Team Concert Plugin

Plugin Information
View Team Concert on the plugin site for more information.

Integrates Jenkins with Rational Team Concert source control and build using the richer features of the build toolkit instead of the command line.

With the build toolkit this plugin adds traceability links from a Jenkins build to an RTC build result, workspace and snapshot. It also publishes links to work items, change sets and file contents captured in the snapshot. It leverages the current RTC features and workflows that users are already familiar with such as, emails, toaster popups, reporting, dashboards, etc.

Documentation

Rational Team Concert Help Topics

1. Rational Team Concert Build Overview
2. Installing the Build System Toolkit
3. Creating encrypted password files
4. Hudson/Jenkins build engine type
5. Dedicated build workspaces

Requirements

Jenkins

- Team Concert Plugin v 1.2.0.5 and above requires Jenkins 1.625.1 and above. Fix for JENKINS-26100 requires Jenkins 2.60 and above, workflow-job 2.12 and above
- Team Concert Plugin v 1.1.9.3 till v 1.2.0.4 requires Jenkins 1.580.1 and above.
- Team Concert Plugin v 1.1.2 and later depends on the Jenkins Credentials plugin version 1.10 or later.

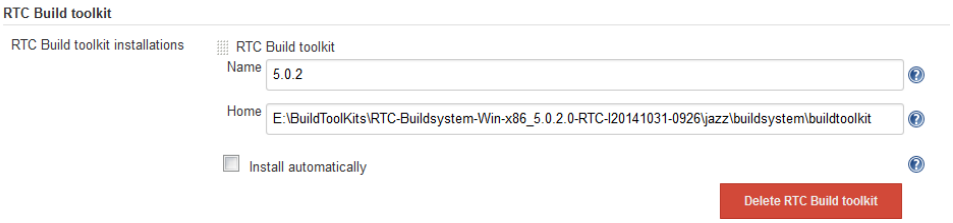
RTC

- This plugin requires Rational Team Concert Build Toolkit version 4.0.7 or newer. Older versions of the plugin supports build toolkit versions starting from 3.0.1.5. See the Installing the Build System Toolkit help topic to learn how to install the build toolkit.
- For all the supported build configurations - Build Definition, Repository Workspace, Stream and Snapshot - **a valid build toolkit should be present on both the master and slave machines** and the Jenkins jobs should be configured to use this toolkit.
- Some features depend on specific Rational Team Concert build toolkit or server versions. See below.
    - Stream configuration works only from build toolkit v. 5.0 or higher.
    - Post Build Deliver for Build Definition configuration introduced in Team Concert Plugin v. 1.2.0.3 depends on Rational Team Concert server version 6.0.4 or higher.
    - Support for Load Rules in build definition has some requirements on the version of RTC client used to create the build definition. See **Load Rules Support** section for more details.
    - If you will be fetching workspaces that contain symbolic links, there is some additional symbolic link setup required. See **Symbolic Link Support** section for more details.
    - Version details of build toolkit can be obtained in the build log only if you are using build toolkit version 5.0.2 and above.

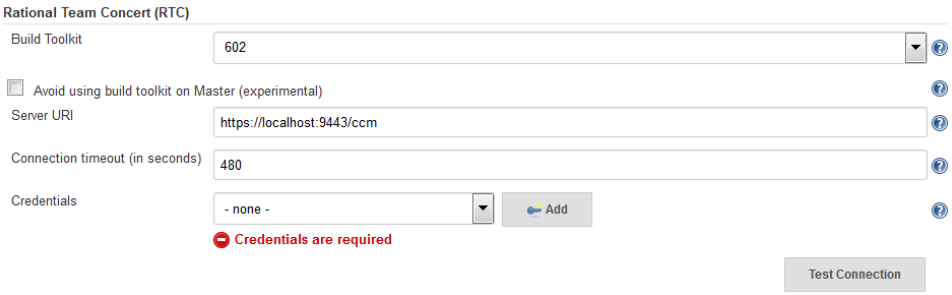
Jenkins Configuration

1. Navigate to the Jenkins Global Tool configure page (Jenkins > Manage Jenkins > Global Tool Configuration) and find the "RTC Build toolkit" section. This section is used to define one or more build toolkits available to the plugin.If you are using Jenkins 1.x, this will be under (Jenkins -> Manage Jenkins -> Configure System)
    - 

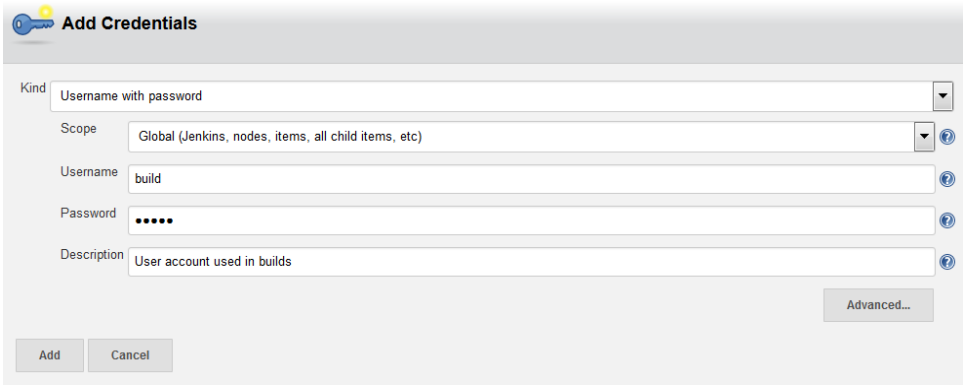
2. Click the "RTC Build toolkit installations..." button and add a new build toolkit.
    - See the Installing the Build System Toolkit help topic to learn how to install the build toolkit.
    - There can be multiple RTC build toolkits associated with one jenkins instance.



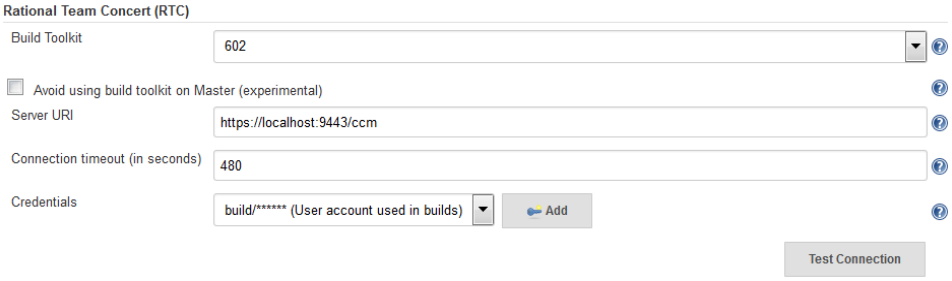
3. Click the "Apply" button to apply the changes.
4. Navigate to the Jenkins Global Configuration page (Manage Jenkins -> Configure System).
5. Find the "Rational Team Concert (RTC)" section. This section is used to define global connection settings that will be the defaults for any jobs created with the plugin. If connection settings will be set on each job, then skip this section.
    - Select a build toolkit



6. Credentials are managed by the Credentials plugin. The Team Concert plugin supports username and password type credentials. Credentials can be defined within a domain or a folder (if you are using the folder's plugin).



7. Choose the credentials to use when logging into RTC for polling and building.

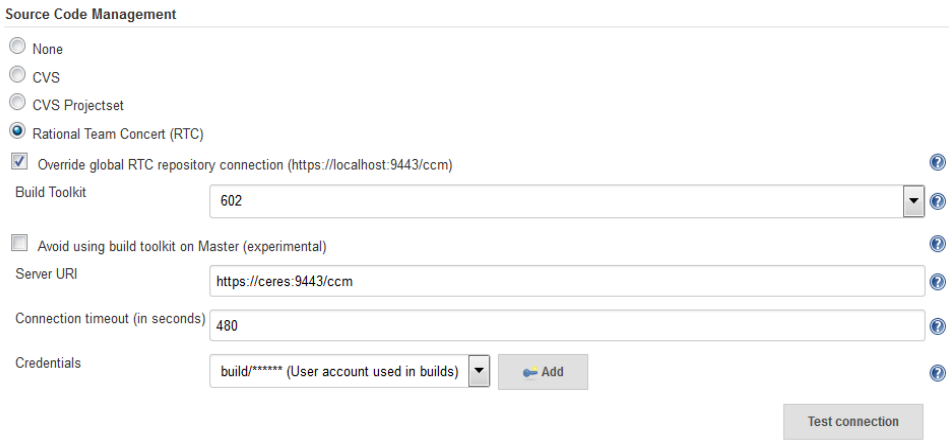


    - If you are using the 1.0.12 (or earlier) version of the Team Concert plugin, instead of credentials, you will need to supply a userId and password or password file.
8. Click the "Test connection" button to verify the repository connection details.

9. Click the "Save" button to save the settings and return to the Jenkins main page.

Job Configuration

1. Create a new free-style software project and find the "Source Code Management" section.
2. Select "Rational Team Concert (RTC)".
3. If global connection settings were not configured above or do not apply to this job, then check the "Override global RTC repository connection" check box and enter the connection settings here.

