xsdplus – reference

Version: 2018-03-10

xsdplus is a command-line tool for processing XSDs in various ways, for example generating location trees and treesheets. For related general information see the paper "Location trees enable XSD based tool development" (doc/location-trees.pdf).

The tool is written in XQuery, version 3.1. For the time being, please use the XQuery processor BaseX for executing xsdplus; BaseX can be downloaded from here:

http://basex.org/products/download/all-downloads/

The first part of this document gives general information about the command-line syntax of xsdplus.

The second part of this document explains all operations offered by xsdplus.

NOTE: Not yet all opeations - as it is work in progress, seeral operations are not yet described.

To summarize, xsdplus offers the following kinds of functionality:

- Schema expansion: returns the result of recursively resolving includes and imports found in a set of head schemas
- Transformation: xsd => location trees
- Transformation: xsd => treesheets
- Creation of frequency trees (or treesheets representing their content)
- Creation of values trees (or treesheets representing their content)

Prerequisites

In order to try out xsdplus, please download or clone the repository. The XQuery command-line tool is provided by `$XSDPLUS/bin/xsdplus.xq`

If you have not yet installed the BaseX XQuery processor, do so now.

General command-line syntax

The XQuery command-line tools is provided by `$XSDPLUS/bin/xsdplus.xq`. Using the BaseX XQuery processor, a call of `xsdplus.xq` has the following general structure:

```
basex -b "request=operation?param1=…,param2=…"   /…/xsdplus/xsdplus.xq
```

where

- `operation` identifies the tool operation (e.g. ltree, treesheet, frequencyTree, valuesTree)
- `param1`, `param2`, … are operation-specific parameter names (e.g. xsd, enames, annos)

Operation and parameter names can be abbreviated, as long as the prefix used is long enough to identify exactly one operation or parameter unambiguously. Operation and parameter names can be entered in any mixture of upper and lower case.

While the set of available parameters is in principle operation specific, the XSDs to be processed are always specified by a parameter named `xsd`. The parameter type is `docFOX`, which means that the value can be a FOXpath expression identifying one or more XSDs. See section "Paramter type `docFOX`" for details about paramter syntax and semantics.

Some parameters have the type `nameFilter` – e.g. the parameters `enames`, `tnames` and `gnames` of operiaton `ltree`. See section "Parameter type `nameFilter`" for details about syntax and semantics.

Parameters typed `xs:boolean` can be specified using an abbreviated syntax (see section "Boolean parameters").

Parameter types

Parameter type `docFOX`

The parameter value is a FOXpath expression. Such expressions can select file system resources with an XPath-like syntax. A general introduction to the FOXpath language is given in doc/foxpath-into.pdf. Examples:

/xsdbase/ota/OpenTravel_2013A_XML/OTA_AirAvailRQ.xsd
/xsdbase/ota//OTA_AirAvailRQ.xsd
/xsdbase/ota//*airavail*.xsd
/xsdbase/ota//(*avail* except *pkg*)
/xsdbase/ota//*avail*[not(self~::*pkg*)]

Parameter type `nameFilter`

The parameter value is a whitespace-separated sequence of name patterns. A *name pattern* is a string of literal characters and/or wildcards. There are two kinds of wildcards: "*" matches a sequence of zero or more arbitrary chacters, and "?" matches exactly one arbitrary character. If the name pattern is not preceded by "~", the pattern selects all names matching the pattern; if the pattern is preceded by "~", the pattern is "negative", that is, it selects all names *not* matching the name pattern.

A namePattern can have a postfix modifying the evaluation of the pattern:
Postfix: #s – case sensitive (default is case insensitive)
Postfix: #r – the pattern is interpreted as a regular expression, rather than a pattern

Examples:

- "*" - matches any name
- "*rq" – matches any name ending with "rq"
- "~*test*" – matches any name not containing the string "test"
- "*rq ~*test*" – matches any name ending with "rq" and not containing the string "test"
- "*RQ#c" – matches any name ending with "RQ", case sensitively
- "*\d{3}#r" – matches any name ending with three digits

Boolean parameters

For parameters types `xs:boolean`, an abbreviated syntax is available: `paramname` is short for `paramname=true`; `~paramname` is short for `paramname=false`.

