# Fox functions

Version: 2025-08-06

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

## Concepts and pitfalls

Many extension functions have semantics which leverage concepts explained in this section.

### Infospace definition document (ispace.xml)

The infospace definition document (recommended name: ispace.xml) is a configuration which defines a mapping of file URIs to a specific parsing approach. The document enables navigation into files without explicit call of a parse function, examples:

.//\*.xml\\foo

.//\*.html\\foo

.//\*.csv\\foo

.//\*.json\\foo

.//\*.docx\\foo

Unless option –s is used, the infospace definition document is the document supplied as part of the Foxpath installation. You can use option –s in order to replace the document with a user-defined document, or option –x in order to extend the standard document with the entries in a user-defined document.

### Unified String Expression

A unified string expression is a string specifying a match condition: any given string either matches or does not match the expression. Function parameters specifying a *filter condition* are often interpreted as unified string expressions. Example: the function call

descendant('\*table\* \*list\* ~\*informal\* ~\*simple\*')

returns all descendant elements with a local name constrained by a unified string expression. More precisely, the name must contain a substring “table” or “list”, but must not contain a substring “informal” or “simple”, where all comparisons are made case-insensitively.

#### Overview

A unified string expression is one of the following:

* A set of glob patterns, each one interpreted as inclusive or exclusive condition
* A set of regular expressions, each one interpreted as inclusive or exclusive condition
* A fulltext expression, representing a fulltext search

A unified string expression consists of a mandatory **pattern string** and an optional **options** string, separated by the first non-doubled # character. In the following

order\d+ cancel-\d+ #rc

the pattern string is interpreted as a set of regular expressions (r), to be evaluated in a case sensitive way (c).

By default, a unified string expression is interpreted as a set of glob patterns. A different interpretation is triggered by options:

* If the options string contain the token fulltext or ft, the expression is interpreted as a fulltext expression. Examples:

versatile markup language #ft

versatile markup lang #ft s-en

* Otherwise, if the options string contains the flag r, the expression is interpreted as a set of regular expressions. Examples:

test\d+ ~\*999\* #r

test\d+ ~\*999\* #rc

* Otherwise, the expression is interpreted as a set of glob patterns. Examples:

table\* \*list ~\*informal\*

table\* \*list ~\*informal\* #c

#### Glob patterns and regular expressions

When the expression is interpreted as a set of glob patterns or regular expressions, the pattern string is evaluated as a whitespace-separated list of patterns/expressions.

Glob syntax:

* The character \* represents zero or more characters
* The character ? represents exactly one character
* The character sequence \s represent a single whitespace character
* Other characters represent themselves

Regular expression syntax: see [XPath functions regex syntax](https://www.w3.org/TR/xpath-functions-31/#regex-syntax).

The individual patterns/expressions are interpreted as inclusive or exclusive conditions:

* Patterns/expressions preceded by a ~ character are interpreted as exclusive conditions
* Other patterns/expressions are interpreted as inclusive conditions

Evaluation rules:

* If the expression contains inclusive conditions, at least one of them must be matched
* If the expression contains exclusive conditions, none of them may be matched.

Examples:

* \*table\* \*list\*

A string must contain either “table” or “list”

* ~\*informal\* ~\*simple\*

A string must not contain “informal” or “simple”.

* \*table\* \*list\* ~\*informal\* ~\*simple\*

A string must contain either “table” or “list”, and it must not contain “informal” or “simple”.

Patterns and regular expressions must not contain whitespace. Whitespace can be represented by “\s”. Example: geo\sdata other\sinfo\*

By default, patterns and regular expressions are interpreted case-insensitively. Case sensitivity is triggered by using flag c. Examples:

* \*table\* \*list\* ~\*informal\* ~\*simple\* #c
* test\d+ ~\*999\* #rc

Note that case sensitivity cannot be controlled on the level of individual patterns or regular expressions.

#### Namespace qualified evaluation

Option “q” triggers namespace qualified evaluation: matching is applied to a **namespace qualified string**, which may (but need not) be an XML node name. A namespace qualified string is supplied as a QName or as a sequence of strings. A sequence of strings is interpreted as follows:

* A singleton item is a string which does not belong to a namespace
* A pair of items represents a string (first item) belonging to a namespace (second item)
* More than two items are not allowed

The individual glob/regex strings of the match pattern are associated with a namespace constraint:

* If containing a colon:
  + If the substring preceding it is a \* character: any or no namespace
  + Otherwise: the namespace URI bound to that prefix

Note that the prefix must not contain wildcard characters.

* Otherwise:
  + If the string consists of a single \* character: any or no namespace
  + Otherwise: no namespace

In order to match a glob/regex associated with a namespace constraint, a namespace qualified string

1. Matches the glob/regex
2. Satisfies the namespace contraint:
   1. Constraint “no namespace”: the string is in no namespace
   2. Constraint is a namespace URI: the string is in that namespace

*Example*

Unified string expression: docbook:tab\* docbook:\*list \*:extension anno-\* #q

Match rules:

* The string must match one of the glob patterns “tab\*”, “\*list”, “extension”, “anno-\*”
* If the string matches “anno-\*”, it must not belong to a namespace; if it is “extension”, it may be in any or no namespace; otherwise, it must belong to the namespace http://docbook.org/ns/docbook

Namespace prefixes contained by glob/regex strings must be resolvable to a namespace URI. This means that the prefix has either been declared as part of the query (declare namespace …='… ';) or it is the prefix of a built-in namespace binding. The following table lists the built-in namespace bindings.

|  |  |
| --- | --- |
| **Prefix** | **Namespace URI** |
| dc | http://purl.org/dc/elements/1.1/ |
| docbook | http://docbook.org/ns/docbook |
| owl | http://www.w3.org/2002/07/owl# |
| rdf | http://www.w3.org/1999/02/22-rdf-syntax-ns# |
| rdfs | http://www.w3.org/2000/01/rdf-schema# |
| svrl | http://purl.oclc.org/dsdl/svrl |
| wsdl | http://schemas.xmlsoap.org/wsdl/ |
| xml | http://www.w3.org/XML/1998/namespace |
| xs | http://www.w3.org/2001/XMLSchema |
| xsi | http://www.w3.org/2001/XMLSchema-instance |
| xsl | http://www.w3.org/1999/XSL/Transform |

#### Fulltext string expression

A **fulltext string expression** is an expression string which can be mapped to a [fulltext search](https://www.w3.org/TR/xpath-full-text-30/) and has the semantics of the corresponding fulltext search expression. Compared to a fulltext search expression, a fulltext string expression is easier to write and read.

A fulltext string expression consists of a **query string**, optionally followed by an **options string**. Query string and options string are separated by a # character.

The basic building block of the query string is a **token list** – a whitespace-separated list of tokens, optionally preceded by a start anchor (^) or followed by an end anchor ($). Start anchor and end anchor signal that matching of the token list must occur at the start / the end of the string.

The query string is either simple or complex:

1. Simple query string: a token list
2. Complex query string: a sequence of **subqueries**, separated by **boolean operators** - / (and), |(or), ~(not)
3. A subquery is either simple or complex:
   1. Simple subquery: a token list, optionally followed by an “@” character followed by **local options**
   2. A complex query string, surrounded by parentheses, optionally followed by an “@” character followed by local options

The evaluation of a token list is controlled by options (or local options):

1. By default, the token list is interpreted as a phrase which must be contained by the tested string
2. Option “W” – all words in the token list must be contained (but not necessarily as a phrase)
3. Option “w” – at least one word from the token list must be contained
4. Option “phrase-1” (phrase-2, …) – the token list is interpreted as an “open phrase”, allowing at most one (two, …) words to occur between two adjacent tokens of the token list
5. Option “phrase-7win” (phrase-10win, …) – the token list is interpreted as an “open phrase”, allowing other words to occur between two adjacent tokens of the token list, yet the length of the phrase including those inserted words must not be greater then the specified number (7, 10, …)

Further options specify other modifications of the matching behaviour:

|  |  |  |
| --- | --- | --- |
| **Options** | **Examples** | **Meaning** |
| c | c | Matching is case-sensitive |
| d | d | Matching is diacritics-sensitive (e.g. ü != u) |
| f  f-$LEVEL | f  f-2  f-3 | Fuzzy matching of individual tokens, tolerance level as specified (1, 2, 3, …) |
| s-$LANG | s-en | Stemming, assuming the specified language |
| wild-$TOKEN | wild-xxx | The token $TOKEN represents any token |
| stop($TOKEN1, $TOK2,…) | stop(xxx,yyy) | The tokens $TOKEN1, $TOK2, … represent any token |
| o | o | The tokens must be matched in the order as given in the token list |
| dist-$RANGE | dist-1  dist-1..  dist-..2  dist-1..2 |  |
| win-$RANGE | win-10  win-..10  win-10..  win-10..12 | The token list must be matched by a substring of the test string containing as many words as specified (e.g. exactly 10 words, at most 10 words, at least 10 words, between 10 and 12 words, …) |
| occ-$RANGE | occ-2  occ-..2  occ-2..  occ-1..2 | The token list must be matched as many times as specified (e.g. exactly twice, at most twice, at least twice, once or twice, …) |

Complex query:

* Single boolean operator
  + subqs @local-options / subqs @local-options # global-options
  + subqs @local-options | subqs @local-options # global-options
* Combinations of several boolean operators, optionally using parentheses
  + (sqs@local-options | sqs@local-options) / sqs@local-options # global-options

Note: use of @local-options and #global-options is optional.