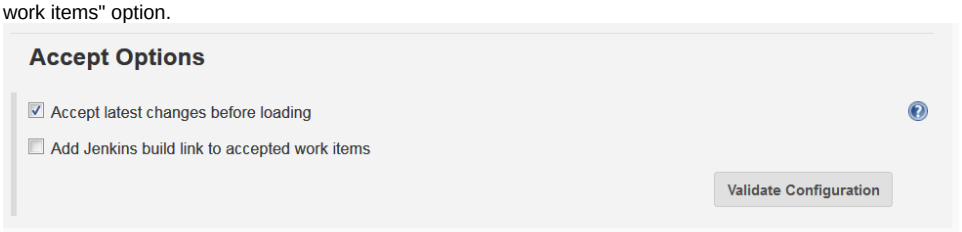


4. Click the "Test connection" button to verify the repository connection details.

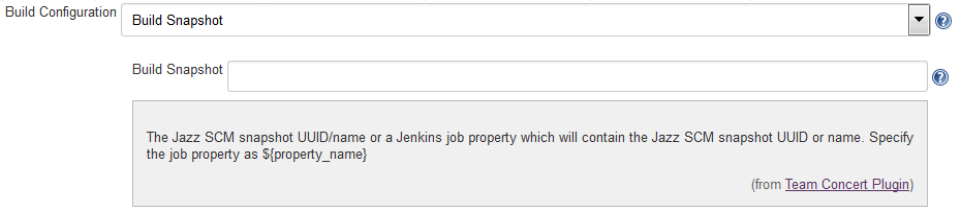
5. Prior to 1.2.0.0 a job can be configured with RTC SCM using either a build definition or a build workspace. In 1.2.0.0 there is support to configure RTC SCM with a build stream or build snapshot also.
6. To benefit most from the integration between this plugin and RTC Build, select "Build Definition" from the Build Configuration dropdown and enter a build definition ID. See the Hudson/Jenkins build engine type help topic to learn how to create a Jenkins build definition. Follow these steps to setup a Jenkins Build Definition and Jenkins Job to avoid a catch-22 situation. A Jenkins job requires a Hudson/Jenkins build definition and a Hudson/Jenkins build definition requires a Jenkins job. RTC actually won't let you save the build definition without a job selected. However, Jenkins will let you save a job without a build definition. So it is important to configure your build definition and job this way.
    a. In Jenkins, create the job first using RTC for source control, but with no build definition. Leave the *Build Definition* text box blank. Save the Jenkins Job.
    b. In RTC, create a Jenkins build engine that connects to the Jenkins server. See Creating a build engine
    c. In RTC, create a build definition that uses the build engine created in step b and select the job created in step a. See **Creating a build definition**
    d. Lastly, in Jenkins, open the Jenkins job and set the *Build Definition* field with the id of the build definition created in step c.

    - Notice the "Build Configuration" dropdown which replaces the radio buttons for build definition and build workspace.
    - Click the "Validate" button to verify the RTC build definition exists.
7. To load the jenkins build workspace using a RTC repository workspace, select "Build Workspace" from the Build Configuration dropdown. See the Dedicated build workspaces help topic to learn how to create a build workspace.

    a.
    b. Click the "Validate" button to verify the RTC build workspace exists.
    c. To add a "Related Artifact" link to a Jenkins build in all the included work items, select the option "Add Jenkins build link to accepted work items" option.

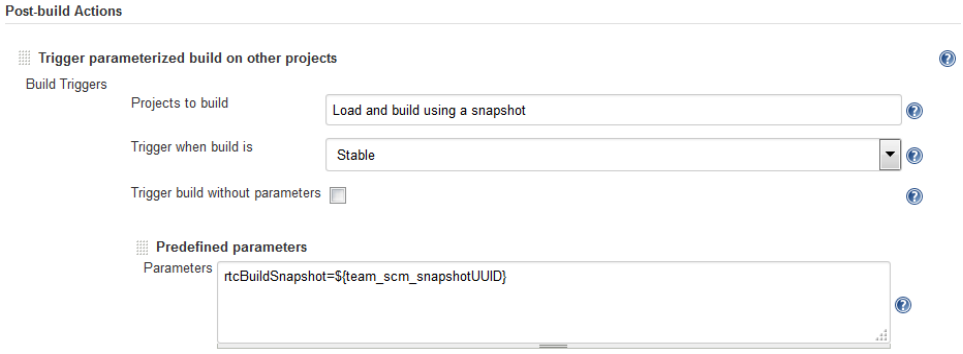


    d.
8. To load the jenkins build workspace using a snapshot, select "Build Snapshot" from the Build Configuration dropdown. This configuration is mainly intended to be used in builds that capture the current state of the RTC SCM workspace/stream in a snapshot and start downstream builds that would populate the jenkins build workspace from the snapshot created and passed from the upstream builds.



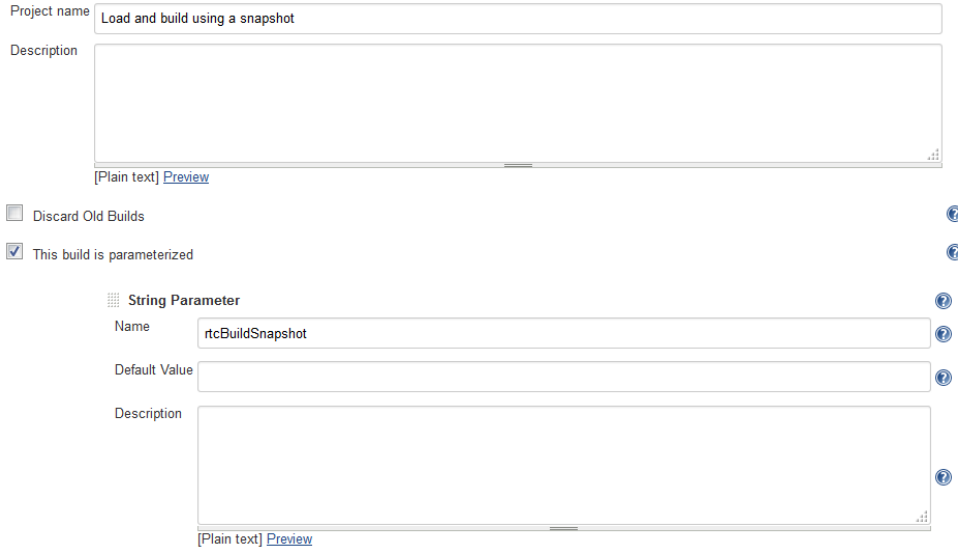
    a.
    b. To start a downstream snapshot build Parameterized Trigger plugin is required.

i. Consider a parent job that is configured to load from a RTC repository workspace. When the build runs, Team Concert Jenkins plugin creates a snapshot on the build workspace. The snapshot uuid is available as the build environment property team_scm_snapshotUUID.
ii. Add a post build action to trigger parametrized build on other projects.

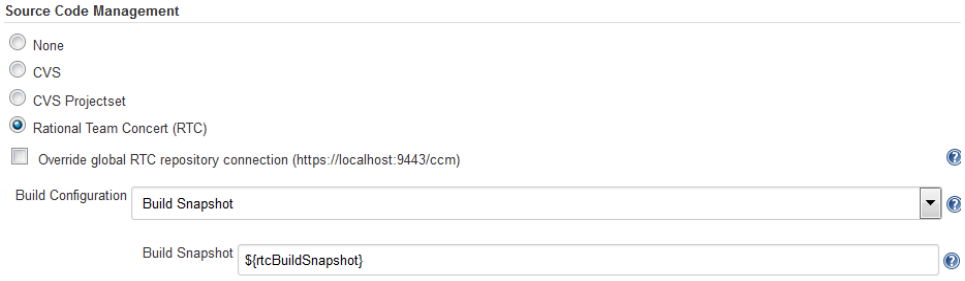


iii.

c. Configure a downstream snapshot build
i. Create a new job and with a string parameter named "rtcBuildSnapshot"



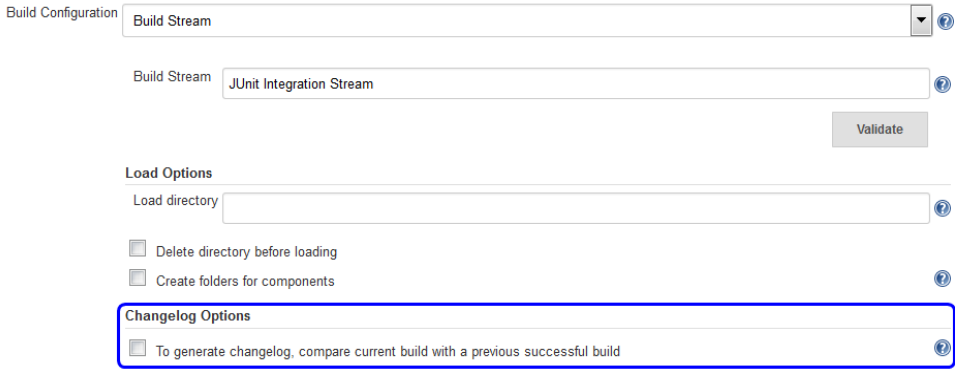
ii.
iii. Configure Rational Team Concert under Source Control options to build from a snapshot.



iv.
d. Now when an upstream build is started and once it is done it will trigger the downstream build with the UUID of the snapshot created on the workspace.
e. Note that the change log is not generated and polling is not supported for load from snapshot as this as an immutable configuration.

9. To load the jenkins build workspace using a stream, select "Build Stream" from the Build Configuration dropdown.

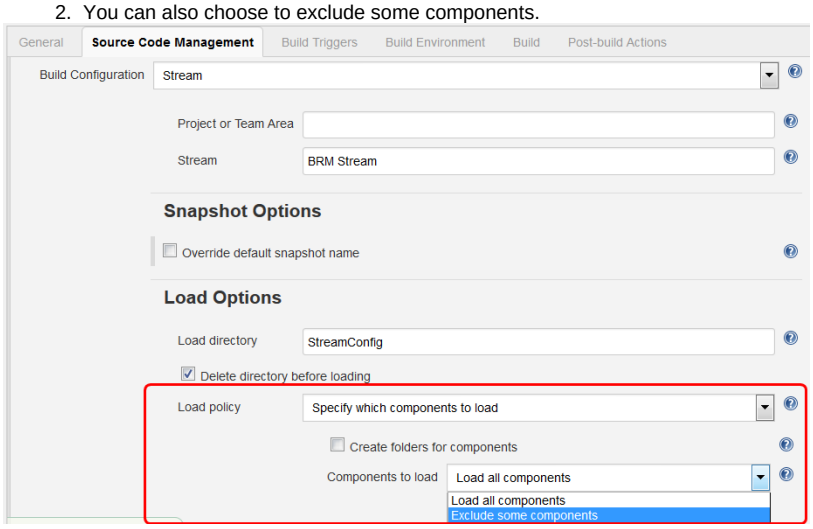
a.
b. Click the "Validate" button to verify the build stream exists.
c. This configuration supports building from the current state of the specified stream.
d. Subsequent builds capture the changes made to the stream since the previous build.
e. In this configuration change log can be chosen to be generated by comparing the current build with the previous successful build. By default this option is unchecked.



f.

