DC 2020 – outline

5‘ Introduction – why file system validation

40’ Guided tour with excercises

15’ Big picture

30’ Selected topics with examples and excercises

## An introduction to Greenfox, a schema language describing file system contents

A tutorial held at: Declarative Amsterdam, 2020-10-08, 14:00 – 15:30

## What is file system validation?

The term **file system validation** is used for an evaluation of a file system tree, defined as a root folder and everything directly or indirectly contained. The evaluation is controlled by a **Greenfox schema**, which is a set of **constraints**. The outcome of validation is a set of **validation results**, one result per validation of a single resource against a single constraint. The validation result is structured information which identifies the resource and the constraint, asserts **conformance** or violation and includes details about a possible violation. The validation results are mapped to a **validation report**, which is a list of results or some derived representation, namely statistical information.

## And why might you care?

We are used to and appreciate declarative validation of files, using well-known schema languages like XSD, RelaxNG, JSON Schema, SHACL. Our true interest, however, is broader – we want systems to be valid, and an important aspect of systems is file system trees – trees of folders and files. Such trees may, for example, accommodate …

* a set of applications in use
* a product to be shipped
* components of infrastructure
* data sources and assets
* a collection of test results
* a collection of observations, obtained by test execution, monitoring sanity checks, etc.
* a mixture of the aforementioned

In all cases we cannot help caring about *whether everything is as expected* - quality and reliability of important processes depend on this: conformance of file system contents to expectations.

Greenfox is a proposal how to validate file system contents declaratively. It is still in an early stage, but at the end of its first year of development I believe the conceptual groundwork to have reached a certain maturity, capable of guiding future development. If the current scope of functionality addresses at least some of your requirements, you may obtain within hours what otherwise would cost you large developmental effort leading to heaps of complex code. Using Greenfox you invest in an executable description of your expectations, rather than code. This characterization may be a slightly idealizing one, as the description may involve complex expressions which can be regarded as a sort of code; but by and large I think the statement is true. And I invite you to put it to a test.

The outline for this tutorial is this:

* We start with a guided tour, to give you an impression of the scope and look and feel of using Greenfox
* This is followed by a brief “big picture” - concepts & major features
* You are given an overview of the available constraint types
* We will briefly take a closer look at a couple of interesting constraint types
* And we conclude with an outlook, especially important because important features have been postponed in the interest of a solid fundament

## Guided tour

Please see folder: $greenfox/declarative-amsterdam-2020/schema

## Big picture

A “big picture” is all important for an understanding of Greenfox. Consider this analogy – how to learn the XQuery language?

1. Learn concepts and principles; (2) Study the catalog of expressions (syntax + semantics)

Same situation here, with Greenfox:

1. Learn concepts and principles; (2) Study the catalog of constraints (syntax + semantics)

Once you have understood concepts and principles, it is easy to extend you knowledge iteratively, familiarizing yourself with the various types of constraints, one at a time.

### Seven things

The big picture which I propose is just a collection of seven things:

* Resources
* Constraints
* Shapes
* Target declarations
* Link definitions
* Results & Reports

We’ll deal with them one by one, and then we’re done.

### Resources

There are two kinds of resources – folders and files. That was easy!

### Constraints

A constraint is a function applied to a single resource and yielding a validation result. A constraint has parameters, syntax and semantics. The syntax describes the representation of the constraint and its parameters; the semantics define how the validation result is determined. We saw various examples, as a reminder here further examples:

<fileSize eq="0"/>

<value exprXP="//@iata" length="3"/>

<value exprXP="//@iata" length="3" distinct="true"/>

<valuePair expr1XP="/project/@mintDate"

expr2XP="//milestoneDate" cmp="le"/>

<valuePair expr1XP="/project/@minDate" count1="1"

expr2XP="//milestoneDate" minCount2="3"

cmp="le"/>

<docSimilar linkName="…">

<skipItem kind="attribute" localName="timestamp"/>

</docSimilar>

There are many kinds of constraints. The kind can be decomposed into a **type** and an optionals **facet**. Examples: FileSizeEq, ValueLength, ValueDistinct, ValuePairCount1, ValuePairMinCount2. Constraints are represented by the content of **constraint elements**. A constraint element has a name equal to a constraint type, and attributes and child elements representing one or more constraints of this type and with different facets. Nodes can be shared by some or all constraints. In the following example, each attribute represents a constraint parameter used by one or more constraints:

