 New Item

 People

 Build History

 Project Relationship

 Check File Fingerprint

 Manage Jenkins

 Credentials

Build Queue

No builds in the queue.

Build Executor Status

1 Idle
2 Idle

Click here for IntelliJ GDSL.

Steps

`archive`: Archive artifacts

Archives build output artifacts for later use.

`includes`

Include artifacts matching this Ant style pattern. Use a comma separated list to add more than one expression.

Type: String

`excludes` (optional)

Exclude artifacts matching this Ant-style pattern. Use a comma-separated list to add more than one expression.

Type: String

`bat`: Windows Batch Script

Executes a Batch script. Multiple lines allowed.

`script`

Type: String

`build`: Build a job

Triggers a new build for a given job.

`job`

Name of a downstream job to build. May be another Pipeline job, but more commonly a freestyle or other project. Use a simple name if the job is in the same folder as this upstream Pipeline job; otherwise can use relative paths like `../sister-folder/downstream` or absolute paths like `/top-level-folder/nested-folder/downstream`.

Type: String

`parameters` (optional)

Array/List

Nested choice of objects

- `$class: 'BooleanParameterValue'`

  `name`

  Type: String

  `value`

  Type:boolean

- `$class: 'CredentialsParameterValue'`

  `name`

  Type: String

  `value`

  Type: String

  `description`

  Type: String

- `$class: 'CvsTagsParamValue'`

  name

  Type: String

  tagName

  Type: String

- `$class: 'FileParameterValue'`

  name

  Type: String

  file

  Nested choice of objects

- `$class: 'ListSubversionTagsParameterValue'`

  name

  Type: String

  tagsDir

  Type: String

  tag

  Type: String

- `$class: 'PasswordParameterValue'`

  name

  Type: String

  value

  Type: String

  description

  Type: String

- `$class: 'RunParameterValue'`

  name

  Type: String

  runId

  Type: String

description

Type: String

- `$class: 'StringParameterValue'`

  name

  Type: String

  value

  Type: String

- `$class: 'TextParameterValue'`

  name

  Type: String

  value

  Type: String

`propagate` (optional)

If set, then if the downstream build is anything but successful (blue ball), this step fails. If disabled, then this step succeeds even if the downstream build is unstable, failed, etc.; use the `result` property of the return value as needed.

Type:boolean

`quietPeriod` (optional)

Optional alternate quiet period (in seconds) before building. If unset, defaults to the quiet period defined by the downstream project (or finally to the system-wide default quiet period).

Type:int

`wait` (optional)

You may ask that this Pipeline build wait for completion of the downstream build. In that case the return value of the step is an object on which you can obtain the following read-only properties: so you can inspect its `result` and so on.

- `number`
- `result` (typically `SUCCESS`, `UNSTABLE`, or `FAILED`)
- `displayName`
- `description`
- `id`
- `timeInMillis`
- `startTimeInMillis`

- `duration`
- `building`
- `inProgress`
- `previousBuild` (another similar object)
- `nextBuild`
- `absoluteUrl`

For a non-Pipeline downstream build, the `buildVariables` property offers access to a map of defined build variables. For a Pipeline downstream build, this property gives access to any variables set globally on `env`.

Administrator-approved scripts outside the sandbox can use the `rawBuild` property to get access to a `hudson.model.Run` with further APIs. You will generally need to do so inside a method marked `@NonCPS` to avoid storing intermediate values.

If you do not wait, this step succeeds so long as the downstream build can be added to the queue (it will not even have been started). In that case there is currently no return value.

Type:boolean

`checkout`: General SCM

This is a special step that allows to run checkouts using any configuration options offered by any Workflow-compatible SCM plugin. To use a concrete SCM implementations, just install the corresponding plugin and check if it is shown in the list below. Then select the SCM to use from the dropdown list and configure it as needed.

Any other specific step to run checkouts (like `svn` or `git`) are simplistic options of this step.

`scm`

Nested choice of objects

- `$class: 'CVSSCM'`

  `repositories`

  Array/List

  Nested object

  `cvsRoot`

  The CVS connection string Jenkins uses to connect to the server. The format is the same as `$CVSROOT` environment variable (`:protocol:user@host:path`)

  Type: String

  `passwordRequired`

  Type:boolean

  `password`

Type: String

repositoryItems

Array/List

Nested object

location

Nested choice of objects

- `$class: 'BranchRepositoryLocation'`

  branchName

  Type: String

  useHeadIfNotFound

  Type:boolean

- `$class: 'HeadRepositoryLocation'`
- `$class: 'TagRepositoryLocation'`

  tagName

  Type: String

  useHeadIfNotFound

  Type:boolean

modules

Array/List

Nested object

remoteName

The name of the module in the repository at `CVSROOT`

Type: String

localName

The name to be applied to this module in the local workspace. If this is left blank then the remote module name will be used. This is similar to the 'checkout-as' function available on many CVS clients.

Type: String

projectsetFileName

The name of the file in this module to parse for projectset entries.

Type: String

excludedRegions

If set, and Jenkins is set to poll for changes, Jenkins will ignore any files and/or folders in this list when determining if a build needs to be triggered.

Each exclusion uses regular expression pattern matching, and must be separated by a new line.

```
src/main/web/.*\.html
src/main/web/.*\.jpeg
src/main/web/.*\.gif
```

The example above illustrates that if only html/jpeg/gif files have been committed to the SCM a build will not occur.

More information on regular expressions can be found here.

Array/List

Nested object

pattern

Type: String

compressionLevel

Type:int

repositoryBrowser

Nested choice of objects

- `$class: 'FishEyeCVS'`

  url

  Specify the root URL of FishEye for this repository (such as this.)

  Type: String

- `$class: 'OpenGrok'`

  url

  Specify the root URL of OpenGrok for this repository.

  Type: String

- `$class: 'ViewCVS'`

`url`

> Specify the root URL of ViewCVS for this repository (such as this).

Type: String

`canUseUpdate`

> If checked, Jenkins will use 'cvs update' whenever possible for builds. This makes a build faster. But this also causes the artifacts from the previous build to remain in the file system when a new build starts, making it not a true clean build.