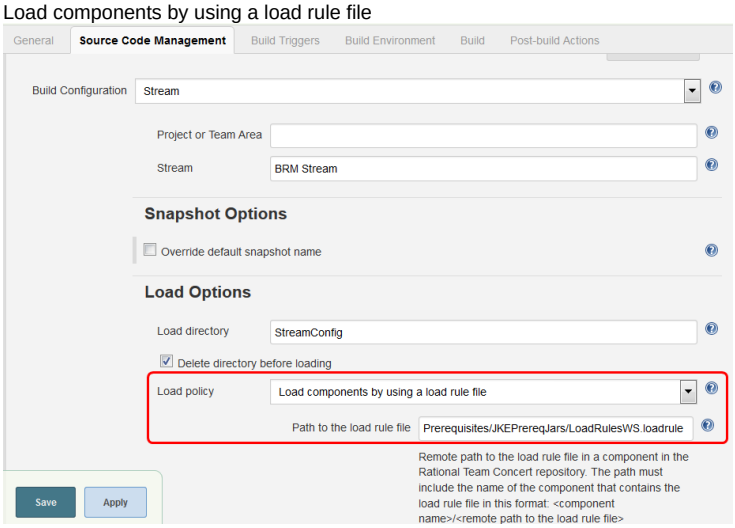
g. For this configuration the RTC user configured globally or for this job needs to have permission to attach snapshots to a stream

10. In 1.2.0.0 some of the load and accept options that were previously configurable only in RTC build definitions, can be specified in the jenkins job configuration. The accept and load options are available for build configurations other than load using a build definition.
    a. The directory on the build machine under which the repository files will be loaded can be specified.
    b. Contents of the load directory can be deleted before reloading
    c. Load Policy field, added in 1.2.0.4, can be used to configure the components to load. You can either specify the components to load or choose to use a remote load rule file or dynamic load rules, to determine which components to load.
        i. Specify which components to load
            1. When specifying components to load you can choose to create folders for components, in which case the load directory would have folders for components at the top level and each of these folders will have the files/folders for that component.
            2. You can also choose to exclude some components.



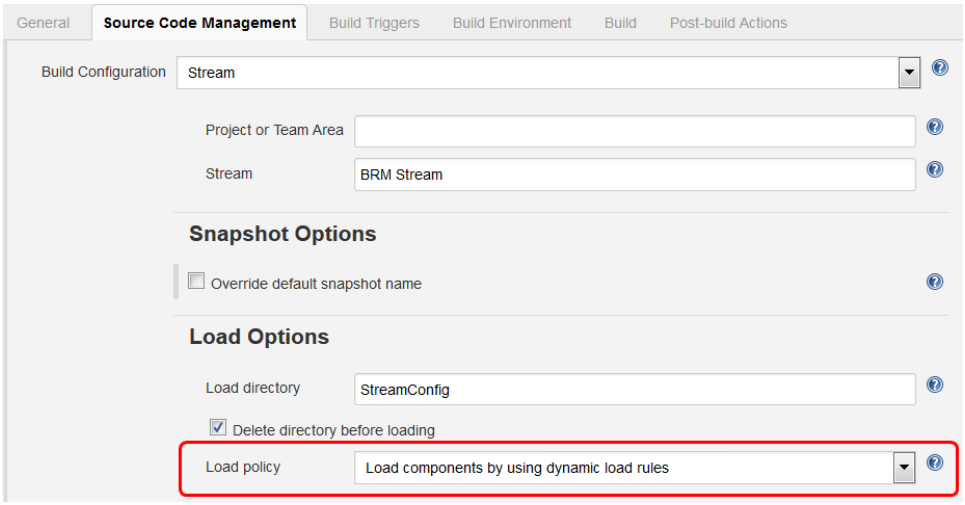
ii.

iii. Load components by using a load rule file

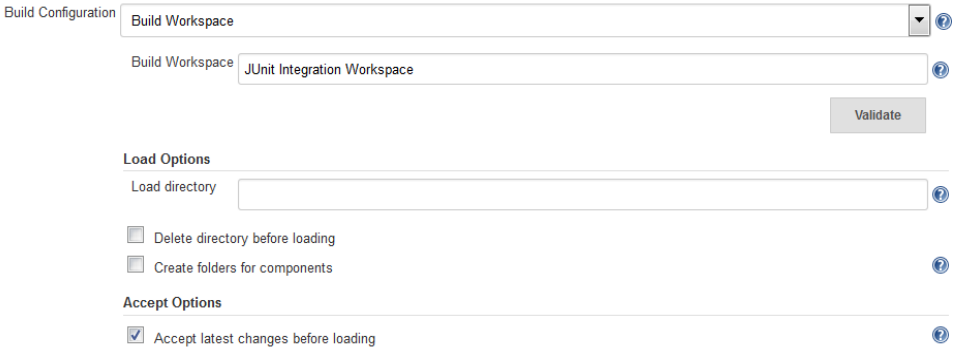


iv.

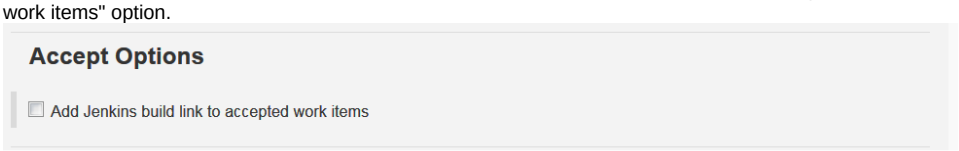
v. Load using dynamic load rules

vi. 

d. For more details on load rules support and how to configure dynamic load rules, see the Load Rules Support section.
e. When loading the jenkins build workspace from a RTC repository workspace, there is an option to configure whether to accept latest changes before loading. By default, this option is selected.



f.
g. To add a "Related Artifact" link to a Jenkins build in all the included work items, select the option "Add Jenkins build link to accepted work items" option.



h.

11. Find the "Build Triggers" section.
12. Check the "Poll SCM" check box to poll for incoming changes to the build workspace.
13. Enter a schedule. Click the help button beside the "Schedule" field to get help with the syntax.
14. Click the "Save" button to save the settings and return to the job page.

Configuring Jenkins job for Post Build Deliver (Build Definition configuration only)

1. In 1.2.0.3, Post Build Deliver is supported for Build Definition configuration. The RTC server version should be 6.0.4 or higher.
2. Configure the RTC Build Definition with Post Build Deliver configuration.
3. In the Jenkins Freestyle job configuration, add the "RTC Post Build Deliver" post build action. Select "Fail on Error", if you want the build to fail if post build deliver fails.
4. Optional : If a Pipeline job is being used, then add the following snippet before the end of the script to perform post build deliver as the last step of the build.
5. **PB deliver snippet**

```
step([$class: 'RTCPostBuildDeliverPublisher', failOnError: true])
```

Using Pipeline as SCM

Team Concert Plugin supports Pipeline as SCM but doesn't support lightweight checkout. Captured below are some ways to use this pipeline feature with RTC SCM.

When you wish to checkout the same content that was initially loaded in the master under pipeline@script folder by Pipeline script from SCM:

Note: In this case, you should set skipDefaultCheckout(true). Otherwise a checkout will happen everytime an agent directive is used. The following snippet can be used to skip default checkout

```
options {skipDefaultCheckout()}
```

Scenario 1 - No load rules in the second checkout

1. Checkout only the JenkinsFile using minimal load rules. That is, configure load rules in the pipeline job's SCM configuration to load just the JenkinsFile and nothing else. This can be done for build definition, repository workspace, snapshot and stream configuration. This keeps the loading time very minimal in the master.

2. Inside the JenkinsFile, if you want to checkout on a slave or master again (not under pipeline@script folder), load the snapshot. For build definition and personal builds triggered from RTC, checkout the build definition once again.

```
For Build Definition

node ("slavexyz") {
if ("${env.personalBuild}" == "true") { // This is true when a personal build is started from RTC.
          echo "Checking out the build definition in node"
          checkout([$class: 'RTCScm', avoidUsingToolkit: false, buildType: [buildDefinition: '<build definition
used in Pipeline script from SCM>', customizedSnapshotName: '', value: 'buildDefinition'], overrideGlobal: false
timeout: 480'])
      } else { // Otherwise we checkout the snapshot created by the Pipeline Script from SCM's checkout.
          echo "Checking out a snapshot in node"
          checkout([$class: 'RTCScm', avoidUsingToolkit: false, buildType: [buildSnapshot: "${env.
team_scm_snapshotUUID}", buildSnapshotContext: [snapshotOwnerType: 'none'], currentSnapshotOwnerType: 'none',
loadDirectory: '.', value: 'buildSnapshot']], timeout: 480])
      }
   }
}

For Stream:
node ("slavexyz") {
          echo "Checking out a snapshot in node from stream configuration"
          checkout([$class: 'RTCScm', avoidUsingToolkit: false, buildType: [buildSnapshot: "${env.
team_scm_snapshotUUID}", buildSnapshotContext: [snapshotOwnerType: 'none'], currentSnapshotOwnerType: 'none',
loadDirectory: '.', value: 'buildSnapshot']], timeout: 480])
}

For Repository workspace:
node ("slavexyz") {
          echo "Checking out a snapshot in node from snapshot configuration"
          checkout([$class: 'RTCScm', avoidUsingToolkit: false, buildType: [buildSnapshot: "${env.
team_scm_snapshotUUID}", buildSnapshotContext: [snapshotOwnerType: 'none'], currentSnapshotOwnerType: 'none',
loadDirectory: '.', value: 'buildSnapshot']], timeout: 480])
}

For snapshot:
node ("slavexyz") {
          echo "Checking out a snapshot in node from snapshot configuration"
          checkout([$class: 'RTCScm', avoidUsingToolkit: false, buildType: [buildSnapshot: "${env.
team_scm_snapshotUUID}", buildSnapshotContext: [snapshotOwnerType: 'none'], currentSnapshotOwnerType: 'none',
loadDirectory: '.', value: 'buildSnapshot']], timeout: 480])
}
```

Scenario 2 - Load rules in the second checkout

1. Checkout the JenkinsFile using minimal load rules. That is, configure load rules in the pipeline job's SCM configuration to load just the JenkinsFile and nothing else. This can be done for build definition, repository workspace, snapshot and stream configuration. This keeps the loading time very minimal in the master.

