# Greenfox – outline

## Getting started with greenfox

Consider **system S** – an imaginary system which is a collection of web services. We are going to validate a *file system representation* which is essentially a set of test results, accompanied by resources supporting validation (XSDs, codelists and data about expected response messages). The following listing shows a file system tree which is an example representation of system S, as observed at a certain point in time:

**system-s**

. resources

. . **codelists**

. . . *codelist-foo-article.xml*

. . **xsd**

. . . *schema-foo-article.xsd*

. testcases

. . **test-t1**

. . . config

. . . . *msg-config.xml*

. . . input

. . . . *getFooRQ\*.xml*

. . . output

. . . . *getFooRS\*.xml*

. . **+test-t2 (contents: see test-t1)**

. . usecases

. . . usecase-u1

. . . . usecase-u1a

. . . . . **+test-t3 (contents: see test-t1)**

The concrete file system tree must be distinguished from the *expected file system tree*, which is described by the following rules.

|  |  |  |
| --- | --- | --- |
| **File or folder** | **Name or pattern** | **Expectation** |
| folder | codelists | Contains one or more codelist files |
| folder | codelists/\* | A codelist file; may have any name; must be an XML document containing <codelist> elements with a |name attribute and <entry> children |
| folder | xsd | Contains one or more XSDs describing services messages |
| file | xsd/\* | An XSD schema file; may have any name |
| folder | test-\* | A test case folder, containing input, output and config folders; apart from these only optional log-\* files are allowed |
| folder | config | A test case config folder, containing file msg.config.csv |
| file | msg.config.csv | A CSV file (separator: comma; with headline; three columns with request file name, response file name, expected return code |
| folder | input | A test case input folder, containg files representing request messages |
| file | input/\* | A file representing a request message; may have any name, but must have name extension .xml or .json; mediatype must correspond to name extension |
| folder | output | A test case output folder, containing files representing response messages |
| file | output/\* | A file representing a response message; may have any name, but must have name extension .xml or .json; mediatype must correspond to name extension |

It should be noted that the number and location of testcase folders (test-\*) are unconstrained. This means that the testcase folders may be grouped and wrapped in any way, althoug they must not be nested, as a testcase folder has exactly three members: folders input, output and config with contents as described by the table. So the use of a testcases folder wrapping all testcase folders - and the use of usecase-\* folders adding additional substructure - is possible, but must not be expected. The placing of XSDs in folder resources/xsd, on the other hand, is obligatory, and likewise the placing of codelist documents in folder resources/codelists. The names of XSD and codelist file, on the other hand, are not constrained.

Besides the structural expectations, there are also content-related expectations:

* For every request message, there must be a response message with a name obtained by replacing in the request file name RQ with RS (e.g. getFooRQ and getFooRS)
* For every response message in XML format, there must be exactly one XSD against which the message can be validated successfully
* In response messages (XML or JSON format) with root element <getFooRS>, the values in elements //fooValue must be found in the codelist with name foo-article.
* In response messages (XML or JSON format), the return code must be as configured by the corresponding row in msg-config.csv.

Now we create a greenfox schema. The first version only checks the existence of non-empty XSD and codelists folders:

<greenfox greenfoxURI="http://www.parsqube.de/ns/greenfox-schema/examples/system-s"  
 xmlns="http://www.greenfox.org/ns/schema">  
   
 <domain path="\tt\greenfox\resources\example-system\system-s" label="system-s">   
   
 <!-- \*\*\* Root folder containing the test system \*\*\* -->  
 <folder foxpath=".">  
   
 <!-- \*\*\* XSD folder -->  
 <folder foxpath=".\\resources\xsd">  
 <targetSize msg="No XSD folder found" count="1"/>  
 <file foxpath="\*.xsd">  
 <targetSize msg="No XSDs found" minCount="1"/>  
 </file>  
 </folder>   
  
 <!-- \*\*\* Codelist folder -->  
 <folder foxpath=".\\resources\codelists">  
 <targetSize msg="No codelist folder found" count="1"/>  
 <file foxpath="\*">  
 <targetSize msg="No codelist files found" minCount="1"/>  
 </file>  
 </folder>   
   
 </folder>  
 </domain>   
</greenfox>

The <domain> element represents the root folder of a file system tree to be validated, which has a file path as specified by the @path attribute.

A <folder> element represents the set of folders matching the foxpath expression given by its @foxpath attribute. Foxpath [1] is an extended version of XPath 3.0 which supports file system navigation. Note that file system navigaton steps are preceded by a backslash operator, rather than a slash, which is used for node tree navigation steps. The foxpath expression is evaluated in the context of the containing folder (or domain, if there is no containing <folder>), so that for example the expression

.\\resources\xsd

resolves to the xsd folders contained by a resources folder found at any depth under the system-s folder. Similarly, a <file> element represents the set of files matching the foxpath expression given by its @foxpath element.

