CheckMate Quick Guide

Introduction

CheckMate is a versatile and easy-to-use Windows application for determining whether an XML or XHTML document conforms to a set of structural rules, even when you don't have a DTD or schema. The rules are written in the CheckMate Scripting Language (CSL). You can choose whether element and attribute names are validated against lists of allowed names. Without name validation CheckMate can be highly permissive when compared with grammar-based approaches to structure validation, and there are situations when this can be a very useful characteristic. Equally CheckMate may be used to enforce an extremely stringent validation regime.

CheckMate has a graphical user interface that makes it easy to locate any structural errors detected, and error reports can be exported in an XML format. Typically CheckMate is used in conjunction with an XML editing tool such as Notepad++.

Setup and configuration

To install CheckMate, you need to copy the CheckMate application (*CheckMate*.exe*) and the CheckMate .ini file (*checkmate.ini*) to somewhere on your computer. The two files should be copied to the same folder. You may find it convenient to create a shortcut to the application and copy it to the Windows desktop.

The CheckMate .ini file tells CheckMate where to look for your XML files and CheckMate rules files, and which allowed elements and allowed attributes lists are to be used. It is also used to store various application settings.

A typical CheckMate .ini file looks something like this:

```
[Locations]
XMLFiles=C:\CheckMateTest\resources\XML files
Scripts=C:\CheckMateTest\resources\scripts

[Settings]
AllowedElements=C:\CheckMateTest\resources\scripts\elements.cml
AllowedAttributes=C:\CheckMateTest\resources\scripts\attribs.cml
  ⋮
```

The .ini file is a text file which you can edit using a text editor such as Windows Notepad in order to specify your own application configuration preferences. It is a good idea to create a backup copy of the file before modifying it.

(*Note*: You can change the XML file and script locations from within the application, *via File | XML file location…* and *File | Script file location…* .)

CheckMate scripts

A CheckMate script is a list of structural rules. Each rule represents a check that CheckMate will carry out on your XML document. Rules are processed sequentially. (The free version of CheckMate has a 50-rule limit. Scripts containing more than 50 rules will be loaded, but only the first 50 rules will be applied.)

Most CheckMate rules take the form 'X {predicate} Y', where X and Y are XML 'element objects' and {predicate} stands for one of a number of terms in the CheckMate Rule Scripting Language (CSL). For example, a script might include the rule:

```
* 1.2: <html> HAS_CHILD <head>
```

This means that the <html> element should always have <head> as a child element, while the rule

```
* 1.3: <html> HAS_ATTRIB @lang
```

means that the <html> element should always have a 'lang' attribute. Predicate terms are written in uppercase text, and underscore characters are used in place of spaces. A rule is said to be passed when its conditions are fulfilled by all the element objects to which it applies. If any such target element object does not meet the rule then the rule is said to fail.

CheckMate is designed to make it easy to define document structures in terms of element/attribute combinations. The term 'element object' is used here to refer to either an element name or an element name in conjunction with an attribute name or name/value pair.

In CheckMate scripts, element objects are written in angle brackets, with any qualifying attribute being given after an '@' symbol. For example, '<p>' refers to all **p** elements, irrespective of the attributes that might be applied to them. '<p@class('biblio')>' means **p** elements that have a class attribute with the value 'biblio'. (In an XML document, the opening tag for such an element would be <p class="biblio">.)

Some expressions involve lists of element objects. These are defined by placing the element objects within a set of square brackets or curly braces, depending on the list type, with commas or vertical bars used to separate the individual element objects. (See examples below.)

(*Note*: If an attribute value is specified then it *must* be placed within single quotation marks, in parentheses following the attribute name.)