2. Inside the JenkinsFile, if you want to checkout on a slave or master again (in a different path), configure different load rules that will load the content required for the build. Note that you can parameterize the load rules using Jenkins Job property instead of directly providing the value.

Note: For Build definition, personal builds are not supported when load rules or components to include/exclude are used in Jazz SCM configuration in the build defintion. Therefore, the following sample will error out when it sees a personal build

Note: You can use load rules or components to exclude. In the sample below, I am assuming load rules. You can substitute loadPolicy with useComponenLoadConfig and provide components to exclude.

```
For Build Definition
node ("slavexyz") {
```

```
if ("${env.personalBuild}" == "true") { // This is true when a personal build is started from RTC.
         error "Personal builds not supported when using load rules or components to include/exclude"
      } else { // Otherwise we checkout the snapshot created by the Pipeline Script from SCM's checkout.
          echo "Checking out a snapshot in node"
          checkout([$class: 'RTCScm', avoidUsingToolkit: false, buildType: [buildSnapshot: "${env.
team_scm_snapshotUUID}", buildSnapshotContext: [snapshotOwnerType: 'none'], currentSnapshotOwnerType: 'none',
loadDirectory: '.', value: 'buildSnapshot' loadPolicy: 'useLoadRules', pathToLoadRuleFile: 'Comp1/loadrules/build.
loadRule'], timeout: 480])
      }
   }
}

Stream:
node ("slavexyz") {
          echo "Checking out a snapshot in node from stream configuration"
          checkout([$class: 'RTCScm', avoidUsingToolkit: false, buildType: [buildSnapshot: "${env.
team_scm_snapshotUUID}", buildSnapshotContext: [snapshotOwnerType: 'none'], currentSnapshotOwnerType: 'none',
loadDirectory: '.', value: 'buildSnapshot', loadPolicy: 'useLoadRules', pathToLoadRuleFile: 'Comp1/loadrules
/build.loadRule'], timeout: 480])
}

For Repository workspace:
node ("slavexyz") {
          echo "Checking out a snapshot in node from repository workspace configuration"
          checkout([$class: 'RTCScm', avoidUsingToolkit: false, buildType: [buildSnapshot: "${env.
team_scm_snapshotUUID}", buildSnapshotContext: [snapshotOwnerType: 'none'], currentSnapshotOwnerType: 'none',
loadDirectory: '.', value: 'buildSnapshot', loadPolicy: 'useLoadRules', pathToLoadRuleFile: 'Comp1/loadrules
/build.loadRule'], timeout: 480])
}

For snapshot:
node ("slavexyz") {
          echo "Checking out a snapshot in node from snapshot configuration"
          checkout([$class: 'RTCScm', avoidUsingToolkit: false, buildType: [buildSnapshot: "${env.
team_scm_snapshotUUID}", buildSnapshotContext: [snapshotOwnerType: 'none'], currentSnapshotOwnerType: 'none',
loadDirectory: '.', value: 'buildSnapshot', loadPolicy: 'useLoadRules', pathToLoadRuleFile: 'Comp1/loadrules
/build.loadRule'], timeout: 480])
}
```

When you want to load new content in every agent (including the master) after the initial Pipeline script from SCM checkout

You can add new checkout steps with different options (like loadrules, or load directory) as required by directly referencing the build definition, stream or build workspace in the JenkinsFile, instead of loading from the last snapshot. If you have a build definition configuration and the build is triggered from RTC, then multiple checkouts on the same build definition will reuse the build result instead of creating new ones. This is different from how it works when the build is triggered from Jenkins where each checkout step with the same build definition will still create new build results.

Considerations when using declarative pipeline

If you are using declarative pipeline, then every agent directive will cause a checkout to happen in that agent using the same configuration asPipeline Script from SCM. When using a build definition, stream or workspace configuration, this will cause an accept to happen in each of those cases, leading to different content being loaded in each agent. In the case of build definition, an additional build result will be created if the build is triggered from Jenkins. This may or may not be what you want. If you want to prevent the extra checkout for every agent directive, add the options directive below the agent directive with skipDefaultCheckout(true)

```
options {skipDefaultCheckout(true)}
```

Generating Pipeline Snippet for Team Concert Plugin from Snippet Generator

For pipeline jobs, you can generate the snippet for Team Concert Plugin's RTCScm using the Pipeline snippet generator.

> *Note : Even if you do not want to override global configuration for Team Concert Plugin, the snippet generator will create a RTCScm snippet with values for **serverUri**, **credentialsId**. If you copy this snippet into your pipeline script, it could create maintenance issues when you intend to change the global server URI, credentials and build tool kit. If you intend to use the global settings for RTCScm configuration, then remove the following attributes in the snippet and then copy it into your pipeline script.*

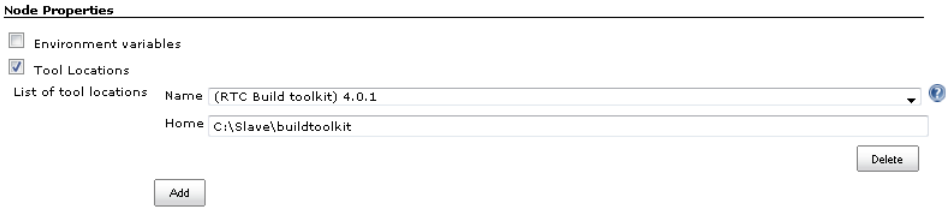
- `serverURI`
- `credentialsId`
- `timeout`
- `buildTool`
- `overrideGlobal`

Master/Slave Configuration

Master and slave configurations are supported by this plugin. See the Jenkins documentation on distributed builds for more information. The RTC build toolkit home path is required for the master to be able to test connections and build artifacts.

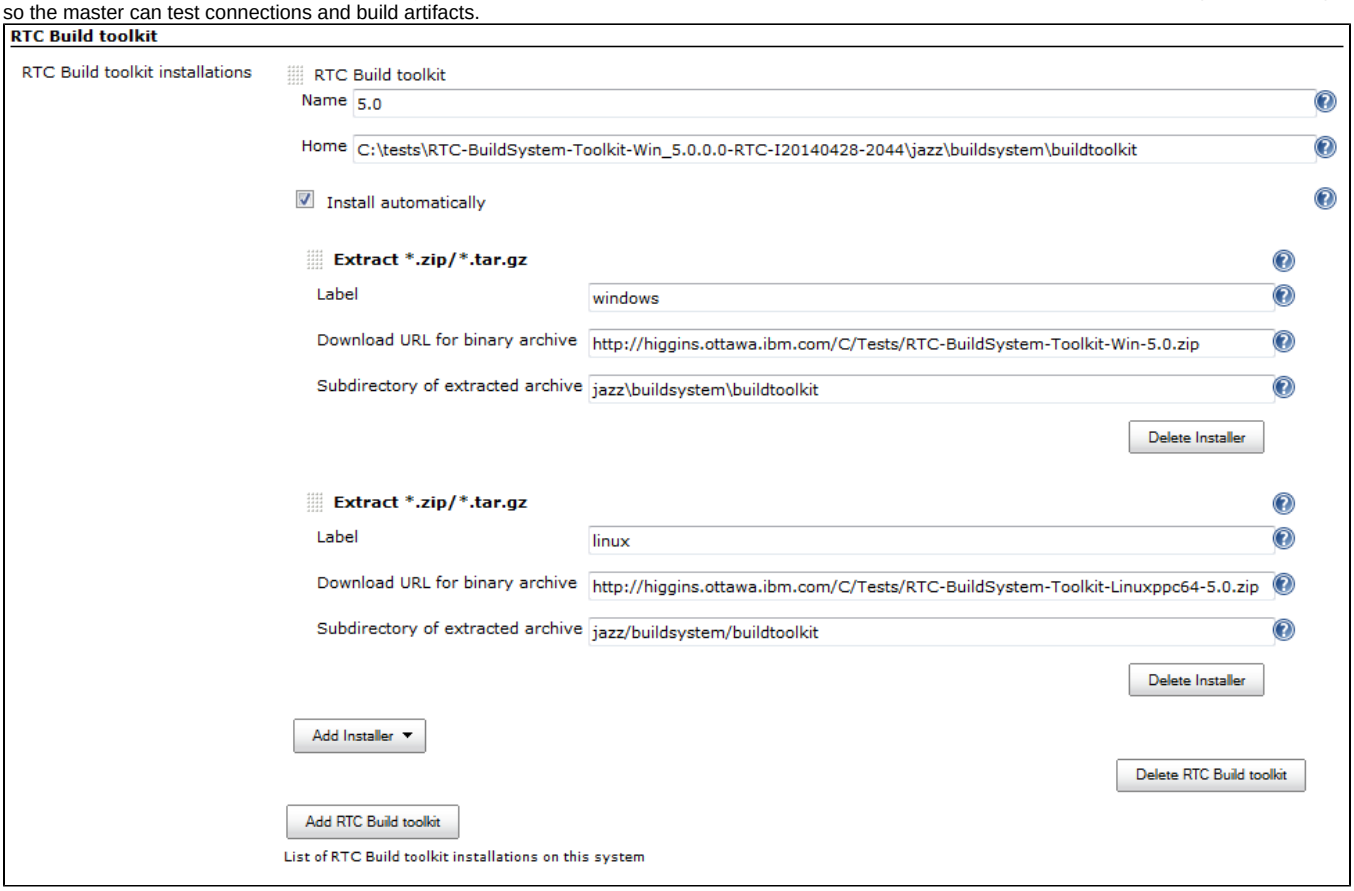
Note: If a password file is being used to authenticate with the RTC server for a particular job, it is unnecessary to copy that file to each of the slaves. The master extracts the password from the password file and passes it to each slave required.

