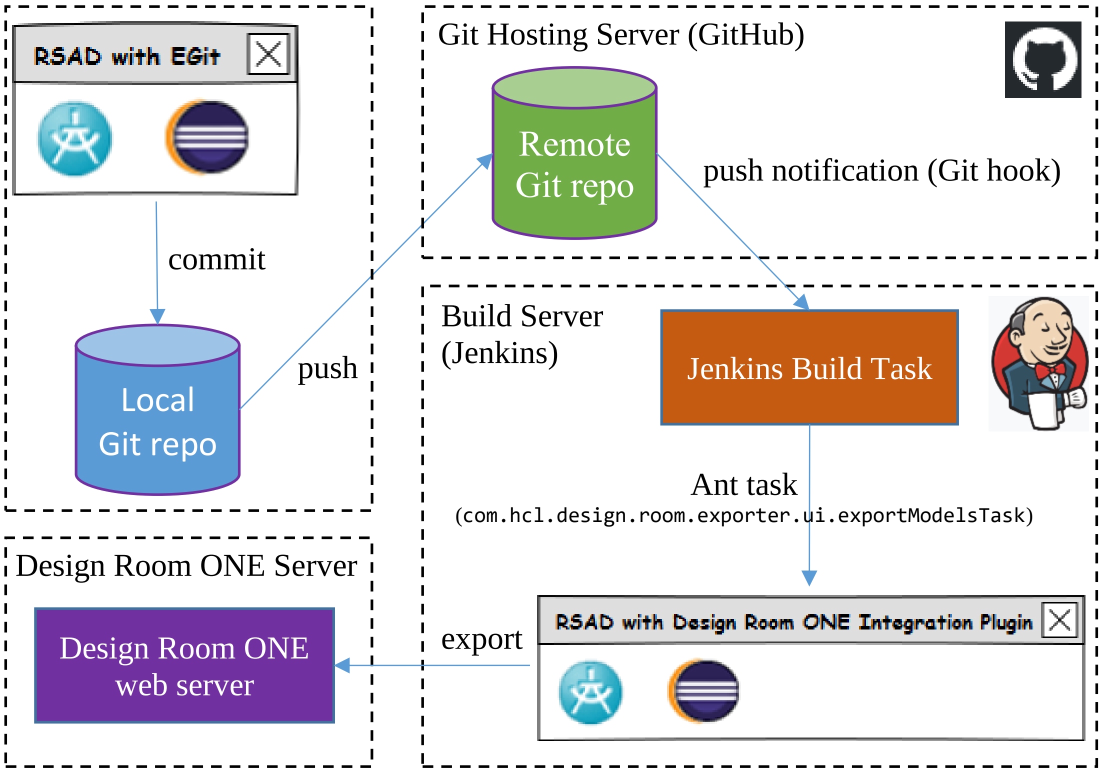
Automated Export to Design Room ONE

Manually exporting a model from Rational Software Architect Designer (RSA) or Rational Software Architect Realtime Edition (RSARTE) or HCL Realtime Software Tooling (RTist) to Design Room ONE is not practical if the model changes frequently, and you want to ensure that the latest version is always available on the Design Room ONE server. In this case you should set-up a scheme where the model is automatically exported. For example, you can choose to export the model once per day, or based on a notification from the SCM system (so that an export takes place as soon as a new version of the model is committed to the SCM system).

In this article we describe how to set-up automatic export to Design Room ONE in an environment where Git is used as SCM system, and Jenkins is used for build automation. We use the EGit integration in RSAD/RSARTE/RTist so that developers can push their model changes to Git from within their modeling tool. A few seconds later, an updated version of the model will be available on the Design Room ONE server.

What we will set-up is illustrated in the picture below:


RSAD with EGit
commit
Local Git repo
push
Git Hosting Server (GitHub)
Remote Git repo
push notification (Git hook)
Build Server (Jenkins)
Jenkins Build Task
Ant task
(com.hcl.design.room.exporter.ui.exportModelsTask)
RSAD with Design Room ONE Integration Plugin
export
Design Room ONE Server
Design Room ONE web server

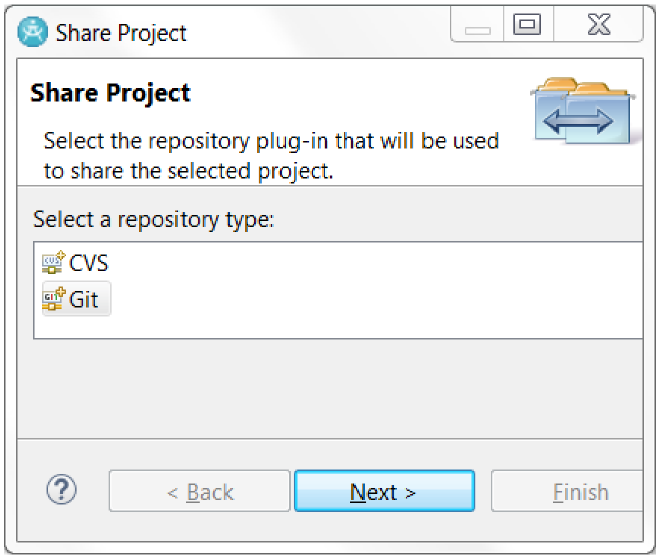
Contents

Create a Git Repository

If you don't already store your models in Git, the first step is to create a Git repository for your model. Here we assume usage of GitHub, the popular web-based Git repository provider. However, any Git repository can of course be used.

