

Variables

Table of contents

1 Variables.....	2
------------------	---

1. Variables

In all text and attribute values of an instruction file variables can be used. That makes instruction files very flexible and independent of a particular environment.

Usually variables should be used if

- the same value is needed at several places in the instruction file
- the value has to be changed more frequently than the instruction file logic itself
- the location of files should be variable and not fixed to a particular environment

1.1. Variables as Placeholder

If a variable is used as placeholder in an attribute value or the text of an XML tag then its name must be enclosed in curly braces.

At execution time the placeholder will be replaced by the current value assigned to the variable name.

Example:

```
<Set name="color" value="red"/>
<ReportBlock message="The preferred color is {color}"/>
```

This example produces the following tag in the output document.

```
<Block message="The preferred color is red"/>
```

1.2. Checking the Values of Variables

It is also possible to apply all assertion commands on the value of variables.

Therefore the variable name with prefix character '^' must be specified in the *element* attribute of the assertion command.

Example:

```
<AssertMatch element="^redirect.url">https://*/</AssertMatch>
```

1.3. Variable Names

Variables can consist of the following characters only:

a-z A-Z 0-9 . _ - \$

Other characters are not allowed and will cause an error with msgid="SERR003".

1.4. Predefined Variables

There are two variables that are always present during the execution of ConfigChecker.

INSTRUCTION_DIR

Contains the absolute directory path of the instruction file (i.e. ".cci" file) that was passed to ConfigChecker with the *-i* start parameter.

Usually that is very helpful to be used in `<Include>` commands if the cci file to be included is located in the same directory or a sub directory of where the main cci file is.

Example:

```
<Include>{INSTRUCTION_DIR}/server_checks.cci</Include>
```

LAST_ASSERTION_RESULT

This variable always contains the result of the last executed assertion. That is, its value is either "true" or "false". Before the first assertion its value is "true".

This variable can be used to control further execution depending on the result of an assertion.

Lets say for example, that the properties of a particular properties file should only be checked if the value of a particular element in an XML configuration file equals "enabled".

```
<XmlFile name="server_cfg.xml">
  <AssertEquals
element="//Connection[@name='backup']/@remote">enabled<AssertEquals/>
</XmlFile>
<ReportBlock if="{LAST_ASSERTION_RESULT}=true">
  <SettingsFile type="properties" name="remote.properteis">
    <AssertEquals element="port">7265<AssertEquals/>
    <AssertGreater element="max.threads">15<AssertGreater/>
  </SettingsFile>
</ReportBlock>
```

OS_FAMILY

This variable gets automatically set to one of the following values:

- "windows"
- "unix"
- "mac"

It can be used in *if* and *ifNot* attributes of the `Include` and `ReportBlock` commands to achieve operating system dependant execution control.

Example:

```
<Include if="unix">other_unix.cci</Include>
```

For the exact operating system name the Java system property *os.name* can be used.

Apart from these variables there are other predefined variable names that are defined by particular command.

For details see the command description of

- <ForEach>
- <ForFiles>

1.5. Scope of Variables

Variables can have a value in *global* and *local*.

The *global* scope is outside a *Data Source Adapter* tag.

The *local* scope is inside a *Data Source Adapter* tag.

If a value has been assigned to a variable in *global* as well as in *local* scope then the corresponding placeholder is always replaced by the *local* value.

That means, the *local* value always overrides the *global* value as long as the execution is inside the *Data Source Adapter* tag where the *local* variable has been set.

When execution reaches the end of a *Data Source Adapter* tag then the local values of these variables are all removed.

1.6. Ways to set Variables

There are many different ways to set the value of a variable.

1. Providing a properties file with start parameter *-v* (see Usage). That loads all properties from the specified file as variables into ConfigChecker.
2. Loading a properties file with the command <SetFromFile> from within an instruction set. This also loads all properties from the specified file as variables into ConfigChecker.
3. Using the command <Set> in an instruction set. This command's purpose is to assign a value to a variable.
4. Using the command <SetFrom> within an instruction set. This command's purpose is to assign a value from a configuration element to a variable.
5. The <ForEach> also sets a variable. That is either the default variable name "*_EACH_*" or the variable specified by its *name* attribute. The scope is always determined from the context the command is used in.
6. Since all Java system properties are generally available as variables for ConfigChecker it is also possible (but not recommended) to use the **-D** option with the JVM execution to set a variable.