ConfigChecker

Table of contents

1. Home

1.1. ConfigChecker

1.1.1. Introduction

ConfigChecker is a tool to analyze the numerous configuration settings of a software system and report the current configuration values or detect misconfiguration.

Essentially the tool can do two things:

1. Report the current configuration settings
2. Check the current configuration against an instruction set which defines what is expected and report the differences

A major strength of ConfigChecker is its capability to extract configuration settings from various sources.
For example it can read or check values in

- Plain text files
- XML files
- Java properties files
- Java manifest files
- Windows ini files
- Windows registry
- Apache conf files
- LDIF files

Get the trial version and see yourself what ConfigChecker can do for you.

1.2. ConfigChecker - History of Changes

1.2.1. Version 2.0

New data source adapter:

- Databse (not in trial version)
- WinRegistry (not in trial version)

New assertion commands:

- AssertMatchAll
- AssertMatchOneOf

Other features:

- Supports now <ValueList> as sub-tag of <AssertOneOf>
- Command <SetFrom> now supports an attribute *default*
- New pre-defined variable OS_FAMILY which containes the operating system family (i.e. "windows", "unix", "mac")
- In <ReportBlock> two new attributes for conditional execution are available: *beforeVersion*, *sinceVersion*
- In the element attribute of commands it is now possible to specify a variable name instead of a data source adapter specific element description path.

Dependencies:

- Now requires JRE 1.5 or later
- Now based on pf.jar 4.0.1

Bugfixes:

- Placeholders in <Value> elements have not been replaced
- NullPointerException if a cci without instructions file was included
- Commands AssertMatch, AssertContains, AssertOneOf didn't support the not-attribute
- Empty settings were not handled correctly
- Empty tags in XML files were not found
- Variable LAST_ASSERTION_RESULT was always true for optional="true" if element was not found

1.2.2. Version 1.0

Supported *Data Source Adapter*:

- SettingsFile
- XmlFile
- HttpdConfFile
- TextFile
- FileSystem
- SettingsObject
- LdifFile

Supported assertion commands:

- AssertExistence
- AssertEquals
- AssertLess
- AssertGreater

- AssertLessOrEqual
- AssertGreaterOrEqual
- AssertContains
- AssertMatch
- AssertOneOf
- AssertNewLine

Supported control commands:

- Include
- Set
- SetFrom
- SetFromFile
- ReportVariable
- ReportValue
- ReportBlock
- ForEach
- ForFiles

Supported reports:

- Minimum XML report that contains only output messages
- Default XML report with and output section
- Extended XML report with header, input and output sections
- Full XML report with header, plug-in list, input and output sections

Other features:

- Bundle with utility *Start Script Shell Generator*
- Automatic location of files inside zipped files
- Remote access to files via http
- Variables/Placeholders
- Automatic authentication
- Pluggable data source adapters and assertion commands

2. Guides

2.1. ConfigChecker

2.1.1. Guides

Find here information about

- Installation
- Usage

2.2. ConfigChecker - Install Guide

2.2.1. Installation

Installation of ConfigChecker is very easy. Just follow these steps:

- Create a new directory wherever you want
- Unzip the contents of *configchecker.zip* into this new directory
- Ensure that the environment variable `JAVA_HOME` points to an existing **JRE 1.5.x** directory (e.g. c:/programs/java1.5.0/jre)
- Generate a shell script that enables invocation of ConfigChecker from any directory (see below)

To learn about how to start ConfigChecker see Usage.

2.2.2. Start Script Shell Generator

There is a little utility program bundled with ConfigChecker called ***Start Script Shell Generator***. The purpose of this tool is to generate a shell script (according to the current operating system) that allows to run ConfigChecker from everywhere in the system without being forced to change the current path or explicitly specify a classpath or modify any environment variables.

To use it just run:

```
java -jar pf-sssgen.jar -f configchecker.jar
```

(under Windows you also can use the *sssgen.bat* instead)
This generates a *run.bat* or *run.sh* depending on the operating system.

To get help about the start parameters of *Start Script Shell Generator* just call it without any parameter.

2.3. Usage

2.3.1. Command-line Parameters

ConfigChecker understands the following command line options which can be specified in any order.

-i *filename*
Specifies the file that contains the instructions to be executed. This start

parameter is **required**.
-o *filename*
Specifies the file to which the output (i.e. the report) must be written. This start parameter is **optional**. If omitted the report will be written to stdout.
-v *filenames*
Specifies the name of one ore more properties files that contain initial variable definitions. Separate multiple files with the OS path separator (';' for Windows and ':' for Unix). The files are loaded in the order as specified from left to right. This start parameter is **optional**. These files can be used to define variable values that are referred to in the instructions set. That simplifies re-using the same instruction set(s) in slightly different environments.
-var
Specifies one ore more variable definitions as name/value pairs. Name and value must be separated by '='. Multiple variables must be separated by the OS path separator (';' for Windows and ':' for Unix). The variables specified with this parameter will be set after the loading of the properties files specified by parameter *-v*. This start parameter is **optional**.
-xsl [*filename*]
Specifies that the generated report should contain a processing instruction for an XSL style-sheet. That allows to display it immediately in a browser. The optional filename specifies the XSL file to refer to. If no filename is given then the default cocheck_default.xsl is used. If necessary it will be copied automatically to the same folder as the XML report. So the XML report can be rendered if opended in a browser. This start parameter is **optional**.
-html
If defined an HTML version of the report will be created. This is only possible if -o and -xsl are set too. This start parameter is **optional**.
-rm
Minimum report. Only the <Output> section will be reported. This start parameter is **optional**.
-rd
Default report. Only the <Header> and <Output> sections will be reported. This start parameter is **optional**.
-re
Enhanced report. The <Header> and <Intput> and <Output> sections will be reported. This start parameter is **optional**.
-rf
Full report. All available data will be reported. This start parameter is **optional**.
-q

Sets the *quiet* mode. That is, no processing monitor window will be shown. This could be useful on Unix servers where no GUI components are supported. This start parameter is **optional**.