1. In the Project Explorer right-click on the model projects that you want to store in Git. Perform the command *Team – Share Project*.

2. Select **Git** in the Share Project dialog that appears.



Share Project

Share Project

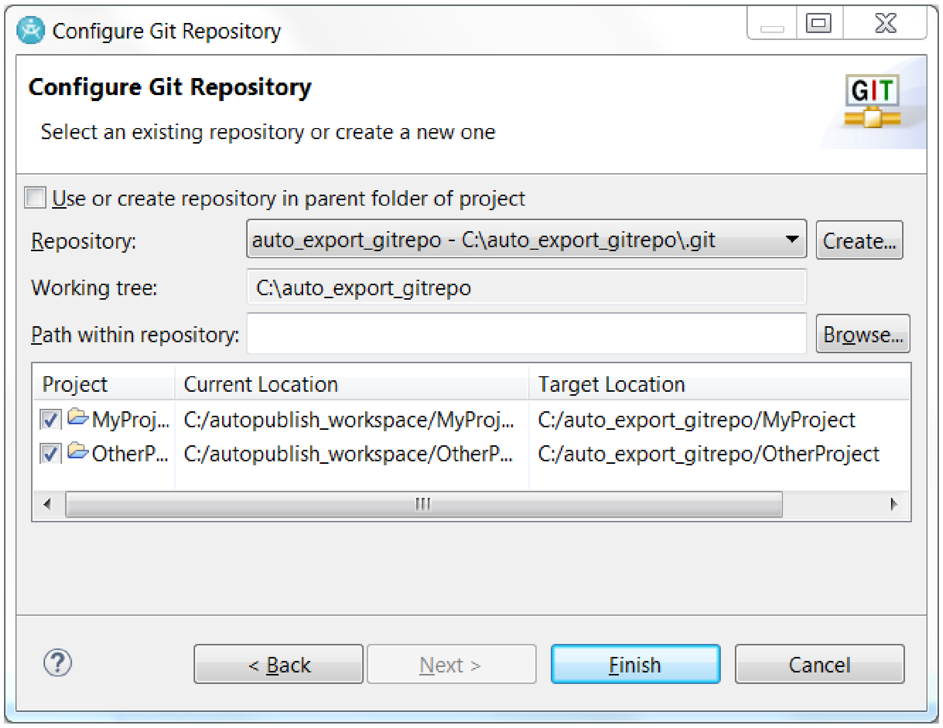
Select the repository plug-in that will be used to share the selected project.

Select a repository type:

CVS

Git

3. In the Configure Git Repository dialog press the **Create** button to specify a folder where to store the local Git repository on your computer. Make sure that all projects that you want to store in the Git repository are marked in the list at the bottom of the dialog. Then press **Finish**.



Configure Git Repository

Configure Git Repository

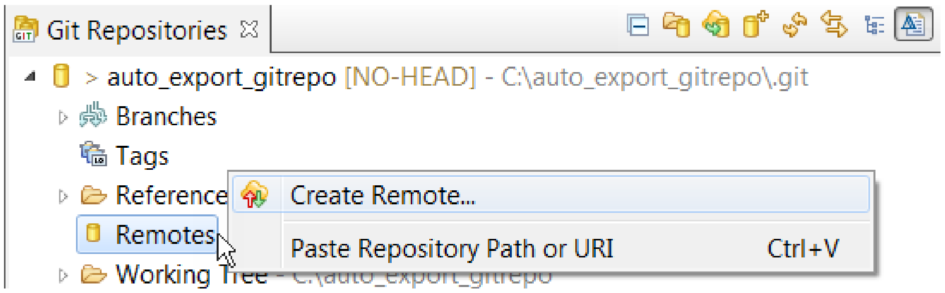
Select an existing repository or create a new one

☐ Use or create repository in parent folder of project

Repository:	auto_export_gitrepo - C:\auto_export_gitrepo\.git
Working tree:	C:\auto_export_gitrepo
Path within repository:	

Project	Current Location	Target Location
✓ MyProj...	C:/autopublish_workspace/MyProj...	C:/auto_export_gitrepo/MyProject
✓ OtherP...	C:/autopublish_workspace/OtherP...	C:/auto_export_gitrepo/OtherProject

4. Switch to the Git perspective and locate your newly created Git repository in the **Git Repositories** view. We now need to connect this local Git repository to a remote Git repository at GitHub. Right-click on the **Remotes** node and do **Create Remote**.

