>
    <!-- The cmsu: taglib has that oh so handy scaleImage tag amongst
    others. -->
    <dependency>
      <groupId>info.magnolia</groupId>
      <artifactId>magnolia-taglib-utility</artifactId>
      <version>${magnolia-version}</version>
    </dependency>
    <!-- This package has a servlet that is used to configure log4j in
    the admin interface. It enables us to chage log4j configs runtime,
    which is nice. -->
    <dependency>
      <groupId>net.sourceforge.openutils</groupId>
      <artifactId>openutils-log4j</artifactId>
      <version>1.0.1</version>
    </dependency>
  </dependencies>
```

web.xml

Magnolia only requires a listener and one filter in web.xml. After Magnolia gets hold of the request/response object it does a lot of internal dispatching to further filters and servlets that it has mapped inside itself.

NOTE Magnolia is mapped to /* and we have it first of all filters in web.xml. This means every request goes through Magnolia and this is a GOOD THING™. Magnolia sets up some stuff in the request object which is required to use the <cms:blah> taglib (though this doesn't happen for bypassed URLs - obviously).

```
...
        <listener>
          <listener-class>info.magnolia.cms.servlets.MgnlServletContextListener</listener-class>
        </listener>
...
       <filter>
          <display-name>Magnolia global filters</display-name>
          <filter-name>magnoliaFilterChain</filter-name>
          <filter-class>info.magnolia.cms.filters.MgnlMainFilter</filter-class>
        </filter>
        <filter-mapping>
          <filter-name>magnoliaFilterChain</filter-name>
          <url-pattern>/*</url-pattern>
          <dispatcher>REQUEST</dispatcher>
          <dispatcher>FORWARD</dispatcher>
          <dispatcher>ERROR</dispatcher>
        </filter-mapping>
...
```

Start the server.

At this point we start up our Magnolia infused webapp. When pointing the browser to it, Magnolia intercepts us and tells us that we need to install a lot of modules - which we happily do. Once finished we can click through to the Magnolia admin user inteface (u/p superuser/superuser).

URL Space

Now that Magnolia is running on /* we can't use any of our old web application URL space. In our case we had plenty of struts actions being mapped to *.do as well as css, javascript. To get our original functionality back we need to start mapping the "old" URL-space back.

Caveat We were in the fortunate position of having no URLs ending ".jsp" - I haven't tried, but I suspect such URL space may require some additional workarounds.

To restore the old URL space we go inside Magnolia: Configuration > server > filters > bypasses

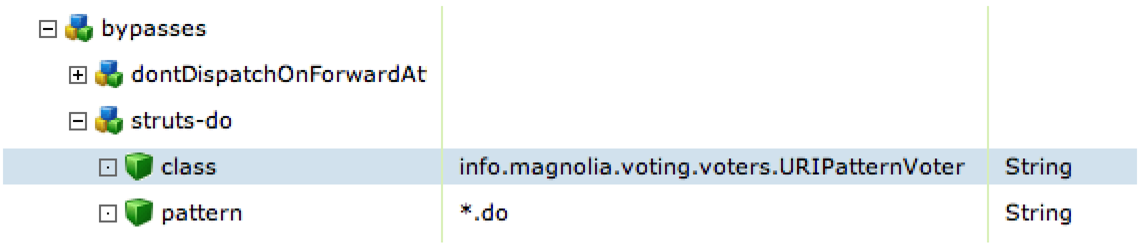
Here we find there is already one bypass "dontDispatchOnForwardAttribute". It's Magnolia internals, we add our bypasses after that. Each bypass needs a new content node with some nice name (such as "struts-do"). Each content node needs a node data "class" that tells the name of a "voter" of which Magnolia has a bunch of different ones. Check the API dock

The ones I find most useful are:

- info.magnolia.voting.voters.URIPatternVoter
- info.magnolia.voting.voters.URIStartsWithVother

Both of which requires an additional node data 'pattern' to configure them.

Example:



⊟ bypasses		
⊞ dontDispatchOnForwardAt		
⊟ struts-do		
class	info.magnolia.voting.voters.URIPatternVoter	String
pattern	*.do	String

WORD OF CAUTION It's very easy to shoot yourself in the foot with the bypass functionality. If you add a new bypass URIPatternVoter and set the class BEFORE you add the pattern node, Magnolia is shafted with a NPE (since it immediately tries to instantiate the new voter which doesn't find the config, and fails). Likewise if you for some reason write just a "*" as the pattern you are now bypassing Magnolia itself - also not a good place to be. We find it easiest to enter the class node with a 'xclass' and remove the 'x' first when we're sure the pattern is good.

From this point we start mapping up our URL space by using Safari's
excellent "Activity" window we find the blocked URLs one by one.