#### Examples

This section presents examples of unified string expressions and their matching behaviour.

**Examples, group 1 – using GLOB patterns.**

|  |  |  |  |
| --- | --- | --- | --- |
| Unified String Expression | Explanation | Test string | Match result |
| Berlin | The string “Berlin”, any case | Berlin | + |
|  |  | BERLIN | + |
|  |  | In Berlin | - |
|  |  | Berlin. | - |
| Berlin#c | The string “Berlin”, case sensitive | Berlin | + |
|  |  | BERLIN | - |
| Ber\* | A string starting with “Ber”, any case | BERLIN. | + |
| Ber\* Ham\* | A string starting with “Ber” or “Ham”, any case | hamburg | + |
|  |  | In Hamburg | - |
| Ber\* Ham\* ~Bern ~Hameln | A string starting with “Ber” or “Ham”, any case, but not equal to “Bern” or “Hameln” | Hamburg | + |
|  |  | Hameln | - |

**Examples, group 2 – using regular expressions.**

|  |  |  |  |
| --- | --- | --- | --- |
| Unified String Expression | Explanation | Test string | Match result |
| test\d+#r | A string starting with “test” (any case), followed by one or more digits. | test01 | + |
|  |  | TEST01 | + |
| test\d+#rc | A string starting with “test” (lower case), followed by one or more digits | test01 | + |
|  |  | TEST01 | - |
| test\d+ custom\d+ ~.\*888 ~.\*999#r | A string starting with “test” or “custom”, followed by one or more digits, but not ending with 888 or 999 | test887 | + |
|  |  | custom887 | + |
|  |  | custom999 | - |

**Examples, group 3 – using fulltext expressions.**

|  |  |  |  |
| --- | --- | --- | --- |
| Unified String Expression | Explanation | Test string | Match result |
| markup language #fulltext | A string containing the phrase “markup language”, any case, anywhere | XML is a versatile markup language. | + |
|  |  | There are several markup languages. | - |
| language markup versatile #fulltext W | A string containing all words from “language”, “markup” and “versatile” | XML is a versatile markup language. | + |
|  |  | XML is a popular markup language. | - |
| language markup versatile #fulltext w | A string containing a word from “language”, “markup” and “versatile” | XML is a popular markup language. | + |
| ^language markup versatile #fulltext w | A string starting with a word from “language”, “markup”, and “versatile” | Markup languages are considered. | + |
|  |  | These markup languages are considered. | - |
| language markup versatile$ #fulltext w | A string ending with a word from “language”, “markup”, and “versatile” | XML is a popular markup language. | + |
|  |  | These markup languages are considered. | - |

### Node name filtering

Various functions support a filtering of nodes by node name. The filtering is specified by a parameter nameFilter, which is evaluated as a [Unified String Expression](#_Unified_String_Expression). The nameFilter parameter is always accompanied by an options parameter, for which the options qname, jname, name and lname are supported. This allows a flexible approach to name filtering:

* By default, the filtering is applied to the local name of the node. This is equivalent to using option lname. Namespace URIs as well as namespace prefixes are ignored.
* Using option qname, filtering is namespace sensitive: each match pattern containing a colon is split into *prefix* (substring preceding the colon) and *local name pattern* (substring following the colon). A match pattern without prefix is interpreted as describing a node name in no namespace, unless the pattern is \*, which matches any node name, regardless of the namespace URI. The prefix may be a wildcard (\*) or a string without wildcard characters. A non-wildcard prefix is resolved to a namespace URI, which means that it must either be a built-in namespace prefix (see below) or a prefix declared as part of the Foxpath query (declare namespace prefix='URI ';).

A node name matches the pattern if its local name matches the local name pattern and either (a) the pattern prefix is a wildcard or (b) the namespace URI of the node is equal to the namespace URI bound to the prefix or (c) the node has no namespace URI and the pattern has no prefix. Special case: the match pattern \* matches any node name, regardless of the namespace.

A non-wildcard pattern prefix is resolved to a namespace URI as follows: (a) if the prefix has been declared as part of the query (declare namespace prefix='…';) the namespace URI specified by the declaration; (b) otherwise the prefix must be a built-in prefix, that is one of dc, docbook, owl, rdf, rdfs, svrl, wsdl, xml, xs, xsi, xsl.

* Using option jname, filtering is applied to the original JSON field name, rather than to the XML name used in the XML representation of the field. Formally, the filtering is applied to the value obtained by applying to the local name of the node extension function decode-key(). For example, the pattern geo data would be matched by an element with local name geo\_0020data.
* Using option name, filtering is applied to the lexical node name, which may include a prefix. The option should be used with care, as the result depends on the actual use of prefixes, which is in sofar unpredictable, as it may be changed without affecting the information content of the document. The option should therefore only be considered when the documents to be processed are known to adhere to a consistent use of namespace prefixes.

**Table.** Examples of node name matching. Note that by default node matching is case insensitive.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pattern  (glob syntax) | Name filter  Option | Node name | Node namespace | Matches |
| tab\* | - | table | http://docbook.org/ns/docbook | + |
| tab\* | - | dbook:table | http://docbook.org/ns/docbook | + |
| tab\* | - | table | - | + |
| Tab\* | - | table | - | + |
| tab\*#c | - | table | - | + |
| Tab\*c# | - | Table | - | + |
| Tab\*#c | - | table | - | - |
| docbook:tab\* | qname | table | http://docbook.org/ns/docbook | + |
| \*:tab\* | qname | table | http://docbook.org/ns/docbook | + |
| docbook:tab\* | qname | db:table | http://docbook.org/ns/docbook | + |
| \*:tab\* | qname | db:table | http://docbook.org/ns/docbook | + |
| tab\* | qname | table | http://docbook.org/ns/docbook | - |
| docbook:tab\* | qname | table | - | - |
| continent\* | jname | continents | - | + |
| geo dat\* | jname | geo\_0020data | - | + |
| geo\_0020data | jname | geo\_0020data | - | - |
| tab\* | name | table | http://docbook.org/ns/docbook | + |
| tab\* | name | table | - | + |
| dbook:tab\* | name | dbook:table | http://docbook.org/ns/docbook | + |
| docbook:tab\* | name | dbook:table | http://docbook.org/ns/docbook | - |
| \*:tab\* | name | Dbook:table | http://docbook.org/ns/docbook | - |

### Node name types – local, lexical, JSON

Several functions deliver node names, or paths containing node names. These functions support several name kinds: local names, lexical names, JSON names. Dependent on the function, the kind of name either depends on the function name (as indicated by the substring –name, -lname or –jname) or by an option value equal to one of the strings name, lname, jname. In both cases, the strings are to be interpreted as follows:

* lname – local name
* name – lexical name, possibly containing a prefix
* jname – JSON name

The JSON is the name from which the given name was obtained by applying the following rules, implemented by function encode-key():

* An empty string is converted to a single underscore (\_)
* Existing underscores are replaced with two underscores (\_\_)
* Characters that are not valid NCName characters are replaced with an underscore and the character’s four-digit Unicode.

The JSON names are obtained from a given name by function decode-key(). Examples:

> fox "'?'/encode-key()"

\_003f

> fox "'\_003f'/decode-key()"

?

### Name paths

The output of several functions contains name paths., e.g. function name-path(). The term means means a path of names separated by slashes similar to an XPath path expression. Variations concern

* The kind of name used (local name, lexical name, JSON name)
* Whether indexes (e.g. [3]) indicating the position among equally named siblings are included
* Whether text nodes are presented as a step (text())
* Whether the proper name path is preceded by information about the containing file
* Whether the proper name path is followed by information about data values

### Function variants

Many functions exist in several variants recognized by a name pattern – either the presence/absence of a postfix, or the use of a particular substring (e.g. –name, -jname, …).

#### \*-ec

Many functions occur in two variants, distinguished by a function name with and without a postfix

-ec. The postfix signals “explicit context”, meaning that the first function parameter receives the items to be processed. This parameter is omitted by the variant without –ec : the first, second, … parameter of this variant correspond to the second, third, … parameter of the variant with –ec. Example:

* xsd-validate($xsds) – validates the document bound to the context item against XSDs bound to parameter $xsds
* xsd-validate-ec($docs, $xsds) – validates the documents bound to parameter $docs against XSDs bound to parameter $xsds

#### Name type dependent - name / lname / jname

Some functions have a name containing one of the substrings:

* -name, -names
* -lname, -lnames
* -jname, -jnames

The output of these functions contain node names, and the substring indicates the kind of name – lexical name (-name), local name (-lname), JSON name (jname).

The term “lexical name” denotes the name returned by the XPath standard function name(); it may contain a prefix. See [Node name types – local, lexical, JSON](#_Node_name_types) for a description of JSON names.

### Function options

Many functions have a last parameter specifying options controlling the processing. The parameter value is a whitespace separated list of entries consisting of an option name, optionally followed by equal sign and option value. Examples:

nosort

order=d width=20

Whitespace is allowed between option name, equal sign and option value, for example:

order = d width = 20

The option value must not contain whitespace. If whitespace is required, it should be represented by the string “%20. Example:

Header=Dir%20name

### Pitfalls

This section is a collection of common pitfalls which can easily confuse a user not yet experienced in the use of Foxpath.