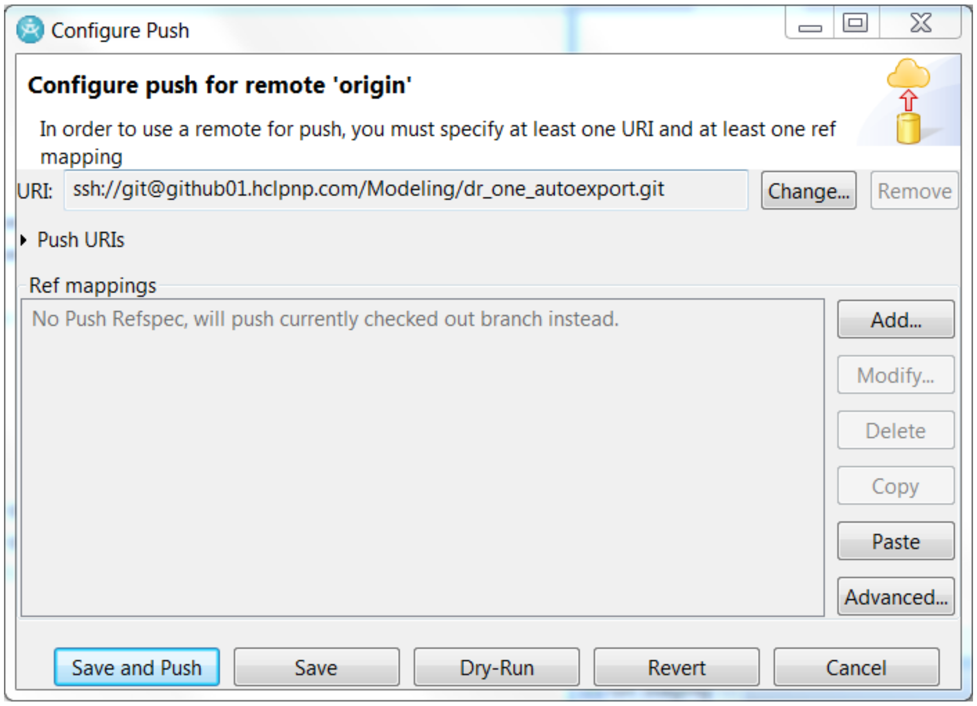


Git Repositories

> auto_export_gitrepo [NO-HEAD] - C:\auto_export_gitrepo\.git

Branches
Tags
References
Remotes
Working Tree - C:\auto_export_gitrepo
Create Remote...
Paste Repository Path or URI Ctrl+V

5. Give a name to the Remote (the default name “origin” is fine). Then press **OK** to configure how to push changes from your local Git repository to the remote Git repository. In the Configure Push dialog enter the URI of your GitHub repository.



Configure Push

Configure push for remote 'origin'

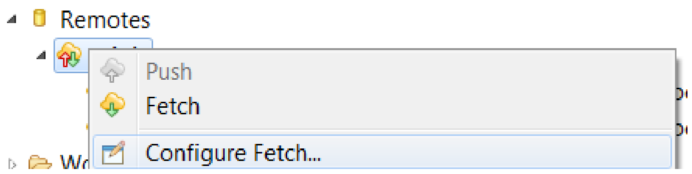
In order to use a remote for push, you must specify at least one URI and at least one ref mapping

URI: ssh://git@github01.hclpnp.com/Modeling/dr_one_autoexport.git

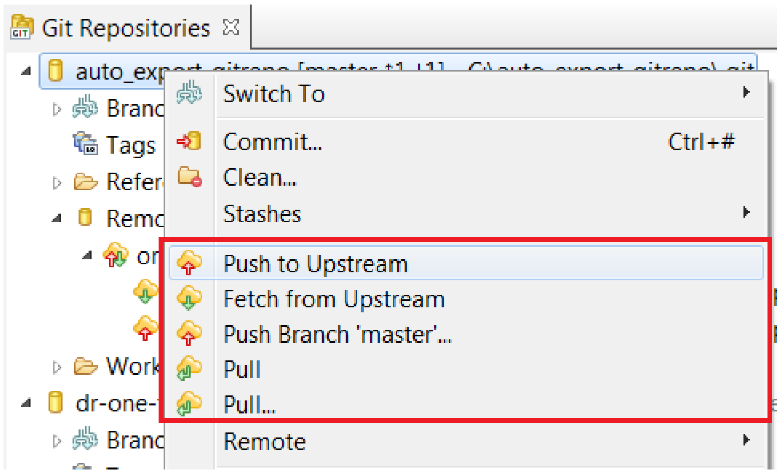
No Push Refspec, will push currently checked out branch instead.

Press **Save** to close the dialog.