-no

Specifies to write no output. Usually that is only useful if ConfigChecker is used embedded inside another application. This start parameter is **optional**.

-a *filename*

Specifies a file that contains authentication credentials for particular URLs or realm. This start parameter is **optional**. If a remote file with a protected URL is to be worked on, ConfigChecker can automatically authenticate if the credentials are specified in the file specified with the -a option.

3. References

3.1. Instruction File Reference

Here you will find the detailed description of all commands that can be used in an instruction set to control the reportig and/or checking of a system.

The reference is split up into the following categories:

- Using variables
- General control commands
- Reporting commands
- Data Source Adapter commands
- Assertion commands

3.2. Variables

3.2.1. 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 variable and not fixed to a particular environment

3.2.1.1. 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"/>
```

3.2.1.2. 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>
```

3.2.1.3. 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".

3.2.1.4. Predefined Variables

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

CONFIGCHECKER_VERSION
Contains the version number (e.g. "2.2.0") of the ConfigChecker that is currently executed (this is not available in ConfigChecker versions before 2.2.0).
Example:

```
<ReportBlock
sinceVersion="{CONFIGCHECKER_VERSION}:2.2.0">...</ReportBlock>
```

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="OS_FAMILY='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>

3.2.1.5. 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.

3.2.1.6. Ways to set Variables

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

1. Providing one or multiple properties files with start parameter -*v* (see Usage). The separator between the filenames is the OS specific path separator (':' for Unix and ';' for Windows). That loads all properties from the specified files as variables into ConfigChecker. The files are loaded from left to right. That implies, that later loaded files may override variables that have been set by previously loaded files.
2. Specifying name/value pairs files with start parameter -*var* (see Usage). These variables are set after the loading of properties files with parameter -*v*. That allows overriding particular variables that have been set by the properties files.
3. 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.
4. Using the command <Set> in an instruction set. This command's purpose is to assign a value to a variable.
5. Using the command <SetFrom> within an instruction set. This command's purpose is to assign a value from a configuration element to a variable.
6. All *AssertXXX* commands support an attribute *resultVar*. It allows to assign the assertion's result (true|false) to a variable with the name specified in that attribute. The scope of this variable can be set with the *scope* attribute.
7. 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.
8. 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.
The better way is using start-parameter *-var*.

3.3. Control Commands

3.3.1. Control Commands

3.3.1.1. Include

The purpose of this command is to include another instruction file at exactly the point where this command is invoked. The *Include* command cannot be put inside a *Data Source Adapter*. It can only be used directly under *Instructions* or one of the following control commands if they are directly placed under *Instructions*.

- ReportBlock
- ForEach
- ForFiles

Attribues

The following table lists all attributes that can be used with the *Include* command.

Attribute Name	Description	required/optional
if	The specified file will only be included if the condition in this attribute is **true**. Here **true** means actually any of the following string values: • true • yes • on • 1	optional
ifNot	The specified file will only be included if the condition in this attribute is **NOT** true. For a description of the syntax of conditions see the *if* attribute.	optional
beforeVersion	The inclusion of the specified file will only be executed if the condition in this attribute is true. The condition specifies the name of a variable and a	optional

	version number. The value of the variable then will be compared against the version number. The condition evaluates to true if the version value of the variable is a lower version than the specified value. The variable name and the version to compare with must be separated by a colon (':'). Example: **beforeVersion="build.version:4** The following values for variable *build.version* will evaluate the example to true: • 1.0.45 • 4.3.27 • 4.2 The following values for variable *build.version* will evaluate the example to false: • 5.1.3 • 4.3.28 • 4.10.2 • 12.2.7 • 4.3.102	
sinceVersion	The inclusion of the specified file will only be executed if the condition in this attribute is true. The condition specifies the name of a variable and a version number. The value of the variable then will be compared against the version number. The condition evaluates to true if the version value of the variable is a higher or equal version compared to the specified value. The variable name and the version to compare with must be separated by a colon (':'). Example: **sinceVersion="build.version:3.** The following values for	optional

	variable *build.version* will evaluate the example to true: • 3.0 • 3.21.7 • 3.21.11 • 11.0.1 • 3.123.2 The following values for variable *build.version* will evaluate the example to false: • 2.4.5 • 3.21.6 • 3.20.46.5	

Examples

Example	Description
<Include>/base/instructions/check12.cci</Include>	Inserts all commands from the instructions file named */base/instructions/check12.cci*
<Include if="servername='mx500'">*{INSTRUCTION_DIR}/r*	If the variable *servername* contains the value *mx500* then (and only then) the file *mx500_checks.cci* from the base directory for instruction files will be included.
<Include sinceVersion="build-version:3.5.1" beforeVersion="build-version:5.2.0">*{INSTRUCTI*	If the variable *build.version* a version that is equal or higher than *3.5.1* and lower than *5.2.0* then (and only then) the file *core_checks.cci* from the base directory for instruction files will be included.

3.3.1.2. ForEach

The purpose of this command is to repeat all enclosed instructions for all values specified in this command.

