Telecom ParisTech
COMELEC Department

Development Infrastructure for TTool

	Document Manager	**Contributors**	**Checked by**
Name	Dominique BLOUIN	Ludovic APVRIL	Ludovic APVRIL
Contact	dominique.blouin@telecom-paristech.fr		
Date	10/02/2017		

Contents

1 Preface

1.1 Table of Versions

Version	Date	Description & Rationale of Modifications	Sections Modified
1.0 1.1	17/10/2016 10/02/2017	First draft Added Eclipse IDE development + tests	

1.2 Table of References and Applicable Documents

Reference	Title & Edition	Author or Editor	Year

1.3 Acronyms and glossary

Term	Description

1.4 Executive Summary

This document describes the development infrastructure that has been setup for the development of TTool. It describes sources configuration management, the development process with basic editor and command lines, with the Eclipse IDE, as well as the testing, building and installation procedures.

2 Source Configuration Management

2.1 Gitlab Server

TTool sources are hosted on the Gitlab server of Telecom ParisTech under the group *mbetools* and project *TTool*. The Gitlab project can be accessed via . Login must be performed using Shibboleth as shown in figure 1, using the credentials from your institution if it is a member of the Federation Education Recherche (`https://services.renater.fr/federation/participants/idp`). Otherwise, please ask us for an account.

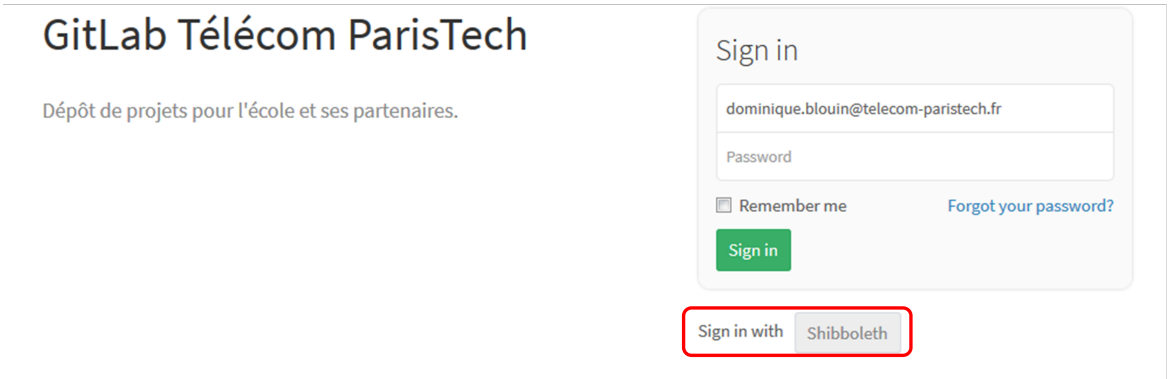

Figure 1:

2.2 Basic Sources Management

The address to access the sources is . For the time being and until further notice, we will keep using a centralized development process, like it was the case for Subversion. In this process, each developer first obtains a clone of the master remote repository. For this, go to the directory where the sources are to be downloaded and issue the following command:

```
git clone git@gitlab.enst.fr:mbe-tools/TTool.git
```

Once modifications are made, they can be committed using command:

```
git commit -m "Commit message"
```

This will only commit the changes in your local repository. In order to move them to the remote repository, use:

```
git push origin master
```

More information on the use of Git can be found here:
`http://rogerdudler.github.io/git-guide/`.

By default, Git will ask credentials for each operation. In order to avoid this, it is possible to upload a public SSH key at:
`https://gitlab.telecom-paristech.fr/profile/keys`.

2.3 Bug Tracking

The Gitlab server provides an issues tracking system to record bugs, evolutions or support demands from users and developers. Issues can be seen at:
`https://gitlab.telecom-paristech.fr/mbe-tools/TTool/issues`

It is suggested to create an issue for every modification that is made to the code, providing with the issue detailed information on the problem including screen shots, log traces and test cases if required. This information is important so that other developers can understand why the changes were made. Ideally, the issue number should be mentioned within the modified code and commit messages.

3 Development with Basic Text Editor and Make

TODO

4 Development with Eclipse

Eclipse is a well-known Integrated Development Environment (IDE) providing many advanced functionalities to support developers and improve code quality by the application of built-in on the fly code analyses. One advantage of Eclipse is that it is a multi-platform application so it can be used on Linux, Windows and Mac. The procedures described in this section are valid for all platforms although some elements such as C++ projects need to be different due to different platform-specific compilation tool chains to be used. More information on this is provided on the concerned subsections.

4.1 Installing and Configuring Eclipse

Download Eclipse IDE for Java developers here:
`http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/neon2`

Unzip the package and launch the eclipse executable. For developing C++ applications such as the DIPLODOCUS simulator, add the C Development Tools (CDT). For this, select menu “Help » Install New Software”. From the dialog box that opens, select the “Neon” update site, unfold the “Programming Languages” category and check the elements as shown in figure 2. Follow the wizard instructions to complete the installation.

