Kuksa IDE - Developers " "Guide Documentation

Pedro Cuadra

Mar 15, 2019

Contents:

CHAPTER 1

Quick Start

Kuksa IDE is built as a full custom Eclipse Che Assembly. Therefore, it includes all assembly components specified and described in Eclipse Che Assembly are included into Kuksa IDE's build system.

1.1 Build the Assembly

Build the assembly by running;

```
cd <kuksa-ide-root-path>
docker run -ti -v ~/.m2:/home/user/.m2 -v `pwd`:/home/user/che-build -v `pwd`:/
↪projects eclipse/che-dev:6.10.0 sh -c "mvn clean install"
```

1.2 Deploying the Assembly

Running as Single User:

Running as Multi User:

CHAPTER 2

Kuksa IDE Custom Assembly

Kuksa IDE is built as a full custom Eclipse Che Assembly. Therefore, it includes all assembly components specified and described in Eclipse Che Assembly are included into Kuksa IDE's build system.

2.1 Build the Assembly

Eclipse Che provides different ways to build a custom assembly and can be seen at Building Che. In this section, two of this procedures are going to be explained.

2.1.1 Build Using Eclipse Che's Docker Image

According to Eclipse Che's documentation one can use `eclipse/che-dev` docker image to build a custom Eclipse Che Assembly. This can be achieved by running;

```
cd <kuksa-ide-root-path>
docker run -ti -v ~/.m2:/home/user/.m2 -v `pwd`:/home/user/che-build -v `pwd`:/
↪projects eclipse/che-dev:6.10.0 sh -c "mvn clean install"
```

Eclipse Che's developers recommend mounting Maven repo (-v ~/.m2:/home/user/.m2) to persist dependencies and make subsequent builds faster.

Note: Building using Eclipse Che's Docker image is the prefereable way if you want to test all the components of the Assembly. For instance, stacks can't be verified when building it inside Eclipse Che itself. Nevertheless, to make use of the built artifacts you first have to deploy it following *Deploying the Assembly within Eclipse Che's Docker*.

Verify your build's correctness

You can check if your assembly was correctly build by checking the content of `<kuksa-ide-root-path>/assembly/assembly-main/target/eclipse-che-<version>/eclipse-che-<version>`. For instance, you can check the that AGL sample projects have been added by running;

```
cd <kuksa-ide-root-path>/assembly/assembly-main/target/eclipse-che-<version>/eclipse-
↪che-<version>
cat templates/samples.json
```

And the output should contain an entry similar to the following;

```
{
    "category": "Samples",
    "commands": [],
    "displayName": "agl-helloworld-service",
    "name": "agl-helloworld-service",
    "links": [],
    "tags": [
        "agl",
        "gcc",
        "cpp"
    ],
    "mixins": [],
    "modules": [],
    "source": {
        "type": "git",
        "location": "https://github.com/iotbzh/helloworld-service.git",
        "parameters": {}
    },
    "path": "/helloworld-service",
    "attributes": {},
    "problems": [],
    "projectType": "c",
    "description": "A binding example for AGL"
}
```

Similarly, to verify that the stacks have been added run;

```
export TEMP_DIR=`mktemp -d`

cd <kuksa-ide-root-path>/assembly/assembly-main/target/eclipse-che-<version>/eclipse-
↪che-<version>

# Copy to temp dir
cp tomcat/webapps/api.war ${TEMP_DIR}

# Change to temp dir
pushd ${TEMP_DIR}
jar xf api.war

tree | grep stacks
```

And the output shouldn't contain `che-core-ide-stacks-<version>.jar`. Instead, it should look like;

```
│   ├── kuksa-stacks-<version>.jar
```

For verifying other included components please review Eclipse Che's documentation to see how your component is packaged into the assembly.

Troubleshooting

The docker image building process can fail because the user ID (uid) of the user issuing the `docker run` command doesn't correspond the uid of the user "`user`" inside the docker. To avoid this add the following flag to the `docker run` command.

```
--user `id -u ${USER}`
```

2.1.2 Build Using Running Eclipse Che

To build our Eclipse Che Assembly you can follow the steps in the article Build Che in Che.

Note: Building Eclipse Che Assembly using the procedure don't allow the user to have a look at anything that is outside the workspace. However, it's easier to start testing and even debugging your IDE extensions.