Attribues

The following table lists all attributes that can be used with this command.

Attribute Name	Description	required/optional
name	Specifies the name of the variable that contains the current value of the value list for each iteration.	optional

	If this attribute is not defined the default variable name **_EACH_** will be used. See the Variables page to find out what characters are allowed in variable names.	
values	Contains the list of values for which to loop over the enclosed instructions. For each value in this list the enclosed instructions will be executed once.	**required**
separator	Defines the separator in the list of values. If this attribute is not specified the default separator (i.e. comma ',') is used.	optional

Examples

```
<ForEach name="filename" values="test.jar:sample.jar:util.jar"
separator=":">
  <FileSystem>
      <AssertExistence label="{filename}"
element="/testdata/lib/{filename}"/>
  </FileSystem>
</ForEach>
```

Description: For each JAR file listed in the *values* attribute it will be checked if it exists in the directory /testdata/lib.

3.3.1.3. ForFiles

The purpose of this command is to iterate over a set of files or directories and execute all enclosed instructions for each file/directory found.

This command always sets some variables for each iteration (i.e. for each found file).

ABSFILENAME
Contains the absolute filename of the current iteration's file.
ABSDIRNAME
Contains the absolute directory name (i.e. without the file's name) of the current iteration's file.
RELFILENAME
Contains the relative file path of the found file. That is relative to the defined start

directory in attribute *dir*.
RELDIRNAME
Contains the relative directory path of the found file. That is relative to the defined start directory in attribute *dir*.
FILENAME
Contains the filename without any path information.
DIRNAME
Contains the relative directory path of the found file. That is relative to the current work directory.

Attribues

The following table lists all attributes that can be used with this command.

Attribute Name	Description	required/optional
dir	Specifies the directory from where to start searching for files matching the given pattern. If not set the search starts in the current working directory.	optional
pattern	Contains one or more name patterns that specify the filter for the files iterate over. A pattern can contain '*' for any number of any character and '?' for single occurance of any character. If more than one pattern is needed they must be separated by ';'.	**required**
recursive	Defines whether or not the file search should go recursivly throgh all sub directories. If not set the default value is "false".	optional
digit	Allows to specify a single wildcard character that represents a digit (i.e. 0-9). Usually '#' is used for this purpose, but any other character can be used as well.	optional

dirsOnly	If this attribute is set to "true" (or "yes, "on", "1") then the command iterates over directories rather than files. If not set the default value is "false".	optional

Examples

```
<ForFiles dir="configuration/local" pattern="*.properties"
recursive="true">
  <FileSystem>
    <ReportValue label="Timestamp of {_FILENAME_}"
element="{_ABSFILENAME_}|@lastModified"/>
  </FileSystem>
</ForFiles>
```

Description: For each found properties file the corresponding timestamp of its last modification gets reported.

```
<ForFiles pattern="*.html" dir="stats" recursive="yes">
  ...
</ForFiles>
```

Description:

```
  Assuming that the current directory is c:/temp and the file
  c:/temp/stats/january/access.html was found with the above command
  then the variables will have the folowing values:

  _ABSFILENAME_    c:/temp/stats/january/access.html
  _ABSDIRNAME_     c:/temp/stats/january
  _RELFILENAME_    january/access.html
  _RELDIRNAME_     january
  _FILENAME_       access.html
  _DIRNAME_        stats/january
```

3.3.1.4. Set

The purpose of this command is to set the value of a variable. It can be used everywhere in an instruction file.
Each setting of a variable overwrites the previous value.
Since variables can have a global and a local scope it is possible to specify the scope with this command. However, it is only reasonable to use this feature when setting a global variable's value from inside a local context (i.e. from inside a data source adapter).

Attribues

The following table lists all attributes that can be used with this command.

Attribute Name	Description	required/optional
name	Specifies the name of the variable to be set. See the Variables page to find out what characters are allowed in variable names.	**required**
value	Any text of any size. Can even be an empty value.	**required**
scope	The scope can either be "global" or "local". If omitted it will be determined from the current context. That is, if the command is executed in a local context (i.e. inside a data source adapter tag) then the scope by default is "local", otherwise "global".	optional

Examples

```
<Set name="basePath" value="sample/data/config"/>
```

Description: Sets the variable *basePath* to the current value *sample/data/config*.

```
<Set name="server.hostname" value="target.example.com" scope="global"/>
```

Description: Sets the global variable *server.hostname* to the current value *target.example.com*.

3.3.1.5. SetFrom

The purpose of this command is to assign the value of a configuration element to a variable.

Note:
This command must be used only inside a *Data Source Adapter* tag.

By default the variable is treated as *local* if not explicitly specified differently.

Attribues

The following table lists all attributes that can be used with this command.

Attribute Name	Description	required/optional
name	Specifies the name of the variable to be set. See the Variables page to find out what characters are allowed in variable names.	**required**
element	The element from which to retrieve the value that should be assigned to the variable. The syntax of this attribute depends on the *Data Source Adapter* inside this command is used. Refer to the documentation of the appropriate *Data Source Adapter*.	**required**
scope	The scope can either be *global* or *local*. If omitted it will be *local*.	optional
range	If the specified element returns more than one value then the range attribute can be used to specify which value(s) assign to the variable. Valid definitions for range are *all* or *first* or *last*. If omitted it will be *all*.	optional
separator	If for multiple values the range was set to *all* then this attribute can be used to define the separator between the values. If omitted the separator will be ','.	optional
default	Allows to specify a default value which will be assigned to the variable if the speified element cannot be found. If this attribute is omitted and the element cannot be found	optional

	then an error will added to the result report.	