<valuePair expr1XP="/project/@minDate" count1="1"

expr2XP="//milestoneDate" minCount2="3"

cmp="le" useDatatype="date"/>

Constraint element: <valuePair>

Constraint parameters

ValuePairCount1 constraint: @expr1XP, @expr2XQ, @count1

ValuePairMinCount2 constraint: @expr1XP, @expr2XQ, @minCount

ValuePairCmp constraint: @expr1XP, @expr2XQ, @cmp, @useDatatype

Parameters are usually atomic, but there are also complex parameters. Example: skipItem parameters of a docSimilar constraint, represented by a child element with attributes:

<docSimilar linkName="referenceDoc">  
 <ignoreValue kind="attribute" localName="eq ne lt le gt ge"/>  
</docSimilar>

### Shapes

A shape is a container for two things: a set of constraints and a target declaration:

* The shape is represented by a <file> or <folder> element
* The constraints by child elements (or descendants) of the element
* The target declaration by attributes of the shape element. Example:

<file foxpath="airports.xml">

<fileDate gt="2020-01-01"/>

<links exprXP="//@href"/>

</file>

In this example, the shape is represented by the <file> element, the target declaration is given by the @foxpath attribute, and the constraints by the child elements of <file>.

### Target declaration

The target declaration may take several different forms – it is not necessarily a Foxpath expression. Here is a different example:

<file exprXP="/\*/(xs:include, xs:import)/@schemaLocation"

recursive="true">

<value exprXP="/xs:schema/xs:redefine" empty="empty"/>

</file>

This target declaration selects the documents obtained by recursively resolving URIs found at locations identified by an XPath. Independent of the kind of target declaration, the basic principle is that the declaration is evaluated repeatedly, in the context of each resource selected by the parent shape. Consider:

<domain path="\projects\abc-service">

<folder foxpath=".\\xsd-\*">

<file foxpath="\*.xsd">

<file exprXP="/\*/(xs:include, xs:import)/@schemaLocation" recursive="true">

<value exprXP="/xs:schema/xs:redefine" empty="empty"/>

</file>

</file>

</folder>

</domain>

Follow the trail of selection:

* The target declaration of the folder shape selects all folders xsd-\* under the domain folder.
* *In each of these* folders, all files \*.xsd selected.
* *For each of these* files, all directly or indirectly imported or included XSDs are selected.

The innermost <file> shape is a child of another <file> shape. There is nothing surprising about that. Keep in mind: parent - child relationships between shapes do not mean that their target resources have a parent - child relationship; it means that the target declaration of the child shape is re-evaluated in the context of each single resource in the target of the parent shape.

### Link definitions

Target declarations are essentially mappings – they map a resource from the target of the parent shape to a set of resources added to the target of the current shape. Such mappings are also required by **pair-based constraints** – constraint types designed to be applied to a pair of resources. An example is the DocSimilar constraint type. Consider:

<docSimilar linkName="reference-response">

<ignoreValue kind="attribute" localName="timestamp"/>

</docSimilar>

A DocSimilar constraint checks whether a given resource has content which is similar to the content of another resource. Namely, the target resource is validated by comparing its contents to the contents of other resources related to it and selected by a link definition. What is a **link definition**? It specifies the mapping of a given resource called the **context resource** to a set of other resources called **link target resources**. (Distinguish the terms *link target resource* and *shape target resource*, often for short just called *target resource*.)

When used by a pair-based constraint, each link target resource yielded by the link definition forms a pair, and all pairs are checked obtained by combining the resource under validation with each one of the resources obtained from the link definition. This logic is the same in both cases – target declarations and pair-based constraints. Therefore, it is the same set of possibilities available for making target declarations or for selecting constraint targets. Syntactically, it’s the same set of attributes and child elements which can be used by a target declaration and by a pair-oriented constraint.

Link definition may be local – defined by attributes and child elements on the element in need of link targets; or it can be referenced by name. The schema may contain global link defintions which can be referenced by name. Example:

<greenfox greenfoxURI="… "xmlns="http://www.greenfox.org/ns/schema">

<!-- \*\*\* Context variables -->  
 <context>…</context>  
   
 <!-- \*\*\* Named link definitions -->  
 <linkDef name="hrefElems" hrefXP="//\*:href"/>

<!-- \*\*\* Domain and its shapes -->  
 <domain path="…" name="dc2020">

<!-- Shapes omitted -->  
 </domain>  
  
</greenfox>

### Results and Reports

Now let us briefly look closer at the outcome of Greenfox validation. The validation of a file system tree against a Greenfox schema is an evaluation of the file system contents which is composed of a fundamental building block - validation of a *single resource* against a *single constraint*. The execution of such a constraint validation produces an element called a **validation result**,:

resource + constraint = validation-result

In the typical case, the validation of a single resource against a single constraint produces a single validation result. In some well-defined cases, more results can be produced. This is the case when the constraint semantics involve a second resource and the set of relevant resources has more than one member.