#### Pitfall “tilt the slash”

***Description***

The user intends a mixed navigation beginning in the file system and continued in file contents - but she fails to use slash and backslash in the right places.

***Example***

Wrong:

fox "cfg//\*.xml//fileSystemUpdates\child-name-seq() => freq()"

Corrected:

fox "cfg**//**\*.xml**\\**fileSystemUpdates**\**child-name-seq() => freq()"

#### Pitfall “use \*-ec”

***Description***

A function ocurring in two variants – with postfix –ec and without – must be handled with care. Simple rules:

* When using the function call as path step, use the form without –ec, as you want the input to be the context item
* When using the function call on the right-hand side of the => operator, use the form with –ec, as the items produced on the left-hand side are implicitly treated as the argument of the first function parameter.

***Examples***

Function used as a path step:

fox "cfg//\*.xml**/**xvalidate(/projects/abc/xsd/\*.xsd)

Function used as right-hand side operator of =>:

fox "cfg//\*.xml **=>** xvalidate-ec(/projects/abc/xsd/\*.xsd)

(More pitfalls under construction.)

## Navigation aids 1 – standard axes

The functions in this section enable enhanced node tree navigation. Each function selects nodes from a particular navigation axis (ancestor, descendant, …). Nodes are selected by a [Unified String Expression](#_Unified_String_Expression) applied to the node name. By default, the selection is applied to the local name of the nodes. Options trigger alternatives:

* option qname – name filtering is applied to the qualified node names
* option jname – name filtering is applied to the JSON field names
* optione name – name filtering is applied to the lexical nodes names

### ancestor (\*-ec)

**ancestor**($nameFilter as xs:string := (),

$options as xs:string := ())

as node()\*

**ancestor-ec**(

$inputItems as item()\*,

$nameFilter as xs:string := ()

$options as xs:string := ())

as node()\*

***Summary***

Returns ancestor nodes, optionally selected by node name and/or position.

**Details**

Input items can be nodes and/or atomic items, but atomic items are ignored, as they are interpreted as document URI and the corresponding document node cannot have ancestor nodes. The function returns the ancestor nodes of the input nodes, optionally filtered by name and/or by position.

Function variant ancestor-ec receives input items as the value of the first parameter. Function variant ancestor processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)). The use of any name filter excludes document nodes from the result.

Further processing details are controlled by *options.* They are provided as option names separated by whitespace.

Options name, jname and lname control the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default). For details, see [Node name types](#_Node_name_types).

Options first, first2, last, last2 specifiy a positional filter: for each input node, only the first (second, last, second last) result node is returned. The positional filter is applied after a name filter.

***Parameters***

Described by the following table.

**Table**. Parameters of function ancestor and ancestor-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are ignored. |
| nameFilter | Name filter selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). Note that use of this parameter excludes document nodes from the result. |
| options | Whitespace-separated list of options.  *Options group 1* – node name type to which the name filter is applied:  lname – local names (default)  jname – JSON names  name – lexical names  *Options group 2* – positional filter; for each input node, at most one result node is returned, selected by its position in reverse document order:  first – for each input node, return only the first result node  first2 – for each input node, return only the second result node  last – for each input node, return only the last result node  last2 – for each input node, return only the second last result node |

***Examples***

Example “no filter”. Inspecting a set of docbook documents - which elements contain 'para' elements?

fox "docbook//\*.xml\\docbook:para\ancestor()\name() => f()"

Example “name filter”. Which section elements ('section', 'sect1'; 'sect2', …) contain 'para' elements?

fox "docbook//\*.xml\\docbook:para\ancestor('sect\*')\name() => f()"

Example “positional filter”. Which top-level elements contain module elements? Note that the last two element ancestors are top-level element and root element.

fox "docbook//\*.xml\\docbook:module\ancestor('\*', 'last2')\name() => f()"

Example “name and positional filters”. Return the nearest ancestor elements of a text node containing a given phrase, skipping any containing 'emphasis' or 'phrase' elements (in search of the smallest containing semantic unit).

fox "docbook//\*.xml\\text()  
 [contains-text('available free space on .+ disk')]  
 \ancestor('~emphasis ~phrase', 'first')   
 => xwrap('elems')"

Example “option name”. List the text nodes containing non-whitespace and contained by an element with an 'svg:' prefix. When using option name, the name filter deals with the lexical name - nodes in the svg namespace but using a different prefix, or no prefix, are not found.

fox "docbook//\*.xml\\text()[nonws()][ancestor('\*svg:\*', 'name')]\truncate() => f()"

### ancestor-or-self (\*-ec)

**ancestor-or-self**(

$nameFilter as xs:string := (),

$options as xs:string := ())

as node()\*

**ancestor-or-self-ec**(

$inputItems as item()\*,

$nameFilter as xs:string := (),

$options as xs:string := ())

as node()\*

***Summary***

Returns ancestor-or-self nodes, optionally selected by node name and/or position.

**Details**

Input items can be nodes and/or atomic items. Atomic items are interpreted as document URI and replaced with the corresponding document node. The function returns the input nodes and their ancestor nodes, optionally filtered by name and/or by position.

Function variant ancestor-or-self-ec receives input items as the value of the first parameter. Function variant ancestor-or-self processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)). The use of any name filter excludes document nodes from the result.

Further processing details are controlled by *options.* They are provided as option names separated by whitespace.

Options name, jname and lname control the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default). For details, see [Node name types](#_Node_name_types).

Options first, first2, last, last2 specifiy a positional filter: for each input node, only the first (second, last, second last) result node is returned. The positional filter is applied after a name filter.

***Parameters***

Described by the following table.

**Table**. Parameters of function ancestor-or-self and ancestor-or-self-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are interpreted as document URI and replaced with the corresponding document node. |
| nameFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). Note that use of this parameter excludes document nodes from the result. |
| options | Whitespace-separated list of options.  *Options group 1* – node name type to which the name filter is applied:  lname – local names (default)  jname – JSON names  name – lexical names  *Options group 2* – positional filter; for each input node, at most one result node is returned, selected by its position in reverse document order:  first – for each input node, return only the first result node  first2 – for each input node, return only the second result node  last – for each input node, return only the last result node  last2 – for each input node, return only the second last result node |

***Examples***

Example “no filter”. Inspecting a set of dita documents - which @xml:lang attribute values are observed in term elements and their ancestors? As no filter is applied, the function call is equivalent to the navigation step ancestor-or-self::node().

fox "dita//\*.dita\\term\ancestor-or-self()\@xml:lang => f()"

Example “name filter”. Find the table, ul and ol elements directly or indirectly containing a @conref. For each element return a relative URI with a name path fragment.

fox "dita//\*.dita

\\\*[@conref]\ancestor-or-self('table ul ol')

\name-path((), 'rel-base-uri')

=> f()"

Example “name and positional filter”. As the previous example, but return only the innermost table, ul and ol elements.

fox "dita//\*.dita

\\\*[@conref]\ancestor-or-self('table ul ol', 'first')

\name-path((), 'rel-base-uri')

=> f()"

### attributes (\*-ec)

**attributes**($nameFilter as xs:string := (),

$options as xs:string := ())

as node()\*

**attributes-ec**(

$inputItems as item()\*,

$nameFilter as xs:string := (),

$options as xs:string := ())

as node()\*

***Summary***

Returns attribute nodes, optionally selected by node name and/or position.

**Details**

Input items can be nodes and/or atomic items, but atomic items are ignored, as they are interpreted as document URI and the corresponding document node cannot have attributes. The function returns the attribute nodes of input nodes, optionally filtered by name and/or by position.

Function variant attributes-ec receives input items as the value of the first parameter. Function variant attributes processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Unified_Filter_Expression). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)).

Further processing details are controlled by *options.* They are provided as option names separated by whitespace.

Options name, jname and lname control the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default). For details, see [Node name types](#_Node_name_types).

Options first, first2, last, last2 specifiy a positional filter: for each input node, only the first (second, last, second last) result node is returned. The positional filter is applied after a name filter.

***Parameters***

Described by the following table.

**Table**. Parameters of function attributes and attributes-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are ignored. |
| nameFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| options | Whitespace-separated list of options.  *Options group 1* – node name type to which the name filter is applied:  lname – local names (default)  jname – JSON names  name – lexical names  *Options group 2* – positional filter; for each input node, at most one result node is returned, selected by its position in reverse document order:  first – for each input node, return only the first result node  first2 – for each input node, return only the second result node  last – for each input node, return only the last result node  last2 – for each input node, return only the second last result node |

***Examples***

Example “no filter”. Inspecting a set of docbook documents - get the names and frequencies of attributes on table elements.

fox "docbook//\*.xml\docbook:article\\docbook:table\attributes()\name() => f()"

Example “name filter”. Inspecting a set of docbook documents - get the names and frequencies of elements which have an. attribute with a local name equal width or matching \*span.

fox "docbook//\*.xml\docbook:article\\docbook:\*[attributes('width \*span')]\name() => f()"

Example “option name”. Inspecting a set of docbook documents - get the names and frequencies of attributes with prefix xml, found in “section” elements (section, sect1, sect2, ...).

fox "docbook//\*.xml\docbook:article\descendant('sect\*')\attributes('xml:\*', 'name')\name() =>f()"

### child (\*-ec)

**child**($nameFilter as xs:string := (),

$options as xs:string := ())

as node()\*

**child-ec**(

$inputItems as item()\*,

$nameFilter as xs:string := (),

$options as xs:string := ())

as node()\*

***Summary***

Returns child element nodes, optionally selected by node name and/or position.

