Creating Modules

The goal of the module system is to allow other developers to write and integrate code into OpenMRS without having to modify the core code base. Our online Add-ons index has been developed to facilitate searching and sharing of modules.

What's in a Module omod?

An omod is simply a renamed jar file. And a jar file is simply a special zip file. OpenMRS expects the file to contain a config.xml file in the root. The root lib folder should contain new/unique jars that the module is providing to the system. All messages.properties files should also be in the root folder.

Getting Started

See the Module Conventions page for existing conventions for naming your module, coming up with a module ID, creating module-specific tables, etc.

For a step-by-step walkthrough, see Creating Your First Module

If you don't feel like reading that, checkout the basicmodule at http://github.com/OpenMRS/openmrs-module-basicmodule.git following the steps described here

You can find and check out a lot of examples from Git Hub at http://github.com/OpenMRS. See projects starting with "openmrs-module-example". Following the steps described here to check out the code

For a more detailed introduction, download the EHSDI Training Course. Module development is covered in lectures 8 and 9 of course EH204 OpenMRS Development.

Module File Structure

If viewed as a single mavenmodule project:

- **.settings** - Eclipse specific folder containing preferences for your environment
- **api** - non web specific 'maven module' project
  - **src**
    - **main** - Java files in the module that are not web-specific. These will be compiled into a distributable mymodule.jar
    - **test** - contains the unit test java files for the generic java classes
  - **target** - folder built at runtime that will contain the distributable jar file for the module
- **omod**
  - **src**
    - **main**
      - **java** - web specific java files like controllers, servlets, and filters
      - **resources** -
        - config.xml
        - *.hbm.xml files
        - liquibase.xml (or the old sqldiff.xml)
        - messages_*.properties files
        - modulesApplicationContext.xml
        - log4j.xml - optional file to control logging in your module
      - **webapp** - jsp and html files included in the omod
        - portlets -
        - resources - image, js, and css files that your jsp files reference
        - **tags** -
        - **taglibs** -
    - **test** - contains java unit test classes that test the controllers in omod/src/main/java
  - **target** - Contains the distributable omod file
- **.classpath** - Eclipse specific file that points to the files necessary for building the omod and jar files on the fly
- **.project** - Eclipse specific file containing the name and properties of your eclipse project
- **pom.xml** - Maven build file. Delegates to pom.xml files in the *omod* and *api* project

Building Your Module

Assuming your module is using Maven (with the layout above), you can build at the command line using mvn package or from within Eclipse by right clicking on the project root, and choosing Run As --> Maven Package

You can find the packaged omod in the /omod/target folder.

Installing Your Module

In a running OpenMRS you can use the Manage Modules page linked to from the Administration page.

If uploads are not allowed from the web (changable via a runtime property), you can drop the omod into the ~/.OpenMRS/modules folder. (Where ~/.OpenMRS is assumed to be the Application Data Directory that the running openmrs is currently using.) After putting the file in there simply restart OpenMRS/tomcat and the module will be loaded and started.

Tips While Developing Your Module

It is a tedious task to build and install the module each time you change something and you want to test it. In order to make the process slightly more efficient you can add the following snippet to your omod/pom.xml:

Deploy-web profile

```
<profiles>
  <profile>
    <id>deploy-web</id>
    <build>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-antrun-plugin</artifactId>
          <executions>
            <execution>
              <phase>package</phase>
              <goals>
                <goal>run</goal>
              </goals>
              <configuration>
                <tasks>
                  <copy todir="${deploy.path}/WEB-INF/view/module/${project.parent.artifactId}">
                    <fileset dir="src/main/webapp" includes="**/*" />
                  </copy>
                </tasks>
              </configuration>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>
  </profile>
</profiles>
```

It will allow you to deploy any changes to your web resources such as jsp or js files without re-installing the module. You only need to select the deploy-web profile and specify the deploy.path variable. The deploy path says where OpenMRS is deployed. It should point to an openmrs directory in Tomcat or to a directory where jetty looks for file changes. The path can be relative (to the omod directory) or absolute. You can do it in your IDE or from the command line as follows:

```
mvn package -P deploy-web -D deploy.path="../../openmrs-1.8.x/webapp/src/main/webapp"
```

It is executed with the package phase thus works for package and higher goals like install.

Other Resources

- http://openmrs.blip.tv
- OpenMRS AOP
- Module Conventions
- Subversion Code of Conduct#Modules

Publishing your Module

The Add-ons index is available to everyone. Read more about the module release process here. Also read more about our rules and regulations for it here: Module Repository .

FAQ

- How do I deploy my module?
  - Run the the target "package-module" in the ant build.xml in the root of your module.
  - Option 1: Go to Manage Modules in the Administration section of the webapp. Upload your compiled .omod file from the /dist folder in your module
  - Option 2: Drop your .omod file into the <openmrs application directory>/modules folder and restart openmrs. (The application directory is either /home/tomcatuser/.OpenMRS or c:/docs and settings/tomcatuser/application data/openmrs)
- How do I redeploy my updated module?

    - Option 1: Choose the "undeploy" link on the Manage Modules page and then deploy your module again
    - Option 2: Drop your new .omod file in the application data directory and restart openmrs
- How do I specify Module Updates?
    - There is a page dedicated to theupdate.rdf file
- How can I allow administration of the modules through the web?
    - You need to be sure this line is in your runtime properties file: module.allow_web_admin=true
- Can I store my module source code in your Git Hub or subversion repository?
    - We welcome and encourage you to use our source code repository. Contact code at openmrs.org for additional help.
- How do I perform unit testing in my module?
    - See Module Unit Testing
- How can my module access the database? (Aka How can my module create its own service?)
    - See page on creating the Module Application Context File
- What parts of OpenMRS does a module have access to override?
    - See Module Access
- How can I add a new hibernate interceptor?
    - See the interceptorexample module's application context file (requires at least v1.6.0.8566)
- What is a good/bad example of a Module adding functionality to OpenMRS?
    - A Module would not be used for to create a javaSwing client, but would be used to provide a RESTful view of the OpenMRS API via web services which could perhaps be connected to that Swing client.
- Where does a Module fit into the larger OpenMRS architecture?
    - A Module is simply extended code in the OpenMRS web application. However modules are not permitted to alter web.xml except via the config.xml file.
- How do I call one module from another? Can one module depend on another?
    - See ?Module Dependencies
- Does a Module have any special relationship to Spring?
    - No. A module is simply additional servlets, JSPs and resources to extend the OpenMRS web application. A module can use Spring, but is not tied to it.
- I can't work on my module anymore, how do I abandon/retire my module?
    - There might be someone else in the community that wants to take over development. Send an email to the developer's list asking for volunteers. If no takers, put a note about the module's abandonment on the module's wiki page and in the module repository (if the omod has been uploaded) If you have not made a release, you can delete the module code from the repository and leave the abandonment note on the module wiki page with a link to the last working changeset in the repository.
- A page requires certain global properties to be filled in, how to force the user to fill those out?
    - RequireConfiguration Taglib
- How to enable logging for my new module in the OpenMRS?
    - OpenMRS loggers are configured only to show the `org.openmrs.api:info` level logs and all error logs. So if you have any INFO level logs in your module, then go to OpenMRS Settings Log and add `org.openmrs.api:info, org.openmrs.yourmodulename:info`.