Operation `load` - expanding schemas

Summary

Returns the result of recursively expanding all includes and imports found in a set of head schemas. The return value is a <z:schemas> element containing a sequence of <xs:schema> elements, which are the provided head schemas as well as all schemas discovered by recursively resolving includes and imports. By default, any chameleon schemas are transformed into schemas with a target namespace equal to the target namespace of the including schema. If parameter $retainChameleons is used, transformation of chameleons is suppressed.

Usage

Operation: load

Parameters:

- xsd – a FOXpath expression specifying one or more XSDs
- retainChameleons – boolean flag indicating that chameleon schemas are retained unchanged, rather than being transformed into a schema with a target namespace equal to the target namespace of the including schema; default = false

Examples

```
basex  -b "request=load?xsd=/xsdbase/ota-2017//*airavail*.xsd"   /tt/xsdplus/xsdplus.xq
```

```
… "request= load?xsd=/xsdbase/ota-2017//*airavail*.xsd,retaincham" …
```
(retain chameleon schemas unchanged)

Concepts

Cyclic includes and imports (e.g. `a.xsd` imports `b.xsd` imports `c.xsd` imports `a.xsd`) are recognized and ignord during expansion.

A *chameleon schema* is a schema without target namespace which is included by a schema with a target namespace. During validation, it is used as if it had a target namespace equal to the including schema's target namespace. Unless parameter $retainChameleons is used, the `load` operation result does not represent chameleon schemas in their original form, but transformed into equivalent schemas with a target namespace equal the including schema's target namespace.

Operation `ltree` - generating location trees

Summary

Creates location trees for a set of elements, or a set of types, or a set of groups. The contents of the location tree can be reduced by several parameters (propertyFilter, stypeTrees, annos). All location trees are delivered as `z:locationTree` elements wrapped in a `z:locationTrees` root element.

Usage

Operation: ltree

Parameters:

- xsd – a FOXpath expression specifying one or more XSDs
- enames – a name filter selecting element names for which location trees are requested
- tnames – a name filter selecting type names for which location trees are requested
- gnames – a name filter selecting group names for which location trees are requested
- global – boolean flag indicating if the elements selected by *enames* must be top-level elements; default: true
- propertyFilter – a name filter selecting property attributes to be included; if the parameter is not used, all property attributes are included
- stypeTrees – boolean flag indicating whether the location tree should contain stypeTrees; default = true
- annos – boolean flag indicating whether the location tree should contain annotation elements; default = true

Examples

basex -b "request=ltree?xsd=/xsdbase/ota//*airavail*,enames=*rq" /tt/xsdplus/xsdplus.xq

… "request=ltree?xsd=/xsdbase/ota//*airavail*,enames=*rq,~stype" …
(suppress simple type trees)

… "request=ltree?xsd=/xsdbase/ota//*airavail*,enames=*rq,~annos" …
(suppress annotations)

… "request=ltree?xsd=/xsdbase/ota//*airavail*,enames=*rq,~stype,~annos,prop=occ" …
(suppress simple type trees and annotations, suppress all location properties except for z:occ))

Concepts

Location trees can be regarded as document models obtained from a set of XSDs. More generally, they provide a tree-structured representation of complex content, which can be identified by an element declaration, a type definition or a group definition. Location trees are intended to be used as intermediaries, which are transformed into artifacts of interest.

Operation `treesheet` - generating treesheets

Summary

Creates treesheets for a set of elements, or a set of types or a set of groups. The kind of item reports shown on the right-hand side of the treesheet is controlled by parameter *report*.