1. Navigate to the Jenkins /computer/? page (Jenkins > Manage Jenkins > Manage Nodes) and click the "New Node" link.
2. Enter a name and create a "Node name", select the "Dumb Slave" radio button and click the "OK" button.
3. In the node configuration page, find the "Node Properties" section and check the "Tool Locations" check box.
4. From the list of tool locations, select the build toolkit you want to define for the node, and set the value in the "Home" field.



a.

Build toolkits can also be installed automatically on slaves. And labels can be used to match build toolkits to slaves. However, a home path is still required so the master can test connections and build artifacts.

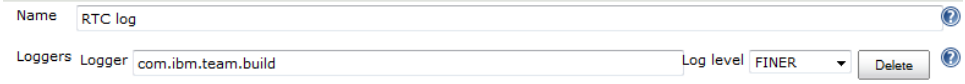


RTC Log

You can capture logs from the Team Concert plugin to debug any problems that you may encounter.

Configuring Java Logging

1. Navigate to the Jenkins /log page (Jenkins > Manage Jenkins > System Log) and click the "Add new log recorder" button.

2. Name it something like "RTC Log" and click the "Add" button to add a logger.
3. Enter a logger of "com.ibm.team.build" and set the log level to "FINER".
4. 

5. Click the "Save" button.
6. Return to this log if a problem is ever experienced using this plugin. The log will help to identify the problem.
7. Logging on Slaves
    a. On the Slave while messages are logged at level FINER, the logs never come back.

Logging in the build console log

1. There is support for a debug flag which will result in the debug output going into a build's console log
2. The environment variable "com.ibm.team.build.debug" with the value "true" will activate the debug logging on a slave.
   To configure on a single Slave
    a. Jenkins > Manage Jenkins > Manage nodes
    b. Hover over the link of the node to configure. Choose Configure from the popup context menu
    c. In the Node properties section, select and check the Environment variables checkbox
    d. Click the Add button beside the List of key value pairs.
    e. Supply "com.ibm.team.build.debug" as the name and "true" as the value
    f. Click the Save button.

   Alternately to configure on the Master and all Slaves
    a. Jenkins > Manage Jenkins > Configure System
    b. In the Global Properties section, select and check the Environment variables checkbox
    c. Click the Add button beside the List of key value pairs.
    d. Supply "com.ibm.team.build.debug" as the name and "true" as the value
    e. Click the Save button.
5. Alternatively, you can add com.ibm.team.build.debug as a parameter to the Job and set its value to true.

The debug flag currently only logs information relating to the class loader setup. The rest of the logic should not be affected by running on a Master or a Slave so if you need those logs, consider running on the Master to get the detailed logs.

Logging the version of build toolkit

If you have turned on the variable "com.ibm.team.build.debug", either through the environment variables or as a Job parameter, then the version of build toolkit used in the master and slave for that build will appear in the build log.

You should see messages such as the following in the build log.

```
Version of build toolkit "<buildtoolkit-name>" on master is "6.0.4".
Version of build toolkit "<buildtoolkit-name>" on "<slave-name>" is "6.0.4".
```

Collecting Metronome Logs for different build configurations

Build Definition

1. Add the following build property to the build definition.
    a. Name - team.build.reportStatistics
    b. Value - true
2. Open the build definition editor in RTC Eclipse or RTC Web UI, click Properties tab and add the property.
3. From the Jenkins console, run a build.
4. Open the build result associated with the Jenkins build.
5. Click the Logs tab.
6. You should see two files statistics-<timestamp>.log and statisticsData-<timestamp>.log

Repository Wokspace, Stream and snapshot

1. Add the following String Job property to the Jenkins job.
    a. Name - team.build.reportStatistics
    b. Value - true
2. From the Jenkins console, run a build.
3. In the machine that hosts the Jenkins master, go to <jenkins config dir>/jobs/<jobname>/builds/<build number>/teamconcert/diagnostics
4. You should see two files statistics-<timestamp>.log and statisticsData-<timestamp>.log

RTC related Environment Variables available to the Build

The following environment variables are available to the build after Rational Team Concert source control step is completed.

property	description
team_scm_changesAccepted	The number of changes accepted or discared during the build.
	UUID of the snapshot created after accepting changes. Not set if no snapshot was created.

team_scm_snapshotUUID	
team_scm_workspaceUUID	The UUID of the Repository workspace used in the build. Only set if the build is using a build definition.
buildResultUUID	UUID of the build result. Only set if the build is using a build definition
RTCBuildResultUUID	UUID of the build result. Only set if the build is using a build definition
requestUUID	UUID of the build request. Only set if the build is using a build definition.
buildDefinitionId	UUID of the build definition being used by the build. Only set if the build is using a build definition.
repositoryAddress	Address of the RTC repository.
buildEngineId	Name of the build engine associated with the build request/result (if there is a build result). An RTC build engine is not actually running, but some ant tasks need the engine id.
buildEngineHostName	Host name of the Jenkins master or slave that the build is running on.
buildRequesterUserId	User id of the RTC user that requested the build be started. Only set if the build is using a build definition
personalBuild	True if the build is a personal build (requested from RTC), otherwise, not set
rtcTempRepoWorkspaceName	The name of the temporary Repository Workspace created during a build using Stream configuration
rtcTempRepoWorkspaceUUID	The UUID of the temporary Repository Workspace created during a build using Stream configuration

Apart from these built-in properties, when using Build definition configuration, all the build properties set in the build definition and potentially modified when requesting the build will be available as environment variables in the Jenkins build after the Team Concert plugin runs.

Accessing RTC Build properties in a Freestyle job

In a freestyle job, after Team Concert Plugin completes, you can access any built-in property (for all configurations) or user defined build property (only for Build definition configuration) using the following syntax:
Windows

%<propertyname>%

Unix

$propertyname

Accessing RTC Build properties in a Pipeline job before checkout step runs (only for Build definition configuration)

When using build definition configuration, you can access build properties set in the RTC build result in the pipeline build even before the checkout step runs. These could be built-in properties set in the build result or user defined RTC build properties. In both cases, you have to create a String parameter in the Jenkins job with the same name as the RTC build property (built-in or user defined). The actual value will be set by the RTC build result that starts the Jenkins build. You can supply different values to the user defined RTC build properties when requesting the RTC build.

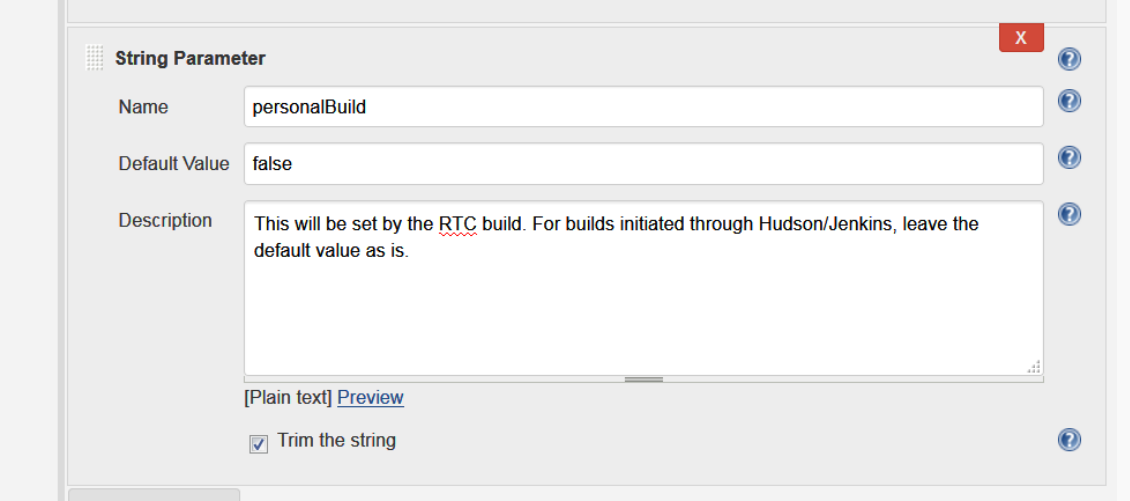
The following built-in properties are available to the Jenkins build even before the checkout step runs.

property	description
buildResultUUID	UUID of the build result. Only set if the build is using a build definition
requestUUID	UUID of the build request. Only set if the build is using a build definition.
buildDefinitionId	UUID of the build definition being used by the build. Only set if the build is using a build definition.
repositoryAddress	Address of the RTC repository.
buildEngineId	Name of the build engine associated with the build request/result (if there is a build result). An RTC build engine is not actually running, but some ant tasks need the engine id.
buildEngineHostName	Host name of the Jenkins master or slave that the build is running on.
	User id of the RTC user that requested the build be started. Only set if the build is using a build definition

buildRequesterUserId	
personalBuild	True if the build is a personal build (requested from RTC), otherwise, not set

For instance, consider the scenario where you want to know if the RTC build result that started this pipeline build is a personal build or not.

1. First create a Job parameter "personalBuild" type is String in the Jenkins pipeline job and set the default value to false.



2. Request a personal build in the RTC build definition associated with the Jenkins job.

3. In your pipeline script, you can check whether the RTC build is a personal build or not as follows

```
if ("${env.personalBuild}" == "true") {
   // Do something } else {   // Do something else}
}
// or

if ("${personalBuild}" == "true") {
   // Do something } else {   // Do something else}
}
```

4. To access the buildRequesterUserId property in your script, define a new String parameter called "buildRequesterUserId" to the Jenkins job and set the default value to an empty string.