Type:boolean

`legacy`

> Hudson 1.20 and earlier used to create redundant directories inside the workspace. For example, if the CVS module name is "foo/bar", it first created "foo/bar" and then put everything below. With this option checked off, there will be no more such unnecessary intermediate directories.
>
> If you have multiple modules to check out, this option is forced (otherwise they'll overlap.)
>
> This affects other path specifiers, such as artifact archivers --- you now specify "build/foo.jar" instead of "foo/build/foo.jar".

Type:boolean

`skipChangeLog`

> Prevent the changelog being generated after checkout has completed. This will stop any changes being shown on the changes screen but reduces load on your CVS server.

Type:boolean

`pruneEmptyDirectories`

> Remove empty directories after checkout using the CVS '-P' option.

Type:boolean

`disableCvsQuiet`

> Instructs CVS to show all logging output. CVS normally runs in quiet mode but this option disables that.

Type:boolean

`cleanOnFailedUpdate`

If the job is configured to use CVS update and the update step fails for any reason then the workspace will be wiped-out and a clean checkout done instead.

Type:boolean

`forceCleanCopy`

If checked, Jenkins will add the 'C' option to the CVS update command to force it to over-write any files with local modifications, rather than attempt a merge or leave them as they are.

Type:boolean

- `$class: 'CvsProjectset'`

`repositories`

Array/List

Nested object

`cvsRoot`

The CVS connection string Jenkins uses to connect to the server. The format is the same as `$CVSROOT` environment variable (`:protocol:user@host:path`)

Type: String

`passwordRequired`

Type:boolean

`password`

Type: String

`repositoryItems`

Array/List

Nested object

`location`

Nested choice of objects

- `$class: 'BranchRepositoryLocation'`

  `branchName`

  Type: String

  `useHeadIfNotFound`

  Type:boolean

- `$class: 'HeadRepositoryLocation'`
- `$class: 'TagRepositoryLocation'`

`tagName`

Type: String

`useHeadIfNotFound`

Type:boolean

`modules`

Array/List

Nested object

`remoteName`

The name of the module in the repository at `CVSROOT`

Type: String

`localName`

The name to be applied to this module in the local workspace. If this is left blank then the remote module name will be used. This is similar to the 'checkout-as' function available on many CVS clients.

Type: String

`projectsetFileName`

The name of the file in this module to parse for projectset entries.

Type: String

`excludedRegions`

If set, and Jenkins is set to poll for changes, Jenkins will ignore any files and/or folders in this list when determining if a build needs to be triggered.

Each exclusion uses regular expression pattern matching, and must be separated by a new line.

```
src/main/web/.*\.html
src/main/web/.*\.jpeg
src/main/web/.*\.gif
```

The example above illustrates that if only html/jpeg/gif files have been committed to the SCM a build will not occur.

More information on regular expressions can be found here.

Array/List

Nested object

`pattern`

Type: String

`compressionLevel`

Type:int

`repositoryBrowser`

Nested choice of objects

- `$class: 'FishEyeCVS'`

  `url`

  > Specify the root URL of FishEye for this repository (such as this.)

  Type: String

- `$class: 'OpenGrok'`

  `url`

  > Specify the root URL of OpenGrok for this repository.

  Type: String

- `$class: 'ViewCVS'`

  `url`

  > Specify the root URL of ViewCVS for this repository (such as this).

  Type: String

`canUseUpdate`

> If checked, Jenkins will use 'cvs update' whenever possible for builds. This makes a build faster. But this also causes the artifacts from the previous build to remain in the file system when a new build starts, making it not a true clean build.

Type:boolean

`username`

> This username will be used for the checkout of any modules parsed from the projectset file if no match

was found against the parsed CVSROOT using the globally configured authentication.

Type: String

`password`

This password will be used for the checkout of any modules parsed from the projectset file if no match was found against the parsed CVSROOT using the globally configured authentication.

Type: String

`browser`

Nested choice of objects

- `$class: 'FishEyeCVS'`

  `url`

  Specify the root URL of FishEye for this repository (such as this.)

  Type: String

- `$class: 'OpenGrok'`

  `url`

  Specify the root URL of OpenGrok for this repository.

  Type: String

- `$class: 'ViewCVS'`

  `url`

  Specify the root URL of ViewCVS for this repository (such as this).

  Type: String

`skipChangeLog`

Prevent the changelog being generated after checkout has completed. This will stop any changes being shown on the changes screen but reduces load on your CVS server.

Type:boolean

`pruneEmptyDirectories`

Remove empty directories after checkout using the CVS '-P' option.

Type:boolean

`disableCvsQuiet`

Instructs CVS to show all logging output. CVS normally runs in quiet mode but this option disables that.

Type:boolean

`cleanOnFailedUpdate`

If the job is configured to use CVS update and the update step fails for any reason then the workspace will be wiped-out and a clean checkout done instead.

Type:boolean

`forceCleanCopy`

Type:boolean

- `$class: 'SubversionSCM'`

Checks out the source code from Subversion repositories. See post-commit hook set up for improved turn-around time and performance in polling.

`locations`

Array/List

Nested object

`remote`

Type: String

`credentialsId`

Type: String

`local`

Specify a local directory (relative to the workspace root) where this module is checked out. If left empty, the last path component of the URL is used as the default, just like the `svn` CLI. A single period (`.`) may be used to check out the project directly into the workspace rather than into a subdirectory.

Type: String

`depthOption`

`--depth` option for checkout and update commands. Default value is `infinity`.
- `empty` includes only the immediate target of the operation, not any of its file or directory children.

- `files` includes the immediate target of the operation and any of its immediate file children.
- `immediates` includes the immediate target of the operation and any of its immediate file or directory children. The directory children will themselves be empty.
- `infinity` includes the immediate target, its file and directory children, its children's children, and so on to full recursion.
- `as-it-is` takes the working depth from the current working copy, allows for setting update depth manually using `--set-depth` option.

More information can be found here.

Type: String

`ignoreExternalsOption`

"--ignore-externals" option will be used with svn checkout, svn update commands to disable externals definition processing.