The validation result is an element named after a **colour** which signals conformance – red, yellow, green. In the very special case that a validation is only performed in order to assist another validation – namelye.g. for checking a condition – this subordinate role is signaled by a composite colour - whitered, whiteyellow and whitegreen results.

The primary outcome of a file system tree against a Greenfox schema is a collection of validation results, which is mapped to a validation report:

File-system-tree + Greenfox-schema = validation-result+

Validation-result+ + call-parameters = validation-report

Currently, the default report is statistical – it does not expose the validation results themselves. Example:

G r e e n f o x r e p o r t s u m m a r y

greenfox: C:/tt/greenfox/declarative-amsterdam-2020/schema/air03.gfox.xml

domain: C:/tt/greenfox/declarative-amsterdam-2020/data/air

#red: 2 (2 resources)

#green: 41 (4 resources)

--------------------------------------------

| Constraint Comp | #red | #green |

|--------------------------|------|--------

| FileSizeEq ............. | 0 | 1 |

| FolderContentClosed .... | 0 | 1 |

| FolderContentMemberFile | 0 | 1 |

| FolderContentMemberFiles | 0 | 6 |

| FolderContentMinCount .. | 0 | 1 |

| TargetCount ............ | 0 | 1 |

| TargetMinCount ......... | 0 | 2 |

| ValueDatatype .......... | 0 | 3 |

| ValueEq ................ | 0 | 3 |

| ValueItemsDistinct ..... | 1 | 2 |

| ValueLt ................ | 1 | 2 |

| ValueMatches ........... | 0 | 3 |

| ValueMinCount .......... | 0 | 15 |

--------------------------------------------

Red resources:

F C:/declarative-amsterdam-2020/data/air/airports/index/airports-denmark.xml (ValueItemsDistinct)

F C:/declarative-amsterdam-2020/data/air/airports/index/airports-ireland.xml (ValueLt)

There are two red results – in order to see these red elements, repeat the call with option –r (for “red”), a report type providing all red validation results, grouped by resource:

<gx:validationReport … reportType="red" reportMediatype="application/xml">

<gx:redResources count="2">

<!--

\*\*\* C:/tt/greenfox/declarative-amsterdam-2020/data/air/airports/index/airports-denmark.xml

-->

<gx:redResource file="C: /declarative-amsterdam-2020/data/air/airports/index/airports-denmark.xml">

<gx:red msg="IDs not distinct"

constraintComp="ValueItemsDistinct"

constraintPath="gx:values[1]/gx:value[5]/@distinct"

resourceShapePath="/gx:greenfox[1]/gx:domain[1]/gx:folder[1]/gx:file[1]"

resourceShapeID="file\_2" distinct="true" valueCount="31" exprLang="xpath"

expr="//airport/@id" quantifier="all">

<gx:value nodePath="/airportsForCountry[1]/airport[1]/@id">607</gx:value>

<gx:value nodePath="/airportsForCountry[1]/airport[2]/@id">607</gx:value>

</gx:red>

</gx:redResource>

<!--

\*\*\* C:/ declarative-amsterdam-2020/data/air/airports/index/airports-ireland.xml

-->

<gx:redResource file="C:/declarative-amsterdam-2020/data/air/airports/index/airports-ireland.xml">

<gx:red msg="Airport too high"

constraintComp="ValueLt"

constraintPath="gx:values[1]/gx:value[2]/@lt"

resourceShapePath="/gx:greenfox[1]/gx:domain[1]/gx:folder[1]/gx:file[1]"

resourceShapeID="file\_2" lt="1000"

useDatatype="integer"

valueCount="3"

exprLang="xpath"

expr="//altitude"

quantifier="all">

<gx:value nodePath="/airportsForCountry[1]/airport[6]/geo[1]/altitude[1]">1319</gx:value>

<gx:value nodePath="/airportsForCountry[1]/airport[16]/geo[1]/altitude[1]">1001</gx:value>

</gx:red>

</gx:redResource>

</gx:redResources>

</gx:validationReport>

A glance suffices to understand that validation results are very fine-grained structured information. A central goal of Greenfox is to ensure access to finest-grained information about the state of the system under investigation.

The evaluation of validation reports may be facilitated by the possibility to filter the constraint types included in the report (option –F). Selection can be very fine-grained, using inclusive and exclusive name filters.

By now you know the basic concepts of Greenfox and are ready to take a look at several important topics.