5. Back in your pipeline script, you can access the property as

```
"${env.buildRequesterUserId}"
// or
"${buildRequesterUserId}"
```

> *Note : This is different from accessing personalBuild property of env object after the checkout step runs. In that case, the personalBuild property was repopulated by the checkout step and can be accessed only through the ${env} variable. Here, the property is set by the RTC when starting the Jenkins build.*

Accessing RTC Environment Variables in a Pipeline Job after checkout step runs

checkout step now returns a map that is populated by Team Concert plugin. For instance, you can do something like the following to get the required values into scmvars variable and access them using the syntax "${scmvars.<rtc environment variable>}". For a list of built-in properties exported to the environment, see this section

checkoutstep
`def scmvars = checkout([$class: 'RTCScm'...])`

Available in plugin version 1.2.0.5 and above and Jenkins 2.60 and above with workflow-cps 2.40 and above. With the fix from Issue 26100,

Note 1:

Even if you are not using the latest version of Team Concert Plugin, with workflow-cps 2.40, the env object is repopulated every time the checkout step runs. See JENKINS-42499 and this Jenkins developers forum post.

Therefore, the issue reported in Defect 370979 - Environment variables for snapshot, build result UUID are null if env object is accessed before running teamconcert checkout step, in a pipeline script and the issue reported in this jazz.net forum post are not seen anymore. As an example, after every checkout, you can save the snapshot UUID value into a separate variable as follows

```
echo "${env.BUILD_NUMBER}"

 node {
   checkout([$class: 'RTCScm'...])
   // At this point, env contains RTC related environment variables from the first checkout
   def snapshotUUID1 = "${env.team_scm_snapshotUUID}"
   echo "${snapshotUUID1}"

   checkout([$class: 'RTCScm' ....])
   // At this point, env contains RTC related environment variables from the second checkout. The environment
variables contributed by the first checkout are overwritten.
   def snapshotUUID2 = "${env.team_scm_snapshotUUID}"
   echo "${snapshotUUID2}"
 }
```

Note 2:

If you are using workflow-cps < 2.40, follow the workaround mentioned below.

In a pipeline job the environment variables published by the Team Concert Jenkins plugin is null if the env object is accessed once before the RTC SCM checkout step. For instance, the following script would return the UUID of the snapshot published by the Team Concert plugin.

```
node('master') {
    // run teamconcert scm step
    echo "${env.team_scm_snapshotUUID}"
 }
```

But in the script given below the env object is accessed once before running the checkout step and hence accessing the snapshot UUID from the env object returns null

```
echo "${env.BUILD_NUMBER}"
node('master') {
    // run teamconcert scm step
    echo "${env.team_scm_snapshotUUID}"
 }
```

Though the Team Concert plugin publishes the environment variables when checkout is invoked, in pipeline scripts the env object once constructed is not refreshed with any of the environment variables, published later.

If you run into issues accessing the environment variables published by the Team Concert plugin, the suggested work around is to access the RTCBuildResultAction object that is added to the build by the Team Concert plugin. The following code returns the build properties stored in RTCBuildResultAction object. This can be used in a pipeline script to obtain snapshot UUID.

```
def action = currentBuild.build().getAction(com.ibm.team.build.internal.hjplugin.RTCBuildResultAction.class)
def buildProps = action.getBuildProperties()
println(buildProps['team_scm_snapshotUUID'])
```

Please note that if you invoke RTC SCM multiple times, then there will be that many RTCBuildResultActions in the build. Therefore, currentBuild.build(). getActions(com.ibm.team.build.internal.hjplugin.RTCBuildResultAction.class) should be used. The action added by the last invocation of RTC SCM should be available at the end of the list. For instance, if there are two RTCScm checkouts, the second RTCBuildResultAction can be accessed as follows.

```
def actions = currentBuild.build().getActions(com.ibm.team.build.internal.hjplugin.RTCBuildResultAction.class)
def buildProps = actions.get(1).getBuildProperties()
println(buildProps['team_scm_snapshotUUID'])
```

Wrapping the code in a Global Shared Library

The above code cannot be directly used in a pipeline script. You can wrap this code inside a method and add it to a Global Shared Library. You can then call the method from your pipeline script.

If you are already using a Global Shared Library in your environment, add the following code in a file called rtcutils.groovy and place the file under the **vars** directory,

```
 def getSnapshotUUID(actionNum) { // The n'th RTCBuildResultAction.
    def actions = currentBuild.build().getActions(com.ibm.team.build.internal.hjplugin.RTCBuildResultAction.
class)
    if (actions != null && actions.size() > 0 && actionNum > 0 && actionNum <= actions.size()) {
        def buildProps = actions.get(actionNum-1).getBuildProperties()
        return (buildProps['team_scm_snapshotUUID'])
    }
    return null
}
```

Then, in your pipeline script, you can write the following to get the snapshotUUID of the checkout step.

```
@Library('your-shared-library')_

node {
   checkout([$class: 'RTCScm'...])

   def snapshotUUID = rtcutils.getSnapshotUUID(2) // pass 2 if the shared library is fetched from RTC,
otherwise pass 1
   echo "${snapshotUUID}"
}
```

If you don't have Global Shared Library in your environment, consult https://jenkins.io/doc/book/pipeline/shared-libraries on how to create and access a shared library in your pipeline script. Note that if you use RTC for hosting the Global Shared Library, then there will be a RTCBuildResultAction added to the build at the point where the library is brought into the pipeline script.

Symbolic Link support

RTC support for symbolic links requires one or two additional libraries (.dll/.so files).

1. RTC file system natives
2. Eclipse file system natives

The reason is Java 6 and earlier doesn't have support for creating/looking at properties of symbolic links. Java 7 has symbolic link support that works on linux, but on Windows there are some limitations when creating links (if the target has not yet been created the type is defaulted to file which is not good if its a directory). If you are running Linux and can use Java 7 you only need the Eclipse natives. Otherwise, you will need both the RTC and Eclipse natives.

In the Build engine directory (<your RTC build install directory>\buildengine\eclipse\plugins), look for (or equivalent jars for your platform/release).

1. `com.ibm.team.filesystem.client_3.1.600.v20130415_0257.jar` (RTC file system natives)
2. `org.eclipse.core.filesystem.win32.x86_1.1.201.R36x_v20100727-0745.jar` (Eclipse file system natives)

From the com.ibm.team.filesystem.client jar you want to extract `winfsnatives.dll` (`libfsnatives.so` on linux). Take all the .dll/.so files from the org.eclipse.core.filesystem jar. Place them directly in a directory (eg. c:\natives\winfsnatives.dll).

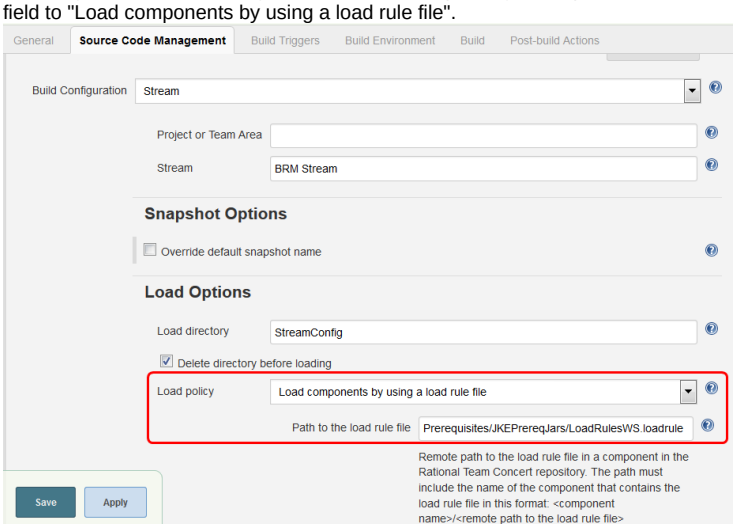
When you start Jenkins, we need to tell java about the directory so that it can load the libraries from it. To this, you can add the directory to the search path. Change the `PATH` variable on Windows or the `LD_LIBRARY_PATH` variable on linux prior to starting Jenkins. Alternatively, you can also specify it when starting Java through the `-Djava.library.path` setting.
eg. `java -Djava.library.path="c:\natives;%Path%" -jar jenkins-1.509.1.war`

If you are running on Windows, you need to be sure that you have permission to create symbolic links. The Symbolic links article in the jazz.net library describes how.

Note: If you are running your jenkins builds on slaves and the symbolic links fail to load, then the native libraries should be included in the JVM library path of slaves too.

Load Rules Support

1. When a jenkins build is configured with an RTC build definition, the component load rules specified in the RTC build definition, if any, will be applied when loading the jenkins build workspace. Component load rules in builds describes how to specify load rules in a build definition.

2. When a jenkins build is configured with an RTC repository workspace, stream, or snapshot load rules can be specified by setting the load policy field to "Load components by using a load rule file".
3. 

4. To configure load policy in a pipeline build, set the "loadPolicy" field to one of - "useComponentLoadConfig", "useLoadRules", or "useDynamicLoadRules".
    a. When loadPolicy is set to useComponentLoadConfig, you can either choose to load all components or exclude some components by setting the value for "componentLoadConfig" to either "loadAllComponents" or "excludeSomeComponents".
5. The load policy field for RTC build definition can be set only using the 6.0.5 RTC client.
6. Component load rules can also be specified through dynamic load rules extension. For more details refer DynamicLoadRulesJenkinsPlugin. Dynamic load rules feature is supported across all build configurations - build definition, repository workspace, stream, and snapshot.
7. In build definition configuration, when load rules are configured in the build definition and dynamic load rules are also provided, dynamic load rules take precedence over the component load rules.
8. Note that the till 1.2.0.4 the behavior of load rules in Jenkins builds, when using the component load rules specified in RTC build definition or the load rules generated by the dynamic load rules extension, is different from how eclipse client enforces the load rules. Say, you have a load rules file that loads some but not all of the components in a workspace. This load rules file when used to load a workspace in the eclipse client, will result in loading of only those components specified in the load rules file. When the same load rules file is configured in an RTC build definition, all components from the workspace, including those not specified in the load rules file, are loaded; those components for which load rules are specified are loaded according to the specified load rules, all the other components are loaded as is. The Components to exclude option, in the RTC build definition can be used to restrict which components are loaded during the build - for more details refer Creating RTC build definitions.
9. From 1.2.0.4 the behavior of load rules in Jenkins builds is at par with RTC SCM. So, only those components for which load rules are specified will be loaded, according to those rules; all the other components for which load rules are not specified will not be loaded. To maintain backward compatibility in Jenkins builds configured with an RTC build definition, old load rules behavior will be enforced unless the load policy field in the build definition is set to use load rules.