More information can be found here.
Note: there is the potential to leverage `svn:externals` to gain read access to the entire Subversion repository. This can happen if you follow the normal practice of giving Jenkins credentials with read access to the entire Subversion repository. You will also need to provide the credentials to use when checking/polling out the svn:externals using the *Additional Credentials* option.

Type:boolean

`workspaceUpdater`

Nested choice of objects

- `$class: 'CheckoutUpdater'`
- `$class: 'UpdateUpdater'`
- `$class: 'UpdateWithCleanUpdater'`
- `$class: 'UpdateWithRevertUpdater'`

`browser`

Nested choice of objects

- `$class: 'Assembla'`

  `spaceName`

  Type: String

- `$class: 'CollabNetSVN'`

  `url`

  The repository browser URL for the root of the project. For example, a Java.net project called `myproject` would use `https://myproject.dev.java.net/source/browse/myproject`.

  Type: String

- `$class: 'FishEyeSVN'`

url

Specify the URL of this module in FishEye. (such as `http://fisheye6.cenqua.com/browse/ant/`)

Type: String

rootModule

Specify the root Subversion module that this FishEye monitors. For example, for `http://fisheye6.cenqua.com/browse/ant/`, this field would be `ant` because it displays the directory "/ant" of the ASF repo. If FishEye is configured to display the whole SVN repository, leave this field empty.

Type: String

- `$class: 'SVNWeb'`

url

Type: String

- `$class: 'Sventon'`

url

Specify the URL of the Sventon repository browser. For example, if you normally browse from `http://somehost.com/svn/repobrowser.svn?name=local`, this field would be `http://somehost.com/svn/`

Type: String

repositoryInstance

Specify the Sventon repository instance name that references this subversion repository. For example, if you normally browse from `http://somehost.com/svn/repobrowser.svn?name=local`, this field would be `local`

Type: String

- `$class: 'Sventon2'`

url

Specify the URL of the Sventon repository browser. For example, if you normally browse from `http://somehost.com/svn/repobrowser.svn?name=local`, this field would be `http://somehost.com/svn/`

Type: String

repositoryInstance

Specify the Sventon repository instance name that references this subversion repository. For example, if you normally browse from `http://somehost.com/svn/repobrowser.svn?name=local`, this field would be `local`

Type: String

- `$class: 'ViewSVN'`

  `url`

  Specify the root URL of ViewSVN for this repository (such as this).

  Type: String

- `$class: 'WebSVN'`

  `url`

  Type: String

`excludedRegions`

If set, and Jenkins is set to poll for changes, Jenkins will ignore any files and/or folders in this list when determining if a build needs to be triggered.

Each exclusion uses regular expression pattern matching, and must be separated by a new line.

```
/trunk/myapp/src/main/web/.*\.html
/trunk/myapp/src/main/web/.*\.jpeg
/trunk/myapp/src/main/web/.*\.gif
```

The example above illustrates that if only html/jpeg/gif files have been committed to the SCM a build will not occur.

More information on regular expressions can be found here.

Type: String

`excludedUsers`

If set, and Jenkins is set to poll for changes, Jenkins will ignore any revisions committed by users in this list when determining if a build needs to be triggered. This can be used to exclude commits done by the build itself from triggering another build, assuming the build server commits the change with a distinct SCM user.

Each exclusion uses literal pattern matching, and must be separated by a new line.

```
auto_build_user
```

The example above illustrates that if only revisions by "auto_build_user" have been committed to the SCM a build will not occur.

Type: String

excludedRevprop

If set, and Jenkins is set to poll for changes, Jenkins will ignore any revisions that are marked with the given revision property (revprop) when determining if a build needs to be triggered. This can be used to exclude commits done by the build itself from triggering another build, assuming the build server commits the change with the correct revprop.

This type of exclusion only works with Subversion 1.5 servers and newer.

More information on revision properties can be found here.

Type: String

excludedCommitMessages

If set, and Jenkins is set to poll for changes, Jenkins will ignore any revisions with commit messages containing any of the given regular expressions when determining if a build needs to be triggered.

Type: String

includedRegions

If set, and Jenkins is set to poll for changes, Jenkins will ignore any files and/or folders that are ***not*** in this list when determining if a build needs to be triggered.

Each inclusion uses regular expression pattern matching, and must be separated by a new line.

This is useful when you need to check out an entire resource for building, but only want to do the build when a subset has changed.

```
/trunk/myapp/c/library1/.*
/trunk/myapp/c/library2/.*
```

If /trunk/myapp is checked out, the build will only occur when there are changes to either the c/library1 and c/library2 subtrees.

If there are also excluded regions specified, then a file is not ignored when it is in the included list and ***not*** in the excluded list.

More information on regular expressions can be found here.

Type: String

ignoreDirPropChanges

If set, Jenkins ignores svn-property only changes of directories. These changes are ignored when

rebuild of a maven project, in spite of incremental build option).

Type:boolean

`filterChangelog`

If set Jenkins will apply the same inclusion and exclusion patterns for displaying changelog entries as it does for polling for changes. If this is disabled, changes which are excluded for polling are still displayed in the changelog.

Type:boolean

`additionalCredentials`

If there are additional credentials required in order to obtain a complete checkout of the source, they can be provided here.

The **realm** is how the repository self-identifies to a client. It usually has the following format:

`<proto://host:port> Realm Name`

- `proto` is the protocol, e.g. `http` or `svn`.
- `host` is the host how it's accessed by Jenkins, e.g. as IP address `192.168.1.100`, host name `svnserver`, or host name and domain `svn.example.org`.
- `port` is the port, even if not explicitly specified. By default, this is `80` for HTTP, `443` for HTTPS, 3690 for the `svn` protocol.
- `Realm Name` is how the repository self-identifies. Common options include `VisualSVN Server`, `Subversion Authentication` or the UUID of the repository.