## Further important topics

If you want to achieve a basic understanding of Greenfox, you are facing two main tasks:

* Familiarize yourself with key concepts and principles
* Get a cursory overview of the available constraint types

You already got an overview of the basic building blocks of validation intput and output: resources, constraints, shapes, target declarations, link definitions, results and reports. Building on this ground, we now proceed to learn about a set of important concepts also required for having an understanding of the potential and limitation of Greenfox. To know:

* What is the role of **expression languages**, which expression languages are used?
* The determination of **evaluation context**, and the use of **focus node elements**
* The dealing with **non-XML mediatypes**
* The **context variables** of a Greenfox schema
* Usage, syntax and semantics of **link definitions**
* A few **syntax principles**

### Expression languages

These expression languages are supported:

* Foxpath
* XPath
* NodePath
* LinesPath

#### Foxpath

Foxpath is an extended version of XPath 3.0, supporting file system navigation, node tree navigation and mixing the both within an expression. This makes it a tool for solving tasks of file system navigation with the ease and elegance you are used to from XPath. As Foxpath supports both, file system and node tree navigation, it uses two step separators, the slash (separating steps of conventional path expressions) and backslash – separating steps of file system navigation. A few examples give you an impression. Note that you can try out the examples yourself when checking out the standalone Foxpath project () and adapt the path leading to declarative-amsterdam-2020. Any linefeeds in the examples below have been added for readability and must not be used on the command-line.

/tt/greenfox/declarative-amsterdam-2020/data/air/airports/index/\*

*Result: all files and folders in the index folder*

/tt/greenfox/declarative-amsterdam-2020/data/air/airports/index/\*[is-file()]

*Result: all files in the index folder*

/tt/greenfox/declarative-amsterdam-2020/data/air/airports/index/\*[is-file()][file-size() eq 0]

*Result: as before, but only files whith are empty*

/tt/greenfox/declarative-amsterdam-2020/data/air/airports/index/\*xml[not(doc-available())]

*Result: as before, but only XML files which are not well-formed*

\tt\greenfox\declarative-\*\data\\airports\index\\*.xml[/airportsForCountry]

*Result: as before, but only XML files with a root element <airportsForCountry>*

\tt\greenfox\declarative-\*\data\\airports\index\\*.xml[.//airport[not(\*)]]]

*Result: as before, but only XML files containing empty <airport> elements*

\tt\greenfox\declarative-\* \data\\airports\index\\*.xml[/airportsForCountry

[.//latitude[xs:decimal(.) lt 10]]]

*Result: as before, but only XML files containing a latitude less than 10*

Foxpath can deal with non-XML formats (JSON, CSV, HTML) as if they were XML, parsing them into node trees:

\tt\greenfox\declarative-\*\data\\airports\index\\*.json[jdoc()//latitude[xs:decimal(.) lt 10]]

*Result: JSON documents in the index folder which contain a latitude < 10*

\tt\greenfox\declarative-\*\data\\airports\index\\*.json[jdoc()//airport/count(\_) gt 35]

*Result: JSON documents in the index folder which contain more than 35 airports*

\tt\greenfox\declarative-\*\data\air\resources\airports\\*.csv

[csv-doc()/csv/record[\*[4] = 'Papua New Guinea']]

*Result: CSV documents with a record which holds in the fourth column the value Papua New Guinea*

\tt\greenfox\declarative-\*\data\air\resources\airports\\*.csv

[csv-doc()/csv/record[not(\*[7] castable as xs:decimal)]

*Result: all CSV documents with a record which holds in the seventh column a non-decimal value*

Using CSV files, parameters are available for dealing with non-comma separators and headlines. Example:

fox -b "\tt\greenfox\declarative-\*\data\resources\geo\cow.csv

\csv-doc(., 'semicolon', 'yes')/csv/record/ISOen\_name

[. ne ../ISOen\_proper]/concat(., ';', ../ISOen\_proper)"

*Result: A sorted list of pairs – ISOen\_name # ISOen\_proper – where the two are different (what is rare).*

In Greenfox, you use Foxpath for various purposes:

* As target declarations, selecting the target resources of a shape
* As link definitions, selecting the link target resources of a link definition
* As resource value to be checked against constraints, possibly in combination with other values produced by Foxpath or another expression language

SYNTAX RULE:

Foxpath expressions are contained by attributes with the name suffix FOX, or with the name foxpath.

#### XPath

XPath (version 3.1) is used for the following purposes:

* As resource value to be checked against constraint, possibly in combination with other values produced by XPath or another expression language
* As focus node, shifting the evaluation context for constraints from the document root to inner nodes
* As part of link definitions, when the link context is given by inner nodes of the starting resource, or the link target is inner nodes of the link target resource

SYNTAX RULE:

XPath expressions are contained by attributes with the name suffix XP, or with the name xpath.

