**Creating Maven Remote Repository**

Some of the third party jars that don’t have maven artifacts for that we can create maven dependency and make it available remotely using github kind of public repositories.

Use below command to install that jar as a local maven artifact

**STEP-1: installing into local repository**

**$/>** mvn install:install-file -Dfile=**sfs2x-client-core.jar** -DgroupId=org.goto.smartfox -DartifactId=**sfs2x-client-core** -Dversion=1.0 -Dpackaging=jar -DgeneratePom=true -DcreateChecksum=true

Above command will install maven artifact into local maven repository by default it will be under Users/{username}/.m2/repository folder.

**STEP-2: creating remote repository**

Create public git repository (in this tutorial we used github.com).

Create one organization as shown in the link <https://github.com/go2andplay>

Create repository <https://github.com/go2andplay/smartfox-dev>

Clone this repository into your local drive

Using

$/>git clone <https://github.com/go2andplay/smartfox-dev.git>

**STEP-3: copy local maven artifact into this git repository and commit**

**$/>**git add .

**$/>**git commit -m ‘write you comment’

**$/>**git push origin master

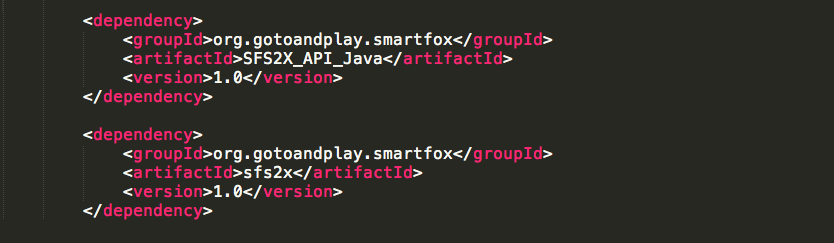
**STEP-4: finally you need add this repository into pom.xml**



NOTE: you need to add raw representation of the URL. For github.com it is <https://raw.githubsercontent.com>

That you can find using raw button on the specific file.

**STEP-5:** add dependencies like any other maven dependencies



**Changing the local maven repository path**

Using below code snippet in the settings.xml

<localRepository>C:/software/maven</localRepository></settings>

Lifecycles

Maven has three built in lifecycles

* **default**: The default lifecycle handles project build and deployment
* **clean**: The clean lifecycle cleans up the files and folders produced by Maven
* **site**: The site lifecycle handles the creation of project documentation

Build lifecycle made up of Phases

**Default lifecycle has following phases**

* **Validate**- validate the project is correct and all necessary information is available
* **Compile**- compile the source code of the project
* **Test**- test the compiled source code using a suitable unit-testing framework. These tests should not require the code be packaged or deployed
* **Package**- take the compiled code and package it in its distributable format, such as a JAR.
* **Verify**- run any checks on results of integration tests to ensure quality criteria are met
* **Install**- install the package into the local repository, for use as a dependency in other projects locally
* **Deploy**- done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

In a development environment, use the following call to build and install artifacts into the local repository. ------ mvn install ------

This command executes each default life cycle phase in order (validate, compile, package, etc.), before executing install>>.

NOTE: You only need to call the last build phase to be executed, in this case, <<<install:

**Build Phase is made up of plugin goals**

However, even though a build phase is responsible for a specific step in the build lifecycle, the manner in which it carries out those responsibilities may vary. This is done by declaring the plugin goals bound to those build phases.

A plugin goal represents a specific task, which contributes to the building and managing of a project. It may be bound to zero or more build phases.

Each phase contains multiple goals associated with that plugin.

Ex: $/>mvn compiler:compile or mvn compiler:testCompile

$/>mvn install:install

**Plugins**

The second way to add goals to phases is to configure plugins in your project. **Plugins are artifacts that provide goals to Maven**. Plugin may have one or more goals wherein each goal represents a capability of that plugin.

Ex: compiler plugin has two goals: compile and testCompile.

**Adding goals to the specific phase when you use custom plugins**

Let's say you have a goal display: time that echos the current time to the commandline, and you want it to run in the process-test-resources phase to indicate when the tests were started. This would be configured like so:

<plugin>

<groupId>com.mycompany.example</groupId>

<artifactId>display-maven-plugin</artifactId>

<version>1.0</version>

<executions>

<execution>

<phase>process-test-resources</phase>

<goals>

<goal>time</goal>

</goals>

</execution>

</executions>

</plugin>

Each lifecycle has following phases

The following lists all build phases of the default, clean and site lifecycles, which are executed in the order given up to the point of the one specified.

**Clean Lifecycle**

|  |  |
| --- | --- |
| pre-clean | execute processes needed prior to the actual project cleaning |
| clean | remove all files generated by the previous build |
| post-clean | execute processes needed to finalize the project cleaning |

**Default Lifecycle**

|  |  |
| --- | --- |
| validate | validate the project is correct and all necessary information is available. |
| initialize | initialize build state, e.g. set properties or create directories. |
| generate-sources | generate any source code for inclusion in compilation. |
| process-sources | process the source code, for example to filter any values. |
| generate-resources | generate resources for inclusion in the package. |
| process-resources | copy and process the resources into the destination directory, ready for packaging. |
| compile | compile the source code of the project. |
| process-classes | post-process the generated files from compilation, for example to do bytecode enhancement on Java classes. |
| generate-test-sources | generate any test source code for inclusion in compilation. |
| process-test-sources | process the test source code, for example to filter any values. |
| generate-test-resources | create resources for testing. |
| process-test-resources | copy and process the resources into the test destination directory. |
| test-compile | compile the test source code into the test destination directory |
| process-test-classes | post-process the generated files from test compilation, for example to do bytecode enhancement on Java classes. For Maven 2.0.5 and above. |
| test | run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed. |
| prepare-package | perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package. (Maven 2.1 and above) |
| package | take the compiled code and package it in its distributable format, such as a JAR. |
| pre-integration-test | perform actions required before integration tests are executed. This may involve things such as setting up the required environment. |
| integration-test | process and deploy the package if necessary into an environment where integration tests can be run. |
| post-integration-test | perform actions required after integration tests have been executed. This may including cleaning up the environment. |
| verify | run any checks to verify the package is valid and meets quality criteria. |
| install | install the package into the local repository, for use as a dependency in other projects locally. |
| deploy | done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects. |

**Site Lifecycle**

|  |  |
| --- | --- |
| pre-site | execute processes needed prior to the actual project site generation |
| site | generate the project's site documentation |
| post-site | execute processes needed to finalize the site generation, and to prepare for site deployment |
| site-deploy | deploy the generated site documentation to the specified web server |

Remembering goals associated with phases might be difficult, hence maven provides default built-in binding for some of the goals

|  |  |
| --- | --- |
| process-resources | resources:resources |
| compile | compiler:compile |
| process-test-resources | resources:testResources |
| test-compile | compiler:testCompile |
| test | surefire:test |
| package | ejb:ejb *or* ejb3:ejb3 *or* jar:jar *or* par:par *or* rar:rar *or* war:war |
| install | install:install |
| deploy | deploy:deploy |

Reference: <https://maven.apache.org/plugins/>

<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>