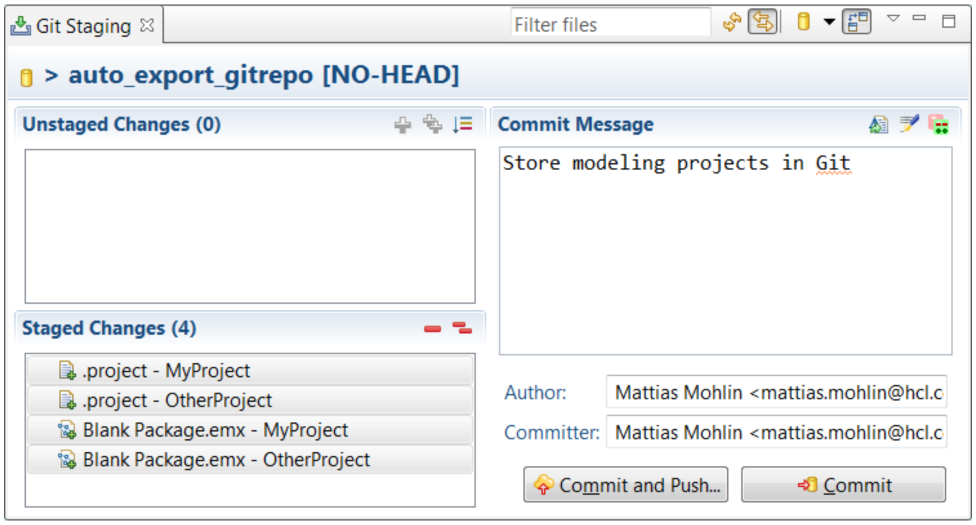
6. Now configure how to fetch changes from the remote Git repository to your local Git repository. Right-click on the Remote node in the Git Repositories view and run the **Configure Fetch** command.



Add a Ref mapping and specify the name of the branch in the remote repository from where you want to fetch changes.
Now you have configured your local Git repository so that you can use the **Push**, **Fetch** and **Pull** commands from the context menu to push changes from the local to the remote Git repository, and to fetch or pull changes in the opposite direction.



7. The first change to push to the remote repository is the initial set of files in your model projects. First stage the files from the **Git Staging** view by dragging all unstaged files into the **Staged Changes** area. Then write a commit message and press the **Commit and Push** button.

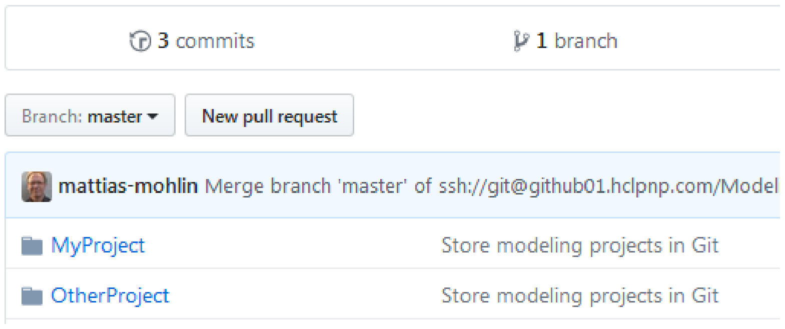


```
Store modeling projects in Git
```

Author: Mattias Mohlin <mattias.mohlin@hcl.c

Committer: Mattias Mohlin <mattias.mohlin@hcl.c

If the push fails, you may first have to pull changes from the remote Git repository that you don’t yet have in your local Git repository.
After a successful push you should be able to see your changes on GitHub:



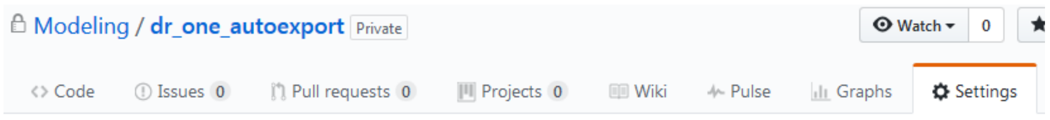
mattias-mohlin Merge branch 'master' of ssh://git@github01.hclpnp.com/Model

MyProject	Store modeling projects in Git
OtherProject	Store modeling projects in Git

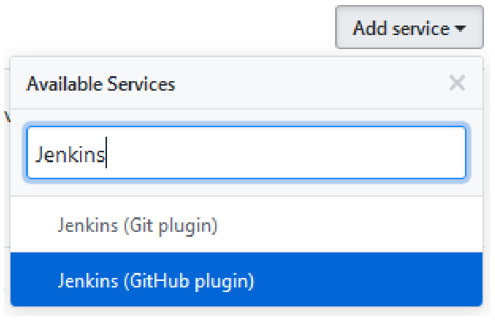
Configure the Git Repository for Push Notifications

The next step is to configure the GitHub repository so that it sends a notification each time a new commit is pushed to it. We will use this notification to trigger a build task in Jenkins which can perform the export to Design Room ONE.