**Details**

Input items can be nodes and/or atomic items. Atomic input items are interpreted as document URI and replaced with the corresponding document node. The function returns the child elements of the input nodes, optionally filtered by name and/or by position.

Function variant child-ec receives input items as the value of the first parameter. Function variant child processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Unified_Filter_Expression). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)).

Further processing details are controlled by *options.* They are provided as option names separated by whitespace.

Options name, jname and lname control the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default). For details, see [Node name types](#_Node_name_types).

Options first, first2, last, last2 specifiy a positional filter: for each input node, only the first (second, last, second last) result node is returned. The positional filter is applied after a name filter.

***Parameters***

Described by the following table.

**Table**. Parameters of function child and child-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are interpreted as document URI and replaced with the corresponding document node. |
| nameFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| options | Whitespace-separated list of options.  *Options group 1* – node name type to which the name filter is applied:  lname – local names (default)  jname – JSON names  name – lexical names  *Options group 2* – positional filter; for each input node, at most one result node is returned, selected by its position in reverse document order:  first – for each input node, return only the first result node  first2 – for each input node, return only the second result node  last – for each input node, return only the last result node  last2 – for each input node, return only the second last result node |

***Examples***

Example “no filter”. Get the names and frequencies of the child elements of docbook root elements.

fox "docbook//\*.xml\docbook:\*\child()\name() => f()"

Example “name filter”. Get the names and frequencies of elements which have “section” child elements - section, sect1, sect2, ...

fox "docbook//\*.xml\\\*[child('sect\*')]\name() => f()"

Example “option name”. Get the names and frequencies of docbook elements containing mml elements.

fox "docbook//\*.xml\\docbook:\*[child('mml:\*', 'name')]\name() => f()"

### descendant (\*-ec)

**descendant**($nameFilter as xs:string := (),

$options as xs:string := ())

as node()\*

**descendant-ec**(

$inputItems as item()\*,

$nameFilter as xs:string := (),

$options as xs:string := ())

as node()\*

**Summary**

Returns descendant element nodes, optionally filtered by node name and/or position.

**Details**

Input items can be nodes and/or atomic items. Atomic input items are interpreted as document URI and replaced with the corresponding document node. The function returns the descendant elements of the input nodes, optionally filtered by name and/or by position.

Function variant descendant-ec receives input items as the value of the first parameter. Function variant descendant processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified String Expression](#_Unified_String_Expression). By default, the filtering is applied to the local names of nodes.

Further processing details are controlled by *options.* They are provided as option names separated by whitespace.

Options qname, jname, name and lname control the kind of node name to which name filtering is applied – qualified name, JSON name, lexical name or local name (default). Note that namespace aware filtering requires option qname. Using option name, filtering is applied to the lexical name, which may contain a prefix, but without considering the namespace URI. Using option jname, filtering is applied to the original JSON field name, rather than its XML representation. For details, see [Unified String Expression](#_Unified_String_Expression).

Options first, first2, last, last2 specifiy a positional filter: for each input node, only the first (second, last, second last) result node is returned. The positional filter is applied after a name filter.

***Parameters***

Described by the following table.

**Table**. Parameters of function descendant and descendant-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are interpreted as document URI and replaced with the corresponding document node. |
| nameFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| options | Whitespace-separated list of options.  *Options group 1* – node name type to which the name filter is applied:  lname – local names (default)  qname – qualified names  jname – JSON names  name – lexical names  *Options group 2* – positional filter; for each input node, at most one result node is returned, selected by its position in reverse document order:  first – for each input node, return only the first result node  first2 – for each input node, return only the second result node  last – for each input node, return only the last result node  last2 – for each input node, return only the second last result node |

***Examples***

Example “no filter”. Get the names and frequencies of the descendant elements of docbook table elements.

fox "ox\*ples/docbook//\*.xml\\docbook:table\descendant()\name() => f()"

Example “name filter”. Get the data paths of "list" elements (itemizedlist, orderedlist).

fox "ox\*ples/docbook//\*.xml/descendant('\*list')/name-path() => f()"

Example “option qname”. Get the names and frequencies of docbook "equation" elements with a math ML child element.

fox "declare namespace math='http://www.w3.org/1998/Math/MathML';  
 ox\*ples/docbook//\*.xml/descendant('docbook:\*equation\*', 'qname')

[child('math:\*', 'qname')]/name-path() => f()"

Example “option jname”. Inspecting a set of JSON documents - get the names and frequencies of elements with a name containing the "/" character, the "#" character or a blank.

fox "json/\*.json/descendant('\*/\* ##\* \*\s\*', 'jname')\jname() => f()"

Example “option name”. Get the names and frequencies of elements with an mml prefix. Note that the selection is based on the use of name prefixes and not affected by the namespace of the element names.

fox "ox\*ples/docbook//\*.xml/descendant('mml:\*', 'name')/name() => f()"

### descendant-or-self (\*-ec)

**descendant-or-self**($nameFilter as xs:string := (),

$options as xs:string := ())

as node()\*

**descendant-or-self-ec**(

$inputItems as item()\*,

$nameFilter as xs:string := (),

$options as xs:string := ())

as node()\*

**Summary**

Returns descendant-or-self nodes, optionally filtered by node name and/or position.

**Details**

Input items can be nodes and/or atomic items. Atomic input items are interpreted as document URI and replaced with the corresponding document node. The function returns the input nodes and their descendant elements, optionally filtered by name and/or by position.

Function variant descendant-or-self-ec receives input items as the value of the first parameter. Function variant descendant-or-self processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)).

Further processing details are controlled by *options.* They are provided as option names separated by whitespace.

Options name, jname and lname control the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default). For details, see [Node name types](#_Node_name_types).

Options first, first2, last, last2 specifiy a positional filter: for each input node, only the first (second, last, second last) result node is returned. The positional filter is applied after a name filter.

***Parameters***

Described by the following table.

**Table**. Parameters of function descendant-or-self and descendant-or-self-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are interpreted as document URI and replaced with the corresponding document node. |
| nameFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| options | Whitespace-separated list of options.  *Options group 1* – node name type to which the name filter is applied:  lname – local names (default)  jname – JSON names  name – lexical names  *Options group 2* – positional filter; for each input node, at most one result node is returned, selected by its position in reverse document order:  first – for each input node, return only the first result node  first2 – for each input node, return only the second result node  last – for each input node, return only the last result node  last2 – for each input node, return only the second last result node |

**Examples**

Example “no filter”. Example “no filter”. Inspecting docbook documents - get the names and frequencies of "list" elements (itemizedlist, orderedlist) and their descendant elements.

fox "docbook//\*.xml\descendant('\*list')\descendant-or-self()\name() => f()"

Example “name filter”. Inspecting a set of XML documents - get the names and frequencies of elements with a name containing the string "object". The name is rendered in Clark notation.

fox ".//\*.xml\descendant-or-self('\*object\*')\clark-name() => f()"

Example “option jname”. Inspecting a set of JSON documents - get the names and frequencies of fields with a name matching \*geo\* or contained by such an element, and with a name containing a blank or a slash.

fox "../json/\*.json\\descendant('\*geo\*')\descendant-or-self('\*/\* \*\s\*', 'jname')\jname() => f()"

### following-sibling (\*-ec)

**following-sibling**($namesFilter as xs:string := (),

$pselecter as xs:integer := (),

$options as xs:string := ())

as node()\*

**following-sibling-ec**(

$inputItems as item()\*,

$namesFilter as xs:string := (),

$pselector as xs:integer := (),

$options as xs:string := ())

as node()\*

**Summary**

Returns following-sibling element nodes, optionally filtered by name and/or position.

**Details**

Input items can be nodes and/or atomic items, but atomic items are ignored, as they are interpreted as document URI and the corresponding document node cannot have sibling nodes. The function returns the following-sibling elements of the input nodes, optionally filtered by name and/or by position.

Function variant following-sibling-ec receives input items as the value of the first parameter. Function variant following-sibling processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)).

A *positional filter* selects at most one result node *per input node*, which is found at the position given by the parameter value, in document order. A negative parameter value is a position counted from the last item backward, with -1, -2, … selecting the last item, second last item, etc.

*Options* are provided as option names separated by whitespace. Supported option names are name, jname and lname, controling the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default).

***Parameters***

Described by the following table.

**Table**. Parameters of function following-sibling and following-sibling-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are ignored. |
| namesFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input item*, only the result node at corresponding position in document order will be returned |
| options | Possible values: name, jname, lname. The name filter is applied to the corresponding kind of node name – lexical name, JSON name or local name. By default, the name filter is applied to local names. |

**Examples**

….

### parent (\*-ec)

**parent**($namesFilter as xs:string := (),

$pselecter as xs:integer := (),

$options as xs:string := ())

as node()\*

**parent-ec**(

$inputItems as item()\*,

$namesFilter as xs:string := (),

$pselector as xs:integer := (),

$options as xs:string := ())

as node()\*

**Summary**

Returns parent nodes, optionally filtered by name and/or position.

**Details**

Input items can be nodes and/or atomic items, but atomic items are ignored, as they are interpreted as document URI and the corresponding document node cannot have parent nodes. The function returns the parent nodes of the input nodes, optionally filtered by name and/or by position.