Examples

```
<SettingsFile type="properties" name="base.properties">
  <SetFrom name="color_config_file" element="color.definitions"
scope="global"/>
</SettingsFile>
<SettingsFile type="ini" name="{color_config_file}">
  <ReportValue element="[Dialogs]/background"/>
</SettingsFile>
```

Description: Reads the filename of the ini-file that contains color configurations from property *color.definitions* in file base.properties and assigns it to variable *color_config_file*. Then it reports the current value of the *[Dialogs]/background* setting in this ini-file.

3.3.1.6. SetFromFile

With this command it is possible to load a set of variables from a properties file.
By default the loaded variables are added to the variable scope the command is executed in. Only if the command is running inside a *Data Source Adapter* (i.e. in local scope) it is possible to specify to load the variables to the global scope anyway.

Attribues

The following table lists all attributes that can be used with this command.

Attribute Name	Description	required/optional
file	Specifies the name of the properties file from which to read the variables.	**required**
scope	The scope can either be *global* or *local*. If omitted it will be the current execution scope.	optional

Examples

```
<SetFromFile file="set2.properties" scope="global"/>
```

Description: Reads all properties into the global variable pool. If some of the variables already exist their value will be modified.

3.4. Report Commands

3.4.1. Report Commands

The purpose of these commands is to explicitly write something to the report. Usually (with assertion commands) only a failed assertion is reported in the output document. The commands explained here are used to add more data and arbitrary text to the output.

3.4.1.1. ReportBlock

This command can be used everywhere in an instruction file. It can be indefinitly nested and can always contain all tags its parent tag would allow. The effect of ReportBlock is that in the output file a corresponding output tag <Block> is added. The tag contains a copy of all attributes from the ReportBlock command. So the ReportBlock can be used to add any arbitrary text to the ouput file. Apart from that the <Block> in the output will also contain all output tags corresponding to the inner tags of the ReportBlock instruction.

An additional feature is, that ReportBlock supports conditional execution of its inner tags. With the attributes *if* and *ifNot* it is possible to make the execution of the inner tags depending on the value of a variable or an even more complex expression.

Attribues

The following table lists all attributes that can be used with the *ReportBlock* command.

Attribute Name	Description	required/optional
if	The inner tags of ReportBlock will only be executed if the condition in this attribute evaluates to **true**. Here **true** actually means any of the following string values: • true • yes • on • 1	optional
ifNot	The inner tags of ReportBlock will only be executed if the condition in this attribute is **NOT** true. For a description of the syntax of conditions see the *if* attribute.	optional

beforeVersion	The inner tags of ReportBlock will only be executed if the condition in this attribute is true. The condition specifies the name of a variable and a version number. The value of the variable then will be compared against the version number. The condition evaluates to true if the version value of the variable is a lower version than the specified value. The variable name and the version to compare with must be separated by a colon (':'). Example: **beforeVersion="build.version:** The following values for variable *build.version* will evaluate the example to true: • 1.0.45 • 4.3.27 • 4.2 The following values for variable *build.version* will evaluate the example to false: • 5.1.3 • 4.3.28 • 4.10.2 • 12.2.7 • 4.3.102	optional
sinceVersion	The inner tags of ReportBlock will only be executed if the condition in this attribute is true. The condition specifies the name of a variable and a version number. The value of the variable then will be compared against the version number. The condition evaluates to true if the version value of the variable is a higher or equal version compared to the specified value. The variable name and the	optional

	version to compare with must be separated by a colon (':'). Example: **sinceVersion="build.version:3.** The following values for variable *build.version* will evaluate the example to true: • 3.0 • 3.21.7 • 3.21.11 • 11.0.1 • 3.123.2 The following values for variable *build.version* will evaluate the example to false: • 2.4.5 • 3.21.6 • 3.20.46.5	

Examples

Example	Description

`<ReportBlock label="First test set">`
`  <AssertExistence element="....."/>`
`</ReportBlock>`
	Simply creates a <Block> element in the output around the report results of the included child elements.

3.4.1.2. ReportVariable

The purpose of this command is to write the current value of a variable to the output.

Attribues

The following table lists all attributes that can be used with this command.

Attribute Name	Description	required/optional
name	Specifies the name of the variable to be reported. The specified variable must exist. Otherwise an error msgid="SERR002" will reported.	**required**

Examples

```
<Set name="title" value="Example"/>
<SettingsFile type="properties" name="config.properties">
  <Set name="title" value="inside"/>
  <ReportVariable name="title"/>
</SettingsFile>
<ReportVariable name="title"/>
```

Description: The first <ReportVariable> command will add

```
<Variable name="title">inside</Variable>
```

and the second <ReportVariable> command will add

```
<Variable name="title">Example</Variable>
```

to the output report file.

3.4.1.3. ReportValue

With this command it is possible to write the current value(s) of a configuration element to the report.

Note:
This command must be used only inside a *Data Source Adapter* tag.

Attribues

The following table lists all attributes that can be used with this command.

Attribute Name	Description	required/optional
element	Specifies the element to be reported. Be aware that the syntax of this attribute depends on the *Data Source Adapter* this command is used with. Refer to the appropriate *Data Source Adapter* documentation for details.	**required**

Examples

```
<SettingsFile type="manifest" name="base.jar/META-INF/MANIFEST.MF">
  <ReportValue element="Specification-Title"/>
  <ReportValue element="Specification-Version"/>
  <ReportValue element="Specification-Vendor"/>
</SettingsFile>
```

Description: Writes the values of the "Specification" settings in the manifest file of base.jar to the report.