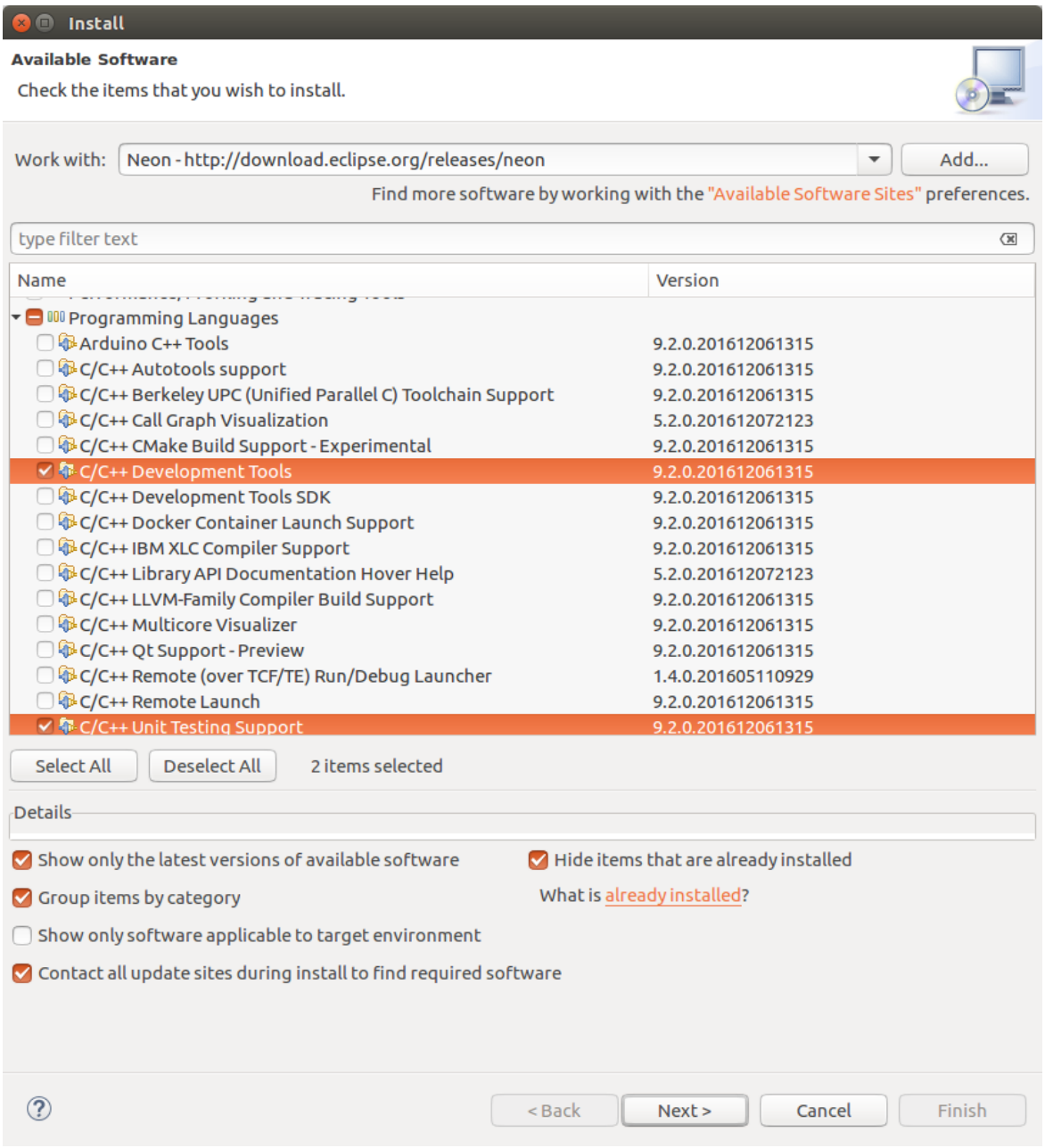

Figure 2:

4.2 Online Help

The first place to look for help is in via menu “Help»Help Content” from Eclipse. A dialog box will show a tree with branches for each integrated plugin or application. Help is provided for the 3 plugins that are used to develop TTool; EGit, Java Development Tools and C/C++ Development Tools.

4.3 Source Configuration Management with EGit

The downloaded Eclipse IDE already includes a plugin named EGit for source configuration management using Git.

4.3.1 Cloning the TTool Repository

First, switch to the Git perspective by clicking menu “Window»Perspective»Open Perspective»Other”. Then select the Git perspective as illustrated in figure 3.

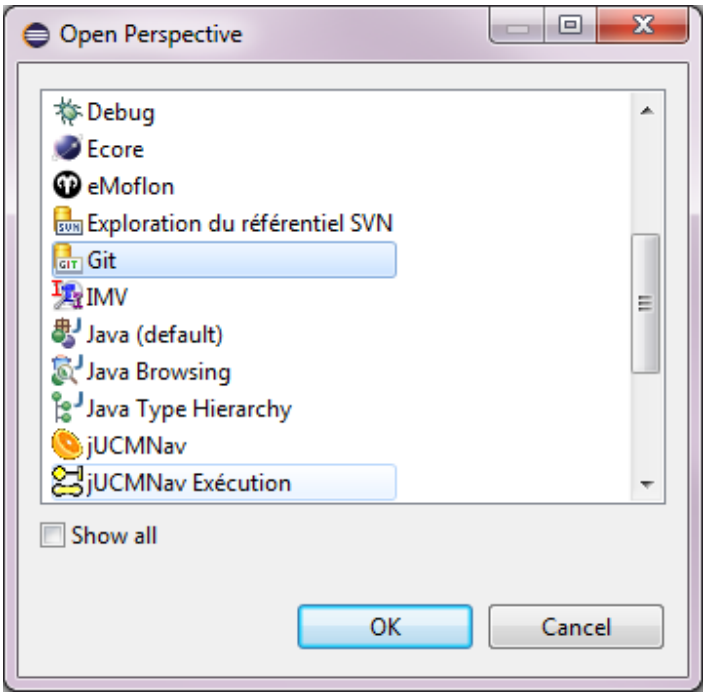

Figure 3:

Then clone the TTool Git repository by clicking the appropriate button as shown in figure 4. Specify the TTool Git repository URI (git@gitlab.enst.fr:mbe-tools/TTool.git) and follow the wizard by also setting the local Git repository path.