To find out the realm, you could do any of the following:

- If you access the repository via HTTP or HTTPS: Open the repo in a web browser without saved credentials. It will use the `Realm Name` (see above) in the authentication dialog.
- Use the command line `svn` program.
  - If you don't have stored the credentials, run e.g. `svn info https://svnserver/repo` and it will tell you the realm when asking you to enter a password, e.g.: *Authentication realm: <svn://svnserver:3690> VisualSVN Server*.
  - If you have already stored the credentials to access the repository, look for the realm name in one of the files in `~/.subversion/auth/svn/simple`; it will be two lines below the line `svn:realmstring`

Make sure to enter the realm *exactly* as shown, starting with a <.

Array/List

Nested object

`realm`

This is the realm that the SvnKit library associates with a specific checkout. For most Subversion servers this will typically be of the format `<scheme://hostname(:port)/path...>`

Type: String

`credentialsId`

Select the credential from the list of relevant credentials in order to use that credential for checking out

the source code.

Type: String

`changelog` (optional)

Type:boolean

`poll` (optional)

Type:boolean

`deleteDir`: Recursively delete the current directory from the workspace

Recursively deletes the current directory and its contents. Symbolic links and junctions will not be followed but will be removed. To delete a specific directory of a workspace wrap the `deleteDir` step in a `dir` step.

`dir`: Change current directory

Change current directory. Any step inside the `dir` block will use this directory as current and any relative path will use it as base path.

`path`

Type: String

`echo`: Print Message

`message`

Type: String

`error`: Error signal

Signals an error. Useful if you want to conditionally abort some part of your program. You can also just `throw new Exception()`, but this step will avoid printing a stack trace.

`message`

Type: String

`fileExists`: Verify if file exists in workspace

Checks if the given file (as relative path to current directory) exists. Returns `true | false`.

`file`

Relative (/-separated) path to file within a workspace to verify its existence.

Type: String

`input`: Wait for interactive input

This step pauses flow execution and allows the user to interact and control the flow of the build. Only a basic "process" or "abort" option is provided in the stage view.

You can optionally request information back, hence the name of the step. The parameter entry screen can be accessed via a link at the bottom of the build console log or via link in the sidebar for a build.

`message`

This parameter gives a prompt which will be shown to a human:

```
Ready to go?
Proceed or Abort
```

If you click "Proceed" the build will proceed to the next step, if you click "Abort" the build will be aborted.

Type: String

`id` (optional)

Every `input` step has an unique ID. It is used in the generated URL to proceed or abort.

A specific ID could be used, for example, to mechanically respond to the input from some external process/tool.

Type: String

`ok` (optional)

Type: String

`parameters` (optional)

Request that the submitter specify one or more parameter values when approving. If just one parameter is listed, its value will become the value of the `input` step. If multiple parameters are listed, the return value will be a map keyed by the parameter names. If parameters are not requested, the step returns nothing if approved.

On the parameter entry screen you are able to enter values for parameters that are defined in this field.

Array/List

Nested choice of objects

- `$class: 'BooleanParameterDefinition'`

  `name`

  Type: String

  `defaultValue`

  Type:boolean

  `description`

  Type: String

- `$class: 'ChoiceParameterDefinition'`

  `name`

  Type: String

  `choices`

  Type: String

  `description`

  Type: String

- `$class: 'CredentialsParameterDefinition'`

  Defines a credentials parameter, which you can use during a build.

  For security reasons, the credential is _NOT_ directly exposed, the UUID of the credential is exposed.

  However, the selected credential is available through variable substitution in some other parts of the configuration. The string value will be the UUID of the credential. A supporting plugin can thus use the UUID to retrieve the selected credential and expose it to the build in an appropriate way.

  `name`

  Type: String

  `description`

  Type: String

  `defaultValue`

  The default credentials to use.

Type: String

credentialType

Type: String

required

When this option is selected, the credentials selection drop down will not provide the empty selection as one of the options. This will not prevent a build without a value if there are no credentials available, for example if the job does not have access to any credentials of the correct type or there is no default value and the user starting the build either does not have any credentials of the correct type in their personal credentials store or they do not have permissions on the job to use credentials from their personal store.

Type:boolean

- `$class: 'CvsTagsParamDefinition'`

name

The name this parameter will be referred to as during any builds.

Type: String

cvsRoot

Type: String

passwordRequired

Type:boolean

password

Type: String

moduleName

The name of the item to retrieve a list of symbolic names for. This could be a module root (e.g. moduleName), subdirectory (e.g. moduleName/sub/directory/) or individual file (e.g. moduleName/sub/directory/file.name).

Type: String

- `$class: 'FileParameterDefinition'`

name

Type: String

description

Type: String

- `$class: 'ListSubversionTagsParameterDefinition'`

  When used, this parameter will display a field at build-time so that the user is able to select a Subversion tag from which to create the working copy for this project.

  Once the two fields **Name** and **Repository URL** are set, you must

  1. ensure the job uses **Subversion** and
  2. set the **Repository URL** field of **Subversion** by concatenating the two fields of this parameter.

  For instance, if **Name** is set to `SVN_TAG` and **Repository URL** is set to `https://svn.jenkins-ci.org/tags`, then **Subversion**'s **Repository URL** must be set to `https://svn.jenkins-ci.org/tags/$SVN_TAG`.

  Notice that you can set the **Repository URL** field to a Subversion repository root rather than just pointing to a `tags` dir (ie, you can set it to `https://svn.jenkins-ci.org` rather than `https://svn.jenkins-ci.org/tags`). In that case, if this repository root contains the `trunk`, `branches` and `tags` folders, then the dropdown will allow the user to pick the trunk, or a branch, or a tag.

  `name`

  Type: String

  `tagsDir`

  Specify the Subversion repository URL which contains the tags to be listed when triggering a new build.

  You can also specify the root of a Subversion repository: If this root contains the `trunk`, `branches` and `tags` folders, then the dropdown will display `trunk`, all the branches and all the tags. If the root does not contain these three folders, then all its subfolders are listed in the dropdown.

  When you enter the URL, Jenkins automatically checks if it can connect to it. If access requires authentication, you'll be prompted for the necessary credential. If you already have a working credential but would like to change it for some other reasons, click this link and specify a different credential.