#### LinePath

LinePath expressions are XPath expressions evaluated in the context of a document obtained by representing the lines of a text file by <line> elements wrapped in a <lines> element. This enables the use of XPath expressions for evaluating text content which cannot be parsed into “normal” node trees.

SYNTAX RULE:

LinePath expressions are contained by attributes with the name suffix LP.

#### NodePath

NodePath is a deliberately simplistic navigation language used for describing document tree structure. It is used by the DocTree and HyperdocTree constraints.

SYNTAX RULE:

LinePath expressions are contained by attributes with the name suffix NP.

### Evaluation context

When expressions are evaluated, it is crucial to have a clear understanding of the evaluation context.

#### Foxpath context

In Greenfox, the evaluation context of a Foxpath expression is always the *resource URI* of the relevant resource – it is *not* a node from resource contents. When used as part of constraint validation, the relevant resource is typically the resource being validated, but it can also be a second resource involved in the evaluation. When used as part of resolving a link definition, the relevant resource is the link context resource.

#### XPath context

In Greenfox, the evaluation context of an XPath expression is most often a document root node, but it can also be an inner node. However, when the rule is when the expression is used by a constraint which is child of a focus node declaration, the evaluation context is a node selected by the focus node declaration. When the expression plays the role of an “expression #2” (in constraints using a pair of expressions), constraint parameters control whether the context is the document root (or focus node) of the relevant resource, or an item returned by the first expression. When used as part of a link definition, the context is either the document node or a focus node of the link context resource.

The following example illustrates the effect of a focus node declarations:

<file foxpath=".\\geo.xml'">

<focusNode xpath=".//continent"> <!-- Visit continents -->

<!-- Put here: continent checks -->

<focusNode xpath=".//country"> <!-- Within the continent: visit countries -->

<!-- Put here: country checks -->

<focusNode xpath=".//province"> <!—Within the country: visit province elements -->

<!-- Put here: pProvince checks -->

</focusNode>

</focusNode>

</focusNode>

</file>

A <focusNode> element selects nodes from the target documents of the containing shape. Constraints contained by a <focusNode> element use those focus nodes as evaluation context of their XPath expressions, rather than the document node.

### Dealing with non-XML mediatypes

Greenfox aims to make XPath-based evaluation of non-XML resources as easy as possible. JSON, CSV and HTML documents are made accessible to XPath by just adding mediatype attributes to the resource shape or Link Defiitions. Example:

* (under construction)

### The context element of a Greenfox schema

(under construction)

### Usage, syntax and semantics of link definitions

(under construction)

### A few syntax rules and patterns

(under construction)

## Constraint types

Having acquired an idea of the basic building blocks and concepts of Greenfox, the next thing to do is familiarize yourself with the major constraint types available (currently 19). The following table summarizes these types, including information whether the constraint can be used for folders or files only, whether it considers the resource in isolation or in the context of other resources, and whether the constraint is concerned with resource properties or resource contents.

**Table.** The constraint types supported by Greenfox. For each constraint type, a varying number of constraint facets is available.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Element** | **File (F) or Folder (D)** | **Isolated (I) or in Context (C)** | **Resource properties (P) or content (C)** |
| FileDate | <fileDate> | F, D | I | P |
| FileName | <fileName> | F, D | I | P |
| FileSize | <fileSize> | F, D | I | P |
| FolderContent | <folderContent> | D | I | C |
| Mediatype | <mediatype> | F | I | C |
| DocTree | <docTree> | F | I | C |
| HyperdocTree | <hyperdocTree> | F, D | C | C |
| XsdValid | <xsdValid> | F | I | C |
| Value | <value> | F | I | C |
| ValuePair | <valuePair> | F | I | C |
| Foxvalue | <foxvalue> | F, D | C | C |
| FoxvaluePair | <foxvaluePair> | F, D | C | C |
| ValueCompared | <valueCompared> | F | C | C |
| FoxvalueCompared | <foxvalueCompared> | F, D | C | C |
| DocSimilar | <docSimilar> | F | C | C |
| FolderSimilar | <folderSimilar> | D | C | C |
| Link | <links> | F, D | C | - |
| TargetSize | <targetSize> | F, D | C | - |
| Conditional | <conditional> | F, D | - | - |

### Constraint types - examples

The folder demo-constraints contains for each constraint type one or several example schemas. Note that the examples are not meant to give a comprehensive overview of the possibilites. Rather, they should give a feeling what can be achieved using that constraint type, and how using the constraint type looks.

### Outlook

Bla