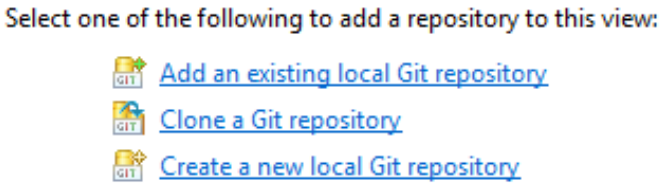

Figure 4:

The content of the cloned repository can be seen from the Git Repository view by unfolding the “Working Tree” folder (figure 5).

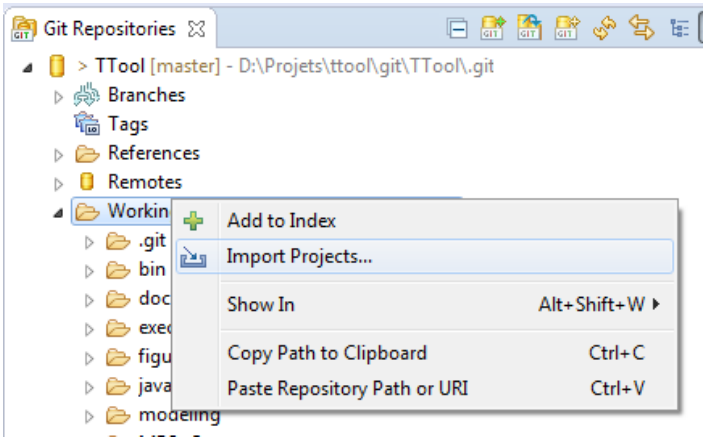

Figure 5:

4.3.2 Importing the Required Projects into the Workspace

Two Eclipse Java projects are needed to develop TTool: the *bin* project that contains the required libraries (jars) and the *src* project that contains the source code. Install these 2 projects in the workspace by right-clicking the "Working Tree" it in the Git repository view and selecting "Import Projects..." as shown in figure 5.

Then select "Import existing Eclipse projects" as shown in figure 6 and follow the steps of the wizard.

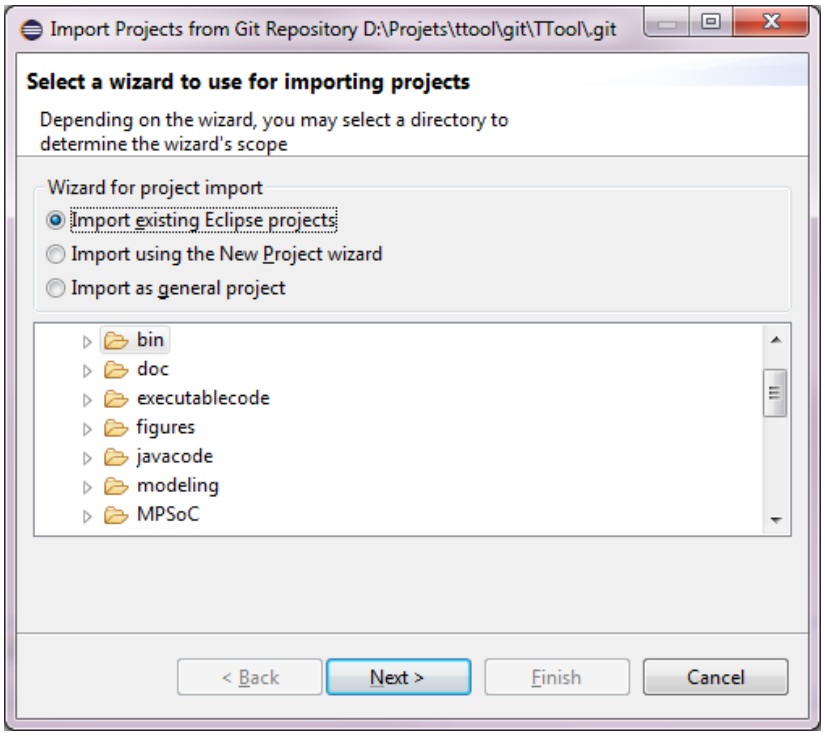

Figure 6:

4.3.3 Committing, Pulling and Pushing Changes

Committing the changes of a file or a directory or a project is performed by selecting this element in the project navigator view then selecting “Team»Commit”. This will open the Git Staging view for where the changed files are listed. Double clicking a file in the unstaged changes view will open an editor allowing visualizing the changes (figure 7). Right click and select “Add to Index”, then enter a commit message and click the “Commit” button to commit the changes. Pushing and pulling can be performed by selecting the repository from the Git repository view or the elements from the project view.