  Type: String

  `credentialsId`

  Type: String

  `tagsFilter`

  Specify a regular expression which will be used to filter the tags which are actually displayed when triggering a new build.

  Type: String

  `defaultValue`

  For features such as SVN polling a default value is required. If job will only be started manually, this

field is not necessary.

Type: String

`maxTags`

The maximum number of tags to display in the dropdown. Any non-number value will default to all.

Type: String

`reverseByDate`

Check this option so that tags are sorted from the newest to the oldest.

If this option is checked, the **Sort Z to A** one won't be taken into account.

Type:boolean

`reverseByName`

Check this option so that tags are displayed in reverse order (sorted Z to A).

Notice that if **Sort newest first** is checked, this option won't be taken into account.

Type:boolean

- `$class: 'PasswordParameterDefinition'`

  `name`

  Type: String

  `defaultValue`

  Type: String

  `description`

  Type: String

- `$class: 'RunParameterDefinition'`

  `name`

  Type: String

  `projectName`

  Type: String

  `description`

Type: String

`filter`

Values:

- ALL
- STABLE
- SUCCESSFUL
- COMPLETED

- `$class: 'StringParameterDefinition'`

  `name`

  Type: String

  `defaultValue`

  Type: String

  `description`

  Type: String

- `$class: 'TextParameterDefinition'`

  `name`

  Type: String

  `defaultValue`

  Type: String

  `description`

  Type: String

`submitter` (optional)

User ID or *external* group name of person or people permitted to respond to the input.

Type: String

`isUnix`: Checks if running on a Unix-like node

Returns true if enclosing `node` is running on a Unix-like system (such as Linux or Mac OS X), false if Windows.

`load`: Evaluate a Groovy source file into the workflow script

Takes a filename in the workspace and runs it as Groovy source text.

The loaded file can contain statements at top level or just load and run a closure. For example:

```
def flow
node('slave') {
    flow = load 'flow.groovy'
    flow.functionA()
}
flow.functionB()
```

Where `flow.groovy` defines functionA and functionB functions (among others) before ending with `return this;`

path

Current directory (`pwd()`) relative path to the Groovy file to load.

Type: String

mail: Mail

Simple step for sending email.

subject

Email subject line.

Type: String

body

Email body.

Type: String

bcc (optional)

BCC email address list. Comma separated list of email addresses.

Type: String

cc (optional)

CC email address list. Comma separated list of email addresses.

Type: String

`charset` (optional)

Email body character encoding. Defaults to `UTF-8`

Type: String

`from` (optional)

From email address. Defaults to the admin address globally configured for the Jenkins instance.

Type: String

`mimeType` (optional)

Email body MIME type. Defaults to `text/plain`.

Type: String

`replyTo` (optional)

Reploy-To email address. Defaults to the admin address globally configured for the Jenkins instance.

Type: String

`to` (optional)

To email address list. Comma separated list of email addresses.

Type: String

`node`: Allocate node

Allocates an executor on a node (typically a slave) and runs further code in the context of a workspace on that slave.

`label`

Computer name, label name, or any other label expression like `linux && 64bit` to restrict where this step builds. May be left blank, in which case any available executor is taken.

Type: String

`parallel`: Execute sub-workflows in parallel

```
org.kohsuke.stapler.NoStaplerConstructorException: There's no @DataBoundConstructor
on any constructor of class org.jenkinsci.plugins.workflow.cps.steps.ParallelStep
```

`pwd`: Determine current directory

Returns the current directory path as a string.

`readFile`: Read file from workspace

Reads a file from a relative path (with root in current directory, usually workspace) and returns its content as a plain string.

`file`

Relative (`/`-separated) path to file within a workspace to read.

Type: String

`encoding` (optional)

Type: String

`retry`: Retry the body up to N times

Retry the block (up to N times) if any exception happens during its body execution.

`count`

Type:int

`sh`: Shell Script

Runs a Bourne shell script, typically on a Unix node. Multiple lines are accepted.

An interpreter selector may be used, for example: `#!/usr/bin/perl`

Otherwise the system default shell will be run, using the `-xe` flags (you can specify `set +e` and/or `set +x` to disable those).

`script`

Type: String

`sleep`: Sleep

Simply pauses the workflow until the given amount of time has expired. Equivalent to (on Unix) `sh 'sleep …'`. May be used to pause one branch of `parallel` while another proceeds.

`time`

Type:int

`unit` (optional)

Values:

- `NANOSECONDS`
- `MICROSECONDS`
- `MILLISECONDS`
- `SECONDS`
- `MINUTES`
- `HOURS`
- `DAYS`

`stage`: Stage

By default, flow builds can run concurrently. The stage command lets you mark certain sections of a build as being constrained by limited concurrency. Newer builds are always given priority when entering such a throttled stage; older builds will simply exit early if they are preëmpted.

`name`

Type: String

`concurrency` (optional)

Concurrency level.

A concurrency of one is useful to let you lock a singleton resource, such as deployment to a single target server. Only one build will deploy at a given time: the newest which passed all previous stages

Type:int

`stash`: Stash some files to be used later in the build

Saves a set of files for use later in the same build, generally on another node/workspace. Stashed files are not otherwise available and are generally discarded at the end of the build.

`name`

Name of a stash. Should be a simple identifier akin to a job name.

Type: String

`excludes` (optional)

Optional set of "Ant-style exclude patterns.

Use a comma separated list to add more than one expression.
If blank, no file will be excluded.

Type: String

`includes` (optional)

Optional set of "Ant-style include patterns.
Use a comma separated list to add more than one expression.
If blank, treated like `**`: all files.
The current working directory is the base directory for the saved files, which will later be restored in the same relative locations, so if you want to use a subdirectory wrap this in `dir`.

Type: String

`useDefaultExcludes` (optional)

If selected, use the default excludes from Ant - see here for the list.

Type:boolean

`step`: General Build Step

This is a special step that allows to call builders or post-build actions (as in freestyle or similar projects), in general "build steps". Just select the build step to call from the dropdown list and configure it as needed.

Note that only workflow compatible steps will be shown in the list.