8. Go to your GitHub repository in a web browser. Click on the **Settings** tab.

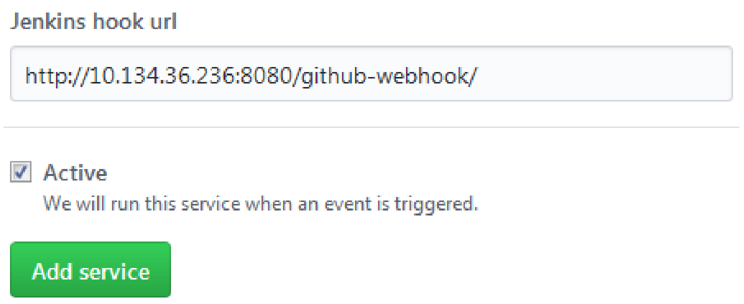


9. Click on **Hooks & services** in the list to the left. Then click the **Add service** button and select the **Jenkins (GitHub plugin)** service.



If your Git repository is not on GitHub but on another server, use the **Jenkins (Git plugin)** instead.

10. Enter the URL of your Jenkins server and append "/github-webhook/" and then press the **Add service** button. For example:


Jenkins hook url

http://10.134.36.236:8080/github-webhook/

☑ Active

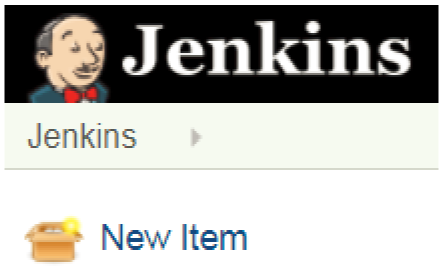
We will run this service when an event is triggered.

The added GitHub service will trigger a Jenkins build task every time new changes are pushed to Git. Now it's time to create that build task in Jenkins.

Create a Jenkins Build Task

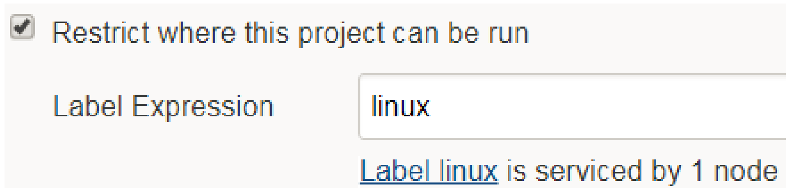
Create a new build task using the Jenkins web page.

11. Open a web browser and log in to Jenkins. Click on **New Item** in the list to the left.



12. Create a new Freestyle project. Enter a name for the build task that does not contain spaces or other "strange" characters. For example, "DRONEAutoExport".

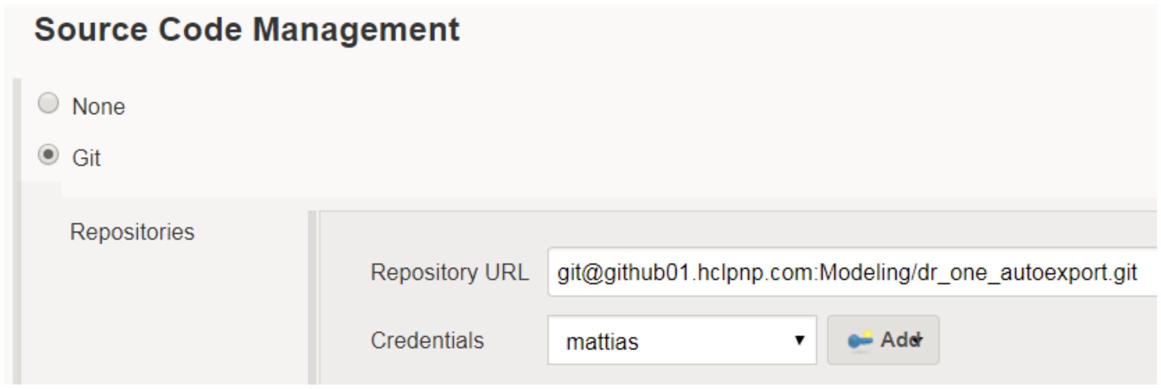
13. The build task will perform an export to Design Room ONE, so it is important that it is executed on an agent machine where RSAD/RSARTE/RTist with the Design Room ONE integration plugin is installed. You can use the **Restrict where this project can be run** option to specify an agent machine to use, in case the modeling tool is not installed on all of them.


