# Greenfox – user manual

This document is a manual describing the use of the greenfox validation language.

*The document is under construction. Frequent updates are to be expected. We apologize for the inconvenience.*

*If a part of the document especially important for you is still incomplete, we encourage you to raise an issue, which will be taken into account when deciding on priorities.*

## What is a greenfox schema?

A **greenfox schema** is a set of conditions constraining a **file system tree**. A file system tree is understood as a file system folder and all folders and files directly or indirectly contained by it. The file system tree can be **validated** against the schema, using a **greenfox processor**. The result of validation is a **validation report**. A validation report indicates **conformance** - whether the file system tree conforms to the schema - and it supplies **validation results**. Each validation result describes the outcome of validating a single resource against a single constraint.

## Implementation

An implementation of a greefox processor is included in the github repository:

<https://github.com/hrennau/greenfox>

The implementation is countained in the bin folder of the repository.

The further text refers to this implementation as the **reference implementation**.

## Usage

The usage information given here refers to the reference implementation included in this repository. The implementation is written in XQuery. Due to its use of BaseX extension functions, it must be executed by the **BaseX** XQuery processor. General usage pattern:

basex –b "request=val?gfox=path/to/schema

[,domain=/path/to/domain]

[,reportType=report-type]

[,params=context-values]

where:

path/to/schema –

name, relative or absolute path of a greenfox schema

/path/to/domain –

the absolute path to the filesystem tree root folder

report-type –

identifies the type of validation report; default value: redTree; available values:

* white – all validation results, no tree structure
* red – only red and yellow results, no tree structure
* whiteTree – all validation results, grouped by resource
* redTree – only red and yellow results, grouped by resource

context-values –

semicolon-separated list of name-value pairs, used by the schema as context parameters

Only the gfox parameter is mandatory. If domain is not specified, the file system tree to be validated is identified within the schema. The default report-type is redTree.

In all examples listed below, after copy-pasting please remove linefeeds.

Example 1: Validate the file system tree identified within the schema identified by gfox.

basex -b "request=val?**gfox=**/tt/greenfox/example-schemas/case-studies/system-s.gfox.xml

/tt/greenfox/bin/greenfox.xq

Example 2: Validate the file system tree identified by domain against the schema identified by gfox.

basex -b "request=val

?**gfox=**/tt/greenfox/example-schemas/case-studies/system-s.gfox.xml,

**domain=**/tt/greenfox/example-data/system-s"

/tt/greenfox/bin/greenfox.xq

Example 3: As example 1, but produce a validation report of type whiteTree.

basex -b "request=val

?**gfox=**/tt/greenfox/example-schemas/case-studies/system-s.gfox.xml,

**domain=**/tt/greenfox/example-data/system-s,

**reportType=**whiteTree"

/tt/greenfox/bin/greenfox.xq

Example 4: As example 2, but setting the context parameter lastModified to the value “2020-02-15”.

basex -b "request=val

?**gfox=**/tt/greenfox/example-schemas/case-studies/system-s.gfox.xml,

**domain=**/tt/greenfox/example-data/system-s,

**reportType=**whiteTree,

**params=lastModified=**2020-02-15"

/tt/greenfox/bin/greenfox.xq

## Fundamentals

This section provides some basic information and concepts required for reading and writing greenfox schemas, and for understanding validation results.

### Data model

The greenfox language is based on the XDM datamodel. Any values used for representing or constraining a resource are represented by XDM values (XDM 3.0).

### Resources

A file system is thought of containing two kinds of resources: **folders** and **files**.

*Note. The raw output of a validation episode (*[*white validation report*](#_White_validation_report)*) is composed of elementary units (*[*validation results*](#_Validation_results)*) which describe the result of validating a single resource against a single* [*constraint*](#_Constraints)*.*

### Resource properties and values

Constraints can refer to *resource properties* and *resource values*.

File system resources are thought of as having a few intrinsic properties, called **resource properties**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Short name** | **Meaning** | **Applicable**  **to files** | **Applicable**  **to folders** |
| File name | Name of the resource | + | + |
| File size | File size as number of bytes | + | - |
| Last modification | Timestamp of the last modification | + | + |
| Mediatype | Format of file contents – constrained to be one of: XML, HTML, JSON, CSV, text, binary | + | - |
| XSD valid | Boolean flag, indicating whether file contents are valid against a given XSD | + | - |
| Folder content | Sequence of child resource names | - | + |

Resources can also be described by **resource values**, which are values obtained by “applying” an expression to it. The meaning of “applying” depends on the expression language used, and it must be precisely defined for each expression language used.