3.5. Assertion Commands

3.5.1. Common Attributes

The following table lists all attributes the assertion commands have in common. However, keep in mind that particularly the syntax of the **element** attribute depends on the *Data Source Adapter* it is used with.

Apart from these common attributes each assertion command might have additional attributes that are specific for a particular *Data Source Adapter*. If such attributes exist they are documented with the corresponding *Data Source Adapter*.

Attribute Name	Description	required/optional
id	A unique identifier which will also be written to the report. It can be set to any value. Its main purpose is to associate a message in the report to its originating instruction in the instruction set. That allows easy tracking which reported value belongs to which instruction command.	optional
label	Sometimes the element's identifier is not quite human readable. In such a case it could be useful to define the *label* attribute with a name that a human being can understand. The contents of the *label* attribute is always copied unchanged to the report.	optional
element	The identifier for the element to check or report.	**required**

	Note: The syntax of this attribute depends on the *Data Source Adapter* it is used with. For an XML based *Data Source Adapter* that could be an XPath expression, for a Java properties file just a simple property name.	
not	If this attribute is set to "true" (or "yes, "on", "1") then the result gets negated. That's a simple way to express that something is expected *not* to be as specified.	optional
optional	If this attribute is set to "true" (or "yes, "on", "1") then the check is optional. That is, in case that the element cannot be found at all then this is not reported as an error or failure. However, if the element was found the check gets executed as usual and reports a failure if it wasn't successful.	optional
useSlashes	If this attribute is set to "true" (or "yes, "on", "1") then all backslashes in the found element's data will be translated into forward slashes. The default is "no".	optional
resultVar	If this attribute is set then the value of the assertion ("true" or "false") will be set to a variable with the name specified in this attribute. Whether the variable is "local" or "global" depends on the *scope* attribute.	optional
scope	Defines the scope of the variable name defined by attribute *resultVar*. The scope can either be "global" or "local".	optional

	The default is "local".	
skipReporting	If this attribute is set to **"all"** or **"failure"** or **"error"** then the outcome of the assertion might not be added to the report. **all** Neither error nor failure will be reported **failure** The failure of the assertion will not be reported **error** An error during the assertion will not be reported This option particularly makes sense in conjunction with the attribute *resultVar* so that the asstion result will be stored into a variable rather than causing a report entry. The default is "false".	optional

3.5.2. Common Child Elements for Multiple Values

The purpose of the following elements is to specify multiple values to check the found value(s) against.
There are several assertion commands that do not just check against a single pre-defined value but against a whole set of values.

Both elements can occur multiple times in any order. All their values are collected into a set which the found element(s) will be checked against.

Element Name	Description
Value	Specifies one value to check against.
ValueList	Specifies a list of values where the comma character is used as separator. However, the separator can be changed by setting the attribute *separator* to the character that should be used to delimit the single values in the list.

Example:

```
        <Assert...... element="...">
          <Value>200</Value>
          <Value>500</Value>
          <ValueList>330,340,350,380</ValueList>
          <Value>700</Value>
          <ValueList
separator="|">1000|2000|9000|11000|{number_list}</ValueList>
        </Assert......>
```

3.5.3. Commands

Note:

The following five commands automatically do an integer comparison if both, the expected and the actual value can be converted to integers. Otherwise, a string comparison is done.

- AssertEquals
- AssertGreater
- AssertGreaterOrEqual
- AssertLess
- AssertLessOrEqual

3.5.3.1. AssertEquals

The purpose of this command is to check whether or not the value of the specified element is equal to the value specified in the body this command. If the element's value is not equal to the specified value then a failure message (**msgid="FAIL0001"**) is written to the report.

Attributes

This command supports all the common attributes.

Example

```
<AssertEquals id="CHECK78" element="MaxConnections">25</AssertEquals>
```

Description: If the element *MaxConnections* ist not *25* then an assertion failure message will be written to the report.

```
<AssertEquals label="Server Name"
element="//Config/server[@id='M19']/host/@fqdn">
  www.testserver.com
</AssertEquals>
```

Description: If the element (i.e the server name) that is specified by the XPath expression is not *www.testserver.com* then an assertion failure message will be written to the report.

3.5.3.2. AssertLess

The purpose of this command is to check whether or not the value of the specified element is less than the value specified in the body of this command. If the element's value is not less than the specified value then a failure message (**msgid="FAIL0002"**) is written to the report.

Attributes

This command supports all the common attributes.

Example

```
<AssertLess id="L210" label="Limit LDAP search result"
element="searchLimit">1000</AssertLess>
```

Description: If the element *SearchLimit* ist not less than *1000* then an assertion failure message will be written to the report.

3.5.3.3. AssertGreater

The purpose of this command is to check whether or not the value of the specified element is greater than the value specified in the body of this command. If the element's value is not greater than the specified value then a failure message (**msgid="FAIL0003"**) is written to the report.

Attributes

This command supports all the common attributes.

Example

```
<AssertGreater id="GR09" element="port">9000</AssertGreater>
```

Description: If the element *port* ist not greater than *9000* then an assertion failure message will be written to the report.

```
<AssertGreater element="Specification-Version">F<AssertGreater>
```

Description: If the element *Specification-Version* is not greater than *F* then an assertion failure message will be written to the report.

3.5.3.4. AssertLessOrEqual

The purpose of this command is to check whether or not the value of the specified element is less or equal compared to the value specified in the body of this command. If the element's value is not less than or equal to the specified value then a failure message (**msgid="FAIL004"**) is written to the report.