2.2 Deploying the Assembly within Eclipse Che's Docker

Running as Single User:

Running as Multi User:

Warning: The previous command includes optional arguments denoted by `[arg]`. If you want to keep any (or all) remove the brackets `[]`.

If you want to run the Kuksa IDE in multi user mode add `-e CHE_MULTIUSER=true`. Similarly, if from the previous run you added news custom stacks to the assembly you'll need to add `-e CHE_PREDEFINED_STACKS_RELOAD__ON__START=true`.

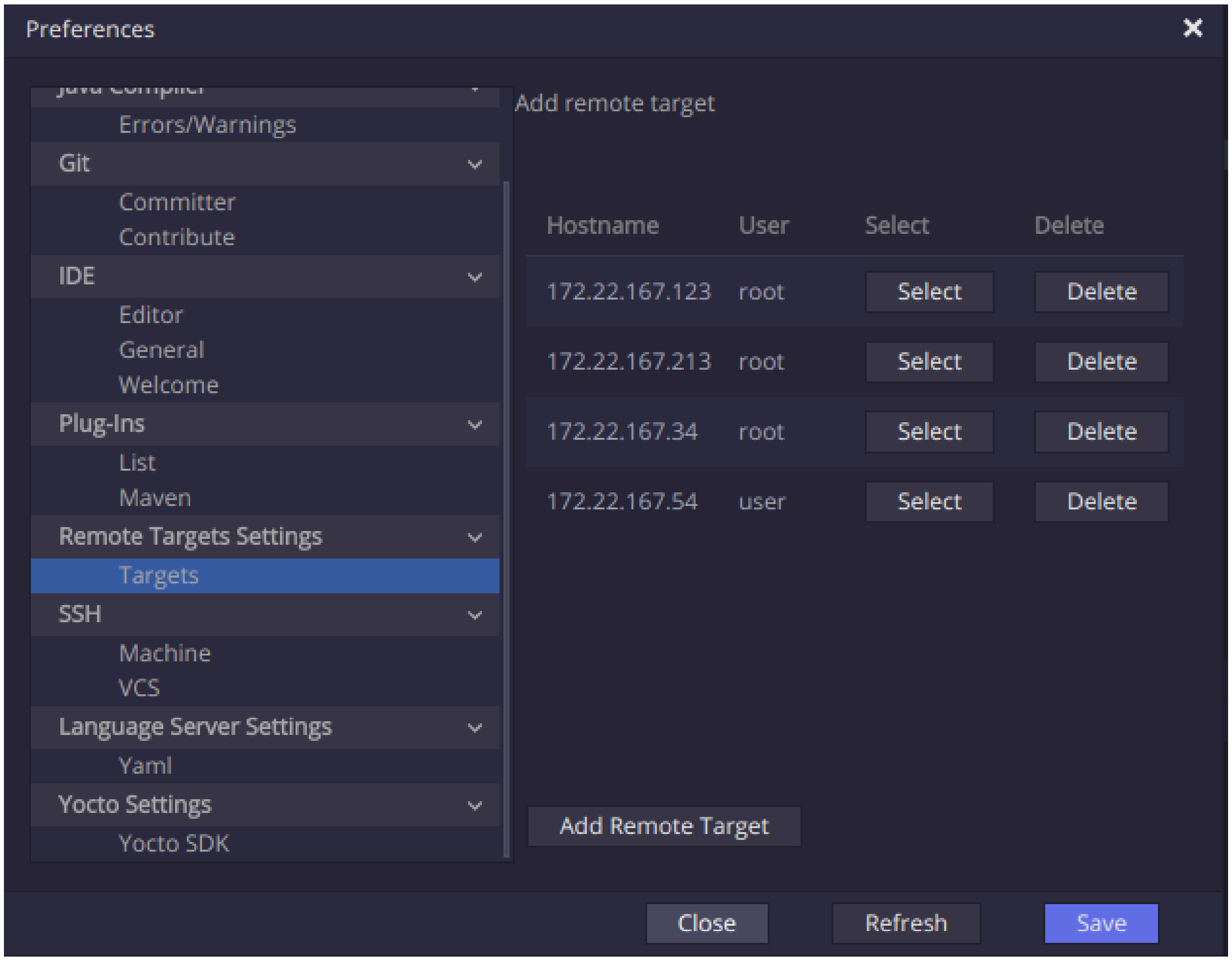
CHAPTER 3

Plugins

Currently, Kuksa IDE includes some plugins extending Eclipse Che's functionality. Each plugin is explained in a sub-section below.

