# Custom Log Messages

Custom Log Messages provide a way for processes executed by Continua CI to interact with the Continua CI build. This includes the ability to set the build number, set variables and send real time status information. This feature will be extended over time.

## Message Format

```
@@continua[messageType parameter='value' parameter='value']
```

Note that the messageType field is not case sensitive. The parameter order is not defined, and parameter values must be quoted (with single quotes). NewLines, blackslashes, [ ] and single quotes must be escaped, e.g

```
@@continua[messageType name='test' value='This is a \nnew line and I am \'quoted\' by a\\b']
```

## Message Types

**message** - Sends a message to to the build log.

Parameters: value, status (debug, information, success, warning, error, fatal)

Example:

```
@@continua[message value='This will be logged' status='information']
```

**startGroup** - Starts a message group (tree node) in the build log.

Parameters: name

Example:

```
@@continua[startGroup name='Copying Files...']
```

**endGroup** - Ends a message group (tree node) in the build log

Parameters: name

Example:

```
@@continua[endGroup name='Copying Files...']
```

**setBuildVariable** (or **setVariable**) - Sets the value of a Continua CI build variable.

Parameters:

- *name*: The variable name. This is required.
- *value*: The new value to set the variable to . This is required.
- *appendOrIncrement***:** When true, append the new value to variable or type text or increment the value of a numeric variable type by the new value. Defaults to false which replaces the current value with the new value.
- *regExPattern*: A regular expression pattern to match against the existing variable value.
- *regExIgnoreCase*: When true, ignore case when matching regular expression. Defaults to false.
- *regExFirstMatchOnly*: When true, replace first regular expression match only. Defaults to false.
- *replacement*: The regular expression replacement text. This may contain back references to captured groups in the regular expression pattern, see Substitutions in Regular Expressions.
- *skipIfNotDefined*: When true, prevents the build failing if the variable has not been defined for the configuration. Defaults to false.
- *expand*: When true, expands any expressions in the value or replacement. Defaults to false.

Examples:

```
@@continua[setBuildVariable name='allowupload' value='true']
```

```
@@continua[setBuildVariable name='possiblyUnDefinedVariable' value='Hello world!' skipIfNotDefined='true']
```

```
@@continua[setBuildVariable name='counter' value='1' appendOrIncrement='true']
```

```
@@continua[setBuildVariable name='counter' regExPattern='v([\d]*)\.([\d]*)\.([\d]*)\.([\d]*)'
replacement='version $1.$2.$3 with build number $4']
```

**setServerVariable** - Sets the value of a Continua CI server variable (Configuration, Project or Application variable)

Parameters:

- *name*: The variable name. This is required.
- *value*: The new value to set the variable to . This is required.
- *appendOrIncrement*: When true, append the new value to variable or type text or increment the value of a numeric variable type by the new value. Defaults to false which replaces the current value with the new value.
- *regExPattern*: A regular expression pattern to match against the existing variable value.
- *regExIgnoreCase*: When true, ignore case when matching regular expression. Defaults to false.
- *regExFirstMatchOnly*: When true, replace first regular expression match only. Defaults to false.
- *replacement*: The regular expression replacement text. This may contain back references to captured groups in the regular expression pattern, see Substitutions in Regular Expressions.
- *skipIfNotDefined*: When true, prevents the build failing if the variable has not been defined for the configuration. Defaults to false.
- *expand*: When true, expands any expressions in the value or replacement.  Defaults to false.
- *updateBuildVariable*: When true, updates the corresponding build variable with the new value of the server variable. Defaults to false.
- *updateAgentVariable*: When true, updates the corresponding agent variable, referred to as %Configuration.VariableName%, %Project.VariableName% and %Application.VariableName%, with the new value of the server variable. Defaults to false.
- *keepBuildWriteLock*: When true, keeps any write lock acquired until the end of the build. Defaults to false.
- *releaseLockOnStageEnd*: When true, releases any write lock acquired at the end of the stage rather than end of build. Defaults to false.

Examples:

```
@@continua[setServerVariable name='allowupload' value='true']

@@continua[setServerVariable name='possiblyUnDefinedVariable' value='Hello world!' skipIfNotDefined='true']

@@continua[setServerVariable name='counter' value='1' appendOrIncrement='true' updateBuildVariable='true'
updateAgentVariable='true' keepBuildWriteLock='true']

@@continua[setServerVariable name='counter' regExPattern='v([\d]*)\.([\d]*)\.([\d]*)\.([\d]*)'
replacement='version $1.$2.$3 with build number $4' keepBuildWriteLock='true' releaseLockOnStageEnd='true']
```

If another build has a lock on the server variable then the setServerVariable message will fail. Due to the one-way nature of custom log message, we cannot pause the running process to respond and retry. We recommend that you use the Acquire Variable Lock action to ensure that you own a lock on the variable before running process which use this message

**loadVariable** - Loads the current value of a server variable into the build variable.

Parameters:

- *name*: The variable name. This is required.
- *expand*: When true, expands any expressions in the loaded value.
- *skipIfNotDefined*: When true, prevents the build failing if the variable has not been defined for the configuration. Defaults to false.
- *skipAgentVariable*: When true, skips updating the corresponding agent variable, referred to as %Configuration.VariableName%, %Project.VariableName% and %Application.VariableName%, with the loaded value.
- *acquireBuildWriteLock*: When true, acquires a write lock on the varaible until the end of the build.
- *releaseLockOnStageEnd*: When true, releases any write lock acquired at the end of the stage rather than end of build.