Known limitations:

1. In the version 1.2.0.0, polling is not supported for stream and snapshot build configurations, when "avoid using toolkit on master (experimental)" is checked.
2. In the version 1.2.0.0 temporary workspaces are created to support loading from a stream and snapshot. Teamconcert plugin deletes the temporary workspaces when the completes. These temporary workspaces could be left behind in case of network issue during the build. The temporary workspaces can be located by searching for workspaces that starts with the prefix "HJP_".
3. In the version 1.1.9.5, validating the connections when "avoid using toolkit on master (experimental)" is checked is broken. This issue seems to be do with maven dependencies. The issue is tracked in the work item Error shown when validating a connection with avoid using toolkit on master option checked
4. You may need to recycle Jenkins and slaves when updating the Team Concert plugin to a new version, or when automatically installing a new build toolkit.
5. Following are knows issues with Workflow support
    a. Deleting a workflow build does not delete the corresponding RTC build result
    b. 365198: [Workflow plugin] Using the groovy script generated by snippet generator for TeamConcert step in a workflow job throws NPE in RTCScm. For a workaround change the generated script from teamconcert([value:"buildDefinition", buildDefinition:"<>")] to teamconcert buildType: [value:"buildDefinition", buildDefinition:"<>"]. For more information on this issue refer to JENKINS-29711
6. Using com.ibm.team.build.debug to know the RTC build toolkit version in a slave for a particular job doesn't work in the first build processed by the slave. Subsequent build of the job on the same slave will output the build tooolkit version in use. See 461155: Logging version of build toolkit on the slave doesn't work in the first build processed by the slave after a slave restart.

Known Limitations (with fixes in newer releases of RTC) :

1. Issue with RTC 6.0 build tool kit and load rules. Due to a breaking change in the RTC 6.0, load rules will not work when using RTC 6.0 build tool kit. ***Fix is available in 6.0 Ifix07 build toolkit (***work item 362564)***.*** Refer to the work item Load rules is broken with Jenkins plugin and RTC 6.0 build tool kit (361926) for more details. If you are using load rules then its recommended to use the RTC 5.0.2 build tool kit and not RTC 6.0 build tool kit. Note that this recommendation if only or the version of the RTC build tool kit and and not for the RTC server. The RTC server can either be 5.0.2 or 6.0, since RTC supports n-1 compatibility (i.e an older client can connect to a later server) a 5.0.2 version of the build tool kit will work with RTC 6.0 server.
2. Each build request initiated from RTC creates a buildResultUUID parameter in the Jenkins workflow job.
    a. **This issue is fixed in RTC v6.0.1 or higher and in 6.0 ifix04, 5.0.2 ifix12.**

b. For a workaround follow the steps listed below
    i. In the workflow job configuration page, delete all but one buildResultUUID parameters.
    ii. Add the following under the <flow-definition> tag in the workflow job's config.xml

```
<actions>
  <hudson.model.ParametersDefinitionProperty>
    <parameterDefinitions>
      <hudson.model.StringParameterDefinition>
        <name>buildResultUUID</name>
        <description>The UUID of the build result in RTC. It is supplied by builds initiated through RTC. For builds initiated through Hudson/Jenkins, no value should be supplied.</description>
        <defaultValue></defaultValue>
      </hudson.model.StringParameterDefinition>
    </parameterDefinitions>
  </hudson.model.ParametersDefinitionProperty>
</actions>
```

    iii. Click Manage Jenkins-> Reload Configuration from Disk.

Tutorial

1. jazz.net wiki topic: Integrating with Jazz SCM and Builds from Hudson and Jenkins using the Team Concert Plugin
2. YouTube video: Team Concert Plugin for Hudson/Jenkins

Best Practices

Refer to the best practices document here.

References

1. Using the Team Concert plugin in Pipeline jobs - https://jazz.net/wiki/bin/view/Main/JenkinsWorkflowPluginSupport
2. Using dynamic load rules in Team Concert plugin - https://jazz.net/wiki/bin/view/Main/DynamicLoadRulesJenkinsPlugin

Releases

1.3.0 May 1, 2019

> *NOTE: The default behavior of creating "Related artifact" link to a Jenkins build in all the accepteed work items when using Repository Workspace or Stream job configuration (introduced by work item 388795) has changed.*
> *In 1.2.0.5, links will be created in all the accepted work items. In 1.3.0, links will NOT be created in all the accepted work items.*
> *There is a new option "Add Jenkins build link to accepted work items" in the Job configuration to create these links and is unchecked by default. You must select the option in the Job configuration to create related artifact links to a Jenkins build in all the accepted work items. See work item 461859 for more details.*

- You can collect metronome information for all build configurations. See Collecting Metronome Logs section for more details.
    - See Work Item 438208: Enhance Team Concert Plugin to collect metronome log like JBE
- In this release, we have changed the behavior of creating "Related artifact" links to Jenkins builds in all the accepted work items originally introduced by work item 388795. You must choose the option "Add Jenkins build link to accepted work items" in the Jenkins job configuration to create "Related artifact" links to Jenkins builds in all the accepted work items.
    - See Work Item 461859: Make the "creation of Jenkins build links to work items in accepted change sets" an opt - in for the users in Repository Workspace and Stream configuration
- We have fixed an incompatibility with Pipeline jobs wherein messages from Team Concert Plugin were not printed in the build log.
    - See Work Item 478877: Pipeline builds do not output messages from RTCScm

GitHub commit - 166456d2a65

1.2.0.5 June 15, 2018

> *Important information : The minimum required version of Jenkins is now 1.625.1. After upgrade, it is recommended to check that the Team Concert plugin (RTCScm) configuration is intact in a few jobs.*

- In repository workspace and stream build configuration, plugin now creates links to the Jenkins build in the work items attached to the change sets
    - See WorkItem 388795: In Team Concert Jenkins Plugin, when using build workspace/stream configuration, create backlinks in included work item (s) to the Jenkins build
- You can view the version of build toolkit used in master and slave in the build log by adding com.ibm.team.build.debug = true to the environment or as a job parameter.
    - See WorkItem 449539: [Jenkins] Log the version of build toolkit in the build log

- You can access the environment variables exported by RTCScm in a checkout step by assigning it to a groovy variable.
    - WorkItem 446242: Adopt changes to SCM from ~~JENKINS-26100~~ - SCM steps should return revision state RESOLVED
- Other fixes
    - WorkItem 398804: Upgrade parent pom version to 2.x
    - WorkItem 448725: Jenkins Build Error: An invalid XML character (Unicode: 0x10) was found
    - WorkItem 458158: Move to Java 7 - upgrade minimum required Jenkins version to 1.625.1

1.2.0.4 December 04, 2017

1. Support for load rules in Jenkins jobs configured with an RTC repository workspace, stream, or, snapshot.
    a. 402834: [CCM] Support for load rules in the Jenkins Integration Plugin
2. Per checkout dynamic load rules configuration.
    a. 403461: Provide an interface in the Jenkins job configuration to check for dynamic load rules during a run
3. Fix for 403254: Dynamic load rules should have precedence over load rules from Build Definition
4. getComponentLoadRules method in dynamic load rules extension is deprecated. Instead dynamic load rules have to be returned by the newly added getPathToLoadRuleFile method. For more information, see DynamicLoadRulesJenkinsPlugin.
5. 367019: [Jenkins-Plugin] Export Build parameter via API
6. 410454: team_scm_workspaceUUID should be available as an environment variable for Repository workspace based builds.

1.2.0.3 Jun 16, 2017

1. In Build Definition configuration, Post Build Deliver is supported when using Rational Team Concert server 6.0.4 or higher. You can edit the Build Definition in RTC to include Post Build Deliver configuration. The configuration information will be used by the plugin to perform post build deliver.
    a. Improve the Team Concert Plugin for Jenkins to support post-build deliver for build definition configuration

1.2.0.2 Dec 6, 2016

1. Support for customising the name of the snapshot created during the build. You can use Jenkins job parameters and/or environment variables in the snapshot name. During the build, the parameters will be resolved to their values to construct the snapshot name.
    a. 368222: Support customization of the name of the generated snapshot
2. In Stream configuration, allow check-in and deliver changes using SCM CLI during the build. The temporary Repository Workspace created for loading content is now deleted at the end of the build, thus permitting check-in and deliver operations. The name and UUID of the temporary Repository Workspace created during the build is available as 'rtcTempRepoWorkspaceName' and 'rtcTempRepoWorkspaceUUID'
    a. 397202: Ability to check-in and deliver changes in Stream configuration based Jenkins build
3. Fixes for the following issues
    a. 398434: RepositoryConnection.accept() is taking unusually long time for workspace and build definition configuration
    b. 401392: Environment variables are missing when loading from Snapshot
    c. 405661: Include workaround for "SQL Duplicate Value exception" when loading from a snapshot into Team Concert Plugin

1.2.0.1 Aug 16, 2016

1. A String parameter can be provided in the text field for Build Definition, Repository Workspace or Stream as '${paramater_name}'. Enhancement 324449- Jenkins Team concert plugin can support parameters for stream,workspace and build definition fields
2. A Snapshot can be scoped to a Repository Workspace or Stream. Task 392790 -For build snapshot configuration, provide options to specify the project area/team area and the owner workspace/stream
3. A Stream can be scoped to a Project Area/Team Area. Task 391633- In the build stream configuration, use the project area/team area value, if configured, to resolve the stream specified by name
4. Support for configuration level validation instead of validating individual fields in the Rational Team Concert section.
5. Temporary Repository Workspace created for Snapshot and Stream configuration have a comment of the form "Created by Team Concert Plugin for job ### in Jenkins server ####". Task 388924 - Add a comment to the temporary workspace so that it becomes easier to identify it as a build workspace
6. Link to the Build Definition, Repository Workspace, Stream used in the build now appears in the build page. Task 396340 - Add links to the current configuration used in a build of a Jenkins job

