# 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**. 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.

## Usage

basex –b "request=val?gfox=path/to/schema,domain=/path/to/filesystem-tree-rootfolder

basex –b "request=val?gfox=path/to/schema,domain=/path/to/filesystem-tree-rootfolder,

reportType=report-type

where:

path/to/schema –

name, relative or absolute path of a greenfox schema

/path/to/filesystem-tree-rootfolder –

the absolute path to the filesystem tree root folder

report-type –

identifies the report structure; 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

Examples:

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

[After copy-pasting, please remove linefeed.]

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

[After copy-pasting, please remove linefeed.]

## 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**. Any validation result refers to an individual folder or file. When a folder or file is validated against a constraint, it is called the **focus resource**.

### 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**:

* Last modification time
* File size (only files)
* Mediatype (only files)
* Folder contents (only folders)
* There is a small set of resource proA resource

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 either the file system path of the resource, or an XDM node tree representing the resource. A resource value can be of any kind or type supported by the XDM data model.

*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 one 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(., ..)!doc(.)

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

*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 current folder.

*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 a *sibling* folder of the current folder. This example demonstrates that resource values of a folder may represent resources not contained by the folder.

*Example 5:* 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 current 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.

### Shapes and targets

Bla

### Constraints

Bla

### Validation results

Bla

### Validation report

Bla

### Expression languages

Bla

## 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

#### Last modification time

Bla

#### File size

Bla

#### Mediatype

Bla

### Folder content

Bla

### 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