3.1 Remote Target

Remote Target plugin enables the user to Manage different development boards (i.e. Raspberry Pi 3). The image below shows the preferences view acccessible through *Profile* > *Preferences* > *Remote Target Settings* > *Targets*.



Add remote target

Hostname	User	Select	Delete
172.22.167.123	root	Select	Delete
172.22.167.213	root	Select	Delete
172.22.167.34	root	Select	Delete
172.22.167.54	user	Select	Delete

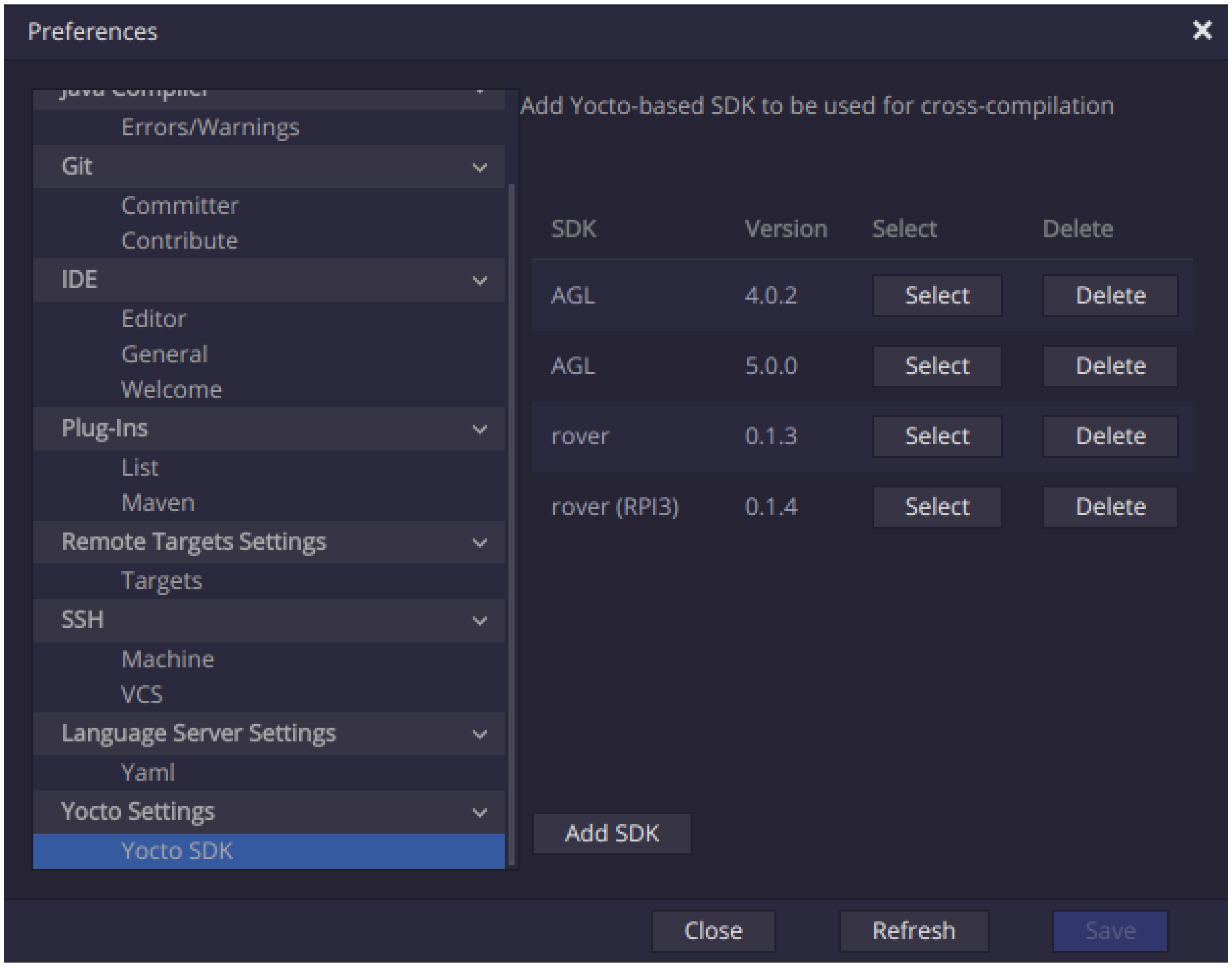
This plugin provides the following macros;

- `${remote.target.hostname}`: Selected remote target's hostname (IP)
- `${remote.target.user}`: Selected remote target's user name

These macros are possible to be resolved once a Remote target is selected in *Profile > Preferences > Remote Target Settings > Targets*.