The current version of greenfox supports two expression languages: **XPath 3.1** and **foxpath 3.0**. In both cases, an expression is *applied* to the resource when the **initial context item** is set to one of the following:

1. the file system path of the resource
2. the root node of an XDM node tree representing the resource
3. an item from a resource value of the resource

*Note. The third possibility is a recursive definition which allows a resource value to be obtained by a chain of expressions, where the first one is evaluated in the context of (1) or (2) and the remaining expressions are evaluated in the context of an item of the preceding expression.*

*Example 1:* Given a file containing a serialized XML document. A resource value can be defined as the value of the XPath expression

count(//airport )

The resource value would be an integer number expressing the number of <airport> elements contained by the document.

*Example 2:* Given a file containing a serialized XML document. A resource value can be defined as the value of the XPath expression

//airport

The resource value would be a sequence of zero or more <airport> elements. This example demonstrates that a resource value can consist of XDM nodes.

*Example 3:* Given a file containing a serialized XML document. A resource value can be defined as the value of the XPath expression

//@schemaLocation/resolve-uri(., ../base-uri(.))!doc(.)

The resource value would be a sequence of document nodes obtained by resolving the schema location URIs. This example demonstrates that resource values of a file may be XDM nodes found in *other* files.

*Example 4:* Given a folder. A resource value can be defined as the value of the foxpath expression:

.\\\*.json

The value would be a sequence of file system paths, belonging to JSON files directly or indirectly contained by the given folder.

*Example 5:* Given a folder. A resource value can be defined as the value of the foxpath expression:

..\\\*.json

The value would be a sequence of file system paths, belonging to JSON files directly or indirectly contained by a *sibling* folder of the given folder. This example demonstrates that resource values of a folder may represent resources not contained by the folder.

*Example 6:* Given a folder. A resource value can be defined as the value of the foxpath expression:

..\\\*.json\jdoc(.)//definitions/\*

The value would be a sequence of XDM nodes, belonging to the XML representations of JSON files directly or indirectly contained by a sibling folder of the given folder. This example demonstrates that also folders (not only files) may have resource values which are XDM nodes, and that these XDM nodes may be found in files both inside and outside of the current folder.

### Resource shapes and targets

A **resource shape** is a set of constraints which apply to a set of resources.

A **file shape** is represented by a <file> element and describes a set of file resources.

A **folder shape** is represented by a <folder> element and describes a set of folder resources.

A **target declaration** is represented by attributes on the shape element (<file> or <folder>).

The following table summarizes the possible forms of target declarations.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Target kind** | **Examples** |
| @path | Literal path. Wildcards are not allowed | resources  resources\xsd |
| @foxpath | Foxpath expression. | resources  resources\xsd  resources\\xsd-\*  resources\\xsd-\*[ota\*.xsd] |
| @linkXPath | Link target. The attribute value is an XPath expression producing the file paths of the link values | /xs:schema/xs:include/@schemaLocation  //@schemaLocation |
| @recursiveLinkXPath | Recursive link target. The attribute value is an expression which is recursively applied to the documents obtained by resolving the links. | /xs:schema/xs:include/@schemaLocation  //@schemaLocation |

### Value shapes

A **value shape** defines a resource value and a set of constraints which apply to the value.

An **xpath shape** defines the resource value as the value of an XPath expression.

A **foxpath shape** defines the resource value as the value of a foxpath expression.

A **link shape** defines the resource value as a set of file URIs derived from the value of an XPath expression.

### Constraints

Bla

### Validation report

Bla

#### White validation report

Bla

#### Final validation report

bla

### Validation results

Bla

### Expression languages

A key feature of greenfox is the use of expressions as building blocks …

* Selecting resources to which a given set of constraints applies
* Defining resource values to which a given set of constraints applies
* Defining constraints

A basic greenfox processor must support XPath expressions, XPath version 3.1.

An advanced greenfox processor must support also foxpath expressions, foxpath version 3.0.

