Module Application Context File

This file allows modules to override/append to the current spring application context definition. It must be named moduleApplicationContext.xml and be located in the omod resources folder (mavenized modules) or the metadata folder (ant task modules).

If you have an ant task module and wish to provide a clean separation between your API- and Web-layer mappings, you can do this by putting your API-layer mappings in **moduleApplicationContext.xml** and your Web-layer mappings in **webModuleApplicationContext.xml**.

The doctype should be set to that of the current spring library's doctype. Currently, that is

```
<\!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
```

The mapping file must have a unique id associated with it.

In order to create your own module's service, the following bean must be used. This is unfortunately a lot of xml code to write, but there isn't a way around it. There are only a few tags at which values need to be manipulated:

```
<bean parent="serviceContext">
        <property name="moduleService">
                <list>
                        <value>org.openmrs.module.formentry.FormEntryService</value> <!-- Your service's
interface class -->
                        <bean class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
                                <property name="transactionManager"><ref bean="transactionManager"/></property>
                                <property name="target">
                                        <bean class="org.openmrs.module.formentry.impl.FormEntryServiceImpl"> <!
-- Your service's concrete class -->
                                                <property name="formEntryDAO"> <!-- Name of the DAO property on
your ServiceImpl -->
                                                        <bean class="org.openmrs.module.formentry.db.hibernate.
HibernateFormEntryDAO"> <!-- Your DAO's concrete class -->
                                                                <property name="sessionFactory"><ref bean="
sessionFactory"/></property>
                                                        </bean>
                                                </property>
                                        </bean>
                                </property>
                                <property name="preInterceptors">
                                        <ref bean="serviceInterceptors" />
                                </property>
                                <property name="transactionAttributeSource">
                                        <bean class="org.springframework.transaction.annotation.
AnnotationTransactionAttributeSource"/>
                                </property>
                        </bean>
                </list>
        </property>
</bean>
```

Redirects can be defined with this file. If we have a file in our formEntry module's web folder named formTaskpane.jsp. Without anything added to the spring context file, this could only be accessed via /openmrs/module/formEntry/formTaskpane.*htm*. However, if we add a mapping for this file like:

```
<bean id="formEntryUrlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
               <props>
                        <prop key="**/formTaskpane.htm">formTaskpaneRedirect</prop>
                </props>
        </property>
</bean>
```

and another bean like so:

```
<bean id="formTaskpaneRedirect" class="org.openmrs.web.controller.RedirectController">
        <property name="redirectView"><value>/module/formEntry/formTaskpane</value></property>
</bean>
```

then we can link to /openmrs/formTaskpane.htm and get redirected *internally* (unbeknownst to the user) to the right jsp file.

Overriding a jsp page

A module can override any page listed in the openmrs-servlet.xml file. Your moduleApplicationContext.xml file just has to define a prop with the same key as the original page. Your applicationContext file also has to have an "order" value lower than 99 so that spring will process your module before the core openmrs-servlet.xml file.

This example will override the encounter form that is shown on the administration page with a custom jsp from this module:

```
<bean id="myModuleMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
     <property name="order"><value>1</value></property>
     <property name="mappings">
        <props>
           <prop key="admin/encounters/encounter.form">encounterFormController</prop>
        </props>
     </property>
</bean>

<bean id="encounterFormController" class="@MODULE_PACKAGE@.web.controller.EncounterFormController">
     <property name="commandName"><value>encounter</value></property>
     <property name="formView"><value>/module/@MODULE_ID@/encounterForm</value></property>
     <property name="successView"><value>../../patientDashboard.form</value></property>
</bean>
```

Note: If it seems like your controller isn't being invoked, make sure you're not using *urlMapping* for the bean id. Instead, use something custom like *myMod uleMapping* above.

Overriding a jsp page with an Annotation-Based Controller

If you want to use an annotation based controller to override one of the built-in jsp pages you only have to make one simple change to your controller. The mapping looks similar:

```
<bean id="myModuleMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
 <property name="order"><value>10</value></property>
 <property name="mappings">
   <props>
     <prop key="admin/patients/newPatient.form">myNewPatientFormController</prop>
   </props>
 </property>
</bean>

<bean id="myNewPatientFormController" class="@MODULE_PACKAGE@.web.controller.MyNewPatientFormController" />
```

In your controller, you don't need to (aka "can't") specify the URL or else Spring complains about duplicate url mappings:

```
@Controller
public class MyNewPatientFormController {

 @RequestMapping(method=RequestMethod.GET)
 public String showThePage() { ... }

 @RequestMapping(method=RequestMethod.POST)
 public String savePatient(@ModelAttribute("patient") NewPatient newPatient) {  ... }

}
```

Adding Multiple Services

You can add several services via your module by simple adding more than one service beans. See reporting module's moduleApplicationContext for an example.

Modifying Beans

BeanFactoryPostProcessor should never be used in OpenMRS since it breaks our interceptors.

It's possible to modify beans which have been set up by the core or other modules. In order to do that you need to create a bean implementing **BeanPostProcessor**. You can see an example of that in the MDS module.

Notes

- A few words of warning: Spring sees all classes loaded by all modules when attempting to load the beans in your moduleApplicationContext.xml file. It is possible to run into classloading issues if a class referenced in one of your beans (or is a property of one of the classes you're loading through a bean) is the same class as a class from a different module when the two modules expect different versions of this class. To give an example, this problem was experienced as an incompatibility between the namephonetics and sync modules, where sync expected apache codecs 1.3, and namephonetics expected 1.4 (namephonetics now doesn't try to load the offending incompatible class through Spring). This manifested itself in the following error: "loader constraint violation: loader (instance of MyClass) previously initiated loading for a different type with name MyClass"