Attributes

This command supports all the common attributes.

Example

```
<AssertLessOrEqual element="logLevel">3</AssertLessOrEqual>
```

Description: If the element *logLevel* ist not *3* or less then an assertion failure message will be written to the report.

```
<AssertLessOrEqual label="Max memory usage"
element="memory">1024<AssertLessOrEqual>
```

Description: If the element *memory* is not greater than *1024* then an assertion failure message will be written to the report.

3.5.3.5. AssertGreaterOrEqual

The purpose of this command is to check whether or not the value of the specified element is greater or equal compared to the value specified in the body of this command. If the element's value is not greater than or equal to the specified value then a failure message (**msgid="FAIL005"**) is written to the report.

Attributes

This command supports all the common attributes.

Example

```
<AssertGreaterOrEqual element="open.files">8</AssertGreaterOrEqual>
```

Description: If the element *open.files* ist not *8* or greater then an assertion failure message will be written to the report.

```
<AssertGreaterOrEqual element="//product[@name='tools']/@patch-level">
  4
<AssertGreaterOrEqual>
```

Description: If the element *//product[@name='tools']/@patch-level* is not greater than *4* then an assertion failure message will be written to the report.

3.5.3.6. AssertExistence

The purpose of this command is to check whether or not the specified element exists. If the element does not exist then a failure message (**msgid="FAIL0006"**) is written to the report.

Attributes

This command supports all the common attributes and additionally the following:

Attribute Name	Description	required/optional
emptyExists	If this attribute is set to "yes", an existing setting with an empty value is accepted as existing. If set to "no", such an empty value will cause a configuration failure message. If the attribute is not explicitly set the default value is "yes".	optional

Example

```
<AssertExistence label="proxy module"
element="LoadModule[@1='proxy_module']/@2"/>
```

Description: Asserts that the LoadModule proxy_module with a second parameter exists. If not a failure message is added to the report.

3.5.3.7. AssertMatch

With this command a value can be checked against a simple pattern. The pattern can be any string. In such a pattern the characters '*', '?' and '#' have specials meanings. The '*' stands for any number and any character. The '?' represents any single occurance of an arbitrary character. The '#' represents a single digit (i.e. 0-9). If the element does not match the pattern then the failure message (**msgid="FAIL0008"**) is added to the report.

Attributes

This command supports all the common attributes.

Example

```
<AssertMatch id="D420" label="Mail address"
    element="[cn=jdoe,ou=users,dc=company]/@mail">*.*@*.*</AssertMatch>
```

Description: If the LDAP element with the distinguished name *cn=jdoe,ou=users,dc=company* must have an attribute named *mail* with a typical eMail address value that is matching the pattern *.*@*.*. The value *john.doe@company.com* will be fine but the value *jdoe@company.com* will cause an assertion failure message being added to the report.

3.5.3.8. AssertContains

This command allows to treat the value of elements as a list of values. It can assert that a particular value is in such a list. The default separator for the list elements is comma (','). If the list value does not contain the expected value then the failure message (**msgid="FAIL0009"**) is added to the report.

Attributes

This command supports all the common attributes.

Example

```
<AssertContains id="ABC" label="Italian"
    element="supported.languages" case-sensitive="no">it</AssertContains>
```

Description: If the property *supported.languages* doesn't contain the value "it" (case insensitive comparison) then a failure message gets reported.

3.5.3.9. AssertOneOf

This command allows to check if the value of an element is equal to at least one of a list of allowed values. If the element's value is not equal to any of the allowed values then failure message (**msgid="FAIL0010"**) is added to the report.
The values to check against have to be specified by child elements <Value> and

<ValueList>.
Both child elements can be used as often as necessary and in arbitrary order. Find the detailed description here.

Attributes

This command supports all the common attributes.

Examples

```
<AssertOneOf label="Language" element="current.language"
case-sensitive="no">
  <Value>it</Value>
  <Value>fr</Value>
  <Value>de</Value>
</AssertOneOf>


<AssertOneOf label="Language" element="current.language"
case-sensitive="no">
  <ValueList>it,fr,de</ValueList>
</AssertOneOf>
```

Description: Both examples are equivalent. They are checking if the property *current.language* is "it" or "fr" or "de" (case insensitive comparison). If that is not the case then a failure message is added to the report.

3.5.3.10. AssertMatchOneOf

This command is useful to check if the value of an element matches at least one of a list of specified patterns. If the element's value does not match any of the defined patterns then failure message (**msgid="FAIL0018"**) is added to the report.
Currently only **'*'** and **'?'** are supported in patterns. Regular expressions are not yet supported.
The values to check against have to be specified by child elements <Value> and <ValueList>.
Both child elements can be used as often as necessary and in arbitrary order. Find the detailed description here.

Attributes

This command supports all the common attributes.

Examples

```
<AssertMatchOneOf label="Port" element="redirect.url" case-sensitive="no">
  <Value>http://*:8080/*</Value>
  <Value>https://*:9443/*</Value>
  <Value>http://*:81/*</Value>
</AssertMatchOneOf>


<AssertMatchOneOf label="Port" element="redirect.url" case-sensitive="no">
  <ValueList
separator=";">http://*:8080/*;https://*:9443/*;http://*:81/*</ValueList>
</AssertMatchOneOf>
```

Description: Both examples are equivalent. They are checking if the property *redirect.url* contains a URL with one of the port numbers "81" or "8080" or "9443". If that is not the case then a failure message is added to the report.

