## Fox functions

Reference documentation of the Foxpath extension functions (“fox functions”).

*[Work in progress]*

## all-descendants

all-descendants($items)

all-descendants()

Returns for each node in $items all descendant elements and their attributes. (Note that the descendant axis of path expressions returns only non-attribute nodes, whereas this function returns elements and attributes. Atomic items in $items are silently ignored, in particular URIs are not resolved to content.

A call without arguments is equivalent to a call with a single argument which is the context item.

**Example**

Get all data paths found in a document or set of documents.

fox "../download/bhub-20210308//product.edmx\all-descendants()\np(.) => frequencies()

## att-lnames

att-lnames($items\*, $nameFilter?, $nameFilterExclude?)

att-lnames()

Returns for each element in $items a sorted, concatenated list of the local attribute names of the element. The attributes to be considered can be filtered by a whitespace separated list of local names or name patterns to be included ($nameFilter), as well as a whitespace separated list of local names or name patterns to be excluded ($nameFilterExclude). The function is often used in combination with frequencies().

A call without parameter is equivalent to a call with a first parameter set to the context item.

Example:

fox -D "../download/bhub-20210308//\*.wsdl\\xs:schema\xs:element\att-lnames() => freq()"

Response:

description, name, prefix ...... (4)

description, name, prefix, type (12)

name ........................... (799)

name, nillable, type ........... (3)

name, type ..................... (1145)

Limiting the analysis to attributes name, type and nillable:

fox -D "../download/bhub-20210308//\*.wsdl\\xs:schema\xs:element\att-lnames(., 'name type nillable') => freq()"

Response:

name ................ (803)

name, nillable, type (3)

name, type .......... (1157)

Limiting the analysis to all attributes *except* name, type and nillable:

fox -D "../download/bhub-20210308//\*.wsdl\\xs:schema\xs:element\att-lnames(., (), 'name type nillable') => freq()"

Response:

................... (1947)

description, prefix (16)

## att-names

att-names($items\*, $nameFilter?, $nameFilterExclude?)

att-names()

Returns for each element in $items a sorted, concatenated list of the lexical attribute names (including prefix) of the element. The attributes to be considered can be filtered by a whitespace separated list of local names or name patterns to be included ($nameFilter), as well as a whitespace separated list of local names or name patterns to be excluded ($nameFilterExclude). The function is often used in combination with frequencies(). For examples see function att-lnames().

A call without parameter is equivalent to a call with a first parameter set to the context item.

## base-dir-name, base-dname, bdname

bdname($node)

bdname()

Returns the name of the folder containing the document containing a given node.

A call without arguments is equivalent to a call with a single argument which is the context item.

**Example**

fox "../data/chemdata/\*[\\text() = 'Oxygen']\bdname()"

**Tip**

Use as argument of hlist-entry(). \_TO\_DO\_ Elaborate!

## base-file-name, base-fname, bfname

bfname($node)

bfname()

Returns the file name extracted from the base URI of a given node.

A call without arguments is equivalent to a call with a single argument which is the context item.

**Example**

fox "../data/chemdata/\*[\\text() = 'Oxygen']\bfname()"

**Tip**

Use as argument of hlist-entry(). \_TO\_DO\_ Elaborate!

## child-jnames

child-jnames($elem, $nosort, $namePatterns, $excludedNamePatterns)

child-jnames()

Returns for a given JSON ($elem) object a concatenated, sorted list of field names, which are the original JSON names, not the names obtained by replacing invalid XML name characters.

A call without parameters is equivalent to a call with a single parameter, which is the context item.

If $nosort is true (“true” or 1), the names are returned in original order, rather than sorted alphabetically.

TO BE CONSIDERED: merge $namePatterns and $excludedNamePatterns into a single filter parameter.