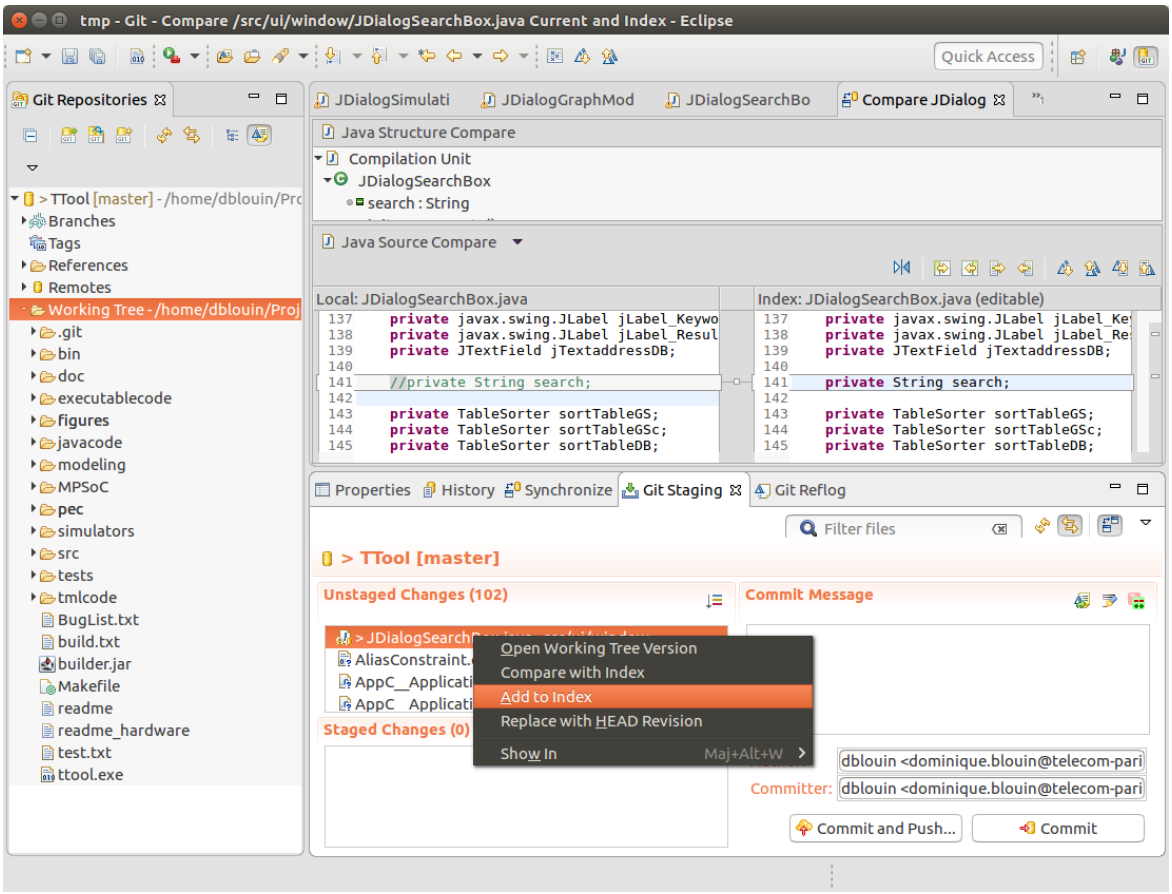

Figure 7:

4.4 Java Development

4.4.1 Coding and Compiling

Switch to the Java perspective to develop TTool with JDT (select menu “Windows»Perspective…” or select the appropriate perspective button from the upper right corner of Eclipse). By default Eclipse will automatically compile all files in the project. However the provided projects have already been configured so that only the required classes are compiled and also to use the required library files from the *bin* project.

JDT provides several advanced functionalities such as automatically navigating from a variable to its declaration, finding its use throughout all the classes, including refactoring capabilities, syntactic coloration, code completion, etc.

4.4.2 Launching TTool

A default launch configuration is provided with the TTool *src* project. It allows for launching TTool from the compiled code. This configuration can be edited by opening the *Run Configurations* dialog box as show in figure 8.