For more information about foxpath see here: <https://github.com/hrennau/foxpath>

## Schema structure

Bla

## Main building blocks

Bla

### Domain shape

Bla

### Folder shape

Bla

### File shape

Bla

### Resource value shape

Bla

## Further building blocks

Bla

### Validation context

Bla

### Focus node

Bla

### Extension component

Bla

## Basic constraints

Bla

### Target size

Bla

### Resource properties

Bla

#### File name

A set of constraints referring to the resource name, which is either a file name or a folder name. The constraints are represented by attributes on a <fileName> element.

Example:

<fileName notLike="\*test\*"

notLikeMsg="File names should not contain string 'test'"/>

Every constraint is represented by a mandatory parameter, as well as several optional parameters. The following table summarizes the available constraints.

**Table: Constraint components checking the resource name (FileName\*).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Constraint component** | **Main parameter** | **Optional parameters** | **Condition** | **Notes** |
| FileNameEq | @eq | @case | The file name is equal to @eq | - |
| FileNameNe | @ne | @case | The file name is not equal to @ne | - |
| FileNameLike | @like | @case, @flags | The file name matches the regex derived from the parameter value | See “Glob valu  es” for details about the regex used |
| FileNameNotLike | @notLike | @case, @flags | The file name does not match the regex derived from the parameter value | See “Glob valu  es” for details about the regex used |
| FileNameMatches | @matches | @case, @flags | The file name matches the regex given by @matches | - |
| FileNameNotMatches | @notMatches | @case, @flags | The file name does not match the regex given by @notMatches | - |

The following table explains optional parameters.

**Table. Optional parameters of constraint components checking the resource name (FileName\*).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Meaning** | **Default** | **Constraint components** |
| @case | xs:boolean | If true, the name is checked case-sensitively | False | FileName\* |
| @flags | xs:string | Flags used when applying a regulare expression | Empty string | FileNameLike  FileNameNotLike  FileNameMatches  FileNameNotMatches |
| @eqMsg  @neMsg  @likeMsg  @notLikeMsg  @matchesMsg  @notMatchesMsg | xs:string | A message included in the validation result in case of a constraint violation | A string reporting the violation | The constraint component identified by the substring preceding the suffix “Msg” |
| @eqMsgOK  @neMsgOK  @likeMsgOK  @notLikeMsgOK  @matchesMsgOK  @notMatchesMsgOK | xs:string | A message included in the validation result in case of successful validation against the constraint | Empty string | The constraint component identified by the substring preceding the suffix “MsgOK” |

The validation result contains the following properties.

**Table. The properties of a validation result for constraint components checking the resource name (FileName\*).**

|  |  |  |
| --- | --- | --- |
| **Result property** | **XML representation** | **Meaning** |
| File path | @filePath | Absolute file path, identifies the file or folder within the file system |
| Resource shape | @resourceShapeID | Identifies the resource shape declaring the constraint |
| Constraint component | @constraintComponent | Identifies the constraint component; the value matches the pattern FileName\* |
| Constraint | @constraintID | Identifies the constraint |
| Comparison | @eq, @ne, @like, @notLike, @matches, @notMatches | The name identifies the kind of comparison, the value the value with which to compare or which to match |
| Case | @case | If true, comparison was carried out case-sensitively |
| Flags | @flags | Flags used when matching the name (only used for constraint components \*Like, \*NotLike, \*Matches, \*NotMatches) |
| Value | gx:value | The file name (only if the constraint is violated) |

#### File size

Bla

#### Last modification time

Bla

#### Mediatype

Bla

### Folder content

Bla

### Links resolvable

Links can be checked so that only such links are valid as can be resolved to a resource. The constraint is represented by a <links> element and its attributes. The element can only be used as child of a <file> element.

The links are identified by an XPath expression (@xpath). An optional @mediatype attribute constrains the link targets to have a mediatype dependent on the attribute value. Valid values and their meaning:

xml – the resource contains well-formed XML

json – the resource constains well-formed JSON

test – the resource can be retrieved using unparsed-text()

binary – only the file existence is checked

If no @mediatype is specified, only the file existence is checked.

***Example***

<links

xpath="//@fileref"  
 msg="Found 'fileref' links which cannot be resolved."  
 minCount="1"  
 minCountMsg="At least one fileref expected"/>

***Validation result values***

In case of constraint violation, every item in the value of @xpath is reported by a <gx:value> element. The reported value is always the string value of the item.