The number of folders or files belonging to the respective set can be constrained by a <targetSize> child element. Attributes @minCount, @maxCount and @count constrain the number in an obvious way. If the constraint is violated, the validation result includes a <gx:red> element which contains the message specified by @msg on the constraint element. The following snippet is from a validation report using a report format which groups errors by the resource on which they occur:

<gx:redResource folder="C:/tt/greenfox/resources/example-system/system-s/resources/xsd">  
 <gx:red navigationPath="\*.xsd"   
 constraintComp="targetCount"   
 constraintID="targetSize\_2"   
 minCount="1"   
 actCount="0"   
 msg="No XSDs found"/>  
 </gx:redResource>

The constraintID is an ID assigned by the greenfox processor. It can be used to identify the violated constraint in an augmented copy of the original schema, which is part of the validation results.

In a second step we extend our schema with an element describing a test case folder:

<folder foxpath=".\\test-\*[input][output][config]">  
 <targetSize msg="No testcase folders found" minCount="1"/>  
 <folderContent msg="Testcase contains member other than input, output, config, log-\*."   
 closed="true">  
 <memberFolders names="input, output, config"/>  
 <memberFile name="log-\*" occ="\*"/>  
 </folderContent>  
 …  
</folder>

The foxpath expression selects all folders found at any depth in the file tree, matching the name pattern test-\* and having (at least) three members input, output and config. The <targetSize> constraint checks that the system contains at least one such folder, and the <folderContent> constraint disallows any additional members except for *optional* files (@occ=\*) with a name matching log-\*.

We proceed with a constraint requiring the config folder to contain a file msg-config.csv:

<!-- \*\*\* msg config -->  
 <file foxpath="config\msg-config.csv" ... >  
 <targetSize msg="Config file missing" count="1"/>

...  
 </file>

We want to be more specific: the file must be a CSV file, and the third column (which according to the header row is called returnCode) must contain a value which is OK or NOFIND or matches the pattern ERROR\_\*. We add attributes to the <file> element which specify how to parse the CSV file into an XML representation (@mediatype, @csv.separator, @csv.header). As with other non-XML mediatypes (e.g. JSON), an XML view enables us to leverage XPath and *express* a selection of content items. An <xpath> constraint associates the selection of items (@expr) with a description of expected values (in this case: <in> child element):

<!-- \*\*\* msg config -->  
 <file foxpath="config\msg-config.csv"

**mediatype="csv" csv.separator="," csv.withHeader="yes"**>  
 <targetSize msg="Config file missing" count="1"/>  
   
 <!-- Check - configured return codes ok? -->  
 <xpath msg="Config file contains unknown return code" expr="//returnCode">  
 <in>  
 <eq>OK</eq>  
 <eq>NOFIND</eq>  
 <like>ERROR\_\*</like>  
 </in>   
 </xpath>   
 </file>

Note that the XPath expression (given by @expr) is evaluated in the context of the document node of the document obtained by parsing the file. Here comes an example of a conformant message definition file:

request,response,returnCode  
getFooRQ1.xml,getFooRS1.xml,OK  
getFooRQ2.xml,getFooRS2.xml,NOFIND

getFooRQ3.xml,getFooRS3.xml,ERROR\_SYSTEM  
  
while this example violates the constraint:

request,response,returnCode  
getFooRQ1.xml,getFooRS1.xml,OK

getFooRQ2.xml,getFooRS2.xml,NOFIND  
getFooRQ3.xml,getFooRS3.xml,ERROR-SYSTEM

We proceed to check request message files: for each such file, there must be a response file in the sibling <output> folder, with a name obtained by replacing in the requests file name the last substring RQ with RS. The constraint is expressed by the following <file> element:

<file foxpath="input\\*">  
 <targetSize msg="Input folder without request msgs" minCount="1"/>  
   
 <!-- Check - request with response ? -->  
 <foxpath msg="Request without response" count="1" expr="  
 let $expFileNameRS := file-name(.) ! replace(., '(.\*)RQ(.\*)\.(xml|json)$', '$1RS$2.$3')   
 return ..\..\output\\*[file-name(.) eq $expFileNameRS]" />  
 </file>

The foxpath expression first determines the file name of the response file ($expFileNameRS), then navigates into the sibling <output> folder and selects the file with such a name (if existant). The expectation that such a response file does exist is expressed by the @count attribute (@count=1) on <foxpath>.

A foxpath can also be used as a child to a folder descriptor, in order to retrieve folder contents across file boundaries and express an expectation about the overall results. We apply this technique in order to constrain the codelists folder to contain <codelist> elements with a @name attribute and at least one non-empty <entry> child:

<domain path="\tt\greenfox\resources\example-system\system-s" label="system-s">   
   
 <!-- \*\*\* Root folder containing the test system \*\*\* -->  
 <folder foxpath=".">   
 ...