Example:

```
@@continua[loadVariable name='buildCounter1']

@@continua[loadVariable name='possiblyUnDefinedVariable' skipIfNotDefined='true']

@@continua[loadVariable name='version' skipAgentVariable='true' acquireBuildWriteLock='true'
releaseLockOnStageEnd='true']
```

If another build has a lock on the server variable and the *acquireBuildWriteLock* parameter is true then the loadVariable message will fail. Due to the one-way nature of custom log message, we cannot pause the running process to respond and retry. We recommend that you use the Acquire Variable Lock action to ensure that you own a lock on the variable before running process which use this message

**acquireVariableLock** - Acquires a write lock on a server variable so that no other builds can modify the value of the server variable until the lock is released, either at the end of the build or by another action

Parameters:

- *name*: The variable name. This is required.
- *releaseLockOnStageEnd*: When true, releases any write lock acquired at the end of the stage rather than end of build.

Example:

```
@@continua[acquireVariableLock name='buildCounter1' releaseLockOnStageEnd='true' ]
```

If another build has a lock on the server variable then the acquireVariableLock message will fail and log an error. Due to the one-way nature of custom log message, we cannot pause the running process to respond and retry. We recommend that you only use this message if you are sure no other builds can hold a lock on the variable, e.g. after acquiring a shared resource lock.

**releaseVariableLock** - Releases any a write lock on a server variable acquired by the current build. A warning will be logged if the current build does not hold a lock on the variable.

Parameters:

- *name*: The variable name. This is required.

Example:

```
@@continua[releaseVariableLock name='buildCounter1' ]
```

**setBuildVersion** - Sets the current build's version string.

Parameters:  value, verbose

Example:

```
@@continua[setBuildVersion value='1.2.3']
```

**setBuildStatus** - Sets the current action's build status, which is shown on the build details page when a build is running.

Parameters: name, verbose

Example:

```
@@continua[setBuildStatus value='Copying Files...']
```

**pinBuild** - Pins the current build, so that it is listed at the top of the build history and excluded from build cleanup. Optionally provide a *comment* to be associated with the pin and set the *appendComment* attribute to 'true' to append it to any existing comment.

Parameters: comment, appendComment, verbose

Example:

```
@@continua[pinBuild comment='Keep this awesome build' appendComment='true']
```

**unpinBuild** - Removes any pins from the current build.

Parameters: verbose

Example :

```
@@continua[unpinBuild]
```

**tagBuild** - Adds the supplied *tag* to the current build. Optionally set the *replaceExisting* attribute to 'true' to remove any existing tags first.

Parameters: tag, replaceExisting, verbose

Example:

```
@@continua[tagBuild tag='deployed' replaceExisting='true']
```

**removeBuildTag** - Remove a *tag* or all tags from current build. Either provide the name of the tag to remove or set the *removeAll* attribute to 'true' to remove all existing tags.

Parameters: tag, removeAll, verbose

Examples:

```
@@continua[removeBuildTag tag='deployed']

@@continua[removeBuildTag removeAll='true']
```

**addBuildComment**- Adds the supplied *comment* to the current build. Optionally set the *replaceExisting* attribute to 'true' to remove any existing comments first.

Parameters: comment, replaceExisting, verbose

Example:

```
@@continua[addBuildComment comment='This build was deployed to staging' replaceExisting='true']
```

**importUnitTestResults** - Imports unit tests and totals from unit test result files and registers them with the server.

Parameters:

- *type***:** One of junit, mstest, nunit or xunit. Identifies the type of unit test runner that created the results file. This is required.
- *filePatterns*: One or more paths to result files separated by the pipe character. Each path may contain ANT pattern wildcards. This is required.
- *failIfTestFails*: When true, the stage fails if any unit test fails. Defaults to true.
- *failIfTestErrors*: When true, the stage fails if any unit test errors. Defaults to true.
- *quiet*: When true, any non-error log messages are suppressed. Defaults to false.

Example:

```
@@continua[importUnitTestResults type='nunit' filePatterns='C:\pathto\tests\**.xml|C:\another\pathto\tests\**.
xml' failIfTestFails='false' failIfTestErrors='false' quiet='false' verbose='false']
```

## Common parameters

If the *verbose* parameter is set to true for any of the message types above, additional text describing the action will be written to the build log.

Example :

```
@@continua[setBuildVersion value='1.2.3' verbose='true']
```