The full list of CSL expressions currently defined is:

CSL expression pattern	**Meaning & example(s)**
ROOT_IS X	Checks whether X is the root element object of the document *Example:* **ROOT_IS** <html>
EXISTS X [IN Y]	Checks whether the element object X occurs in the document *Examples:* **EXISTS** <section@class('book-meta')> **EXISTS** <p@class('title')> **IN** <section@class('chapter')>

X [IN Y] HAS_PARENT Z	Checks whether element object X has element object Z as its parent *Examples:* <p@class('chap-title')> **HAS_PARENT** <section@class('chap-meta')>
X [IN Y] HAS_CHILD Z	Checks whether element object X has element object Z as one of its children *Example:* <head> **HAS_CHILD** <link>
X [IN Y] HAS_SIBLING Z	Checks whether element object X has element object Z as a sibling *Example:* <meta@name('dc.title')> **HAS_SIBLING** <meta@name('dc.creator')>
X [IN Y] DESCENDS_FROM Z	Checks whether element object X is a descendant of element object Z, i.e. Z must be the parent or grandparent or great-grandparent (…) of X *Example:* <body> **DESCENDS_FROM** <html>
X [IN Y] CONTAINS Z	Checks whether element object Z descends from element object X *Example:* <div@class('copyright-container')> **CONTAINS** <span@class('copyright-holder')>
X [IN Y] HAS_ATTRIB @z	Checks whether element object X has attribute @z *Examples:* <html> **HAS_ATTRIB** @lang <html> **HAS_ATTRIB** @xmlns('http://www.w3.org/1999/xhtml')
X [IN Y] HAS_ALL_CHILDREN_IN <list>	Checks whether element object X's children include *all* the element objects in the specified list *Example:* <article@class('book')> **HAS_ALL_CHILDREN_IN** **[**<section@class('book-meta')>**,** <section@class('book-front')>**,** <section@class('book-body')>**,** <section@class('book-back')>**]**
X [IN Y] HAS_CHILDREN_ONLY_FROM <list>	Checks whether *all* of element object X's children are drawn from the specified list *Examples:* <body> **HAS_CHILDREN_ONLY_FROM [**<article@class('book')>**]** <section@class('book-meta')> **HAS_CHILDREN_ONLY_FROM** **[**<section@class('book-series-info-sec')>**,** <section@class('book-title-page')>**,** <section@class('book-pub-rights')>**]**
X [IN Y] IS_FOLLOWED_BY Z	Checks whether element object X has element object Z as its immediate successor sibling *Example:*

	<div@class('copyright-container')> **IS_FOLLOWED_BY** <div@class('publisher-container')>
COUNT X [IN Y] <num-op> <number> COUNT A [IN B] <num-op> COUNT C [IN D]	Checks whether the number of times element object X occurs in the document stands in the specified numerical relation to the specified number Checks whether the number of times element object A occurs in the document stands in the specified numerical relation to the number of times element object C occurs in the document *Examples:* **COUNT** <title> **IN** <head> **EQ** 1 **COUNT** <cite> **GTE COUNT** <ref>
@x [ON Y [IN Z]] HAS_VALUE_MATCH @a [ON B [IN C]]	Checks whether attribute @x has the same value as attribute @a *Examples:* @rid **HAS_VALUE_MATCH** @id @rid **ON** <cite> **IN** <body> **HAS_VALUE_MATCH** @id **ON** <ref> **IN** <back>
@x [ON Y [IN Z]] MATCHES %<regex>%	Checks whether the value of attribute x is a match with the regular expression specified on the RHS between the % symbols Example: @id ON <p> **MATCHES** %^p-[0-9]+$%

Context

More specific contextual constraints can be expressed by qualifying the left hand side of many predicate expressions using the '**IN**' qualifier. To say that an element/attribute combination B occurs 'in' a particular element/attribute combination A means that B is a descendant of A. So for example if we say

<meta> **IN** <head> HAS_CHILD <link>

then we mean that when a <meta> element occurs in the context of a <head> element it should have a <link> element as a child.

Alternatives

Alternatives may be specified, using curly braces and vertical bar(s), for predicates that take a single value rather than a list on the right hand side. (Those predicates are HAS_CHILD, HAS_PARENT, CONTAINS, DESCENDS_FROM, IS_FOLLOWED_BY, HAS_ATTRIB.) For example:

<ref> IS_FOLLOWED_BY **{** <ref> **|** <ref-head> **}**

means that a <ref> element must have as its immediate successor element (at the same level, i.e. as a sibling rather than a child) either another <ref> or a <ref-head> element.