☑ Restrict where this project can be run

Label Expression: linux

Label linux is serviced by 1 node

14. In the Source Code Management section mark **Git** and enter the URL to you GitHub repository.


Source Code Management

None

[✓ Git]

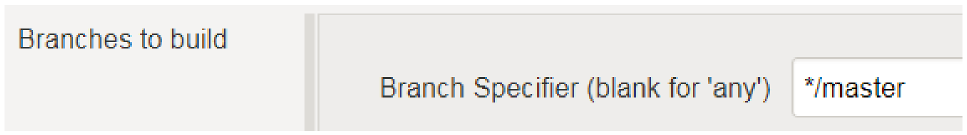
Repositories

Repository URL git@github01.hclpnp.com:Modeling/dr_one_autoexport.git

Credentials mattias

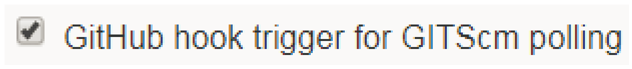
If you login to GitHub with SSH you can click the **Add** button and specify your private key to let Jenkins use it for authenticating with GitHub.

15. You can specify the branch to monitor in the **Branch Specifier** field. If you use the master branch just leave the default value unchanged.


Branches to build

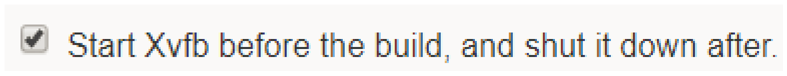
Branch Specifier (blank for 'any') */master

16. In the Build Triggers section mark the **GitHub hook trigger for GITScm polling** checkbox.

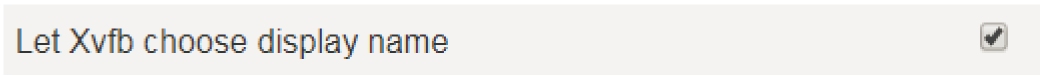

✓ GitHub hook trigger for GITScm polling

If you don't use GitHub but another Git server you would instead mark the **Poll SCM** checkbox and specify a schedule for when the Git repository should be polled for changes.

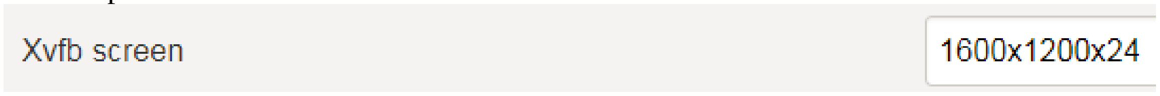
17. Our build task will perform the model export to Design Room ONE by invoking the Ant task `com.hcl.design.room.exporter.ui.exportModelsTask` provided by the Design Room ONE integration plugin. This Ant task requires a display to be available on the agent machine. If the agent machine you specified above does not have a display you can scroll down to the Build Environment section and mark the **Start Xvfb before the build, and shut it down after** checkbox. This will ensure that a display server is started before running the build task.


✓ Start Xvfb before the build, and shut it down after.

Also press the **Advanced** button next to this checkbox and mark the checkbox **Let Xvfb choose the display name**.

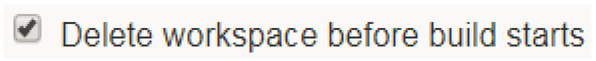

Let Xvfb choose display name ✓

Finally, set the option **Xvfb screen** to specify an appropriate screen resolution and color depth:


Xvfb screen 1600x1200x24

Without this setting exported diagrams may get a different appearance in Design Room ONE.

18. In the same section also mark the checkbox **Delete workspace before build starts**.


✓ Delete workspace before build starts

This is not strictly necessary, but it ensures that each time Jenkins runs the build task, it will start with a new workspace. It is recommended to have this checkbox marked while working on the set-up since it can make it easier to troubleshoot problems. However, once the set-up is ready and works, you can unmark this checkbox to make the build task run a bit faster.

19. In the Build section click the **Add build step** button and choose **Execute shell**. Write a script that starts an eclipse application.
For RSAD the application id should be
`org.eclipse.ant.ui.antRunner`
In order to better capture real-time specific details about the model for RSARTE and RTist the application id should be
`com.hcl.design.room.exporter.ui.DRExporter`

Here is an example of what a bash script for Linux and RSAD could look like. Adjust it as necessary depending on the agent machine you use for executing the build task.

```
#!bash -x
/storage/IBM/SDP961/eclipse -nosplash -data $WORKSPACE/tmp_ws -application
org.eclipse.ant.ui.antRunner -file $WORKSPACE/export.xml -
Dworkspace=$WORKSPACE
```

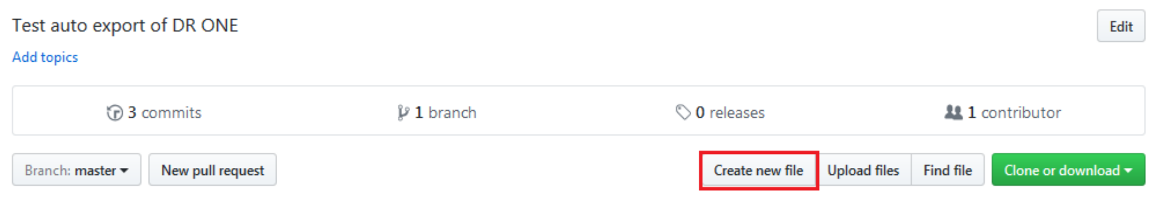
$WORKSPACE is an environment variable set by Jenkins to the absolute path of the folder assigned to the build. The build task pulls files from GitHub into that folder. We also create a temporary Eclipse workspace (`tmp_ws`) in this folder and run Eclipse in headless mode on that workspace with an Ant file `export.xml` as input. We will create that file shortly, but for now just notice that we use the Ant runner that requires a display (`org.eclipse.ant.ui.antRunner`). That is the reason why the build task must start the display server Xvfb.

