Creating Your First Module

Automatically Using the SDK

> The instructions on this page are outdated, and you will have difficulty following them verbatim; this page is kept around for archival purposes.
>
> **Please use the OpenMRS SDK to create modules!**
>
> See OpenMRS SDK and OpenMRS SDK Step By Step Tutorials.

(These instructions are kept here for archival purposes. Please use the SDK instead.)

Manual Step-by-Step Instructions

There is really no reason to do anything other than use the SDK (see above), but if you want to create the module structure manually:

1. Check out a local copy of the basicmodule *using Maven* (view source)
    a. To checkout the module from github see using git, **remember** not to share the project yet after checking it out
2. Create a new repository for your module on github
    a. Go to https://github.com/new, specify the name, the naming convention is 'openmrs-module-module_id' e.g openmrs-module-myFirstModule
    b. Check the Initialize this repository with a README because you will need it, Click the 'Create Repository' button, be sure to get the URL of your repository next to the 'Git Read-Only' button
3. Disconnect the project from the master origin to the new repo on github you created above
    a. From eclipse or file system open the config file under the '.git' folder and change the value of the remote origin url to what it was in step 2b above.
4. Replace all references to the Basic Module with your new modules name, using the same Conventions. (For example, rename BasicModule to MyFirstModule).
    a. Rename the project to something unique e.g myFirstModule
    b. edit the pom.xml file in the project root folder
        i. find the "artifactId" tag, edit the value to your module id in this case myFirstModule
        ii. find the "name" tag, edit the value to your module name in this case My First Module
        iii. find the "url" tag, edit the value to that of your module
        iv. find the "scm" tag, edit the values of its nested tags appropriately to point to your repository
        v. find the "dependencyManagement" tag and remove any dependencies you don't need and add what you need for your module.
        vi. find the "properties" tag and change the value of the nested "openMRSVersion tag" to what your module actually requires
    c. edit omod/src/main/resources/config.xml
        i. find the "<id>" tag, edit the value (example: change "<id>basicmodule</id>" to " <id>myfirst</id>")
        ii. find the <name> tag, edit the value (example: change "<name>Basic Module</name>" to "<name>My First Module</name>")
        iii. find the <author> tag, change the value from Ben Wolfe to your name ... Ben shouldn't get *all* the credit
        iv. find the <description> tag, edit the value if you want,
        v. you can change the <version> value, following the Versioning conventions
        vi. find the <activator> tag, change the activator name. Must be a class in your module. Case sensitive!
    d. edit omod/src/main/resources/moduleApplicationContext.xml
        i. This file is really only needed if you want either 1) your module to provide jsp files or 2) your module to have its own ___Service.java class
    e. edit omod/src/main/resources/liquibase.xml
        i. This is only needed if you are adding your own tables to the openmrs data model (or modifying others)
    f. rename the packages
        i. expand api/src/main/java
        ii. right-click on org.openmrs.module.basicmodule "Refactor->Rename..."
        iii. the "Rename Package" wizard appears
        iv. edit the name (example: change to "org.openmrs.module.myfirst")
        v. select all of the check boxes (Put a * in the "File name patterns" text field)
        vi. click "Preview"
        vii. (wait patiently)
        viii. click "OK" on the final screen of the wizard
    g. repeat the previous for omod/src/main/java and omod/src/test/java
    h. Be certain to rename all files that have BasicModule (or basicmodule) in their name (for example basicmoduleForm.jsp under web/module). Also look in the contents of the files (e.g. AdminList.java and build.xml)
    i. Change the contents of omod/src/main/resources/messages_*.properties
    j. Edit the README file appropriately
    k. Build your module using maven. "mvn package" at command line or "right click on project, run as maven package"
5. Try out your module
    a. Go to your openmrs installation Administration Manage Modules
    b. Choose Upload and browse to the "omod/target" folder in your checked out code. choose the latest.omod file
6. Document your module

- Modules should have a home page that describes their purpose, features, instructions, known issues, etc.
- Feel free to create a wiki page for your module on this wiki. Your page should be called "FooBar Module" if you have created a module called FooBar.
- Create it as a child of the Active Projects page

7. Publish the resulting .omod file
    - Consider using the OpenMRS Module Repository app. Read more about it here: Module Repository
8. Committing your code to the remote repository you created in step 2 above
    a. Using EGit:
        i. From eclipse, right-click on project name, select commit, in the next pop window, enter a commit message, be sure to uncheck files with changes you don't wish to commit and check those you wish to commit and then click commit, at this point the code resides in your local git repository, you want to push to your github repo.
        ii. Right-click on project name, select remote push, the "Configured remote repository" option is selected, click next
        iii. On the next screen under the "Add create/update specification" section 'master[branch]' as the source ref, click 'Add Spec' button, click next and then click finish
    b. Form the command line:
        i. Navigate to the project root, enter: **git commit -m"your message"**, this commits the code to your local repo
        ii. To push the code to the module's repo on github enter: **git push origin master**