The null element, <~>

If you want to say that an element may be empty, i.e. need not have any child element, or you wish to stipulate that an element may be followed by one of a number of elements or by *no element at all*, then the null element, represented as <~>, is used. The null element is short-hand for 'no element'.

For example, if you wish to say that a paragraph must be followed by another paragraph, a heading, or by nothing at all, then this can be accomplished by a rule like this:

```
<p> IS_FOLLOWED_BY { <p> | <head> | <~> }
```

(*Note*: The null element may be used only with the HAS_CHILD, HAS_CHILDREN_ONLY_FROM and IS_FOLLOWED_BY predicates.)

Numerical operators

The following numerical operators are defined for use where relevant (at present only in conjunction with COUNT):

EQ	:	equal to
GT	:	greater than
LT	:	less than
GTE	:	greater than or equal to
LTE	:	less than or equal to
NE	:	not equal to

Overall script structure

CheckMate scripts are plain text files, and should conform to a number of rules:

- They should have the file extension '.cms', and should be saved to the scripts folder specified in the CheckMate .ini file (see above).
- Each script rule should be placed on a new line, and the line should begin with an asterisk character (*).
- There should then be a space, followed by a numerical string (a combination of digits and the period character, e.g. '1.5.2') to identify the rule.
- The identifier is followed by a colon and a space. After that comes the rule itself.
- Rules should not contain any line breaks.

Example:

```
* 1.1.2: @class ON <div> IN <section@class('book-toc')> MATCHES %^toc-[a-z]+$%
```

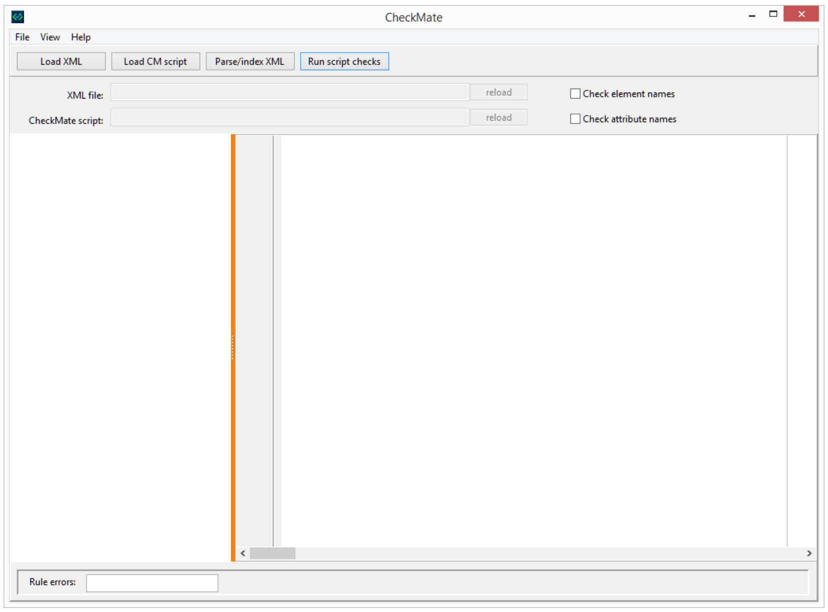
CheckMate treats lines beginning with a hash character as comments, and they are ignored. Comments may be used to create headings, which can be useful for grouping your rules into different categories.

Rules are placed within a defined section of the script file, the start of which is indicated by a line bearing the text '%RULES_START' and the end of which is indicated by '%RULES_END'.

(*Note*: CheckMate validates rules when they are loaded, and some additional validation may take place when a rule is applied to content. The application will report any errors it detects, and in general it is a good idea to correct any incorrect rules before attempting to apply a rules script to your content.)