<!-- \*\*\* Codelist folder -->  
 <folder foxpath=".\\codelists">  
 <targetSize msg="No codelist folder found" count="1"/>  
 **<foxpath expr="\*.xml/codelist[entry/@code/string()]/@name" minCount="1"/>** </folder>

Note the mixed navigation of the foxpath expression – starting with file system navigation to all \*.xml files, then drilling down into their content, ending at their <codelist> elements.

Now we turn to the files representing response messages. They must be “fresh”, that is, have a timestamp of last modification which is after a timestamp specified by a call parameter of the validation operation. This is accomplised by a <lastModified> constraint:

<!-- \*\*\* Response message \*\*\* -->   
<file foxpath="output\\*">  
 <targetSize msg="Input folder without request msgs" minCount="1"/>

<!-- \*\*\* Check - response fresh? \*\*\* -->  
 <lastModified msg="Stale output file" ge="${lastModified}"/>   
</file>

In this code, the placeholder ${lastModified} is substituted by the value supplied to the greenfox processor as input parameter and declared (and defaulted) in the schema as a context parameter:

<greenfox greenfoxURI="http://www.parsqube.de/ns/greenfox-schema/examples/system-s"  
 xmlns="http://www.greenfox.org/ns/schema">  
  
 <context>  
 <field name="lastModified" value="2019-12-07"/>  
 </context>   
 ...

We proceed with several expectations related to XML responses. First, they must be valid against some XSD found in the XSD folder. XSD validation is triggered by a <xsdValid> constraint, with a foxpath expression locating the XSD(s) to be used:

<!-- \*\*\* XML responses -->  
<file foxpath="output\\*.xml">

<!-- \*\*\* Check - XML response XSD valid? -->

<xsdValid msg="Response msg not XSD valid"   
 xsdFoxpath="$domain\resources\xsd\\\*.xsd"/>  
</file>

It is not necessary to specify an individual XSD – the greenfox processor inspects all XSDs matching the expression and selects for each document to be validated the appropriate XSD element declaration. (If not exactly one element declaration is found, an error is reported.) Note the variable reference $domain, which can be referenced in any XPath or foxpath expression and which points to the domain folder.

The next constraint checks if certain values from the response are found in a particular codelist.

<!-- \*\*\* Check - known article number? -->   
<xpath msg="Unknown foo article number"  
 expr="//\*:fooValue"   
 inFoxpath="$domain\\codelists\\*.xml/codelist[@name eq 'foo-article']/entry/@code"/>

This means comparing the results of a navigation into the document be checked (//\*:fooValue) with a navigation across the file system into aggregated folder contents:

$domain\\codelists\\*.xml/codelist[@name eq 'foo-article']/entry/@code

In order to check the return code, we must first read it from the document being checked, then navigate to the message config of the current test case, which is a CSV file, and retrieve the expected return code as the value in the third column (named returnCode) in the row where the second column (named response) matches the file name of the response. Such an operation can be expressed using foxpath:

<!-- \*\*\* Check - return code ok? \*\*\* -->   
<foxpath msg="Return code not the configured value" eq="true" expr="  
 let $actReturnCode := doc(.)//\*:returnCode  
 let $expReturnCode := ..\..\config\msg-config.csv\csv-doc(., ',', 'yes')   
 //record[response eq $fileName]/returnCode   
 return $actReturnCode = $expReturnCode"/>

The last two checks are also applied - in slightly adapted form - to JSON responses. Note the @mediatype attribute on the <file> element which ensures that the file is parsed as a JSON file and transformed into an XML representation:

<!-- \*\*\* JSON response message \*\*\* -->   
<file foxpath="output\\*.json" **mediatype="json"**>  
   
 <!-- \*\*\* Check - known article number? -->  
 <xpath msg="Unknown foo article number"  
 expr="//fooValue"   
 inFoxpath="$domain\\codelists\\*.xml/codelist[@name eq 'foo-article']/entry/@code"/>

<!-- \*\*\* Check - return code ok? \*\*\* -->   
 <foxpath msg="Return code not the configured value" eq="true" expr="  
 let $actReturnCode := json-doc(.)//\*:returnCode  
 let $expReturnCode := ..\..\config\msg-config.csv\csv-doc(., ',', 'yes')   
 //record[response eq $fileName]/returnCode   
 return $actReturnCode eq $expReturnCode"/>   
</file>

The complete schema is shown in the appendix A2. It demonstrates several basic types of constraints which are available for validating file system contents. These constraints are related to:

* Folder tree structure and folder contents (<targetSize>)
* File properties (<lastUpdate>)
* File contents of XML and non-XML files (<xpath>)
* File content relationships across document boundaries (<xpath>, <foxpath>).
* File content relationships involving different mediatypes (<xpath>, <foxpath>).