Examples:

fox "/projects/bhub/download/bhub-20210225//(\*.json except (wsdl\*,edmx\*))\\*\info\child-jnames() => f()"

=>

contact, description, license, termsOfService, title, version ..................................... (215)

contact, description, license, termsOfService, title, version, x-document-meta-data ............... (4)

contact, description, license, termsOfService, title, version, x-document-meta-data, x-olp-service (1)

contact, description, license, title, version ..................................................... (7)

contact, description, termsOfService, title, version .............................................. (8)

contact, description, termsOfService, title, version, x-document-meta-data ........................ (18)

contact, description, title, version .............................................................. (34)

contact, description, title, version, x-targetEndpoint ............................................ (2)

description, license, title, version .............................................................. (11)

description, termsOfService, title, version ....................................................... (9)

description, title, version ....................................................................... (531)

description, title, version, x-element-hub, x-element-id, x-element-key, x-element-name ........... (16)

description, title, version, x-targetEndpoint ..................................................... (29)

Only x-\*; show percent, rather than absolute numbers

fox "/projects/bhub/download/bhub-20210225//(\*.json except (wsdl\*,edmx\*))\\*\info\child-jnames(.,'x-\*') => pf()"

=>

.......................................................... (92.1)

x-document-meta-data ...................................... (2.5)

x-document-meta-data, x-olp-service ....................... (0.1)

x-element-hub, x-element-id, x-element-key, x-element-name (1.8)

x-targetEndpoint .......................................... (3.5)

Exclude x-\*:

fox "/projects/bhub/download/bhub-20210225//(\*.json except (wsdl\*,edmx\*))\\*\info\child-jnames(.,(),'x-\*') => pf()"

=>

contact, description, license, termsOfService, title, version (24.9)

contact, description, license, title, version ................ (0.8)

contact, description, termsOfService, title, version ......... (2.9)

contact, description, title, version ......................... (4.1)

description, license, title, version ......................... (1.2)

description, termsOfService, title, version .................. (1.0)

description, title, version .................................. (65.1)

Example: explore how “components” are used by OpenAPI v3 schemas; ignore security\*, head\*, exam\*:

fox "/projects/bhub/download/bhub-20210225//(\*.json except (wsdl\*,edmx\*))\\*[openapi\starts-with(., '3')] \components\child-names(., (), 'security\* head\* exam\*') => pf()"

parameters, responses, schemas (13.0)

parameters, schemas ........... (7.4)

responses, schemas ............ (14.8)

schemas ....................... (64.8)

## child-lnames

child-lnames($elem, $nosort, $namePatterns, $excludedNamePatterns)

child-names()

Returns for a given element ($elem) a concatenated, sorted list of child element local names. All details as in child-jnames().

A call without parameters is equivalent to a call with a single parameter, which is the context item.

## child-names

child-names($elem, $nosort, $namePatterns, $excludedNamePatterns)

child-names()

Returns for a given element ($elem) a concatenated, sorted list of child element lexical names. All details as in child-jnames().

A call without parameters is equivalent to a call with a single parameter, which is the context item.

## content-deep-equal

***Usage***

content-deep-equal($items as item()+) as xs:boolean?

content-deep-equal($items as xs:string?, $scope) as xs:boolean?

***Summary***

Returns false if $items contains at least two items with content which is not deep-equal. The meaning of “content” is controlled by $scope, which can mean the item itself ($scope value s), its content comprising attributes and child nodes (c), its child nodes (n) or its attributes (a).

***Details***

The single argument is a sequence of items, which may be a node or a string. Nodes are used without change, strings are interpreted as document URIs and replaced with the document node of the document found at that URI.

If $items has less than two items the empty sequence is returned, otherwise true or false.

***Parameters***

Described by the following table.

**Table**. Parameters of function content-deep-equal.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | A sequence of two or more items to be checked for content equality. Atomic items are interpreted as file URIs. |

***Examples***

Comparing three documents specified by URI:

fox "(a1.xml, a1-copy1.xml, a1-copy2.xml) => content-deep-equal()"

Comparing the *content of the root elements* - attributes and child nodes, but ignoring the names of the root elements:

fox "(a1-att8.xml, b1-att8.xml)\\* => content-deep-equal()"

The same as before, making the scope of comparison – content – explicit:

fox "(a1-att8.xml, b1-att8.xml)\\* => content-deep-equal('c')"

Comparing the *child nodes of the root elements*, but ignoring the names of the root elements as well as their attributes:

fox "(a1-att8.xml, b1-att8.xml)\\* => content-deep-equal('n')"

Comparing the *attributes root elements*, but ignoring the names of the root elements as well as their child nodes:

fox "(a1-att8.xml, b1-att8.xml)\\* => content-deep-equal('a')"