20. To test that your build task works as expected you can now trigger a build manually from Jenkins. Click on **Build Now**. When the build has finished click its link in the **Build History** list and then click **Console Output**. Read the printouts and ensure that

the build task successfully cloned the Git repository and then ran the script you wrote. However, you should expect to see an error message:

```
BUILD FAILED
Buildfile: /storage/jenkins/workspace/DRONEAutoExport/export.xml does not
exist
```

This is expected since we didn't yet write that file. Now let's do it directly in GitHub. Press the button **Create new file** on your main repository page in GitHub:



Create a file `export.xml` with this content:

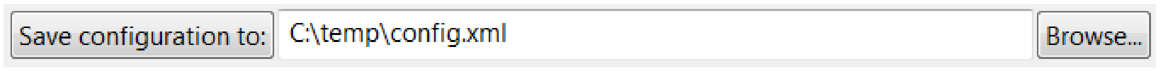
```
<project name="automaticExport" default="export" basedir=".">
  <target name="export">
    <com.hcl.design.room.exporter.ui.exportModelsTask
      configuration="${workspace}/config.xml"
      importFrom="${workspace}">
    </com.hcl.design.room.exporter.ui.exportModelsTask>
  </target>
</project>
```

21. Scroll down to the bottom of the page and press the button **Commit new file**. If you now switch back to Jenkins you will see that our commit of `export.xml` triggered a build. However, it still fails, but now because of another file that is missing:

```
BUILD FAILED
/storage/jenkins/workspace/DRONEAutoExport/export.xml:5: Cannot find file:
/storage/jenkins/workspace/DRONEAutoExport/config.xml
```

The file `config.xml` is referenced by the Ant build file `export.xml` and contains the settings that control how to export from RSAD/RSARTE/RTist to Design Room ONE.

22. Create the file `config.xml` in GitHub in the same way as you created `export.xml`. The easiest way to get the contents of this file is to use the wizard for exporting designs in RSAD/RSARTE/RTist. Follow the normal steps for performing a manual export to Design Room ONE, but on the 3rd wizard page do not perform the export, but instead enter a file name for a configuration file and press the **Save configuration to** button.



Here is an example of what the file could look like when exporting two models (your Design Room ONE server URL will be different):

```
<?xml version="1.0" encoding="UTF-8"?>
<drexport designName="RSAD_AutoExported_Designs" insecure="true"
logFile="${workspace_loc}/drexport.log" serverURL="https://dr-
one.hclpnp.com/dr">
<resources>
  <resource path="/MyProject/Blank Package.emx"/>
  <resource path="/OtherProject/Blank Package.emx"/>
</resources>
<workingSets/>
</drexport>
```

To avoid having to update this file each time you add a new model file to your projects, you can modify the file slightly to specify patterns instead of hardcoded paths:

```
<resource pattern="/MyProject/*.emx"/>
<resource pattern="/OtherProject/*.emx"/>
```

23. As soon as you commit the file `config.xml` to GitHub you should notice another build being run by Jenkins. And this time it should be successful with a printout similar to this:

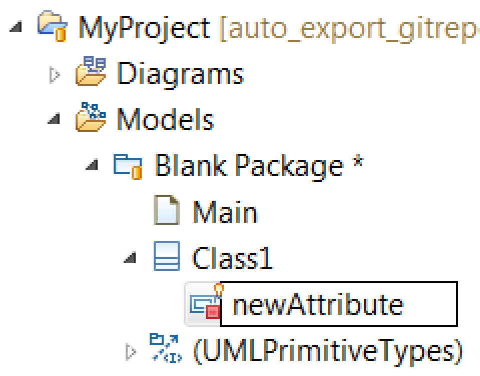
```
[com.hcl.design.room.exporter.ui.exportModelsTask] 08:27:49 : EXPORT
COMPLETED. ELAPSED TIME 6886 ms ( 0 h 0 min 6 sec)
BUILD SUCCESSFUL
```

Our set-up is complete, and we are now ready to test everything from within RSAD!