`delegate`

Nested choice of objects

- `$class: 'ArtifactArchiver'`

  Archives the build artifacts (for example, distribution zip files or jar files) so that they can be downloaded later. Archived files will be accessible from the Jenkins webpage.
  Normally, Jenkins keeps artifacts for a build as long as a build log itself is kept, but if you don't need old artifacts and would rather save disk space, you can do so.

  Note that the Maven job type automatically archives any produced Maven artifacts. Any artifacts configured here will be archived on top of that. Automatic artifact archiving can be disabled under the advanced Maven options.

  #### `artifacts`

  You can use wildcards like 'module/dist/**/*.zip'. See the includes attribute of Ant fileset for the exact format. The base directory is the workspace. You can only archive files that are located in your workspace.

Type: String

`allowEmptyArchive` (optional)

Normally, a build fails if archiving returns zero artifacts. This option allows the archiving process to return nothing without failing the build. Instead, the build will simply throw a warning.

Type:boolean

`caseSensitive` (optional)

Artifact archiver uses Ant `org.apache.tools.ant.DirectoryScanner` which by default is case sensitive. For instance, if the job produces *.hpi files, pattern "**/*.HPI" will fail to find them.

This option can be used to disable case sensitivity. When it's unchecked, pattern "**/*.HPI" will match any *.hpi files, or pattern "**/cAsEsEnSiTiVe.jar" will match a file called caseSensitive.jar.

Type:boolean

`defaultExcludes` (optional)

Type:boolean

`excludes` (optional)

Optionally specify the 'excludes' pattern, such as "foo/bar/**/*". A file that matches this mask will not be archived even if it matches the mask specified in 'files to archive' section.

Type: String

`fingerprint` (optional)

Type:boolean

`onlyIfSuccessful` (optional)

Type:boolean

- `$class: 'Fingerprinter'`

  Jenkins can record the 'fingerprint' of files (most often jar files) to keep track of where/when those files are produced and used. When you have inter-dependent projects on Jenkins, this allows you to quickly find out answers to questions like:
  - I have `foo.jar` on my HDD but which build number of FOO did it come from?
  - My BAR project depends on `foo.jar` from the FOO project.
    - Which build of `foo.jar` is used in BAR #51?
    - Which build of BAR contains my bug fix to `foo.jar` #32?

  To use this feature, all of the involved projects (not just the project in which a file is produced, but also the projects in which the file is used) need to use this and record fingerprints.

  See this document for more details.

`targets`

Can use wildcards like `module/dist/**/*.zip` (see the @includes of Ant fileset for the exact format). The base directory is the workspace.

Type: String

- `$class: 'JUnitResultArchiver'`

Jenkins understands the JUnit test report XML format (which is also used by TestNG). When this option is configured, Jenkins can provide useful information about test results, such as historical test result trends, a web UI for viewing test reports, tracking failures, and so on.

To use this feature, first set up your build to run tests, then specify the path to JUnit XML files in the Ant glob syntax, such as `**/build/test-reports/*.xml`. Be sure not to include any non-report files into this pattern. You can specify multiple patterns of files separated by commas.

Once there are a few builds running with test results, you should start seeing something like this.

`testResults`

Type: String

`healthScaleFactor` (optional)

The amplification factor to apply to test failures when computing the test result contribution to the build health score.
The default factor is `1.0`

- A factor of `0.0` will disable the test result contribution to build health score.
- A factor of `0.1` means that 10% of tests failing will score 99% health
- A factor of `0.5` means that 10% of tests failing will score 95% health
- A factor of `1.0` means that 10% of tests failing will score 10% health
- A factor of `2.0` means that 10% of tests failing will score 20% health
- A factor of `2.5` means that 10% of tests failing will score 25% health
- A factor of `5.0` means that 10% of tests failing will score 50% health
- A factor of `10.0` means that 10% of tests failing will score 0% health

The factor is persisted with the build results, so changes will only be reflected in new builds.

Type:double

`keepLongStdio` (optional)

If checked, any standard output or error from a test suite will be retained in the test results after the build completes. (This refers only to additional messages printed to console, not to a failure stack trace.) Such output is always kept if the test failed, but by default lengthy output from passing tests is truncated to save space. Check this option if you need to see every log message from even passing tests, but beware that Jenkins's memory consumption can substantially increase as a result, even if you never look at the test results!

Type:boolean

`testDataPublishers` (optional)

```
java.lang.UnsupportedOperationException: do not know how to categorize
attributes of type
hudson.util.DescribableList<hudson.tasks.junit.TestDataPublisher,
hudson.model.Descriptor<hudson.tasks.junit.TestDataPublisher>>
```

- `$class: 'Mailer'`

If configured, Jenkins will send out an e-mail to the specified recipients when a certain important event occurs.

1. Every failed build triggers a new e-mail.
2. A successful build after a failed (or unstable) build triggers a new e-mail, indicating that a crisis is over.
3. An unstable build after a successful build triggers a new e-mail, indicating that there's a regression.
4. Unless configured, every unstable build triggers a new e-mail, indicating that regression is still there.

For lazy projects where unstable builds are the norm, Uncheck "Send e-mail for every unstable build".

`recipients`

Type: String

`notifyEveryUnstableBuild`

Type:boolean

`sendToIndividuals`

If this option is checked, the notification e-mail will be sent to individuals who have committed changes for the broken build (by assuming that those changes broke the build).

If e-mail addresses are also specified in the recipient list, then both the individuals as well as the specified addresses get the notification e-mail. If the recipient list is empty, then only the individuals will receive e-mails.

Type:boolean

`svn`: Subversion

SVN step. It performs a checkout from the specified repository.

Note that this step is shorthand for the generic SCM step:

```
checkout([$class: 'SubversionSCM', remote: 'http://sv-
server/repository/trunk']])
```

`url`

Type: String

`changelog` (optional)

Type:boolean

`poll` (optional)

Type:boolean

`timeout`: Enforce time limit

Executes the code inside the block with a determined time out limit. If the time limit is reached, an exception is thrown, which leads in aborting the build (unless it is catched and processed somehow).