1.2.0.0 April 22, 2016

1. Enhancement 376827: Support Load Directory and Delete before loading in Jenkins Job
2. Enhancement 382347: Support RTC BuildDefinition's Accept Options in Jenkins job
3. Enhancement 366909: Support for loading from a snapshot
4. Enhancement 375548: Support for loading from stream
5. Enhancement 376098: Provide dropdown combo box support for various build configurations
6. Fixes for the following issues
    a. 346653: Jenkins plugin repeatedly resets the "Quiet period"
    b. 380220: Rework the Jenkins Plugin messages to display the error trace
    c. 388284: Loading a jenkins build workspace with a RTC build definition configuration fails in Jenkins 1.655
    d. 383194: Insufficient error handling or error logging for dynamic load rule generation - with this fix, implementations of dynamic load rules can propagate any exceptions to the teamconcert jenkins plugin.
    e. 387320: Validating workspace/connection during job configuration fails if the job is created under a folder with global credentials scoped to the folder

1.1.9.9 January 25, 2016

1. Enhancement 338976 Provide a mechanism to generate and input the Load Rules file in the Jenkins Team Concert plugin. Dynamic Load Rules feature allows users to provide load rules for components during the build. For more information, see DynamicLoadRulesJenkinsPlugin
2. Fixes for the following issues
    a. 377090: Team Concert plugin for Jenkins triggers builds even there are no real changes
    b. 379521: RTC Jenkins plugin leaving .jazzlock file in the workspace
    c. 380589: [Jenkins Integration] Build Toolkit on Slave not found (1.1.9.8)
    d. 380708: During delta computation for determining if a build has to be fired, ignore outgoing changes in the build workspace
    e. 381693: When starting a Jenkins job from RTC and if the Jazz source control Load directory is specified as . and delete before loading is checked , build fails
    f. 381794: Fix for .jazzlock on abandoning the build (from work item 379521) doesn't work as expected

1.1.9.8 December 21, 2015

Fix for work item 379521- RTC Jenkins plugin leaving .jazzlock file in the workspace, is not available in 1.1.9.8. The issue has been fixed 1.1.9.9

1. Fixes for the following issues
    a. Jenkins Jobs config.xml file broken when upgrading the plugin from 1.1.9.4 to 1.1.9.7
    b. Jenkins Plugin v1.1.9.7 doesn't save the credentials of the global RTC configuration (Manage Jenkins)

1.1.9.6 and 1.1.9.7 October 26, 2015

1. Fixes for the following issues
    a. RTCScmStep uses serverUri but RTCScm uses serverURI
    b. RTC Jenkins integration for "Recent Changes" does not work properly when a changeset is related to two workitems
2. Note that 1.1.9.6 and 1.1.9.7 releases are the same, 1.1.9.7 is a respin of the 1.1.9.6 release to fix the error in the release 1.1.9.6.1

1.1.9.6.1 October 26, 2015

1. Invalid plugin release, do not use

1.1.9.5 September 21, 2015

1. Fixed multiple issues with supporting WorkFlow projects
    a. Add visual support for the snippet generator when using TeamConcert step
    b. Add setters for optional parameters in the teamconcert step
    c. Personal build for a build definition connected to a workflow job shows up in the Changes section of the job
    d. In the changes section of a workflow job and build, work item numbers, change sets are not displayed as links
    e. [Expose RTC build information to the environment so that it can be used in the workflow script | https://jazz.net/jazz/resource/itemName/com.ibm.team.workitem.WorkItem/363665
    f. Using snippet generator in the workflow definiton section for Rational Team Concert plugin generates incorrect groovy script
    g. Improve logging in RTC Jenkins plugin - additional logging statements

1.1.9.4 August 04, 2015

1. Fixed issue with load rules and RTC 5.x build tool kit Remote load rules not working using Jenkins Team Concert Plugin 1.1.9.3 (364161)

1.1.9.3 July 26, 2015

1. Implement Quite period support for FreeStyle project types. work item 362725
2. Initial implementation for Workflow jobs. 362121: RTC Jenkins plugin - workflow support. Refer to Usage guide and documentation for more details.
3. Note that this is a initial implementation with some limitations and issues, refer to the the limitation section for know issues and workarounds.

1.1.9.2 June 11, 2015

1. Translation update and release for RTC 6.0work item 360197

1.1.9.1 March 26, 2015

1. Provide a Group ID in the Team Concert plugin work item 336266
2. RTC build plugin for Jenkins repeatedly resets the "Quiet period" work item 350379

1.1.9 October 9, 2014

There is a migration impact for this release. See the Migrations section below.

1. When a Jenkins build is deleted, the corresponding RTC build result(s) (if there are any) are deleted from RTC. The RTC build result will not be deleted it it is flagged as deletion is not allowed. work item 330249
2. Improve support for "Multiple SCMs" plugin. You can now specify multiple RTC SCM configurations referencing different servers (when builds are started from Jenkins). work item 300164
3. Support so RTC's Build Definition editor can warn the user if the Jenkins job doesn't point to that definition work item 276139
4. Thread's context class loader not reset properly + work around for unexpected failure to load LogFactory class work item 322272

1.1.8 July 10, 2014

1. Main Jenkins configuration page was not showing the chosen Build toolkit work item 320832
2. Add warning to console log when build workspace has components not visible to the build user work item 203294
3. When the SCM provider is the "Multiple SCMs" plugin, the detailed changes for a build does not list the change details work item 323307

1.1.2 June 11, 2014

1. Support for Jenkins credentials has been added which introduced a dependency on the Credentials plugin. If a job is already configured to use user ID and password (or password file) , it will continue to run but these fields are read only. Any changes will require credentials going forward. Using the Credentials plugin offers more flexibility and solves some issues.
    a. Multiple credentials can be defined and used in multiple jobs
    b. Can use a global user ID and password (or password file) when the RTC URL is overridden issue 21537
    c. Improves Security issue 21038work item 295009
2. Work related to starting a build with an RTC build result has been moved from the Master to the Slave (assuming the job is running on the Slave). Work item 306172
3. When builds are started within RTC, the server will manage the lifecycle of the build result by periodically polling Jenkins to determine if the build is completed. With RTC 5.0, build definitions support the boolean property `com.ibm.rational.connector.hudson.queueOnly`. When used in conjunction with this release of the plugin, the plugin will terminate the RTC build result when the build completes (just as it does when the build is started in Jenkins). If a lot of builds are started from within RTC, this will be more efficient. Requires RTC version 5.0 or later. Work item 308749
4. New option to use rest service calls to communicate with the RTC Server when performing configuration, polling and build result management (as opposed to the build toolkit). This means if all the jobs have this configured and run on slaves, the toolkit classes will not be loaded on the master. Requires RTC version 5.0 or later.

1.0.12 (and earlier) October 23, 2013

1. This plugin version does not have any Jenkins specific dependencies.
2. To authenticate against an Team Concert server a user id and password is required. The password can be supplied directly or it can be placed in a password file.
3. The RTC build toolkit is used perform build related tasks within Jenkins Master and Slave processes (as opposed to using a command line client). The RTC related tasks include validating the configuration, polling and working with the RTC build result as well as performing the Accept and Checkout phases of the build.
4. Support for a simple build workspace
    a. Changes are accepted into the build workspace from the stream(s) referenced by the flow target(s)
    b. Snapshot of the workspace is created for a build
    c. Change log is created
    d. Build workspace is loaded
5. Integrated support for build definitions
    a. Traceability links from a Jenkins build to an RTC build result, workspace and snapshot.
    b. Publishes links to work items, change sets and file contents captured in the snapshot.
    c. Build workspace is identified by the Build definition
    d. Changes are accepted into the build workspace from the stream(s) referenced by the flow target(s)
    e. Snapshot of the workspace is created for a build
    f. Additional SCM configuration options available in the build definition
    g. RTC Build result is created for a deeper integration with the work items included in the build
    h. Builds (including personal builds) can be started from RTC
    i. Environment variables defined in the RTC build definition are available in the build environment
6. RTC build environment variables are available in the build environment

property	description
team_scm_changesAccepted	How many changes were accepted. Not set if there were no changes.
team_scm_snapshotUUID	UUID of the snapshot created after accepting changes. Not set if no snapshot was created.
RTCBuildResultUUID	UUID of the build result. Only set if the build is using a build definition
requestUUID	UUID of the build request. Only set if the build is using a build definition.
buildDefinitionId	UUID of the build definition being used by the build. Only set if the build is using a build definition.
repositoryAddress	Address of the RTC repository.
buildEngineId	Name of the build engine associated with the build request/result (if there is a build result). An RTC build engine is not actually running, but some ant tasks need the engine id.

buildEngineHostName	Host name of the Jenkins master or slave that the build is running on.
buildRequesterUserId	User id of the RTC user that requested the build be started. Only set if the build is using a build definition
personalBuild	True if the build is a personal build (requested from RTC), otherwise, not set

Migrations

1.1.9

1. The environment variable buildResultUUID is a parameter that is supplied to the Jenkins job when the build started from RTC. It was sometimes also being updated (contributed by this plugin) even if the build was started in Jenkins. In order to better support building multiple projects with the Multiple SCM plugin, the environment variable will not be updated by this plugin. The build result UUID is still available from the RTCBuildResultUUID regardless of where the build was started from.

1.1.2

1. Jenkins Credentials Plugin is now used for storing the user ID and password. For an existing global configuration and jobs, the user ID and password (or password file) fields will be read-only. If a job is using a password file and needs to change a password, the password file contents can be replaced. Otherwise, to update the password the job will need to start using credentials. If this not acceptable, the plugin can work in the old mode by setting the system/environment property: com.ibm.team.build.credential.edit=true.

1.0.10

1. On Linux, a build definition with a load directory starting with "/" (i.e. "/any/folder") used to be interpreted as a relative path, but is now correctly interpreted as an absolute path. So, any build definition relying on the previous behavior need only prefix the load directory with a "." (i.e. "./any /folder").