```
*.do
/img/*
/images/* (don't ask)
/css/*
/js/*
```

And then we add URIPatternVoters for all of these until we gradually get our old application back.

Page Activation

The standard Magnolia way is to have an authoring instance that publishes the activated pages using a subscriber onto the public instance. We don't want that, but we still want pages to be active (visible to the public) or not - we want Magnolia's activate functionality, but not send the activations anywhere.

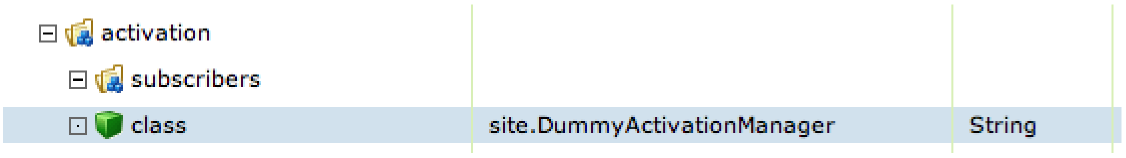
DummyActivationManager

If your instance doesn't have any subscribers, you will not even get the activate/deactivate buttons (and status indicators). To remedy this we create a dummy plug that tricks Magnolia into thinking there is something to activate when there isn't:

DummyActivationManager.java

```
public class DummyActivationManager implements ActivationManager {

    public void addSubscribers(Subscriber arg0) {
    }

    public String getConfigPath() {
        return "/server/activation/subscribers";
    }

    public Collection getSubscribers() {
        return Collections.EMPTY_SET;
    }

    public boolean hasAnyActiveSubscriber() {
        return true; // this is the trick
    }

    public void setSubscribers(Collection arg0) {
    }

}
```

In Configuration > Subscribers we find the "activation" node and change the class to be our dummy.



activation		
subscribers		
class	site.DummyActivationManager	String

At this point we get the activation buttons on pages that we create and we can set the status indicators to red or green.

Is page public?

When constructing templates in accordance with the standard Magnolia deployment, we use the <cms:adminOnly> and <cms:publicOnly> tags to switch on /off bits in the templates. When doing single instance deployment this is not enough - we want not activated pages to remain invisible to the public. There's no such support on Magnolias bundled tags, but it's not very hard to solve.

NOTE Doing installations this way means once a page is public and you make changes to it, the changes are visible immediately. Also the system doesn't really distinguish between green and yellow states since yellow (has local changes) doesn't make sense for single instance deployments.

What we need is:

- One tag to see if a user is an admin.
- One tag to check if a page is activated (I call it public).
- One tag to check if a page is not activated.

Here's three such tags done using tag file. I've put these files under WEB-INF/tags/site.

isadmin.tag

```
<%@ tag import="info.magnolia.context.MgnlContext"%>
<%@ tag import="info.magnolia.cms.security.*"%>
<%@ tag import="java.util.*"%>
<%@ tag body-content="scriptless" %>

<%--
  The user needs to be in one of the below defined ADMIN_ROLES.

  If user is, the body is executed.
--%>

<%!
  static String[] ADMIN_ROLES = {"superuser"};
%>

<%
    User user = MgnlContext.getUser();

    LinkedList inAdminRoles = new LinkedList(Arrays.asList(ADMIN_ROLES));
    inAdminRoles.retainAll( user.getRoles() );

    if ( !inAdminRoles.isEmpty() ) {
%>
  <jsp:doBody/>
<%
    }

%>
```

ifpublic.tag

```
<%@ tag import="info.magnolia.context.MgnlContext"%>
<%@ tag import="info.magnolia.cms.core.*"%>
<%@ tag import="info.magnolia.cms.security.*"%>
<%@ tag import="info.magnolia.cms.util.Resource"%>
<%@ tag import="java.util.*"%>
<%@ tag body-content="scriptless" %>
<%@ taglib prefix="site" tagdir="/WEB-INF/tags/site" %>

<%--
  This tag checks that the current Magnolia content node is activated as
  well as all parents are activated. If all are active, no restriction are
  put on the current user roles.

  If node is not activated or any of its parents, the user needs to be
  in one of the defined ADMIN_ROLES.

  If user is allowed, the body is executed.
--%>

<%!
  public boolean isAllParentsActivated( Content content ) throws Exception {

    if ( !content.getMetaData().getIsActivated() ) return false;

    if ( content.getLevel() > 1 ) {
      return isAllParentsActivated( content.getParent() );
    } else {
      return true;
    }

  }
%>

<%

  Content content = Resource.getCurrentActivePage();
  boolean activated = isAllParentsActivated( content );

  if ( activated ) {
%>
    <jsp:doBody/>
<%
  } else {
%>
    <site:isadmin>
      <jsp:doBody/>
    </site:isadmin>
<%
  }
%>
```

ifnotpublic.tag

```
<%@ tag body-content="scriptless" %>
<%@ taglib prefix="site" tagdir="/WEB-INF/tags/site" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%--
  The opposite of ifpublic.
--%>

<site:ifpublic>
  <c:set var="public" value="${true}"/>
</site:ifpublic>

<c:if test="${not public}">
  <jsp:doBody/>
</c:if>
```

Examples:

This I use to build a navigation of only public pages.

navigation.jsp

```
...
    <ul>
    <cms:loadPage path="/info"/>
    <cms:pageIterator>
      <site:ifpublic>
        <cms:out nodeDataName="path" var="path"/>
        <li><a href="${pageContext.request.contextPath}${path}.html">
          <cms:out nodeDataName="title"/>
        </a></li>
      </site:ifpublic>
    </cms:pageIterator>
    <cms:unloadPage/>
    </ul>
...
```

singlecolumnpage.jsp

```
...
  <site:ifnotpublic>
    <site:senderror code="404"/>
  </site:ifnotpublic>
...
  <h1><cms:out nodeDataName="title"/></h1>

  <cms:contentNodeIterator
       contentNodeCollectionName="contentParagraphs">

     <cms:editBar adminOnly="true" />

     <div class="cmsparagraph">
       <cms:includeTemplate />
     </div>

  </cms:contentNodeIterator>
```

Thanks to the Magnolia guys for making the software powerful enough to flex itself this way - amazing work.