`time`

Type:int

`unit` (optional)

Values:

- `NANOSECONDS`
- `MICROSECONDS`
- `MILLISECONDS`
- `SECONDS`
- `MINUTES`
- `HOURS`
- `DAYS`

`tool`: Use a tool from a predefined Tool Installation

Binds a tool installation to a variable (the tool home directory is returned). Only tools already configured in `Configure System` are available here. If the original tool installer has the auto-provision feature, then the tool will be installed as required.

`name`

Type: String

`type` (optional)

Type: String

`unstash`: Restore files previously stashed

Restores a set of files previously `stash`ed into the current workspace.

`name`

Name of a previously saved stash.

Type: String

`waitUntil`: Wait for condition

Runs its body repeatedly until it returns `true`. If it returns `false`, waits a while and tries again. (Subsequent failures will slow down the delay between attempts.) There is no limit to the number of retries, but if the body throws an error that is thrown up immediately.

`withEnv`: Set environment variables

Sets one or more environment variables within a block. These are available to any external processes spawned within that scope. For example:

```
node {
  withEnv(['MYTOOL_HOME=/usr/local/mytool']) {
    sh '$MYTOOL_HOME/bin/start'
  }
}
```

(Note that here we are using single quotes in Groovy, so the variable expansion is being done by the Bourne shell, not Jenkins.)

See the documentation for the `env` singleton for more information on environment variables.

`overrides`

A list of environment variables to set, each in the form `VARIABLE=value` or `VARIABLE=` to unset variables otherwise defined. You may also use the syntax `PATH+WHATEVER=/something` to prepend `/something` to `$PATH`.

Array/List

Type: String

`wrap`: General Build Wrapper

This is a special step that allows to call build wrappers (also called "Environment Configuration" in freestyle or similar projects). Just select the wrapper to use from the dropdown list and configure it as needed. Everything inside the wrapper block is under its effect.

Note that only workflow compatible wrappers will be shown in the list.

`delegate`

Nested choice of objects

`writeFile`: Write file to workspace

Write the given content to a named file in the current directory.

`file`

Type: String

`text`

Type: String

`encoding` (optional)

Type: String

`ws`: Allocate workspace

Allocates a workspace. Note that a workspace is automatically allocated for you with the `node` step.

`dir`

A workspace is automatically allocated for you with the `node` step, or you can get an alternate workspace with this `ws` step, but by default the location is chosen automatically. (Something like `SLAVE_ROOT/workspace/JOB_NAME@2`.)

You can instead specify a path here and that workspace will be locked instead. (The path may be relative to the slave root, or absolute.)

If concurrent builds ask for the same workspace, a directory with a suffix such as `@2` may be locked instead. Currently there is no option to wait to lock the exact directory requested; if you need to enforce that behavior, you can either fail (`error`) when `pwd` indicates that you got a different directory, or you may enforce serial execution of this part of the build by some other means such as `stage name: '…', concurrency: 1`.

If you do not care about locking, just use the `dir` step to change current directory.

Type: String

Advanced/Deprecated Steps

`catchError`: Catch error and set build result

If the body throws an exception, mark the build as a failure, but nonetheless continue to execute the workflow from the statement following the `catchError` step. This is *only* necessary when using certain post-build actions (notifiers) originally defined for freestyle projects which pay attention to the result of the ongoing build.

```
node {
    catchError {
        sh 'might fail'
    }
    step([$class: 'Mailer', recipients: 'admin@somewhere'])
}
```

If the shell step fails, the flow build's status will be set to failed, so that the subsequent mail step will see that this build is failed. In the case of the mail sender, this means that it will send mail. (It may also send mail if this build *succeeded* but previous ones failed, and so on.) Even in that case, this step can be replaced by the

following idiom:

```
node {
    try {
        sh 'might fail'
    } catch (err) {
        echo "Caught: ${err}"
        currentBuild.result = 'FAILURE'
    }
    step([$class: 'Mailer', recipients: 'admin@somewhere'])
}
```

For all other cases, use plain `try-catch(-finally)` blocks:

```
node {
    sh './set-up.sh'
    try {
        sh 'might fail'
        echo 'Succeeded!'
    } catch (err) {
        echo "Failed: ${err}"
    } finally {
        sh './tear-down.sh'
    }
    echo 'Printed whether above succeeded or failed.'
}
// …and the workflow as a whole succeeds
```

See this document for background.

`dockerFingerprintFrom`: Record trace of a Docker image used in FROM

Normally used implicitly by method calls on the `docker` global variable. Records the fact that a Docker image was built from another.

`dockerfile`

Workspace-relative path of a `Dockerfile` which will be parsed for its `FROM` instruction.

Type: String

`image`

ID or tag of the image which was just built, like `--tag` of `docker build`.

Type: String

`toolName` (optional)

Type: String

`dockerFingerprintRun`: Record trace of a Docker image run in a container

Normally used implicitly by method calls on the `docker` global variable. Records the fact that a Docker image was used by this build.

`containerId`

Type: String

`toolName` (optional)

Type: String

`unarchive`: Copy archived artifacts into the workspace

`mapping` (optional)

```
java.lang.UnsupportedOperationException: do not know how to categorize attributes of type java.util.Map<java.lang.String, java.lang.String>
```

`withDockerContainer`: Run build steps inside a Docker container

Normally used implicitly by method calls on the `docker` global variable. Takes an image ID or symbolic name *which must already have been pulled locally* and starts a container based on that image. Runs all nested `sh` steps inside that container. The workspace is mounted read-write into the container.

`image`

Type: String

`args` (optional)

Any other arguments to pass to `docker run`.

Type: String

`toolName` (optional)

Type: String

`withDockerRegistry`: Sets up Docker registry endpoint

Normally used implicitly by method calls on the `docker` global variable. Sets up connection details for a Docker registry.

`registry`

Nested object

`url`

URL to the Docker registry you are using. May be left blank to use the public DockerHub registry (currently `https://index.docker.io/v1/`).

Type: String

`credentialsId`

Type: String

`withDockerServer`: Sets up Docker server endpoint

Normally used implicitly by method calls on the `docker` global variable. Sets up connection details for a Docker server.