3.2 Yocto Support

Yocto Support plugin enables the user to manage different Yocto SDKs. The image below shows the preferences view acccessible through *Profile > Preferences > Yocto Settings > Yocto SDK*.



Add Yocto-based SDK to be used for cross-compilation

SDK	Version	Select	Delete
AGL	4.0.2	Select	Delete
AGL	5.0.0	Select	Delete
rover	0.1.3	Select	Delete
rover (RPI3)	0.1.4	Select	Delete

As part of the managing of the Yocto SDK, these are;

- Being downloaded and installed, when added.
- Being uninstalled, when deleted

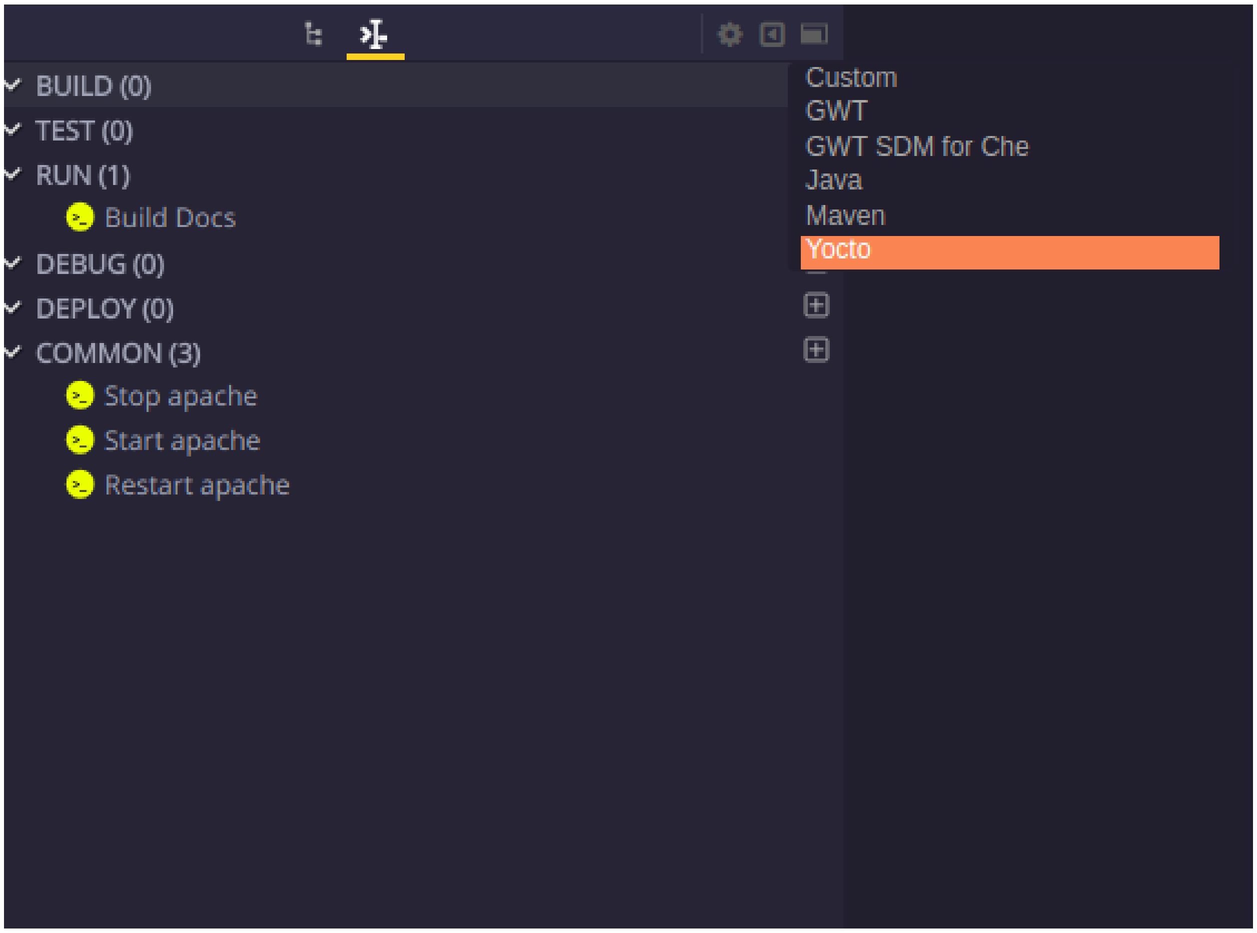
For dowloading, installing, and uninstalling the Yocto SDK, a `CustomSilentCommandExecutor` that executes command line commands in the Workspace's running machine.