3.5.3.11. AssertMatchAll

With this command it is possible to ensure that the value of an element matches **all** of a list of specified patterns. If the element's value does not match any one of the defined patterns then failure message (**msgid="FAIL0016"**) will be added to the report.
If the assertion is negated (not="true") then failure message (**msgid="FAIL0015"**) will be added to the report if the found element's value matches at least one of the specified patterns.
Currently the wildcard characters **'*'** and **'?'** are supported in patterns. Regular expressions are not yet supported.
The patterns to check against have to be specified by child elements <Value> and <ValueList>.
Both child elements can be used as often as necessary and in arbitrary order. Find the detailed description here.

Attributes

This command supports all the common attributes.

Examples

```
<AssertMatchAll not="true" label="Logo" element="image.main.logo"
case-sensitive="no">
  <Value>*.gif</Value>
  <Value>*.jpg</Value>
  <Value>*.png</Value>
</AssertMatchAll>


<AssertMatchAll not="yes" label="Logo" element="image.main.logo"
```

```
case-sensitive="no">
  <ValueList>*.gif,*.jpg,*.png</ValueList>
</AssertMatchAll>
```

Description: Both examples are equivalent. They are checking if the property *image* matches none of the file name patterns "*.gif" or "*.jpg" or "*.png". If the property matches one of the patterns then a failure message is added to the report.

3.5.3.12. AssertNewLine

This assertion command is only reasonable within a text file data source adapter. It can be used to check the end-of-line characters in a text file. If at least one line is terminated by a different one than the expected the failure message (**msgid="FAIL0011"**) is added to the report.

Attributes

Attribute Name	Description	required/optional
element	The element name for this assertion command must always be **EOL** (i.e. "End Of Line").	**required**
value	The value attribute specifies the expected line-end character(s). Valid values are: • "CR" -> carriage return • "CRLF" -> carriage return/ line feed • "LF" -> line feed • "LFCR" -> line feed/carriage return	**required**

Example

```
<AssertNewLine label="Unix style line end" element="EOL" value="LF"/>
```

Description: Checks all lines in a file to end with line-feed (LF) character. If any line has different line-end character(s) then the assertion fails.

3.6. Data Source Adapter Commands

3.6.1. Data Source Adapter Commands

3.6.1.1. SettingsFile

This data source adapter can be used to work on files that contain settings specified by key/value pairs. That is, Java properties files, Java manifest files and Windows ini files.

Attribues

The following table lists all attributes that can be used with this data source adapter.

Attribute Name	Description	required/optional
name	Contains the name of the file to work on.	**required**
type	Specifies the the type of the file. Currently the following values are supported: • properties • manifest • ini	**required**

Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Instructions version="1">
  <Set name="testdata.dir">testdata</Set>
  <Set name="BasePath">{testdata.dir}/ini</Set>

  <SettingsFile name="{BasePath}/sample1.ini" type="ini" label="Base
Configuration" id="I01" case-sensitive="no">
    <AssertEquals id="AE01"
element="[Configuration]/Editor">notepad.exe</AssertEquals>
    <AssertExistence id="AX01" element="[Packer]/InternalUnzip"/>
    <ReportValue id="RV01" element="[Packer]/ZIP"/>
    <AssertGreater id="AG01" element="[1280x1024
(8x16)]/divider">400</AssertGreater>
    <ReportValue id="RV02" element="[1280x1024 (8x16)]/divider"/>
  </SettingsFile>
</Instructions>
```

3.6.1.2. TextFile

This data source adapter can be used to check text files. In general it allows to search for specific text lines in a file and check the found line(s) against a pattern.

Another useful feature is to check the line-end convention (i.e. LF or CRLF).

Attribues

The following table lists all attributes that can be used with this data source adapter.

Attribute Name	Description	required/optional
name	Contains the name of the file to work on.	**required**

3.6.1.3. XmlFile

The purpose of this data source adapter is to report or check values in XML files.

Attribues

The following table lists all attributes that can be used with this data source adapter.

Attribute Name	Description	required/optional
name	Contains the name of the file to work on.	**required**

Element Syntax

The element attribute in all enclosed commands must specify a valid **XPath** expression. With such an expression the element is identifed that has to be checked or of which the current value should be reported.

3.6.1.4. HttpdConfFile

With this data source adapter it is possible to check values in files that comply with the Apache configuration file format as for example the httpd.conf of the Apache Web Server.

Attribues

The following table lists all attributes that can be used with this data source adapter.

Attribute Name	Description	required/optional
name	Contains the name of the file to work on.	**required**

Element Syntax

The element attribute in all enclosed commands must use the following syntax to identify an element:

- *Section::CommandName[@param='...']/@param*
- *CommandName/@param*
- *CommandName[@param='...']/@param*
- *CommandName*
  This is the same as *CommandName/@1*

Where *Section* is the (optional) section name which is enclosed <...> tags like XML. *CommandName* is the command. It must always be specified. Finally *param* is the number of the command's parameter (starting with 1).
If '*' is used for *param* then all parameters are treated as one string.

Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Instructions version="1">
  <Include>{INSTRUCTION_DIR}/defaults.cci}</Include>
  <HttpdConfFile name="{RootPath}/httpd/sample1.conf">
    <AssertEquals element="AccessFileName">.htaccess</AssertEquals>
    <AssertEquals element="DirectoryIndex/@3">welcome.html</AssertEquals>
    <AssertEquals
element="LoadModule[@1='proxy_module']/@2">modules/ApacheModuleProxy.dll</AssertEquals>
    <AssertEquals element="Directory
''/usr/ApacheGroup/Apache/cgi-bin''::AllowOverride/@1">None</AssertEquals>
    <ReportValue element="CustomLog/@*"/>
  </HttpdConfFile>
</Instructions>
```

3.6.1.5. LdifFile

With this data source adapter it is possible to report and check values in LDIF files. It allows to select one or more directory objects in the LDIF file by either a distinguished name (see RFC 2253) or a search filter (**not** LDAP search query - RFC 2254).

