Tycho builds for Odysseus

New. You can now use tycho to build Odysseus modules.

Important: Due to a bug in tycho currently maven version 3.6.2 cannot be used. So use 3.6.1 instead.

If you want to create a module that should sometimes be added to the Odysseus eco system (building and updatesite) you need to follow some rules for the tycho build.

Pomless build

First of all, we still want to use the Eclipse platform so we use a pomless build, i.e. maven is only used to organize the project.

To define pomless builds create a new folder `.mvn` on the root level of the project and add file `extensions.xml` with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<extensions>
  <extension>
    <groupId>org.eclipse.tycho.extras</groupId>
    <artifactId>tycho-pomless</artifactId>
    <version>1.4.0</version>
  </extension>
</extensions>
```

Global and local builds

There are different ways to build Odysseus (implemented with maven profiles).

1. Each module can be build (so called solobuild)
2. A group of modules can be grouped together to be build. The stable and the incubation updatesite is build on this way.

Main Pom

To allow local and solobuilds there must be a pom.xml on the main level of the project. In the following there is an example from odysseus_core.

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>de.uniol.inf.is.odysseus</groupId>
    <artifactId>de.uniol.inf.is.odysseus_core.root</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <parent>
        <groupId>de.uniol.inf.is.odysseus.tycho</groupId>
        <artifactId>de.uniol.inf.is.odysseus.tycho.configuration</artifactId>
        <version>1.0.0-SNAPSHOT</version>
        <relativePath>./odysseus_dev/de.uniol.inf.is.odysseus.tycho.configuration</relativePath>
    </parent>

    <modules>
        <module>common</module>
        <module>resource</module>
        <module>client</module>
        <module>server</module>
        <module>monolithic</module>
        <module>wrapper</module>
    </modules>

    <profiles>
        <profile>
            <id>solobuild</id>
            <activation>
                <activeByDefault>true</activeByDefault>
            </activation>
            <modules>
                <module>odysseus_dev</module>
                <module>releng/de.uniol.inf.is.odysseus.update</module>
            </modules>
        </profile>
        <profile>
            <id>test</id>
            <modules>
                <module>test</module>
            </modules>
        </profile>
    </profiles>
</project>
```

The are different parts in this maven pom. On the top is the base information. For your project you must change the <artifactId> to a unique name.

In the parent clause the parent is defined. In the parent things such as the target platform and the different base configurations are handled.

Remark: The parent is part of the odysseus_dev submodule. It is importent to keep this module up to the latest version.

The modules part contains all directories, where plugins can be found. In this case there is a module for most subdirectories.

There are two maven profiles below the modules section

- solobuild: This profile is used by default and will be used if the module is build allone. In case of a global build, the given maven modules are not used.
- test: This profile is used the create a test product

As you can see, there is a module named releng/de.uniol.inf.is.odysseus.update. This is used to create an update site for this Odysseus module. This can be copied to a web-server and can be used to install new features in Odysseus.

This folder is not available by default, so you will need to create it.

Add the following pom to this folder. It is important to update the artifactId. This must be the same name as given above.

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>de.uniol.inf.is.odysseus</groupId>
        <artifactId>de.uniol.inf.is.odysseus_core.root</artifactId>
        <version>1.0.0-SNAPSHOT</version>
        <relativePath>../../pom.xml</relativePath>
    </parent>

    <artifactId>de.uniol.inf.is.odysseus_core.update</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>eclipse-repository</packaging>