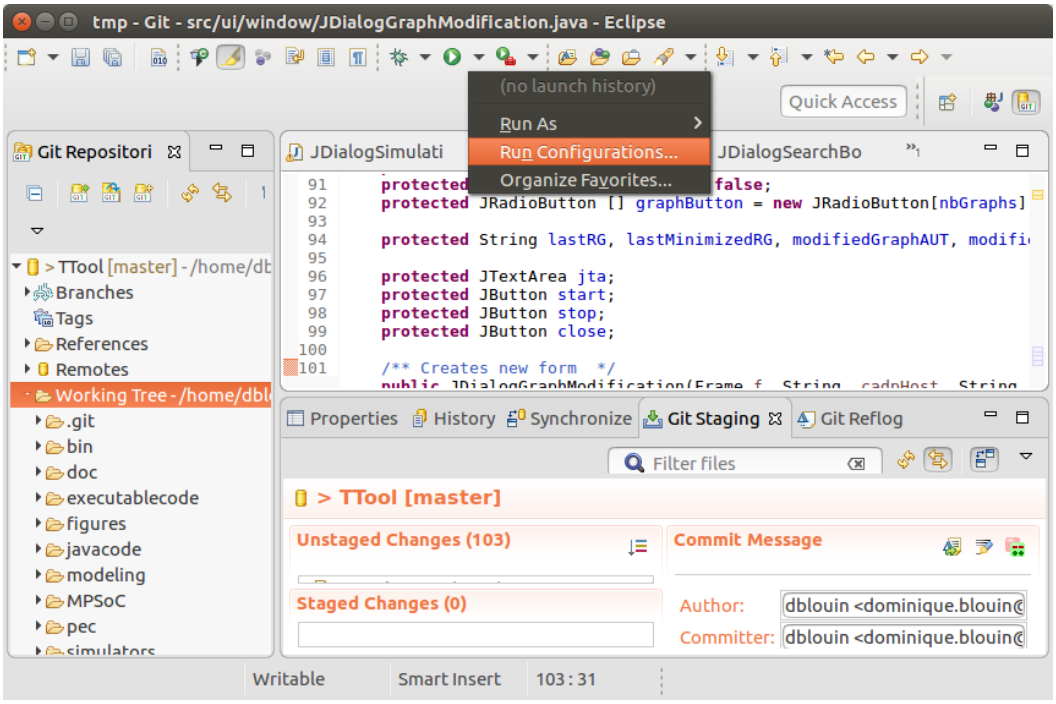

Figure 8:

The TTool launch configuration is displayed in figure 9. It specifies the main class to execute as well as the program arguments, working directory and optionally additional environment variables to be added to the default system environment. Click *Run* to launch TTool. The output from TTool will be displayed in the console view. Once TTool has been launched once, simply clicking the button to open the launch dialog box will directly launch TTool so that it is not necessary to open the launch dialog box every time.

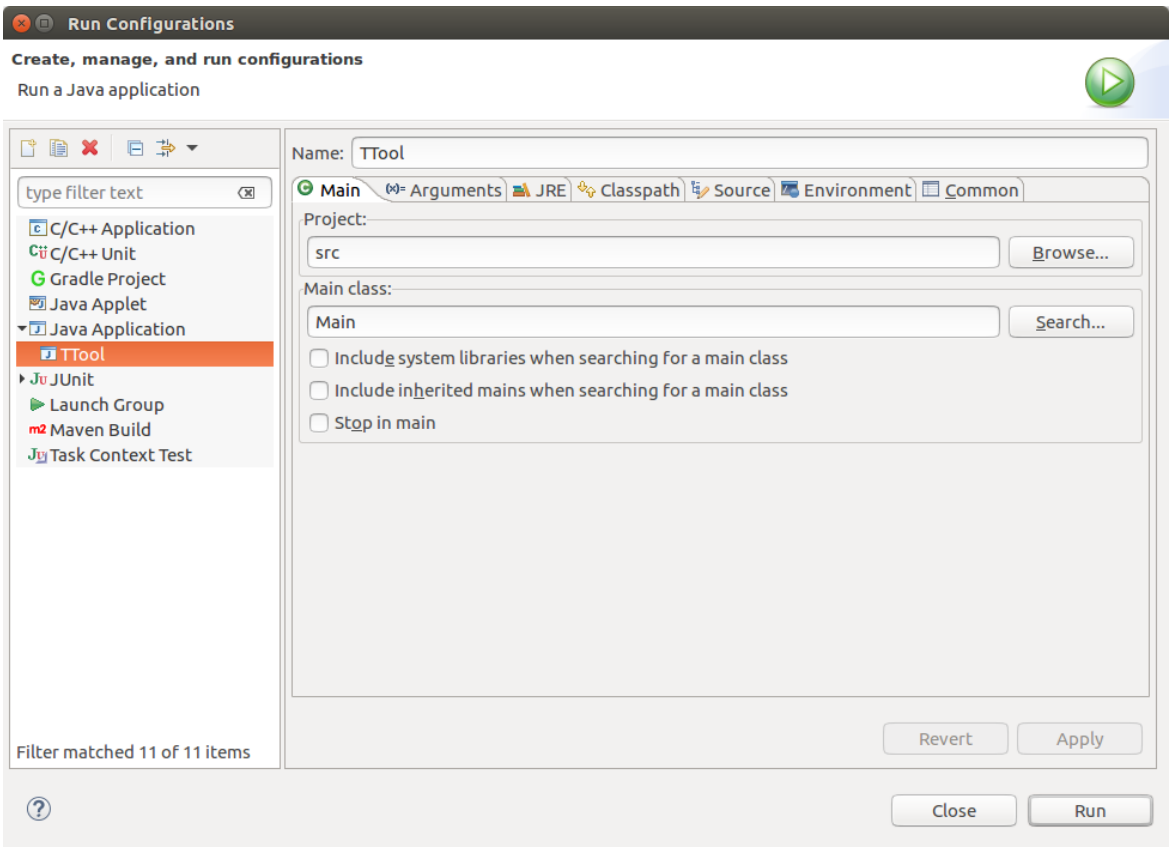

Figure 9:

4.4.3 Using the Debugger

In addition, JDT provides an enhanced debugger allowing executing the program step by step with sophisticated breakpoints to control the execution and to examine variable contents etc. However to use the debugger, the proper launch configurations must be used by clicking the *debug* button just left of the launch configuration button of figure 8.

4.5 C++ Development

One advantage of IDEs is that once a user has learned how to use a first tool, other integrated tools are faster to learn because the tools share so much functionality.

4.5.1 Coding and Compiling

Like for JDT, CDT also has its own perspective and the first thing to do is to switch to this perspective as shown in figure 10.