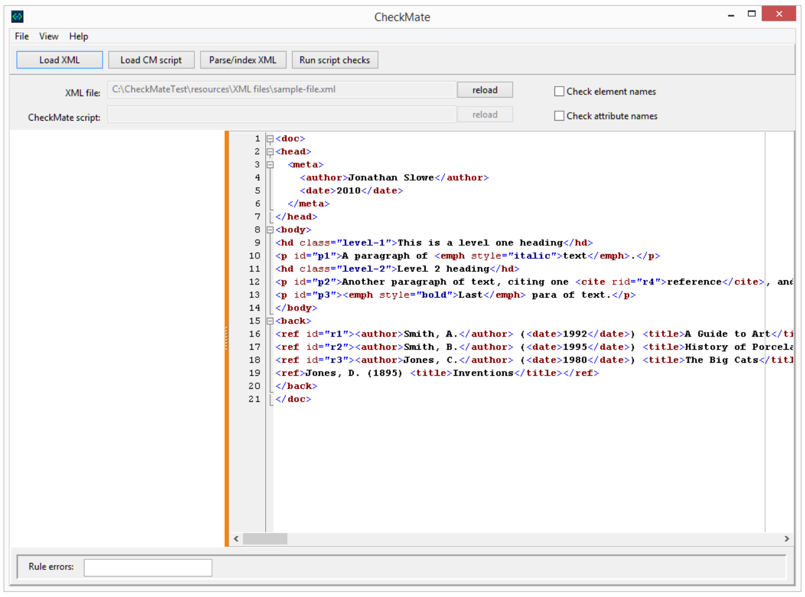
Running CheckMate

To launch CheckMate, navigate to the folder into which you copied the file CheckMate*.exe and double click on it, or click on the desktop shortcut if you created one. You will see the main CheckMate window:



Checking the structure of an XML or XHTML document using CheckMate consists of four main steps:

(1) **Load the document** via the *Load XML* button. The document will be visible in the right hand pane and the filename will be displayed in the upper text box:



XML file: C:\CheckMateTest\resources\XML files\sample-file.xml

CheckMate script:

```
<doc>
<head>
  <meta>
    <author>Jonathan Slowe</author>
    <date>2010</date>
  </meta>
</head>
<body>
<hd class="level-1">This is a level one heading</hd>
<p id="p1">A paragraph of <emph style="italic">text</emph>.</p>
<hd class="level-2">Level 2 heading</hd>
<p id="p2">Another paragraph of text, citing one <cite rid="r4">reference</cite>, an
<p id="p3"><emph style="bold">Last</emph> para of text.</p>
</body>
<back>
<ref id="r1"><author>Smith, A.</author> (<date>1992</date>) <title>A Guide to Art</ti
<ref id="r2"><author>Smith, B.</author> (<date>1995</date>) <title>History of Porcel
<ref id="r3"><author>Jones, C.</author> (<date>1980</date>) <title>The Big Cats</titl
<ref>Jones, D. (1895) <title>Inventions</title></ref>
</back>
</doc>
```

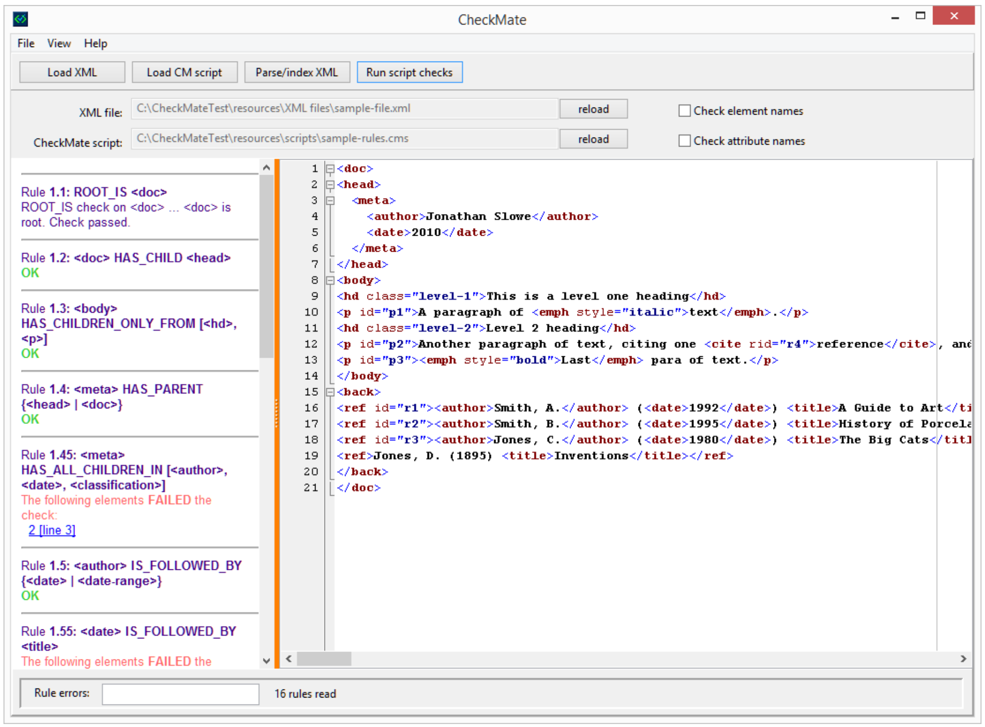
(2) **Load the CheckMate script** against which you wish to validate the XML document's structure, via the *Load CM script* button. The script name will be displayed in the lower text box in the upper panel. If any of the rules are found to contain errors, this is reported by the application. The numbers of any incorrect rules, if there are any, are listed in the rule error box within the lower information panel. Once all rules have been loaded the number of rules read is also stated in the information panel.