Comparing *inner elements* themselves, taking their names, attributes and child nodes into account:

fox "(a-att8-b1.xml, b-att9-b1.xml)\\*\b => content-deep-equal('s')"

Given the following files:

|  |  |
| --- | --- |
| **Name** | **Content** |
| a1.xml | <a>1</a> |
| a1-att8.xml | <a p="8">1</a> |
| a1-att9.xml | <a p="9">1</a> |
| a-att8-b1.xml | <a p="8"><b>1</b></a> |
| a-att9-b1.xml | <a p="9"><b>1</b></a> |
| a2.xml | <a>2</a> |
| b1.xml | <b>1</b> |
| b1-att8.xml | <b p="8">1</b> |
| b1-att9.xml | <b p="9">1</b> |
| b2.xml | <b>2</b> |

Several Foxpath expressions yield values as shown below:

|  |  |
| --- | --- |
| **Foxpath** | **Value** |
| fox "(a1.xml, b1.xml) => content-deep-equal()" | false |
| fox "(a1.xml, b1.xml)\. => content-deep-equal()" | false |
| fox "(a1.xml, b1.xml)\\* => content-deep-equal()" | true |
| fox "(a1.xml, b1.xml)\\* => content-deep-equal('c')" | true |
| fox "(a1.xml, b1.xml)\\* => content-deep-equal('s')" | false |
| fox "(a1-att8.xml, b1-att8.xml)\\*\@p => content-deep-equal('s')" | true |
| fox "(a1-att8.xml, a1-att9.xml)\\*\@p => content-deep-equal('s')" | false |
| fox "(a-att8-b1.xml, a-att9-b1.xml) => content-deep-equal()" | false |
| fox "(a-att8-b1.xml, a-att9-b1.xml)\\* => content-deep-equal()" | false |
| fox "(a-att8-b1.xml, a-att9-b1.xml)\\* => content-deep-equal('n')" | false |
| fox "(a-att8-b1.xml, a-att9-b1.xml)\a => content-deep-equal()" | false |
| fox "(a-att8-b1.xml, a-att9-b1.xml)\a\b => content-deep-equal()" | true |
| fox "(a-att8-b1.xml, b-att9-b1.xml) => content-deep-equal()" | false |
| fox "(a-att8-b1.xml, b-att9-b1.xml)\\* => content-deep-equal()" | false |
| fox "(a-att8-b1.xml, b-att9-b1.xml)\\*\b => content-deep-equal()" | true |

## count-chars

count-chars($string, $char)

count-chars($char)

Returns the number of occurrences of character $char in string $string.

If the first argument is omitted, it defaults to the context item.

Tip: A typical use is a predicate selecting items containing a separator, or at least a certain number of separators.

Examples:

fox "count-chars('a b c', ' ')"

fox "doc.xml\\@foo[count-chars(., ',')]"

fox "doc.xml\\@foo[count-chars(',')]"

## csv-doc, cdoc

csv-doc($uri, $separator, $header, $names, $quotes, $backslashes)

csv-doc()

Parses the document found at $uri into the XML representation of a CSV document.

A call without arguments is equivalent to a call with a single argument which is the context item.

Parameters:

* $uri – the URI of the document
* $separator – the separator; either specified as a single character, or one of the strings semicolon, colon, comma, tab, space; default: comma
* $header – if yes, the first record is interpreted as a table header; default: no
* $names – if direct, field names are represented as element names; if attributes, field names are provided by @name attributes; default: direct
* $quotes – if yes, quotes at the begin and end of a field value are treated as control characters; newline and separators within the value will be treated as parts of the value, not as a delimiter; default: yes
* $backslashes – if yes, \r, \n and \t will be replaced by the corresponding control character; otherwise, two consecutive quotes will be replaced by a single quote

Example: to parse the following CSV document:

2021-03-29,111  
2021-04-05,153

no parameters are required (except for $uri, if the URI is not the context item):

data.csv/csv-doc()

csv-doc(data.csv)

Example: to parse the following CSV document:

week;count  
2021-03-29;111  
2021-04-05;153

the separator should be specified and the used of a table header should be indicated using $header set to yes:

csv-doc(., 'semicolon', 'yes')

## file-copy , fcopy

file-copy($fileUris, $targetUri, $flags?)

file-copy($fileUris, $targetUri)

***Summary***

Copies files and/or folders.