Function variant parent-ec receives input items as the value of the first parameter. Function variant parent processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)).

A *positional filter* selects at most one result node *per input node*, which is found at the position given by the parameter value, in reverse document order. A negative parameter value is a position counted from the last item backward, with -1, -2, … selecting the last item, second last item, etc.

*Options* are provided as option names separated by whitespace. Supported option names are name, jname and lname, controling the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default).

***Parameters***

Described by the following table.

**Table**. Parameters of function parent and parent-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are ignored. |
| namesFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input item*, only the result node at corresponding position in document order will be returned. |
| options | Possible values: name, jname, lname. The name filter is applied to the corresponding kind of node name – lexical name, JSON name or local name. By default, the name filter is applied to local names. |

**Examples**

….

### preceding-sibling (\*-ec)

**preceding-sibling**($namesFilter as xs:string := (),

$pselecter as xs:integer := (),

$options as xs:string := ())

as node()\*

**preceding-sibling-ec**(

$inputItems as item()\*,

$namesFilter as xs:string := (),

$pselector as xs:integer := (),

$options as xs:string := ())

as node()\*

**Summary**

Returns preceding-sibling element nodes, optionally filtered by name and/or position.

**Details**

Input items can be nodes and/or atomic items, but atomic items are ignored, as they are interpreted as document URI and the corresponding document node cannot have sibling nodes. The function returns the preceding-sibling elements of the input nodes, optionally filtered by name and/or by position.

Function variant preceding-sibling-ec receives input items as the value of the first parameter. Function variant preceding-sibling processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)).

A *positional filter* selects at most one result node *per input node*, which is found at the position given by the parameter value, in reverse document order. A negative parameter value is a position counted from the last item backward, with -1, -2, … selecting the last item, second last item, etc.

*Options* are provided as option names separated by whitespace. Supported option names are name, jname and lname, controling the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default).

***Parameters***

Described by the following table.

**Table**. Parameters of function preceding-sibling and preceding-sibling-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are ignored. |
| namesFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input item*, only the result node at corresponding position in reverse document order will be returned |
| options | Possible values: name, jname, lname. The name filter is applied to the corresponding kind of node name – lexical name, JSON name or local name. By default, the name filter is applied to local names. |

**Examples**

….

### self (\*-ec)

**self**($namesFilter as xs:string := (),

$pselecter as xs:integer := (),

$options as xs:string := ())

as node()\*

**self-ec**(

$inputItems as item()\*,

$namesFilter as xs:string := (),

$pselector as xs:integer := (),

$options as xs:string := ())

as node()\*

**Summary**

Returns self nodes, optionally filtered by name and/or position.

**Details**

Input items can be nodes and/or atomic items. Atomic input items are interpreted as document URI and replaced with the corresponding document node. The function returns the input nodes, optionally filtered by name and/or by position.

Function variant self-ec receives input items as the value of the first parameter. Function variant self processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)).

A *positional filter* selects at most one result node *per input node*, which is found at the position given by the parameter value, in document order. A negative parameter value is a position counted from the last item backward, with -1, -2, … selecting the last item, second last item, etc.

*Options* are provided as option names separated by whitespace. Supported option names are name, jname and lname, controling the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default).

***Parameters***

Described by the following table.

**Table**. Parameters of function self and self-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are interpreted as document URI and replaced with the corresponding document node. |
| namesFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input item*, only the result nodes at corresponding position in document order will be returned |
| options | Possible values: name, jname, lname. The name filter is applied to the corresponding kind of node name – lexical name, JSON name or local name. By default, the name filter is applied to local names. |

**Examples**

….

## Navigation aids 2 – compound axes

The functions in this section enable enhanced node tree navigation. Each function selects nodes from a “compound” navigation axis, which is a combination of standard axes. An example is the content axis, containing the node and its descendants together with their attributes.

Nodes are selected by a [Unified String Expression](#_Unified_String_Expression) applied to the node name. By default, the selection is applied to the local name of the nodes. Options trigger alternatives:

* option qname – name filtering is applied to the qualified node names
* option jname – name filtering is applied to the JSON field names
* optione name – name filtering is applied to the lexical nodes names

### content (\*-ec)

**content**($namesFilter as xs:string := (),

$pselecter as xs:integer := (),

$options as xs:string := ())

as node()\*

**content-ec**(

$inputItems as item(),

$namesFilter as xs:string := (),

$pselector as xs:integer := (),

$options as xs:string := ())

as node()\*

**Summary**

Returns descendant elements and their attributes, optionally filtered by name and/or position.

**Details**

Input items can be nodes and/or atomic items. Atomic input items are interpreted as document URI and replaced with the corresponding document node. The function returns nodes which are descendant elements of the input items, as well as the attributes of descendant elements. The nodes returned by the function can be filtered by name and/or by position.

Function variant all-descendant-ec receives input items as the value of the first parameter. Function variant all-descendant processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Unified_Filter_Expression). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)).

A *positional filter* selects at most one result node *per input node*, which is found at the position given by the parameter value, in document order. A negative parameter value is a position counted from the last item backward, with -1, -2, … selecting the last item, second last item, etc.

*Options* are provided as option names separated by whitespace. Supported option names are name, jname and lname, controling the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default).

***Parameters***

Described by the following table.

**Table**. Parameters of function all-descendant and all-descendant-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are interpreted as document URI and replaced with the corresponding document node. |
| namesFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input item*, only the result node at corresponding position in document order will be returned. |
| options | Possible values: name, jname, lname. The name filter is applied to the corresponding kind of node name – lexical name, JSON name or local name. By default, the name filter is applied to local names. |

**Examples**

Gets the element and attribute paths in selected files, along with their frequences.

fox "\*stud\*.xml\**all-descendant**()\name-path() => frequencies()"

Returns the names of files with XML content and containing an element or attribute with a name containing “font”.

fox "../output-convert-mass/\*fibook.xml[all-descendant('\*font\*')]"

### content-or-self (\*-ec)

**content-or-self**($namesFilter as xs:string := (),

$pselecter as xs:integer := (),

$options as xs:string := ())

as node()\*

**content-or-self-ec**(

$inputItems as item()\*,

$namesFilter as xs:string := (),

$pselector as xs:integer := (),

$options as xs:string := ())

as node()\*

**Summary**

Returns descendant-or-self nodes and their attributes, optionally filtered by name and/or position.

**Details**

Input items can be nodes and/or atomic items. Atomic input items are interpreted as document URI and replaced with the corresponding document node. The function returns the input nodes and their descendant elements, as well as the attributes of input nodes and descendant elements, optionally filtered by name and/or by position.

Function variant all-descendant-ec receives input items as the value of the first parameter. Function variant all-descendant processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Unified_Filter_Expression). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)).

A *positional filter* selects at most one result node *per input node*, which is found at the position given by the parameter value, in document order. A negative parameter value is a position counted from the last item backward, with -1, -2, … selecting the last item, second last item, etc.

*Options* are provided as option names separated by whitespace. Supported option names are name, jname and lname, controling the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default).

***Parameters***

Described by the following table.

**Table**. Parameters of function all-descendant-or-self and all-descendant-or-self-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are interpreted as document URI and replaced with the corresponding document node. |
| namesFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input item*, only the result node at corresponding position in document order will be returned |
| options | Possible values: name, jname, lname. The name filter is applied to the corresponding kind of node name – lexical name, JSON name or local name. By default, the name filter is applied to local names. |

**Examples**

Gets the element and attribute paths in selected files, along with their frequences.

fox "\*stud\*.xml\**all-descendant-or-self**()\name-path() => frequencies()"

### sibling (\*-ec)

**sibling**($namesFilter as xs:string := (),

$pselecter as xs:integer := (),

$options as xs:string := ())

as node()\*

**sibling-ec**(

$inputItems as item()\*,

$namesFilter as xs:string := (),

$pselector as xs:integer := (),

$options as xs:string := ())

as node()\*

**Summary**

Returns sibling element nodes, optionally filtered by name and/or position.

**Details**

Input items can be nodes and/or atomic items, but atomic items are ignored, as they are interpreted as document URI and the corresponding document node cannot have sibling nodes. The function returns the sibling elements of the input nodes, optionally filtered by name and/or by position.

Function variant sibling-ec receives input items as the value of the first parameter. Function variant sibling processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). By default, the filtering is applied to the local names of nodes. If option name is used, filtering is applied to lexical names, which may include a name prefix. If option jname is used, filtering is applied to JSON names (see [Node name types](#_Node_name_types)).

A *positional filter* selects at most one result node *per input node*, which is found at the position given by the parameter value, in document order. A negative parameter value is a position counted from the last item backward, with -1, -2, … selecting the last item, second last item, etc.

*Options* are provided as option names separated by whitespace. Supported option names are name, jname and lname, controling the kind of node name to which name filtering is applied – lexical name, JSON name or local name (default).

***Parameters***

Described by the following table.

**Table**. Parameters of function sibling and sibling-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  The input items to be evaluated. Atomic items are ignored. |
| namesFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input item*, only the result node at corresponding position in document order will be returned |
| options | Possible values: name, jname, lname. The name filter is applied to the corresponding kind of node name – lexical name, JSON name or local name. By default, the name filter is applied to local names. |

**Examples**

….

### child-text (\*-ec)

**child-text**($options as xs:string := ())

as xs:string?