(3) **Parse your XML file**. This process creates indexes of all the elements and attributes in the document, and it is these indexes that the software examines when it carries out the checks defined in your script. Use the *Parse/index XML* button to initiate parsing. If the parser detects

errors in the basic structure of your XML file, e.g. mis-matched open and close tags, incorrect element nesting, etc, then you should correct the errors and re-parse the file until no errors are reported.

Optionally you may choose to have the parser validate the names of the elements and attributes in your file, by checking the *Check element names* and/or *Check attribute names* check boxes in the upper panel. Suitable lists of allowed element and attribute names must be defined and available to the application. These lists are text files that end with the extension '.cml'. (See sample files provided with the application.) To load an allowed element and/or attribute file select *File | Load allowed elements files* or *File | Load allowed attributes file* as appropriate. Once a file has been successfully loaded you can select *View | Allowed elements* or *View | Allowed attributes* to inspect the list. (*Note*: you must use an external text editor to modify the contents of an allowed list.)

(4) **Click on *Run script checks***. Checking is generally very quick, although the time taken will depend on the number of rules, the length of the document being checked, and various other factors.[1] While checks are carried out the number of the rule currently being applied is shown in the lower information panel. Once all rules have been applied a checking report is displayed in the left hand pane:



XML file: C:\CheckMateTest\resources\XML files\sample-file.xml

CheckMate script: C:\CheckMateTest\resources\scripts\sample-rules.cms

☐ Check element names

☐ Check attribute names

Rule **1.1: ROOT_IS <doc>**
ROOT_IS check on <doc> ... <doc> is root. Check passed.

Rule **1.2: <doc> HAS_CHILD <head>**
OK

Rule **1.3: <body> HAS_CHILDREN_ONLY_FROM [<hd>, <p>]**
OK

Rule **1.4: <meta> HAS_PARENT {<head> | <doc>}**
OK

Rule **1.45: <meta> HAS_ALL_CHILDREN_IN [<author>, <date>, <classification>]**
The following elements **FAILED** the check:
2 [line 3]