Morevoer, the plugin provides the following macros

- `${yocto.sdk.env.path}`: Path to the environement source file
- `${yocto.sdk.path}`: Path to the installation root directory

These macros can be only resolved when a Yocto SDK is selected through *Profile* > *Preferences* > *Yocto Settings* > *Yocto SDK*.

Additionally, this plugin provides a custom Eclipse Che command type, which gives an initial template for using some Yocto SDK's plugin macros. This template is accessible as shown in the image below.


BUILD (0)
TEST (0)
RUN (1)
Build Docs
DEBUG (0)
DEPLOY (0)
COMMON (3)
Stop apache
Start apache
Restart apache
Custom
GWT
GWT SDM for Che
Java
Maven
Yocto

CHAPTER 4

Stacks

Eclipse Che doesn't provide a standard mechanism to add custom stacks during build time. Therefore, Kuksa IDE provides an easy and straight forward mechanism to append custom stacks to the ones provided by Eclipse Che during build time.

4.1 Adding New Stacks

To add a new stack simply write its JSON. And add it to the repository at `stacks/src/main/resources/stacks`. During build time Maven will unpack `che-core-ide-stacks-<version>` and append any JSON content from `stacks/src/main/resources/stacks/*` to it's main `stacks.json`.

For instance, by the time of writing two stacks were already added. These are shown below.

```
$ tree stacks/src/main/resources/stacks
stacks/src/main/resources/stacks
    ├── agl.json
    └── yocto.json
```

4.2 Debugging

For verifying that your stack is being added to the main `stacks.json` you can first check `stacks/target/classes/stacks.json`.

Note: `stacks/target/classes/stacks.json` is created during the building of `kuksa-stacks` Maven module.

You can also just build `kuksa-stacks` Maven module instead of the entire assembly for quicker debuggin by running;

```
mvn clean install -rf :kuksa-stacks
```

You can also check if your stack was in fact added to the assembly by checking if `kuksa-stacks-<version>.jar` was included into the `api.war` as explained in *Verify your build's correctness*.

Additionally, you can extract `kuksa-stacks-<version>.jar`'s content to verify it your stack is in `stacks.json` by running;

```
pushd ${TEMP_DIR}
jar xf WEB-INF/lib/kuksa-stacks-<version>.jar
cat stacks.json
```

Note: ${TEMP_DIR} is obtaine by following the steps from *Verify your build's correctness*

CHAPTER 5

Sample Projects

Eclipse Che doesn't provide a standard mechanism to add custom sample projects during build time. Therefore, Kuksa IDE provides an easy and straight forward mechanism to append them to ones provided by Eclipse Che during build time.

5.1 Adding New Sample Projects

To add a new sample projects simply write its JSON. And add it to the repository at `samples/src/main/resources/samples`. During build time Maven will unpack `che-core-ide-templates-<version>` and append any JSON content from `samples/src/main/resources/samples/*` to it's main `samples.json`.

For instance, by the time of writing a sample project was already added. This is shown below.

```
$ tree samples/src/main/resources/samples
samples/src/main/resources/samples
   └── helloworld_service.json
```

5.2 Debugging

For verifying that your sample project is being added to the main `samples.json` you can first check `samples/target/classes/samples.json`.

Note: `samples/target/classes/samples.json` is created during the building of `kuksa-samples` Maven module.

You can also just build `kuksa-samples` Maven module instead of the entire assembly for quicker debuggin by running;

```
mvn clean install -rf :kuksa-samples
```

You can also check if your sample project was in fact added to the assembly by following the steps explained in *Verify your build's correctness*.