</project>
```

You need to add file named category.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<site>
    <feature id="de.uniol.inf.is.odysseus.common.feature">
        <category name="common"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.keyvalue.datahandler.feature">
        <category name="common"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.keyvalue.datatype.feature">
        <category name="common"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.slf4j.feature">
        <category name="common"/>
    </feature>
    <feature  id="de.uniol.inf.is.odysseus.cache.feature">
        <category name="server"/>
    </feature>
    <feature  id="de.uniol.inf.is.odysseus.core.server.feature">
        <category name="server"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.memstore.feature">
        <category name="server"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.server.platform.feature">
        <category name="server"/>
    </feature>
    <feature  id="de.uniol.inf.is.odysseus.security.feature">
        <category name="server"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.server.fragmentation.base.feature">
        <category name="server"/>
    </feature>
    <feature  id="de.uniol.inf.is.odysseus.keyvalue.feature">
        <category name="server"/>
    </feature>
    <feature  id="de.uniol.inf.is.odysseus.latency.feature">
        <category name="server"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.planmgmt.standard.feature">
        <category name="server"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.relational.feature">
        <category name="server"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.rest.feature">
        <category name="server"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.scheduling.feature">
        <category name="server"/>
    </feature>
    <feature  id="de.uniol.inf.is.odysseus.script.feature">
```

```
        <category name="server"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.usermanagement.store.feature">
        <category name="server"/>
    </feature>
    <feature  id="de.uniol.inf.is.odysseus.monolithic.feature">
        <category name="monolithic"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.monolithic.sdp.feature">
        <category name="monolithic"/>
    </feature>
    <feature  id="de.uniol.inf.is.odysseus.monolithic.studio.feature">
        <category name="monolithic"/>
    </feature>
    <feature  id="de.uniol.inf.is.odysseus.client.feature">
        <category name="client"/>
    </feature>
    <feature  id="de.uniol.inf.is.odysseus.client.platform.feature">
        <category name="client"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.client.studio.feature">
        <category name="client"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.rcp.dashboard.colors.feature">
        <category name="client"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.rcp.dashboard.feature">
        <category name="client"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.rcp.dashboard.soccer.feature">
        <category name="client"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.rcp.dashboard.windrose.feature">
        <category name="client"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.server.feature">
        <category name="server"/>
    </feature>
    <feature  id="de.uniol.inf.is.odysseus.studio.feature">
        <category name="client"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.datarate.feature">
        <category name="common"/>
    </feature>
    <feature id="de.uniol.inf.is.odysseus.wrapper.json.feature">
        <category name="wrapper"/>
    </feature>
    <category-def name="wrapper" label="wrapper"/>
    <category-def name="odysseus_core" label="odysseus_core"/>
    <category-def name="client" label="client">
        <category name="odysseus_core"/>
    </category-def>
    <category-def name="common" label="common">
        <category name="odysseus_core"/>
    </category-def>
    <category-def name="server" label="server">
        <category name="odysseus_core"/>
    </category-def>
    <category-def name="monolithic" label="monolithic">
        <category name="odysseus_core"/>
    </category-def>
</site>
```

This file should contain any feature of your project that should be available on the update site. Each feature should have a category. The categories are defined in the lower part of the file.

Define poms in modules

For each element in the <modules> section of the root pom there must be a definition of the real plugins and features that can be found in the folder.

Here is an example of the common folder of odysseus_core

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>de.uniol.inf.is.odysseus</groupId>
    <artifactId>de.uniol.inf.is.odysseus_core.common</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>pom</packaging>

    <parent>
        <groupId>de.uniol.inf.is.odysseus</groupId>
        <artifactId>de.uniol.inf.is.odysseus_core.root</artifactId>
        <version>1.0.0-SNAPSHOT</version>
    </parent>

    <modules>
        <module>core</module>
        <module>bugreport</module>
        <module>compiler</module>
        <module>datatype.interval</module>
        <module>de.uniol.inf.is.odysseus.common.feature</module>
        <module>de.uniol.inf.is.odysseus.console.executor</module>
        <module>de.uniol.inf.is.odysseus.report</module>
        <module>de.uniol.inf.is.odysseus.report.executor</module>
        <module>de.uniol.inf.is.odysseus.report.general</module>
        <module>de.uniol.inf.is.odysseus.slf4j</module>
        <module>de.uniol.inf.is.odysseus.slf4j.feature</module>
        <module>de.uniol.inf.is.odysseus.sweeparea</module>
        <module>de.uniol.inf.is.odysseus.datarate</module>
        <module>de.uniol.inf.is.odysseus.datarate.feature</module>
        <module>intervalapproach</module>
        <module>keyvalue.datahandler</module>
        <module>keyvalue.datahandler.feature</module>
        <module>keyvalue.datatype</module>
        <module>keyvalue.datatype.feature</module>
        <module>keyvalue.mep</module>
        <module>mep</module>
        <module>mep.matrix</module>
    </modules>
</project>
```

As you can see, there are different places where you need to adapt the file to your situation:

First of all, there must be a unique name for this pom (<artifactId>), but the name is not important. To avoid problems in the global build, the artifactId should start with the name of global artifact. If there is a plugin with the same name as the folder (e.g. common) you could name this pom e.g. ...common. project to avoid the same name.

As second, you must update the <artifactId> in the <parent> clause. This must be the same as in the root pom.

Finally, you need to add the folders where the eclipse plugins and features can be found. There is no pom in this folders needed, because a pom-less build is defined (see above). Of course, you can also structure this modules further (i.e. create subfolders for combined elements). In this case you need to add another pom.xml to the subfolder similar to the one in this folder.

Building

After you defined every necessary pom you can create a build with maven:

```
mvn clean verify -Dtargetfilename=${PLATFORMTARGETFILE}
```

Replace ${PLATFORMTARGETFILE} with the required PLATFORMTARGETFILE.

Here are some examples

```
// Odysseus core
mvn clean verify -Dtargetfilename=platform_core
// Stable development build
mvn clean verify -Dtargetfilename=platform_development_stable
// All master build
mvn clean verify -Dtargetfilename=platform_master_all
```

The build typically takes some time as dependencies are resolved and the artifacts are build.

After this, you will find the p2 based update site under: releng/de.uniol.inf.is.odysseus.update/target/repository

Creating Server, Client and Monolithic product

You can use the maven build to create products that can be installed and that contains already all required features. To do this, you need to add you features top the product files that are provided in odysseus_dev/products.

E.g. add a feature to de.uniol.inf.is.odysseus.monolithic.product.product. This can e.g. be done with eclipse

After that you can build the products with the following maven call

```
mvn clean verify -Pproduct -Dtargetfilename=${PLATFORMTARGETFILE}
```

Here all version for all supported platform are created. You can create the products for the current platform only, by calling

```
mvn clean verify -Pproduct,solobuild,currentOS -Dtargetfilename=${PLATFORMTARGETFILE}
```

Attention: The are no blancs in the "P-section"!

Here are some example

```
// Odysseus core
mvn clean verify -Pproduct,solobuild,currentOS -Dtargetfilename=platform_core
// Stable development build
mvn clean verify -Pproduct,solobuild,currentOS -Dtargetfilename=platform_development_stable
// All master build
mvn clean verify -Pproduct,solobuild,currentOS -Dtargetfilename=platform_master_all
```

See The Odysseus Operator Test Framework for information how to create integration tests for your module.

Global build

If you want to combine different modules to a global build (e.g. as we do in jenkins to create all odysseus_stable and incubation modules) you need to do the following:

0) Create a new git project

1) Create a root folder with a root pom

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>de.uniol.inf.is.odysseus</groupId>
    <artifactId>de.uniol.inf.is.odysseus.odysseus_all</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <parent>
        <groupId>de.uniol.inf.is.odysseus.tycho</groupId>
        <artifactId>de.uniol.inf.is.odysseus.tycho.configuration</artifactId>
        <version>1.0.0-SNAPSHOT</version>
    </parent>

    <modules>
        <module>stable</module>
        <module>incubation</module>
        <module>odysseus_dev</module>
    </modules>
</project>
```

2) Add odysseus_dev as submodule

```
git submodule add https://mgrawunder@git.swl.informatik.uni-oldenburg.de/scm/ody/odysseus_dev.git
```

3) Add all required modules as submodules

4) Do something like in the following:

```
# Install the parent
cd ${WORKSPACE}/odysseus_dev/de.uniol.inf.is.odysseus.tycho.configuration
mvn clean install

# Do processing
cd ${WORKSPACE}
# Combine all category.xml of the submodules to a new category.xml
rm -rf tooling
git clone https://git.swl.informatik.uni-oldenburg.de/scm/ODYJENK/tooling.git
javac "${WORKSPACE}/tooling/ci/de.uniol.inf.is.odysseus.creatermap/src/de/uniol/inf/is/odysseus/creatermap
/CreateRMap.java"
java -cp "${WORKSPACE}/tooling/ci/de.uniol.inf.is.odysseus.creatermap/src" de.uniol.inf.is.odysseus.creatermap.
CreateRMap "${WORKSPACE}" "${WORKSPACE}/tooling/ci/de.uniol.inf.is.odysseus.updatesite" "${WORKSPACE}/tooling/ci
/de.uniol.inf.is.odysseus.update.p2"
cp "${WORKSPACE}/tooling/ci/de.uniol.inf.is.odysseus.update.p2/category.xml" ${WORKSPACE}/releng/de.uniol.inf.
is.odysseus.update/

#Run the build
mvn  clean verify -P!\solobuild -Dtargetfilename=platform_core
```

Install Features via Command Line

It is possible to install features without starting the product (e.g. to create a docker image with a distinct combination of features), e.g.:

```
eclipsec.exe -application org.eclipse.equinox.p2.director -repository https://odysseus.informatik.uni-oldenburg.
de/updatesite/odysseus_all_new/stable/origin/master/latest/ -installIU de.uniol.inf.is.odysseus.wrapper.
rabbitmq.feature.feature.group -destination H:/git/odysseus_core/odysseus_dev/products/monolithic/target
/products/de.uniol.inf.is.odysseus.studio.product.monolithic/win32/win32/x86_64 -profile SDKProfile
```

See: https://help.eclipse.org/2020-03/index.jsp for further information.