Rule **1.5: <author> IS_FOLLOWED_BY {<date> | <date-range>}**
OK

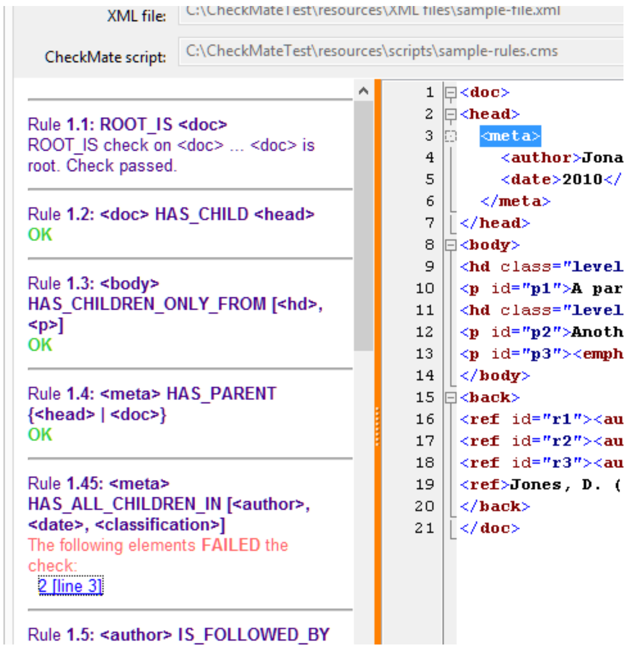
Rule **1.55: <date> IS_FOLLOWED_BY <title>**
The following elements **FAILED** the

```
 1  <doc>
 2  <head>
 3    <meta>
 4      <author>Jonathan Slowe</author>
 5      <date>2010</date>
 6    </meta>
 7  </head>
 8  <body>
 9  <hd class="level-1">This is a level one heading</hd>
10  <p id="p1">A paragraph of <emph style="italic">text</emph>.</p>
11  <hd class="level-2">Level 2 heading</hd>
12  <p id="p2">Another paragraph of text, citing one <cite rid="r4">reference</cite>, an
13  <p id="p3"><emph style="bold">Last</emph> para of text.</p>
14  </body>
15  <back>
16  <ref id="r1"><author>Smith, A.</author> (<date>1992</date>) <title>A Guide to Art</ti
17  <ref id="r2"><author>Smith, B.</author> (<date>1995</date>) <title>History of Porcela
18  <ref id="r3"><author>Jones, C.</author> (<date>1980</date>) <title>The Big Cats</titl
19  <ref>Jones, D. (1895) <title>Inventions</title></ref>
20  </back>
21  </doc>
```

Rule errors: 16 rules read

The checking report shown in the application lists the rules and provides feedback on whether the document conformed to them. Links to erroneous structural elements are given, and clicking on them will highlight the corresponding place in the document:

[1] Some regex expressions may take some time to be evaluated.



XML file: C:\CheckMateTest\resources\XML files\sample-file.xml

CheckMate script: C:\CheckMateTest\resources\scripts\sample-rules.cms

Rule **1.1**: ROOT_IS <doc>
ROOT_IS check on <doc> ... <doc> is root. Check passed.

Rule **1.2**: <doc> HAS_CHILD <head>
OK

Rule **1.3**: <body> HAS_CHILDREN_ONLY_FROM [<hd>, <p>]
OK

Rule **1.4**: <meta> HAS_PARENT {<head> | <doc>}
OK

Rule **1.45**: <meta> HAS_ALL_CHILDREN_IN [<author>, <date>, <classification>]
The following elements FAILED the check:
2 [line 3]

Rule **1.5**: <author> IS_FOLLOWED_BY

```
<doc>
<head>
  <meta>
    <author>Jona
    <date>2010</
  </meta>
</head>
<body>
<hd class="level
<p id="p1">A par
<hd class="level
<p id="p2">Anoth
<p id="p3"><emph
</body>
<back>
<ref id="r1"><au
<ref id="r2"><au
<ref id="r3"><au
<ref>Jones, D. (
</back>
</doc>
```

The error report may be saved as a well-formed XML file which provides a basis for automatic creation (e.g. via XSLT) of an HTML file containing embedded error messages. To save the report select *File* | *Save error report*.

If you modify the XML document in another application (e.g. Notepad++) and wish to check it again, you can do so by loading the document again via the *Load XML* button or by simply clicking the *reload* button next to the upper text box. You must parse the document once more, since by editing it you will have made structural changes to the document that make the generated indexes obsolete. Then, assuming that you have already loaded a script, you can just click on the *Run script checks* button to re-check the document.

That covers the basics of using CheckMate. Additional functionality is accessible via the menu items, and it is worth exploring those. Further exciting features are planned for future versions of the software.

Your feedback matters!
Please send any comments or suggestions to
epistemicsystems@gmail.com.