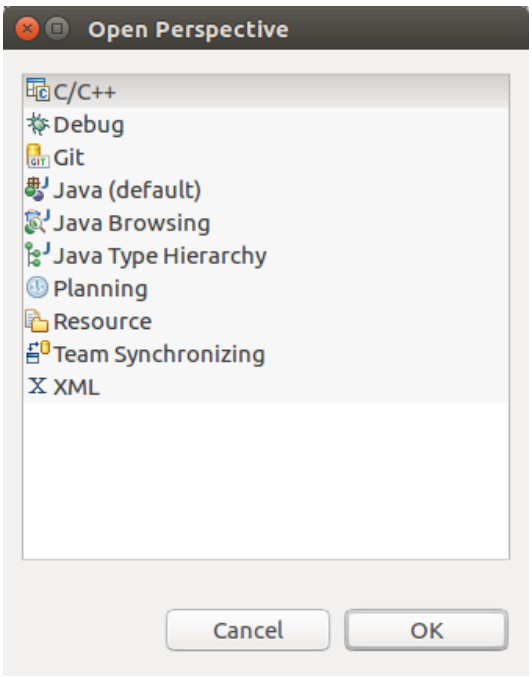

Figure 10:

For developing the TTool C++ applications with CDT, the predefined CDT projects must first be imported into the workspace as explained in section 4.3.2. Note that for CDT, the projects are platform dependent due to the different compilation toolchains (Cygwin is used for Windows) so there is one project per platform. For instance, for developing the DIPLODOCUS simulator on Linux, import the *c++2* project. For windows, import the *c++2 _ windows* project (TODO to be provided later). CDT provides the ability to define several build configurations. For the *c++2* project, two configurations are provided as shown in figure 11.

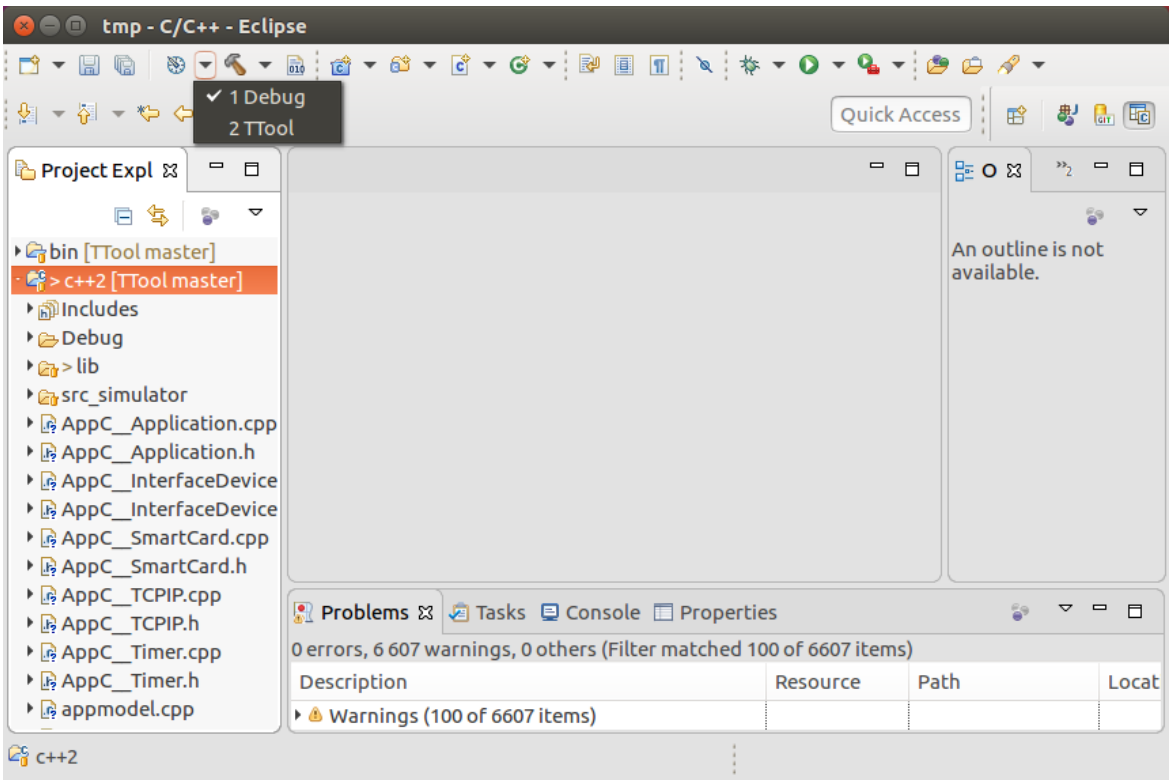

Figure 11:

The *TTool* configuration will compile the code using the provided make file as if the simulator were compiled from TTool. The *Debug* configuration allows for compiling TTool such that it can be executed in debug mode. The selected button in figure 11 allows changing the configuration being used. The hammer button on the right hand side of it allows compiling and linking the code.

The settings for these build configurations can be edited by selecting the *c++2* project in the project navigator and clicking “Properties”. For the *Debug* configuration, the CDT internal builder is used. This means that the compilation and linking options can be changed via the form editor shown in figure 12.