Usage

Operation: treesheet

Parameters:

- xsd – a FOXpath expression specifying one or more XSDs
- enames – a name filter selecting element names for which location trees are requested
- tnames – a name filter selecting type names for which location trees are requested
- gnames – a name filter selecting group names for which location trees are requested
- global – boolean flag indicating if the elements selected by *enames* must be top-level elements; default: true
- report – specifies the type of item reports to be displayed on the right-hand side of the treesheet; available values:
    - tname – type names
    - tdesc – simple type descriptors
    - anno – schema annotations
- lang – if parameter *report* = “anno”, the annotation language; if annotation in this language are available, these are displayed; otherwise english annotations, if available; otherwise the first annotation available, regardless of its language

Examples

```
basex  -b "request=treesheet?xsd=/xsdbase/ota//*airavail*,enames=*rq, report=tdesc"
                        /tt/xsdplus/xsdplus.xq
          (treesheet displaying type descriptions)

… "request=treesheet?xsd=/xsdbase/ota//*airavail*,enames=*rq, report=tname"
          (treesheet displaying type names)

… "request=treesheet?xsd=/xsdbase/ota//*airavail*,enames=*rq, report=anno"
          (treesheet displaying annotations)

… "request=treesheet?xsd=/xsdbase/ota//*airavail*,enames=*rq,report=anno, lang=fr"
          (treesheet displaying French annotations)
```

Concepts

A treesheet is a document model report. It is a text file with a left-hand side representing the document tree structure and a right-hand side containing item reports.

Operation `frequencyTree` - generating frequency trees

Summary

Creates frequency trees for a set of instance documents, or a set of instance document elements, described by a set of schemas.

Usage

Operation: frequencyTree

Parameters:

- xsd – a FOXpath expression specifying one or more XSDs
- doc – a FOXpath expression specifying one or more XML documents (described by the XSDs)
- format – if "xml", an XML report is returned, if "treesheet", a treesheet is returned; default: treesheet
- rootElem – if specified, the frequency trees are not created for the document roots, but for elements found in the documents with a name equal to the parameter value

Examples

basex –b "request=freq?xsd=gateway.xsd, doc=msgs/*.xml "   /tt/xsdplus/xsdplus.xq

… "request=freq?xsd=gateway.xsd, doc=msgs/*.xml, format=treesheet" …
        (frequencyTree represented as treesheet)

… "request=freq?xsd=gateway.xsd, doc=msgs/*.xml, format=xml" …
        (frequencyTree represented as tree)

… "request=freq?xsd=gateway.xsd,doc=msgs.xml,rootElem=BestPackageOfferForHotelRQ" …
        (frequencyTree is created for the elements in msgs.xml with the name
        "BestPackageOfferForHotelRQ")

Concepts

A frequency tree is an extended document model, augmenting item locations with information about the fraction of instance documents containing items described by the location. A frequency tree thus shows the actual use of optional items and choice branches.

Operation valuesTree - generating value trees

Summary

Creates values trees for a set of instance documents, or a set of instance document elements, described by a set of schemas.

Usage

Operation: valuesTree

Parameters:

- xsd – a FOXpath expression specifying one or more XSDs
- doc – a FOXpath expression specifying one or more XML documents (described by the XSDs)
- format – if “xml”, an XML report is returned, if “treesheet”, a treesheet is returned; default: treesheet
- rootElem – if specified, the frequency trees are not created for the document roots, but for elements found in the documents with a name equal to the parameter value
- nterms – the maximum number of values shown for an item; default: 5
- inamesTokenize – a nameFilter selecting all items whose values are treated in tokenized form (split at whitespace); if not specified, all values are treated in original form

Examples

basex –b "request=values?xsd=gateway.xsd, doc=msgs/*.xml" /tt/xsdplus/xsdplus.xq

… "request=values?xsd=gateway.xsd, doc=msgs/*.xml, format=treesheet" …
(valuesTree represented as treesheet)

… "request=values?xsd=gateway.xsd, doc=msgs/*.xml, format=xml" …
(frequencyTree represented as tree)

… "request=values?xsd=gateway.xsd, doc=msgs.xml, rootElem=BestPackageOfferForHotelRQ" …
(valuesTree is created for the elements in msgs.xml with the name “BestPackageOfferForHotelRQ”)

… "request=values?xsd=gateway.xsd, doc=msgs/*.xml, format=xml, nvalues=10" …
(maximum number of terms shown for each item is 10, rather than 5, the default)

… “request=values?xsd=gateway.xsd,doc=msgs/*.xml,format=xml, inamesTokenize=*" …
(the values of all items are considered in whitespace-tokenized form)

Concepts

A values tree is an extended document model, augmenting item locations with information about the data values found within a given set of documents in items described by the location.