***Details***

Copies files and/or folders to a target URI. If a source URI is a folder URI, the target URI must be a folder URI or a non-existing URI. If all source URIs are file URIs, the target URI may be a folder URI or a file URI. If the target URI does not exist and flag d is used, the target URI is interpreted as folder URI and the corresponding folder is created, also creating any non-existent parent folders. If the URI does not exist and flag d is not used, the target URI is interpreted as file URI. If the non-existing file URI belongs to a non-existing folder, an error is returned, unless flag c is used, in which case all non-existing parent folders are created. If the target URI is an existing file, an error is returned, unless flag o is used, in which case the file is overwritten.

***Parameters***

Described by the following table.

**Table**. Parameters of function file-copy.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| fileUris | File URIs or paths of the files to be copied |
| targetUri | File URI of the copy target – may be a folder URI or a file URI |
| flags | String of characters interpreted as follows:  o – overwrite an existent file  d – a non-existing target URI is interpreted as folder URI and the folder is created; non-existing parent folders are also created  c – a non-existing target URI is interpreted as file URI and non-existing parent folders are created |

***Examples***

// Copies doc.xml to doc2.xml; does not work if doc.xml already exists

fox doc.xml => file-copy('doc2.xml ')

// File doc2.xml is overwritten, if it already exists

fox doc.xml => file-copy('doc2.xml', 'o')

// Does not work unless folder copies already exists

fox doc.xml => file-copy('copies/doc2.xml')

// Folder copies is created, if non-existing

fox \*.xml => file-copy('copies/doc2.xml ', 'c ')

// The target URI is treated as a folder, which is created, if non-existing

fox \*.xml => file-copy('/other/copies ', 'd')

// Using a more complex selection

fox "../work/stages/\*d2cx//(\*.xml except \*docbook\*) => fcopy(../d2cx)"

## file-extension , fext

file-extension($fileUri)

Returns the file extension, that is, the last occurrence in the file name of a dot and all following characters.

A call without arguments is equivalent to a call with a single argument which is the context item.

Example: frequency distribution of all file extensions

fox "/programme/oxygen\*/frameworks/dita//\*/fext() => f()"

## file-info , finfo

file-info()

file-info($format)

Returns a string describing the context resource.

The structure of the info string is configured by $content. The value is a whitespace-separated list of display components. A display component specifies the kind of information item (first character) and the format of its display (following characters).

Item kind:

* p – URI
* n - file name
* s - file size
* d - file date

Display format:

* number... - right-pad to this length; padding character is the character following the number
* -number... - left-pad to this length; padding character is the character following the number
* () - put value into parentheses

Some useful display formats can be identified by their name, rather than specifying its parts:

* #nsd - “p60. s-10\_ d”
* #dn - “d28 p”
* #dns - “d28 p s()”

Default display: #nsd

Examples:

fox "../examples-operations//\*.ps1/file-info()"

fox "../examples-operations//\*.ps1/file-info('#dn')"

fox "../examples-operations//\*.ps1/file-info('#dns')"

fox "../examples-operations//\*.ps1/file-info('#dns')"

fox "../examples-operations//\*.ps1/file-info('d26 s-8 n')"

## fox-child ; fchild

fox-child($names)

fox-child($names, $namesExcluded?)

Returns the child URIs of the context URI with a name matching a name or name pattern from $names, and not matching a name or name pattern from $namesExcluded.

Example: all child resources with a name matching \*concur\* or \*fieldglass\*

fox "/projects/bhub/fox-child('\*concur\* \*fieldglass\*')"

Example: all child resources with a name matching \*concur\*, but not matching \*expense\*

fox "/projects/bhub/fox-child('\*concur\*', '\*expense\*')"

## fox-descendant ; fdescendant

fox-descendant($names)

fox-descendant($names, $namesExcluded?)

Returns the descendant URIs or the context URI with a name matching a name or name pattern from $names, and not matching a name or name pattern from $namesExcluded.

## fox-descendant-or-self ; fdescendant-or-self

fox-descendant-or-self($names)

fox-descendant-or-self($names, $namesExcluded?)