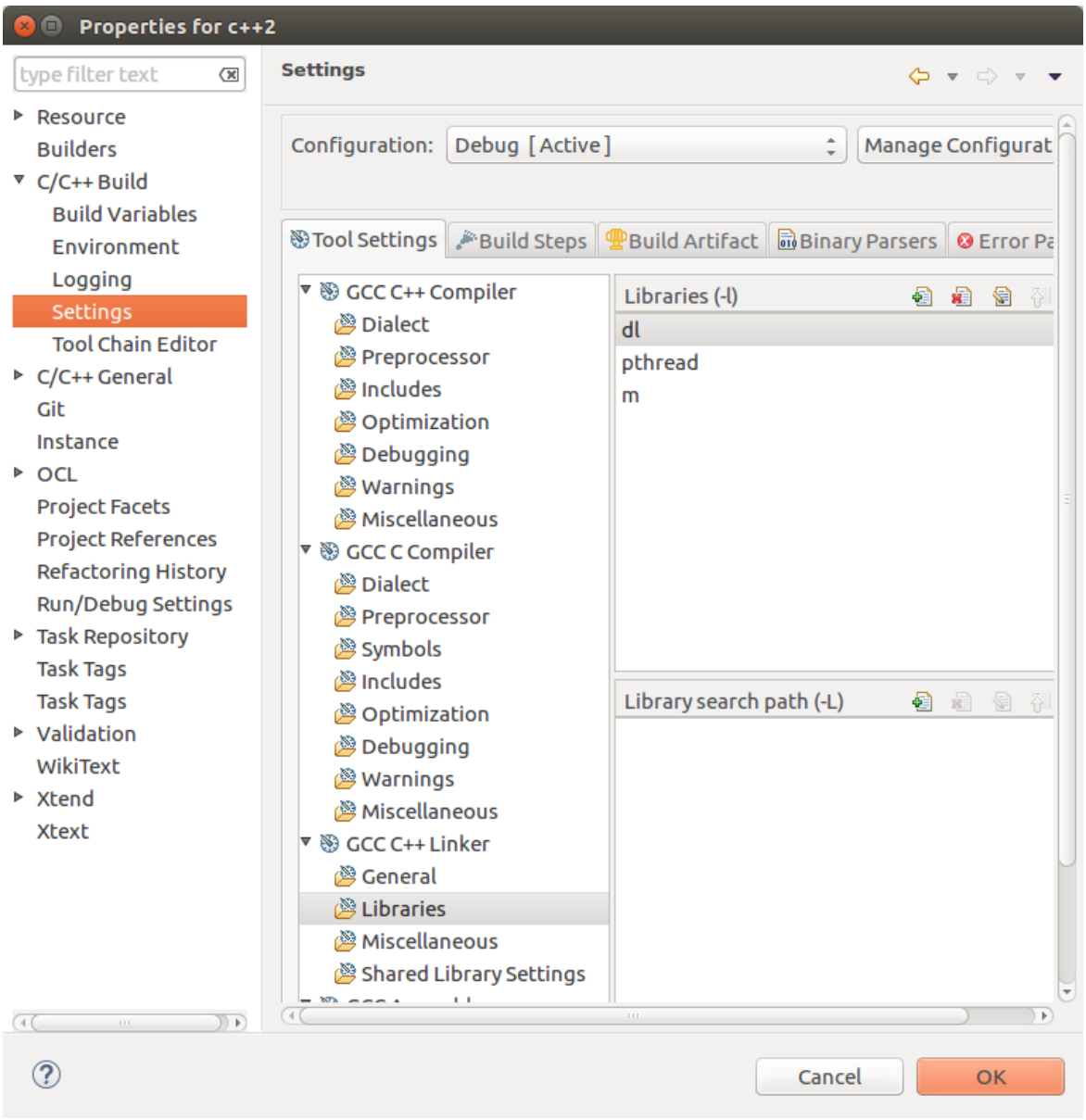

Figure 12:

Note that the code provided by default in the *c++2* project from the Git repository does not compile by itself. TTool must first be executed to generate the classes specific to the system being simulated. Once this is done, the simulator can be compiled and executed from CDT for the desired build configuration.

4.5.2 Launching the Simulator

Like for JDT, the launch of C++ applications can be specified using the launch configurations dialog as introduced in section 4.4.2. One launch configuration is provided for each build configuration and named *c++2 TTool* and *c++2 Debug*.

4.5.3 Using the Debugger

The CDT debugger is very similar to the JDT debugger. However to be able to use it only the *c++2 Debug* launch configuration should be used and triggered from the *Debug As* button. By default, the program execution will automatically stop at the first instruction of the program. This behavior can be changed by editing the debug launch configuration.

5 Testing

It is planned to develop more and more tests for TTool in order to improve the product quality.

5.1 Java

The TTool Java code is tested using the JUnit framework.

5.1.1 Test Projects

Test projects can be found under the *test* directory of the TTool repository. The projects are typically named fr.tpt.ttool.tests.*component _ name* where *component _ name* is the name of the component of TTool being tested.

An example project is *fr.tpt.ttool.tests.util*, which provides test cases for the TTool utility classes such as those for client-server remote communication.

5.1.2 Executing the Tests

Manual Execution

Like for C++ and Java applications, JUnit launch configurations can be defined in Eclipse as shown in figure 13. This configuration will execute all unit tests for class *RSH-Client* and display within the IDE a view of the status of the tests (passed of failed) as shown in figure 14.