`server`

Nested object

`uri`

URI to the Docker Host you are using. May be left blank to use the Docker default (defined by DOCKER_HOST environment variable) (typically `unix:///var/run/docker.sock` or `tcp://127.0.0.1:2375`).

Type: String

`credentialsId`

Type: String

Variables

`docker`

The `docker` variable offers convenient access to Docker-related functions from a Pipeline script.

Methods needing a slave will implicitly run a `node {…}` block if you have not wrapped them in one. It is a good idea to enclose a block of steps which should all run on the same node in such a block yourself. (If using a Swarm server, or any other specific Docker server, this probably does not matter, but if you are using the default server on localhost it likely will.)

Some methods return instances of auxiliary classes which serve as holders for an ID and which have their own methods and properties. Methods taking a body return any value returned by the body itself. Some method parameters are optional and are enclosed with `[]`. Reference:

`withRegistry(url[, credentialsId]) {…}`

Specifies a registry URL such as `https://docker.mycorp.com/`, plus an optional credentials ID to

connect to it.

withServer(uri[, credentialsId]) {…}

Specifies a server URI such as `tcp://swarm.mycorp.com:2376`, plus an optional credentials ID to connect to it.

withTool(toolName) {…}

Specifies the name of a Docker installation to use, if any are defined in Jenkins global configuration. If unspecified, `docker` is assumed to be in the `$PATH` of the slave agent.

image(id)

Creates an `Image` object with a specified name or ID. See below.

build(image[, dir])

Runs `docker build` to create and tag the specified image from a `Dockerfile` in the current directory (or `dir`). Returns the resulting `Image` object. Records a `FROM` fingerprint in the build.

Image.id

The image name with optional tag (`mycorp/myapp`, `mycorp/myapp:latest`) or ID (hexadecimal hash).

Image.run([args])

Uses `docker run` to run the image, and returns a `Container` which you could `stop` later. Additional `args` may be added, such as `'-p 8080:8080 --memory-swap=-1'`. Records a run fingerprint in the build.

Image.withRun[(args)] {…}

Like `run` but stops the container as soon as its body exits, so you do not need a `try-finally` block.

Image.inside[(args)] {…}

Like `withRun` this starts a container for the duration of the body, but all external commands (`sh`) launched by the body run inside the container rather than on the host. These commands run in the same working directory (normally a slave workspace), which means that the Docker server must be on localhost.

Image.tag([tagname[, force]])

Runs `docker tag` to record a tag of this image (defaulting to the tag it already has). By default will rewrite an existing tag; pass `false` for the second argument to fail instead.

Image.push([tagname[, force]])

Pushes an image to the registry after tagging it as with the `tag` method. For example, you can use `image.push 'latest'` to publish it as the latest version in its repository.

Image.pull()

Runs `docker pull`. Not necessary before `run`, `withRun`, or `inside`.

Image.imageName()

The `id` prefixed as needed with registry information, such as `docker.mycorp.com/mycorp/myapp`. May be used if running your own Docker commands using `sh`.

Container.id

Hexadecimal ID of a running container.

`Container.stop`

Runs `docker stop` and `docker rm` to shut down a container and remove its storage.

env

Environment variables are accessible from Groovy code as `env.VARNAME`. You can write to such properties as well:

```
env.MYTOOL_VERSION = '1.33'
node {
  sh '/usr/local/mytool-$MYTOOL_VERSION/bin/start'
}
```

These definitions will also be available via the REST API during the build or after its completion, and from upstream Pipeline builds using the `build` step.

However any variables set this way are global to the workflow build. For variables with node-specific content (such as file paths), you should instead use the `withEnv` step, to bind the variable only within a `node` block.

A set of environment variables are made available to all Jenkins projects, including workflows. The following is a general list of variables (by name) that are available; see the notes below the list for Pipeline-specific details.

BUILD_NUMBER
: The current build number, such as "153"

BUILD_ID
: The current build ID, identical to BUILD_NUMBER for builds created in 1.597+, but a YYYY-MM-DD_hh-mm-ss timestamp for older builds

BUILD_DISPLAY_NAME
: The display name of the current build, which is something like "#153" by default.

JOB_NAME
: Name of the project of this build, such as "foo" or "foo/bar". (To strip off folder paths from a Bourne shell script, try: `${JOB_NAME##*/}`)

BUILD_TAG
: String of "jenkins-*${JOB_NAME}-${BUILD_NUMBER}*". Convenient to put into a resource file, a jar file, etc for easier identification.

EXECUTOR_NUMBER
: The unique number that identifies the current executor (among executors of the same machine) that's carrying out this build. This is the number you see in the "build executor status", except that the number starts from 0, not 1.

NODE_NAME
: Name of the slave if the build is on a slave, or "master" if run on master

NODE_LABELS
: Whitespace-separated list of labels that the node is assigned.

WORKSPACE
: The absolute path of the directory assigned to the build as a workspace.

JENKINS_HOME
: The absolute path of the directory assigned on the master node for Jenkins to store data.

JENKINS_URL
: Full URL of Jenkins, like `http://server:port/jenkins/` (note: only available if *Jenkins URL* set in system configuration)

BUILD_URL
: Full URL of this build, like `http://server:port/jenkins/job/foo/15/` (*Jenkins URL* must be set)

JOB_URL

Full URL of this job, like `http://server:port/jenkins/job/foo/` (*Jenkins URL* must be set)

The following variables are currently unavailable inside a workflow script:

- `NODE_LABELS`
- `WORKSPACE`
- SCM-specific variables such as `SVN_REVISION`

As an example of loading variable values from Groovy:

```
mail to: 'devops@acme.com',
    subject: "Job '${env.JOB_NAME}' (${env.BUILD_NUMBER}) is waiting for input",
    body: "Please go to ${env.BUILD_URL} and verify the build"
```

currentBuild

The `currentBuild` variable may be used to refer to the currently running build. It is an object similar to that documented for the return value of the `build` step (when `wait` is enabled). Additionally, for this build only (but not for other builds), the following properties are writable:

- `result`
- `displayName`
- `description`