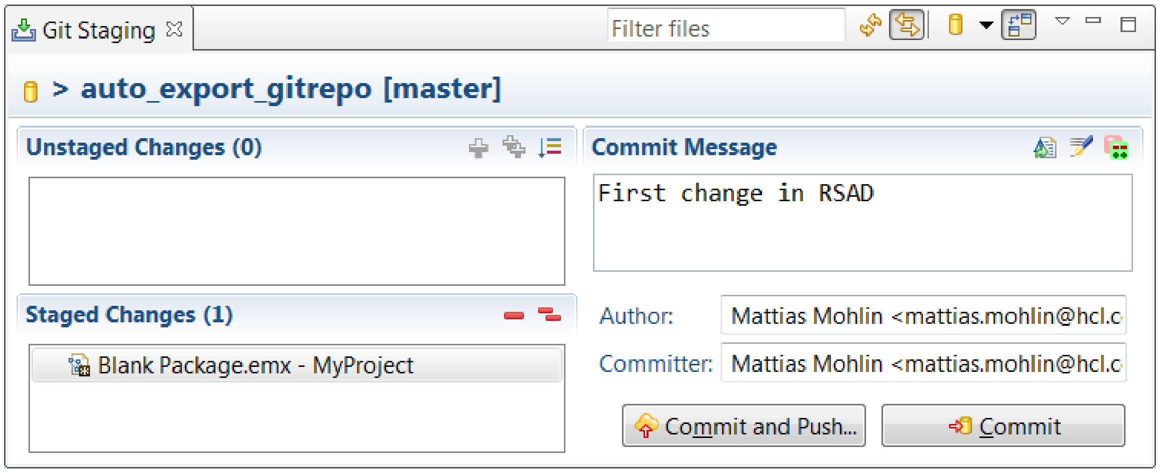
Test Automatic Export from RSAD/RSARTE/RTist

We have now created a set-up where a user can work in the modeling tool and push changes to Git, and it will immediately trigger an export of the updated model to Design Room ONE. Let’s try with a small change.

24. Create a new model element in one of the projects you store in Git. For example, create a new attribute:



25. Save the model and switch to the Git perspective. In the Git Staging view drag the modified file to the **Staged Changes** area, write a commit message and press the **Commit and Push** button.



```
First change in RSAD
```

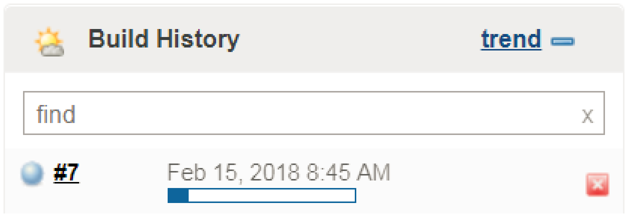
Blank Package.emx - MyProject

Author: Mattias Mohlin <mattias.mohlin@hcl.c

Committer: Mattias Mohlin <mattias.mohlin@hcl.c

If the push fails, you first have to pull changes from the remote Git repository that you don't yet have in your local Git repository (remember those files `export.xml` and `config.xml` that you created?). Then you can push again (use the context menu of the Git repository in the Git Repositories view both for pulling and pushing changes).

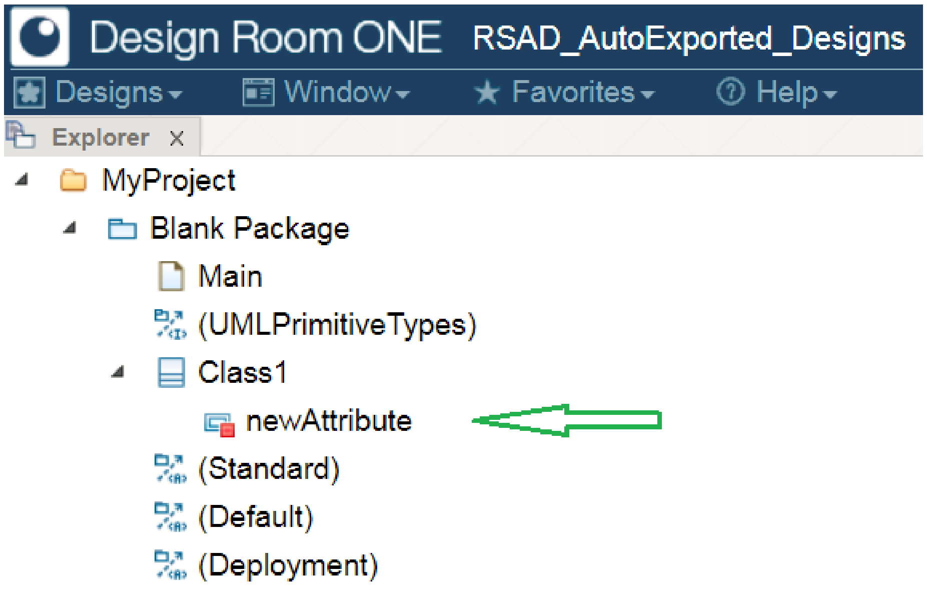
26. Right after pushing your changes switch to Jenkins and note that a new build was started:



#7 Feb 15, 2018 8:45 AM

Wait until it has finished.

27. Now go to the Design Room ONE web application and open the design you exported your models to (the name is specified in the file `config.xml`).



The added attribute is there!