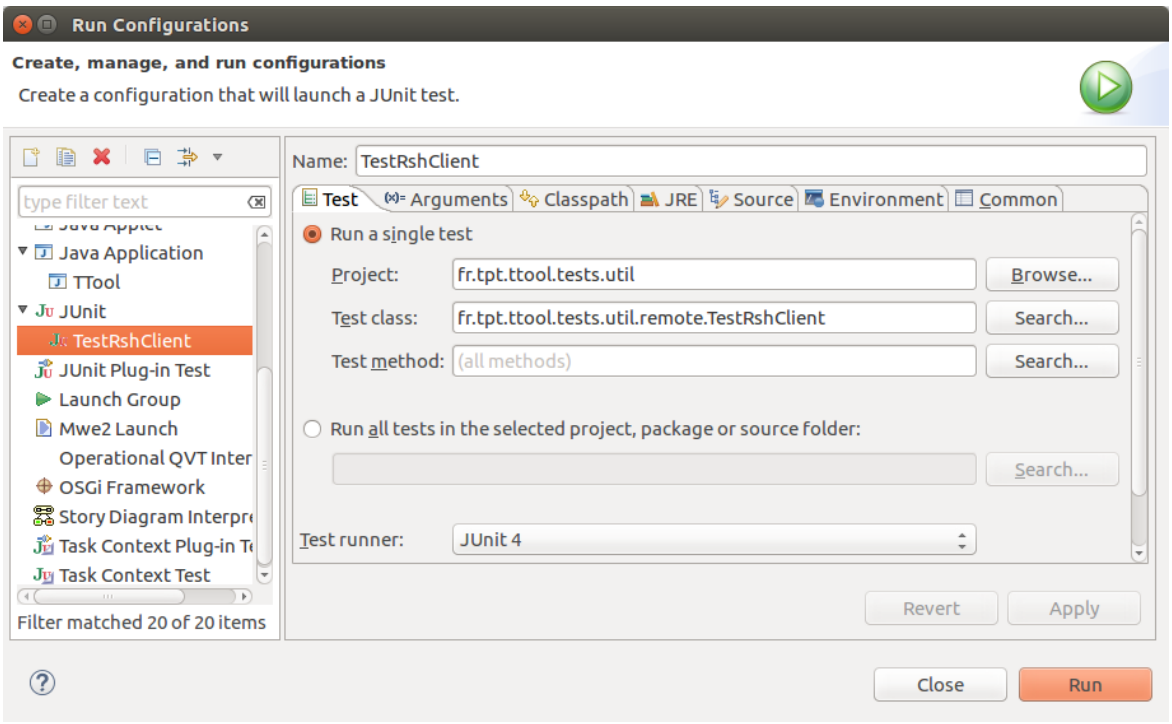

Figure 13:

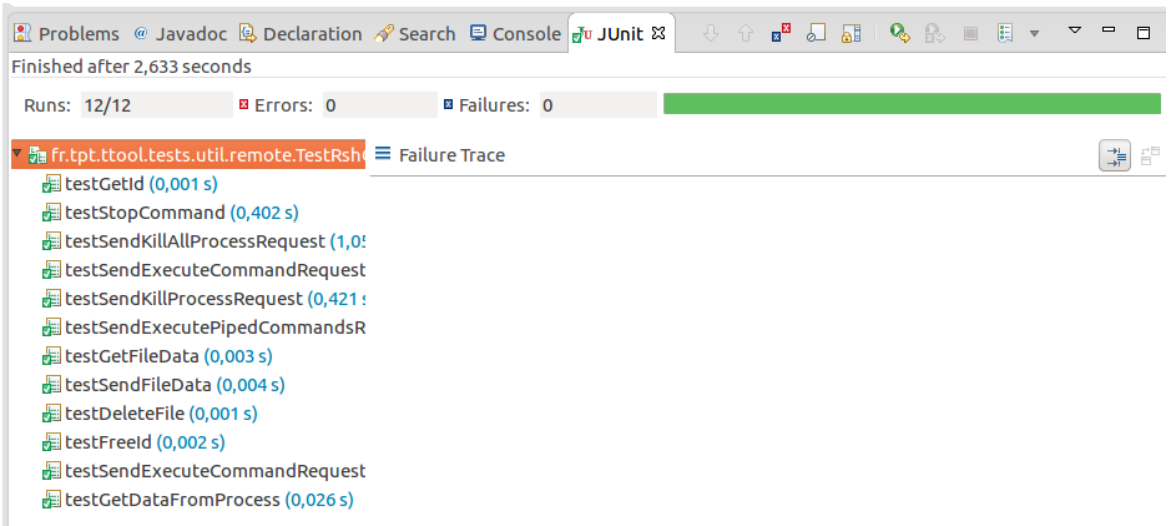

Figure 14:

Automated Execution

JUnit tests can also be launched outside the IDE for automated tests executed for example at the beginning of a tool release. An example code snippet is provided in the following:

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;
import fr.tpt.ttool.tests.util.remote.TestRshClient;
```

```
public class TToolUtilTestsRunner  {

    public static void main(String[] args) {
        Result result = JUnitCore.runClasses(TestRshClient.class);

        for ( final Failure failure : result.getFailures() ) {
            System.err.println( "Test failed : " + failure.toString() );
        }

        if ( result.wasSuccessful() ) {
            System.out.println( "All tests passed." );
        }
        else {
            System.err.println( "Some of the tests failed!" );
        }

        System.exit( 0 );
    }
}
```

5.2 C++

TODO

6 Building

TODO

7 Installing

TODO