***Constraint component signature***

Constraint component name: LinksResolvable

Constraint parameter:

* linkExpression - @xpath (mandatory)
* linkMediatype - @mediatype (optional)

Message attributes:

* @msg
* @msgOK

### Link size

These are “collaterate constraints” which may accompany a *Links resolvable* constraint. They are declared by additional attributes on the <links> element declaring the *Links resolvable* constraint. These attributes are:

* @minCount – the expression value must have at least the given number of items
* @maxCount – the expression value must have at most the given number of items
* @count – the expression value must have exactly the given number of items

***Constraint component signature***

Constraint component name: LinksMinCount

Constraint parameters:

* minCount (@minCount)

Message attributes:

* @minCountMsg
* @minCountMsgOK

***Constraint component signature***

Constraint component name: LinksMaxCount

Constraint parameters:

* maxCount (@maxCount)

Message attributes:

* @maxCountMsg
* @maxCountMsgOK

***Constraint component signature***

Constraint component name: LinksMaxCount

Constraint parameters:

* count (@count)

Message attributes:

* @countMsg
* @countMsgOK

### Schema valid

Bla

#### XSD valid

Bla

#### JSON Schema valid

This constraint component has not yet been implemented. It can be used to check whether a file conforms to a JSON Schema. The JSON Schema will be provided by a file identified by a constraint parameter.

#### SHACL valid

This constraint component has not yet been implemented. It can be used to check whether a file contains a data graph which conforms to a SHACL shapes graph. The SHACL shapes graph will be provided by a file identified by a constraint parameter.

## Expression based constraints

Bla

### Generic constraint parameters

Bla

#### quant – all items or some items

Bla

#### useDatatype – datatype used for comparison

bla

### Datatype

Bla

### Value items unique

Checks that the value items are unique. More precisely, there must not be two items whose string value is equal.

### Value items count

Bla

#### minCount – at least … items

bla

#### maxCount – at most … items

bla

#### count – exactly … items

bla

#### exists – at least one item

If the the value is "true", the value must have one or more items. If the value is "false", the value must be empty.

#### empty – value is empty

If the the value is "true", the value must be empty. If the value is "false", the value have one or more items.

### String length

The value items must have a string length equal to the value of @length. Before measuring the string length, any node items are atomized. Items which are maps or arrays are treated as violating the constraint.

#### minLength – length greater or equal

The value items must have a string length greater than or equal to the value of @minLength. Before measuring the string length, any node items are atomized. Items which are maps or arrays are treated as violating the constraint.

#### maxLength – length less or equal

The value items must have a string length less than or equal to the value of @maxLength. Before measuring the string length, any node items are atomized. Items which are maps or arrays are treated as violating the constraint.

#### length – length equal

The value items must have a string length equal to the value of @length. Before measuring the string length, any node items are atomized. Items which are maps or arrays are treated as violating the constraint.

### Comparison

Bla

#### eq – equal to

The string value of the items must be equal to the value of @eq. Items which are maps or arrays are treated as violating the constraint.

#### ne - not equal to

The string value of the items must not be equal to the value of @eq. Items which are maps or arrays are treated as violating the constraint.

#### gt - greater than

Bla

#### ge - greater or equal

Bla

#### lt – less than

Bla

#### le - less or equal

Bla

#### matches – matches a regular expression

Bla

#### notMatches – does not match a regular expression

Bla

#### like – matches a Glob pattern

Bla

#### notLike – does not match a Glob pattern

Bla

#### in – value items in a value list

Bla

#### notin – value items not in a value list

Bla

#### contains – value items contains given items

Bla

### Relating two expressions

Bla

#### eqXPath, eqFoxpath – value equal to an expression value

bla

#### ltXPath, ltFoxpath – value less than an expression value

bla

#### leXPath, leFoxpath – value less than or equal to an expression value

bla

#### gtXPath, gtFoxpath – value greater than an expression value

bla

#### geXPath, geFoxpath – value greater than or equal to an expression value

bla

#### inXPath, inFoxpath – value items contained by an expression value

bla

#### containsXPath, containsFoxpath – value contains another expression value

bla