Attribues

The following table lists all attributes that can be used with this data source adapter.

Attribute Name	Description	required/optional
name	Contains the name of the file to work on.	**required**

Element Syntax

The element attribute of all enclosed commands must use the following syntax to identify an element:

- *dn/@attrname*
- *[serach filter]/@attrname*

The first variant identifies exactly one object by its distinguished name.
The second variant uses a search filter with a simple syntax. It uses attribute names of the searched objects and compares them with static values. The following operators are supported:

Operator	Purpose	Example
=	string match	name='Jo*'
+	AND	name='Fred' + surname='Flintstone'
|	OR	location='London' | location='Paris'
!	NOT	country='US' + ! telephoneNumber='555*'
()	grouping	(a=7 | b=9) + (m='text' | m='ascii')

In both variants the *@attrname* then is the name of the attribute of the found object(s) to be reported or checked.

Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Instructions version="1">
  <LdifFile name="{RootPath}/ldif/sample1.ldif">
    <AssertEquals element="uid=tmorris,ou=People,
dc=example,dc=com/@roomNumber">4117</AssertEquals>
    <AssertOneOf element="[givenName='Gern'|sn='W*']/@location">
      <Value>Santa Clara</Value>
      <Value>Cupertino</Value>
      <Value>Sunnyvale</Value>
    </AssertOneOf>
    <AssertMatch
element="[(sn='B*'|sn='F*')+!(sn='Burton'|sn='Fisher')]/@telephoneNumber">
      +1 ### ### ####
    </AssertMatch>
  </LdifFile>
</Instructions>
```

3.6.1.6. FileSystem

The purpose of this data source adapter is to check/report the existence of files. It also allows to refer to the "last modified" date or the size of a file.

Attribues

This data source adapter has no attributes.

Extra Command Attribues

The following table lists attributes that can be used with assertion or report commands when used inside this data source adapter.

Attribute Name	Description	required/optional
format	Specifies the date format for the *lastmodified* field of a file. Use "dd" for day, "MM" for month and "yy" or "yyyy" for year. If the time should also be considered use "hh" for hours, "mm" for minutes and "ss" for seconds. Examples: "MM/dd/yy", "dd.MM.yyyy", "dd/MM/yyyy hh:mm:ss" Generally everything specified for java.text.SimpleDateFormat should work.	**optional**

Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Instructions version="1">
  <Set name="testdata.dir">/usr/ben/testdata</Set>
  <Set name="BasePath">{testdata.dir}/config</Set>

  <FileSystem>
    <AssertExistence label="default configuration"
element="{BasePath}/default.cfg"/>
    <ReportValue id="F99"
element="{BasePath}/sample.jar/base.properties|@lastmodified"/>
    <AssertGreater
```

```
/install.log|@size">800</AssertGreater>
    <ReportValue label="Size of install.log"
element="{BasePath}/install.log|@size"/>
    <AssertGreaterOrEqual element="{BasePath}/install.log|@lastmodified"
format="dd.MM.yyyy">10.07.2004</AssertGreaterOrEqual>
  </FileSystem>
</Instructions>
```

4. Examples

4.1. Examples

4.1.1. Tomcat 5 Environment

Here you can find an example of a ConfigChecker report and the corresponding ConfigChecker instruction file that was used to create the report about an installation of Tomcat 5.

- Tomcat 5 Installation: Instruction files tomcat5.cci and jar_info.cci => Report (tomcat5.xml)

5. Try and Buy

5.1. How to get ConfigChecker

5.1.1. Trial Version

ConfigChecker is available for immediate download as a trial version.
The trial version is fully functioning but does not contain all data source adapters.

5.1.2. Full Version

In contrary the full version of ConfigChecker contains all data source adapters but cannot be downloaded from this site.
It must be ordered from Caprica Ltd.

5.1.3. Instruction File Editor (Eclipse Plugin)

To simplify creation and editing of ConfigChecker instruction files (cci) there is an eclipse plugin available for download. It supports content assist with help for the supported instructions.

5.2. Trial Version

5.2.1. V2.2.0

Download

Included are the following data sourec adapters:

- SettingsFile
- XmlFile
- FileSystem

5.3. Full Version

5.3.1. Buy the full version of ConfigChecker

To purchase the full version of ConfigChecker please send an eMail to orders@configchecker.com.
Please provide your postal address and the number of licenses you want to buy.
We'll then send you an invoice. After the amount is paid onto our account we send you a URL from where you can download the full version of ConfigChecker.

5.3.2. Pricing

Number of Licenses	Price w/o VAT	Price incl. 19.0% VAT
1-5	198.00 EUR	229.68 EUR
6-10	180.00 EUR	208.80 EUR
11-100	160.00 EUR	185.60 EUR
101-unlimited	150.00 EUR	174.00 EUR

5.4. Instructions Editor

5.4.1. Eclipse Plugin

This version of the ConfigChecker instructions editor was created for Eclipse 3.1 up to 3.5.x.

Download ConfigChecker Instructions Editor V2.3.0