**child-text-ec(**

$elems as element(),

$options as xs:string := ())

as xs:string?

**Summary**

Returns the concatenated text of the text nodes immediately contained by given elements.

**Details**

The text node values are concatenated without separating character.

If option ign-wsonly is used, only text nodes containing non-whitespace are considered.

***Parameters***

Described by the following table.

**Table**. Parameters of function child-text and child-text-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| elems | NOTE: this parameter is only expected by function variant \*-ec.  The input elements to be evaluated. Items which are not elements are ignored. |
| options | Possible values: ign-wsonly.  ign-wsonly – text nodes containing only whitespace are ignored |

**Hint**

The function can be used in order to select “leaf elements”, containing text, using a predicate:

[child-text('ign-wsonly')]

However, if mixed content can be excluded, a simpler filter would be:

[not(\*)]

**Examples**

Return the frequency distribution of the name paths of all elements containing text:

fox "octo\*.xml\\\*[child-text('ign-wsonly')]\name-path((), 'value') => freq()"

/octocheck/control/inputFolder=../../input-data/cloud-convert... (1)

/octocheck/control/outputFolder=output.......................... (1)

/octocheck/control/reportFolder=output.......................... (1)

…

## File system navigation

The functions in this section support a more concise expression of complex file system navigation. Resources are selected by a [Unified String Expression](#_Unified_String_Expression) applied to the file name.

### fancestor (\*-ec)

**fancestor**($names as xs:string?,

$pselector as item()?)

as xs:string\*

**fancestor**($names as xs:string?)

as xs:string\*

**fancestor**()

as xs:string\*

**fancestor-ec**(

$uris as xs:string\*,

$names as xs:string?,

$pselector as item()?)

as xs:string\*

…

***Summary***

Returns ancestor URIs, optionally filtered by file name and/or position.

***Details***

The function returns the ancestor URIs of the input URIs, optionally filtered by name and/or by position. Duplicate URIs are removed. Result URIs are returned in file system order.

Function variant fancestor-ec receives input URIs as the value of the first parameter. Function variant fancestor processes a single input URI, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). When used, only URIs with a matching file or folder name are returned.

A *positional filter* selects at most one result URI *per input URI*, which is found at the position given by the parameter value, in reverse file system order. A negative parameter value is a position counted from the last item (in reverse file system order) backward, with -1, -2, … meaning the last item, second last item, etc.

***Parameters***

Described by the following table.

**Table**. Parameters of function fancestor and fancestor-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uris | NOTE: this parameter is only expected by function variant \*-ec.  Input URIs |
| names | Resource name filter selecting result URIs. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input URI,* only the result URI at corresponding position in reverse file system order will be returned |

**Examples**

Get the first ancestor URI with a file name matching proj\*.

fox "../\*config\*/anchor\*.xml/fancestor('proj\*', 1) => distinct-values()"

### fancestor-or-self (\*-ec)

**fancestor-or-self**($names as xs:string?,

$pselector as item()?)

as xs:string\*

**fancestor-or-self**($names as xs:string?)

as xs:string\*

**fancestor-or-self**()

as xs:string\*

**fancestor-or-self-ec**(

$uris as xs:string\*,

$names as xs:string?,

$pselector as item()?)

as xs:string\*

…

***Summary***

Returns ancestor-or-self URIs, optionally filtered by file name and/or position.

***Details***

The function returns the input URIs and their ancestor URIs, optionally filtered by name and/or by position. Duplicate URIs are removed. Result URIs are returned in file system order.

Function variant fancestor-or-self-ec receives input URIs as the value of the first parameter. Function variant fancestor-or-self processes a single input URI, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). When used, only URIs with a matching file or folder name are returned.

A *positional filter* selects at most one result URI *per input URI*, which is found at the position given by the parameter value, in reverse file system order. A negative parameter value is a position counted from the last item (in reverse file system order) backward, with -1, -2, … meaning the last item, second last item, etc.

***Parameters***

Described by the following table.

**Table**. Parameters of function fancestor-or-self and fancestor-or-self-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uris | NOTE: this parameter is only expected by function variant \*-ec.  Input URIs |
| names | Resource name filter selecting result URIs. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input URI,* only the result URI at corresponding position in reverse file system order will be returned |

**Examples**

Find folders containing a file anchor-config.xml and return the first ancestor-or-self URI with a folder name matching proj\*.

fox ".//\*[anchor-config.xml]/fancestor-or-self('proj\*', 1) => distinct-values()"

### fchild (\*-ec)

**fchild**($names as xs:string?,

$pselector as item()?)

as xs:string\*

**fchild**($names as xs:string?)

as xs:string\*

**fchild**()

as xs:string\*

**fchild-ec**(

$uris as xs:string\*,

$names as xs:string?,

$pselector as item()?)

as xs:string\*

…

***Summary***

Returns child URIs, optionally filtered by file name and/or position.

***Details***

The function returns the child URIs of the input URIs, optionally filtered by name and/or by position. Duplicate URIs are removed. Result URIs are returned in file system order.

Function variant child-ec receives input URIs as the value of the first parameter. Function variant child processes a single input URI, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). When used, only URIs with a matching file or folder name are returned.

A *positional filter* selects at most one result URI *per input URI*, which is found at the position given by the parameter value, in file system order. A negative parameter value is a position counted from the last item (in file system order) backward, with -1, -2, … meaning the last item, second last item, etc.

***Parameters***

Described by the following table.

**Table**. Parameters of function fchild and fchild-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uris | NOTE: this parameter is only expected by function variant \*-ec.  Input URIs |
| names | Resource name filter selecting result URIs. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input URI,* only the result URI at corresponding position in file system order will be returned |

**Examples**

Get the child URIs with a file or folder name matching sap-\*., yet not matching \*-201? or \*-200?.

fox "/projects/fchild('sap-\* ~\*-201? ~\*-200?')"

### fdescendant (\*-ec)

**fdescendant**($names as xs:string?,

$pselector as item()?)

as xs:string\*

**fdescendant**($names as xs:string?)

as xs:string\*

**fdescendant**()

as xs:string\*

**fdescendant-ec**(

$uris as xs:string\*,

$names as xs:string?,

$pselector as item()?)

as xs:string\*

…

***Summary***

Returns descendant URIs, optionally filtered by file name and/or position.

***Details***

The function returns the descendant URIs of the input URIs, optionally filtered by name and/or by position. Duplicate URIs are removed. Result URIs are returned in file system order.

Function variant fdescendant-ec receives input URIs as the value of the first parameter. Function variant fdescendant processes a single input URI, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). When used, only URIs with a matching file or folder name are returned.

A *positional filter* selects at most one result URI *per input URI*, which is found at the position given by the parameter value, in file system order. A negative parameter value is a position counted from the last item (in file system order) backward, with -1, -2, … meaning the last item, second last item, etc.

***Parameters***

Described by the following table.

**Table**. Parameters of function fdescendant and fdescendant-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uris | NOTE: this parameter is only expected by function variant \*-ec.  Input URIs |
| names | Resource name filter selecting result URIs. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input URI,* only the result URI at corresponding position in file system order will be returned |

**Examples**

Get the descendant URIs of files with a name matching \*.xsl or \*.xsd, yet not matching tmp\* or \*scratch\*.

fox "/projects/london/fdescendant('\*.xsl \*.xsd ~tmp\* ~\*scratch\*')"

### fdescendant-or-self (\*-ec)

**fdescendant-or-self**($names as xs:string?,

$pselector as item()?)

as xs:string\*

**fdescendant-or-self**($names as xs:string?)

as xs:string\*

**fdescendant-or-self**()

as xs:string\*

**fdescendant-or-self-ec**(

$uris as xs:string\*,

$names as xs:string?,

$pselector as item()?)

as xs:string\*

…

***Summary***

Returns descendant-or-self URIs, optionally filtered by file name and/or position.

***Details***

The function returns the input URIs and their descendant URIs, optionally filtered by name and/or by position. Duplicate URIs are removed. Result URIs are returned in file system order.

Function variant fdescendant-or-self-ec receives input URIs as the value of the first parameter. Function variant fdescendant-or-self processes a single input URI, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). When used, only URIs with a matching file or folder name are returned.

A *positional filter* selects at most one result URI *per input URI*, which is found at the position given by the parameter value, in file system order. A negative parameter value is a position counted from the last item (in file system order) backward, with -1, -2, … meaning the last item, second last item, etc.

***Parameters***

Described by the following table.

**Table**. Parameters of function fdescendant-or-self and fdescendant-or-self-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uris | NOTE: this parameter is only expected by function variant \*-ec.  Input URIs |
| names | Resource name filter selecting result URIs. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input URI,* only the result URI at corresponding position in file system order will be returned |

**Examples**

Find folders containing WSDLs and return these folders along with descendant folders, filtered by name \*apidef\*.

fox "/projects//\*[\*.wsdl]/fdescendant-or-self('\*apidef\*')"

### ffollowing-sibling (\*-ec)

**ffollowing-sibling**($names as xs:string?,

$pselector as item()?)

as xs:string\*

**ffollowing-sibling**($names as xs:string?)

as xs:string\*

**ffollowing-sibling**()

as xs:string\*

**ffollowing-sibling-ec**(

$uris as xs:string\*,

$names as xs:string?,

$pselector as item()?)

as xs:string\*

…

***Summary***

Returns following-sibling URIs, optionally filtered by file name and/or position.