Returns from the context URI and its descendants those URIs with a name matching a name or name pattern from $names, and not matching a name or name pattern from $namesExcluded.

## fox-self ; fself

fox-self($names)

fox-self ($names, $namesExcluded?)

Returns the context URI, if its names matches a name or name pattern from $names and does not match a name or name pattern from $namesExcluded.

## fox-sibling ; fsibling

fox-sibling($names)

fox-sibling($names, $namesExcluded?)

fox-sibling($names, $namesExcluded?, $fromSubstring?, $toSubstring?)

Returns the sibling URIs of the context URI with a name matching a name or name pattern from $names, and not matching a name or name pattern from $namesExcluded. If $fromSubstring and $toSubstring are specified, the names to be matched are obtained by replacing in the names or name patterns in $names – or in the context resource name, if $names is not specified - substring $fromSubstring with substring $toSubstring.

Example: all sibling resources with a name matching \*concur\* or \*fieldglass\*

fox "/projects/bhub/fox-sibling('\*concur\* \*fieldglass\*')"

Example: all sibling resources with a name matching \*concur\*, but not matching \*expense\*

fox "/projects/bhub/fox-sibling('\*concur\*', '\*expense\*')"

Example: all sibling resources with a name obtained by replacing “Recruiting” with “Onboarding”

fox "/projects/bhub/sfRecruiting/fox-sibling((), (), 'Recruiting', 'Onboarding')"

## frequencies, freq, f

frequencies($values, $minFreq?, $maxFreq?, $order?, $format?)

frequencies($values, $minFreq?, $maxFreq?, $order?)

frequencies($values, $minFreq?, $maxFreq?)

frequencies($values, $minFreq?)

frequencies($values)

***Summary***

Reports distinct values and their frequencies.

***Parameters***

Described by the following table.

**Table**. Parameters of function frequencies.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| values | The values to be analyzed |
| minFreq | Report only values with a frequence >= the parameter value |
| maxFreq | Report only values with a frequence <= the parameter value |
| order | Sort order:  a – order by ascending frequence  b – order by descending frequence  t – order alphabetically (default) |
| format | Result format:  text\* – text file;  each distinct value a line, value padded to the length of the longest value string  textNN – text file;  each distinct value a line, value padded to a length of NN characters (e.g. text40)  csv – CSV document  json – JSON document  xml – XML document (root element name values, child element name value) |

***Examples***

Example 1: Report the element names occurring in a set of documents.

fox "a/b/c/\*.edmx\\\*\name() =>f(())"

## hlist-entry

hlist-entry($value …)

Create an entry for a hlist.

## hlist

hlist($hlistEntries, $emptyLinesSpec)

$hlistEntries: a sequence of concatenated strings representing value tuples to be output, where the concatenated items correspond to the values on each level; must be created using function hlist-entry().

$emptyLinesSpec: the nth digit gives the number of empty lines after each value on the nth level.

Example: hlist($entries, '010')

Insert an empty line after each value on the second level.

## jname-path ; jnpath ; jnp

jnpath($nodes, $numberOfSteps, $context)

Example:

fox foo.json\\allOf\jnpath()

Criticism:

We should have a special function for giving the paths of all decendants of a context, relative to that context; perhaps with filter criteria, too. Something like  
jcontent-paths()

## jnode-child ; jchild

jnode-child($nodes, $names, $namesExcluded, $ignoreCase)

Returns selected child nodes of a sequence of given nodes ($nodes). Selected child nodes have a JSON name matching a name filter ($names) and not matching a name filter defining excludions. Both name filters are optional, and absent name filters have no impact on the selection. If $ignoreCase is true, matching is performed ignoring character case.

Examples:

Get all media types used by Media Type Objects in a set of OpenAPI documents:

fox -D "../apis/\*.json\oas-keywords(., 'content')\jchild(., '\*xml\*')\jname() =>f()"

…

## jnode-location

jnode-location($nodes, $numFolders?)

Reports the locations and (if existent) text content of JSON nodes. The location includes the names of containing folders (optionally), the file name, the node JSON name and the path of node JSON names within the file. Parameter $numFolders specifies the number of containing folders to be included in the location. The parameter value must be an integer greater or equal to one.

Example: get the locations of allOf elements with siblings.

fox "../report3//\*\\\*:allOf[preceding-sibling::\*, following-sibling::\*] => jnode-location(2)"

Result:

====================================================================================================  
Folder  
. Folder  
. . File  
. . . Name  
. . . . Path  
. . . . . Value  
====================================================================================================  
  
download  
. bhub-20210329  
. . AccountMembersManagementAPI.json  
. . . allOf  
. . . . /oas/msgs/msg/schema/allOf  
. . EPD\_VISUALIZATION\_CONVERSION.json  
. . . allOf  
. . . . /oas/msgs/msg/schema/schema/allOf  
. . . . /oas/msgs/msg/schema/schema/allOf/schema/properties/systemSettings/schema/allOf  
. . EPD\_VISUALIZATION\_INTEGRATION.json  
. . . allOf  
. . . . /oas/msgs/msg/schema/schema/allOf (5)

## jpath-content ; jpcontent

jpath-content($context, $includedNames, $excludedNames, $excludedNodes)

## jpath-compare; jpathcmp

jpath-compare($nodes, options)

## resolve-json-ref ; jsonref ; jref

jref()

jref($ref)

jref($ref, $mode)

bla

## left-value-only ; left-value

left-value($value1, $value2)

***Summary***

Returns all atomic items occurring in the first value, but not in the second.

***Details***

The items of both values are atomized. Returns the atomized items occurring in the first value, but not in the second.

***Parameters***

Described by the following table.

**Table**. Parameters of function left-value-only.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| value1 | A value |
| value2 | Another value |

***Examples***

// Copies doc.xml to doc2.xml; does not work if doc.xml already exists

fox doc.xml => file-copy('doc2.xml ')

left-value($value1, $value2)

Returns all items contained in $value1, but not in $value2. All items are atomized.

Example:

Returns file names found in one folder (at any depth), but not in another folder (at any depth):

fox "/projects/bhub/download/left-value(bhub-20210510//\*.json/fname(), bhub-20210517//\*.json/fname())"

Example:

Return the paths of all OpenAPI documents containing tag declarations which are not used:

fox "/projects/bhub/download/bhub-20210517//\*.json[\\*\left-value(tags\\_\name, paths\\*\\*\tags\\_)]"

## lname-path ; lnpath ; lnp

lname-path($nodes, $numberOfSteps, $context)

Example:

fox foo.xml\\bar\lnpath()

## lnode-location

lnode-location($nodes, $numFolders?)

Reports the locations and (if existent) text content of file content nodes. The location includes the names of containing folders (optionally), the file name, the local node name and the path of local node names within the file. Parameter $numFolders specifies the number of containing folders to be included in the location. The parameter value must be an integer greater or equal to one.

## lpath-content ; lpcontent

lpath-content($context, $includedNames, $excludedNames, $excludedNodes)

## name-path ; npath ; np

name-path($nodes, $numberOfSteps, $context)

Example:

fox foo.xml\\bar\npath()

## node-location

node-location($nodes, $numFolders?)

Reports the locations and (if existent) text content of file content nodes. The location includes the names of containing folders (optionally), the file name, the node name and the name path within the file. Parameter $numFolders specifies the number of containing folders to be included in the location. The parameter value must be an integer greater or equal to one.

For an example see function jnode-location.

## non-distinct-file-names , non-distinct-fnames

non-distinct-file-names($uris, $ignoreCase)

Returns the URIs which have a non-unique file name, that is, a file name contained by at least two URIs. If $ignoreCase is true, distinctness check is performed ignoring case differences.

## non-distinct-file-names , non-distinct-fnames

non-distinct-file-names($uris, $ignoreCase?)

Returns the URIs in $uris which contain a non-distinct file name, that is, which contain a file name also contained by a different URI. If $ignoreCase is true, distinctness is checked ignoring case.

## non-distinct-values , non-distinct

non-distinct-values($items, $ignoreCase?)

Returns the items which are not distinct, that is, occurring in $items at least twice. If $ignoreCase is true, distinctness check is performed ignoring case differences. If $ignoreCase is true, distinctness is checked ignoring case.