***Details***

The function returns the following-sibling URIs of the input URIs, optionally filtered by name and/or by position. Duplicate URIs are removed. Result URIs are returned in file system order.

Function variant ffollowing-sibling-ec receives input URIs as the value of the first parameter. Function variant ffollowing-sibling processes a single input URI, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). When used, only URIs with a matching file or folder name are returned.

A *positional filter* selects at most one result URI *per input URI*, which is found at the position given by the parameter value, in file system order. A negative parameter value is a position counted from the last item (in file system order) backward, with -1, -2, … meaning the last item, second last item, etc.

***Parameters***

Described by the following table.

**Table**. Parameters of function ffollowing-sibling and ffollowing-sibling-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uris | NOTE: this parameter is only expected by function variant \*-ec.  Input URIs |
| names | Resource name filter selecting result URIs. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input URI,* only the result URI at corresponding position in file system order will be returned |

**Examples**

Get the following sibling files of the last file with a name containing a date in May.

fox "report.202205\*[last()]/ffollowing-sibling()"

Get the following sibling files matching a name pattern.

fox "report.20220501.xml/ffollowing-sibling('\*202206\*')"

### fparent (\*-ec)

**fparent**(…)

as xs:string\*

***Summary***

…

### fparent-sibling (\*-ec)

**fparent-sibling**(…)

as xs:string\*

…

### fpreceding-sibling (\*-ec)

**fpreceding-sibling**(…)

as xs:string\*

…

### fself (\*-ec)

**fself**($names as xs:string?,

$namesExcluded as xs:string,

$pselector as item()?,

$caseSensitive as xs:boolean?)

as node()\*

**…**

***Summary***

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.

### fsibling (\*-ec)

**fsibling**($names as xs:string?,

$pselector as item()?)

as xs:string\*

**fsibling**($names as xs:string?)

as xs:string\*

**fsibling**()

as xs:string\*

**fsibling-ec**(

$uris as xs:string\*,

$names as xs:string?,

$pselector as item()?)

as xs:string\*

…

***Summary***

Returns sibling URIs, optionally filtered by file name and/or position.

***Details***

The function returns the sibling URIs of the input URIs, optionally filtered by name and/or by position. Duplicate URIs are removed. Result URIs are returned in file system order.

Function variant fsibling-ec receives input URIs as the value of the first parameter. Function variant fsibling processes a single input URI, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). When used, only URIs with a matching file or folder name are returned.

A *positional filter* selects at most one result URI *per input URI*, which is found at the position given by the parameter value, in file system order. A negative parameter value is a position counted from the last item (in file system order) backward, with -1, -2, … meaning the last item, second last item, etc.

***Parameters***

Described by the following table.

**Table**. Parameters of function fsibling and fsibling-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| Uris | NOTE: this parameter is only expected by function variant \*-ec.  Input URIs |
| names | Resource name filter selecting result URIs. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input URI,* only the result URI at corresponding position in file system order will be returned |

**Examples**

Get all sibling file names.

fox "config.xml/fsibling()/file-name()"

Get the file names of siblings \*.xml, excluding \*tmp and \*scratch\*.

fox "config.xml/fsibling('\*.xml ~tmp\* ~\*scratch\*')/file-name()"

### fparent-shifted (\*-ec)

**fparent-shifted**($shiftedParent as xs:string,

$nameReplaceSubstring as xs:string?,

$nameReplaceWith as xs:string?)

as xs:string\*

**fparent-shifted-ec**($contextUris as xs:string\*,

shiftedParent as xs:string,

$nameReplaceSubstring as xs:string?,

$nameReplaceWith as xs:string?)

as xs:string\*

***Summary***

Returns URIs of resources found in the shifted parent folder, with equal or related names.

***Details***

Bla

***Parameters***

Described by the following table.

**Table**. Parameters of function fparent-shifted.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| contextUris | [Parameter only used by variants ec-\*] The context URIs to be mapped to URIs of resources under the shifted parent |
| shiftedParent | URI of shifted parent, or Foxpath expression returning the shifted parent URI (Foxpath in {}) |
| nameReplaceSubstring | The name of the returned resource is obtained by replacing this substring with the value found in $nameReplaceWith |
| nameReplaceWith | The name of the returned resource is obtained by replacing the substring specified by $nameReplaceSubstring with this value |

**Examples**

Bla

### bsibling (\*-ec)

**bsibling**($names as xs:string?,

$pselector as item()?)

as xs:string\*

**bsibling**($names as xs:string?)

as xs:string\*

**bsibling**()

as xs:string\*

**bsibling-ec**(

$nodes as item()\*,

$names as xs:string?,

$pselector as item()?)

as xs:string\*

…

***Summary***

Returns sibling URIs of the files containing the input nodes.

***Details***

The function returns the sibling URIs of the files containing the input nodes, optionally filtered by name and/or by position. Duplicate URIs are removed. Result URIs are returned in file system order.

For every input item which is a node, sibling URIs of the file containing the node are returned. Atomic input items are interpreted as URIs, sibling URIs of which are returned.

Function variant bsibling-ec receives input items as the value of the first parameter. Function variant bsibling processes a single input item, which is the context item of the function call (for more information see [ec – variant](#_ec_–_variant)).

The *name filter* is a [Unified Filter Expression](#_Filter_syntax_1). When used, only URIs with a matching file or folder name are returned.

A *positional filter* selects at most one result URI *per input item*, which is found at the position given by the parameter value, in file system order. A negative parameter value is a position counted from the last item (in file system order) backward, with -1, -2, … meaning the last item, second last item, etc.

***Parameters***

Described by the following table.

**Table**. Parameters of function bsibling and bsibling-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| Uris | NOTE: this parameter is only expected by function variant \*-ec.  Input items, which can be nodes or URIs. |
| names | Resource name filter selecting result URIs. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| pselector | An integer number. *For each input URI,* only the result URI at corresponding position in file system order will be returned |

**Examples**

Navigate from a node to the sibling files of the file containing the node.

fox "config.xml\\service[1]\bsibling()/file-name()"

Navigate from nodes to the sibling files of the files containing the nodes.

fox ".//config.xml\\service => bsibling-ec('service-\*')"

Navigate from a node to content in a sibling file of the file containing the node.

fox "config.xml\\service\bsibling('service-'||replace('-service', '.xml'))\\endpoint\string()"

## Doc functions

The functions in this section parse files into XDM node trees. The file is represented by its resource URI.

### doc

**doc**($uri as xs:string)

as document-node()?

**doc**()

as document-node()?

***Summary***

Parses a file into an XDM node tree.

***Details***

The file need not be an XML document – the appropriate parsing approach is selected in accordance to the [ispace definition](#_Infospace_definition_document).

***Parameters***

Described by the following table.

**Table**. Parameters of function doc.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | Document URI.  Note: if omitted, the parameter defaults to the context item. |

***Examples***

Read data from a JSON file.

fox "frameworks/tei/xml/tei/odd/p5subset\_it.json/doc(.)\\ident => freq()"

### idoc

**idoc**($uri as xs:string,

$grammar as xs:string)

as document-node()?

**idoc**($grammar as xs:string)

as document-node()?

***Summary***

Parses a file into an XDM node tree, using an ixml grammar.

***Details***

The grammar can be specified by resource URI or by the name assigned to it in the [infospace definition](#_Unified_String_Expression) definition.

***Parameters***

Described by the following table.

**Table**. Parameters of function idoc.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | Document URI.  Note: if omitted, the parameter defaults to the context item. |
| grammar | The name or resource URI of a grammar. If a name is supplied (example: #xpath31), it must be prefixed by a # character, followed by a grammar name found in a <grammar> element of the [Infospace definition](#_Unified_String_Expression). |

***Examples***

Read data from an CSS file.

fox "samples//\*.css/idoc('#css')\\declaration[property eq 'color'}\\expr"

### json-doc, jdoc

**json-doc**($uri as xs:string)

as document-node()?

**json-doc**()

as document-node()?

***Summary***

Parses a JSON document into an XDM node tree.

***Details***

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

The node tree corresponds to the node tree returned by the BaseX processor when calling function [json:doc](https://docs.basex.org/main/JSON_Functions#json:doc) with option format equal direct. For the reader’s convenience, the rules are repeated here:

* The resulting document has a json root node.
* Object pairs are represented via elements. The name of a pair is encoded, as described for the [Conversion Functions](https://docs.basex.org/main/Conversion_Functions#keys), and used as element name.
* Array entries are also represented via elements, with \_ as element name.
* Object and array values are stored in text nodes.
* The types of values are represented via type attributes:
  + The existing types are *string*, *number*, *boolean*, *null*, *object*, and *array*
  + As most values are strings, the *string* type is by default omitted.

***Parameters***

Described by the following table.

**Table**. Parameters of function json-doc.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | Document URI.  Note: if omitted, the parameter defaults to the context item. |

***Examples***

Read data from a JSON document.

fox "frameworks/tei/xml/tei/odd/p5subset\_it.json/json-doc(.)\\ident => freq()"

### html-doc, hdoc

**html-doc**($uri as xs:string)

as document-node()?

**html-doc**()

as document-node()?

***Summary***

Parses an HTML document into an XDM node tree.

***Details***

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

The node tree corresponds to the node tree returned by the BaseX processor when calling function [html:doc](https://docs.basex.org/main/HTML_Functions#html:doc). It uses the tagsoup parser.