Example: return all OAS

/projects/bhub/download/bhub-20210225//(\*.json except (wsdl\*,edmx\*))  
\\*\paths\(  
 let $ndv := non-distinct-values(\*\\*\operationId) return   
 hlist-entry(bdname(), bfname(), $ndv => sort() => string-join(', '))[$ndv]  
) => hlist()

Response:

AlertNotification

. cf\_configuration\_api.json

. . create, delete, get, getAll, update

. neo\_configuration\_api.json

. . create, delete, get, getAll, update

NFEAPIS

. nfe\_authorize.json

. . downloadNFe

SAPCustomerDataCloud

. GigyaAPI\_accounts\_b2b\_registerOrganization.json

. . accounts.b2b.registerOrganization

## oas-msg-schemas ; oasmsgs

oas-msg-schemas($nodes)

$nodes – nodes from OpenAPI documents

Returns the message schema objects of given OpenAPI documents. Processes all documents containing at least one node from $nodes. Usually, this will be the root element of the document, but any nodes from the document may be used as well - the output for a given document is not influenced by the number and kind of nodes used to identify it.

Pitfall: as the input must be nodes, not URIs, make sure to pass nodes to the function. In the example below, note the use of the backslash operator, ensuring that a node is passed to the function, rather than the URI produced by the preceding step.

Example: get all names of message schema fields –

fox -D "../apis/\*.json\oas-msg-schemas()\\*\jname() => f()"

## path-content ; pcontent

path-content($context, $includedNames, $excludedNames, $excludedNodes)

path-content()

Lists the relative data path of all items directly or indirectly contained by a context node. The paths are relative to the node given by $context (e.g. foo/bar indicating an item named bar which is child of an item named foo which is child of the context node.

A call without parameters is equivalent to a call with a single parameter, which is the context item.

## PROJECT: path-template ; ptemplate

path-template($paths, $tvarSpec …)

$paths: paths obtained with path-name or path-content

$tvarSpec:

number whitespace varName

| negnumber whitespace varName

| pathPattern:

examples:

paths/{path}

{path}/get

{path}/(get, post, put, delete, options, head, patch, trace)

paths/{path}/(get, post, put, delete, options, head, patch, trace)

foo\*/{varname}/bar\*

foo\*/{varname: a\*}/bar\*

Leading / - pattern anchored

Trailing / - pattern anchored

Steps can contain wildcards

A step may contain several names: (), names comma separated

{varname: a\*} – var name assignment requires this step to match the pattern a\*

{varname: (a\*, b\*, c\*)} – …s

## right-value-only ; right-value

right-value($value1, $value2)

Returns all items contained in $value2, but not in $value1. All items are atomized.

Example:

Returns file names found in one folder (at any depth), but not in another folder (at any depth):

fox "/projects/bhub/download/right-value(bhub-20210510//\*.json/fname(), bhub-20210517//\*.json/fname())"

Example:

Returns attribute values found in one file, but not in another file:

fox "right-value(\*bootstrap.xml\\@value, \*gefeg.xml\\@value)"

## values-distinct

values-distinct($values)

Returns true, is the string values of $values are distinct, false otherwise.

Example: clarify if the folder names containing given files are distinct.

## write-file

write-file($items, $fileName, $encoding)

## xwrap

write-file($values, $rootName, $flags, $itemName)

Transforms a sequence of values into an XML document.

Options:

*Options available for node items*

a – if the item is an attribute: turn it into an element with the same name

A – if the item is an attribute: turn it into an element with the same local name and without namespace

b – add an attribute @xml:base, giving the base URI of the item

p – add an attribute @path, giving the name path of the item

j – add an attribute @jpath, giving the JSON name path of the item

f – use a flat copy of the item, with child nodes discarded

*Options available for atomic items*

d – the item is interpreted as a URI and an attempt is made to parse the document retrieved from that URI; if a document is obtained, it is used as the item to be included in the result; otherwise, an element <PARSE-ERROR uri= "… "> is used instead.

b – in combination with option d: add an attribute @xml:base, giving the base URI of the document

w – the item is interpreted as a URI and an attempt is made to retrieve the text content of the resource thus identified; if text can be retrieved, it is wrapped in an element and the element is used as the item; the name of the element is \_text\_ by default, but can be controlled by parameter $name2

t – as w, but the text retrieved from the URI is not wrapped in an element

c – atomic item is wrapped in an element; the name of the element is \_text\_ by default, but can be controlled by parameter $name2