***Parameters***

Described by the following table.

**Table**. Parameters of function html-doc.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | Document URI.  Note: if omitted, the parameter defaults to the context item. |

***Examples***

Read data from an HTML document.

fox "frameworks/dita//considerations.html/hdoc()\\(h1, h2)"

### csv-doc, cdoc (\*-ec)

**csv-doc**($separatoras xs:string,

$header as xs:boolean)

as document-node()?

**csv-doc**($separator as xs:string)

as document-node()?

**csv-doc**()

as document-node()?

**csv-doc-ec**($uri as xs:string,

$separator as xs:string,

$header as xs:boolean)

as document-node()?

…

*Abbreviations - t*he function name can be abbreviated:

cdoc

cdoc-ec

***Summary***

Parses a CSV document into an XDM node tree.

***Details***

If the argument is omitted, it defaults to the context item. By default, the field delimiter is a comma and the first record is treated as a data record, not as a table header.

***Parameters***

Described by the following table.

**Table**. Parameters of function csv-doc, csv-doc-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | Document URI.  Parameter only used by function csv-doc-ec. |
| separator | The character used as field delimiter. Default value: a comma. The character can be supplied literally, or via one of the following names: comma, semicolon, colon, space. |
| header | If 'yes ', the first record is interpreted as a table header, not a data record. |

***Examples***

Read data from a CSV document. Field delimiter is the comma, and no table header expected.

fox "samples//\*.csv/cdoc()"

As before, but expecting a table header.

fox "samples//\*.csv/cdoc((), 'yes')"

Read data from a CSV document. Field delimiter is a semicolon, and no table header expected.

fox "samples//\*.csv/cdoc('semicolon ')"

### docx-doc, docx

**docx-doc**($uri as xs:string)

as document-node()?

**docx-doc**()

as document-node()?

***Summary***

Parses an MS Office Word document into an XDM node tree.

***Details***

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

The function parses the document found in the docx archive at the path word/document.

***Parameters***

Described by the following table.

**Table**. Parameters of function docx-doc.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | Document URI.  Note: if omitted, the parameter defaults to the context item. |

***Examples***

Read data from a docx document.

fox "frameworks/dita//\*.docx/docx()\\w:r\string()"

## Base URI, base file, base dir

The functions in this section return the base URI or the file path or file name of the file or folder containing nodes.

### base-dir (\*-ec)

**base-dir**()

as xs:string\*

**base-dir-ec**($inputItems as item()\*)

as xs:string\*

***Summary***

Returns for each input node the normalized file path of the folder containing it.

***Details***

…

***Params***

…

***Examples***

…

### base-dir-name (\*-ec)

**base-dir-name**()

as xs:string\*

**base-dir-name-ec**($inputItems as item()\*)

as xs:string\*

***Summary***

Returns for each input node the name of the folder containing it.

***Details***

…

***Params***

…

***Examples***

…

### base-dir-rel(\*-ec)

**base-dir-rel**($pathContext as xs:string := ())

as node()\*

**base-dir-rel-ec**(

$inputItems as item()\*,

$pathContext as xs:string := ())

as node()\*

***Summary***

Returns for each input node the relative file path of the folder containing it.

***Details***

Tile file context for which the relative file paths are determined defaults to the current working directory. The context can be specified explicitly as a name filter, selecting the closest containing folder with a name matching the filter.

Note: the difference between this function and base-uri-rel() is subtle: this function does not URI-escape the path steps, whereas base-uri-rel() does. Example:

$node/base-dir-rel() => Oxygen XML Editor 25/frameworks/tei/README.txt

$node/base-uri-rel() => Oxygen%20XML%20Editor%2025/frameworks/tei/README.txt

***Params***

…

***Examples***

…

### base-file (\*-ec)

**base-file**()

as xs:string\*

**base-file-ec**($inputItems as item()\*)

as xs:string\*

***Summary***

Returns for each input node the normalized file path of the file containing it.

***Details***

…

***Params***

…

***Examples***

…

### base-file-name (\*-ec)

**base-file-name**()

as xs:string\*

**base-file-name-ec**($inputItems as item()\*)

as xs:string\*

***Summary***

Returns for each input node the name of the file containing it.

***Details***

…

***Params***

…

***Examples***

…

### base-file-rel(\*-ec)

**base-file-rel**($pathContext as xs:string := ())

as node()\*

**base-file-rel-ec**(

$inputItems as item()\*,

$pathContext as xs:string := ())

as node()\*

***Summary***

Returns for each input node the relative file path of the file containing it.

***Details***

The file context for which the relative file paths are determined defaults to the current working directory. The context can be specified explicitly as a name filter, selecting the closest containing folder with a name matching the filter.

***Params***

…

***Examples***

…

### base-uri-rel(\*-ec)

**base-uri-rel**($pathContext as xs:string := ())

as node()\*

**base-uri-rel-ec**(

$inputItems as item()\*,

$pathContext as xs:string := ())

as node()\*

***Summary***

Returns for each input node the relative URI of the file containing it.

***Details***

The URI context for which the relative URIs are determined defaults to the current file URI of the working directory. The context can be specified explicitly as a name filter, selecting the closest containing folder with a name matching the filter.

Note: the difference between this function and base-uri-rel() is subtle: this function URI-escapes the path steps, whereas base-dir-rel() does not. Example:

$node/base-uri-rel() => Oxygen%20XML%20Editor%2025/frameworks/tei/README.txt

$node/base-dir-rel() => Oxygen XML Editor 25/frameworks/tei/README.txt

***Params***

…

***Examples***

…

## Validation

The functions in this section validate document.

### xsd-validate (xvalidate) (\*-ec)

**xsd-validate**($xsds as item()\*,

$options as xs:string?)

as element(\*)

**xsd-validate-ec**($docs as item()\*,

$xsds as item()\*,

$options as xs:string?)

as element(\*)

***Summary***

Validates documents against XSDs.

***Details***

Validates the input documents against XSD schemas and returns a validation report, which includes a summary of validation results. Using function variant xsd-validate, the input document is supplied by the context item and the first parameter supplies the XSDs. Using function variant xsd-validate-ec, the first parameter supplies input documents, the second parameter supplies the XSDs.

Note that *several* input documents and *several* XSDs can be specified. For each input document, the function selects the appropriate XSD, which is the first XSD containing an element declaration matching the root element of the input document. If no XSD is found with such an element declaration, the first XSD is selected. In this case it is assumed that the element declaration is found in a schema imported or included by the schema.

Input documents and schemas can be supplied as nodes or document URIs. IMPORTANT: when input documents are supplied as nodes, validation messages may contain incorrect line numbers.

***Parameters***

Described by the following table.

**Table**. Parameters of function xsd-validate and xsd-validate-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| docs | NOTE: this parameter is only expected by function variant \*-ec.  The documents to be validated. Documents can be supplied as URIs or nodes. Warning – if supplied as nodes, messages may contain incorrect line numbers. |
| xsds | The schemas against which to validate. Schemas can be supplied as URIs or nodes. |
| options | fname – the report includes file names, rather than file URIs |

***Examples***

Validate a single document against a single schema.

fox "data/airports01.xml/xsd-validate(../../xsd/airports.xsd)"

Validate a single document against a set of schemas. The appropriate schema is selected automatically.

fox "data/airports01.xml/xsd-validate(../../xsd/\*.xsd)"

Validate a set of documents against a set of schemas. For each document, the appropriate schema is selected automatically.

fox "data/airport\*.xml => xsd-validate-ec(xsd/\*.xsd)"

Validate a set of documents against a set of schema nodes, supplied as nodes, rather than URIs.

fox "data/airport\*.xml => xsd-validate-ec(xsd/\*.xsd\xs:schema)"

Validate documents; the report should use file names, rather than URIs.

fox "data/airport\*.xml => xsd-validate-ec(xsd/\*.xsd, 'fname')"

Validate selected inner nodes, not documents.

fox "data/airports02.xml\\airport[\*] => xsd-validate-ec(xsd/\*.xsd, 'fname')"

Validate selected inner nodes extracted from multiple documents.

fox "data/airports\*.xml\\airport[city = 'Kikala'] => xsd-validate-ec(xsd/\*.xsd, 'fname')"

### dtd-validate (dval) (\*-ec)

**dtd-validate**($dtd as item(),

$options as xs:string?)

as element(\*)

**dtd-validate-ec**($docs as item()\*,

$dtd as item(),

$options as xs:string?)

as element(\*)

***Summary***

Validates documents against a DTD.

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## Creation of file trees

The functions in this section create file trees, which is a tree-structured representation of file system contents, representing folders and files by <fo> and <fi> elements, respectively..

### ftree (\*-ec)

**ftree**($fileProperties as item()\*)

as element()

**ftree-ec**(

$dirs as item()\*,

$fileProperty as item()\*)

as element()

***Summary***

Returns a tree representation of folder contents.

***Details***

For each input folder, a tree representation is generated, consisting of an <ftree> element with <fo> and <fi> descendants representing contained folders and files. Using function variant ftree-ec, input folder URIs are specified by the first parameter ($dirs); using function variant ftree, the input folder URI is supplied by the context item. If several input folders are specified, the corresponding <ftree> elements are wrapped in an <ftrees> element; if a single folder is specified, the corresponding <ftree> element is returned without a wrapper element.

If a $fileProperties argument is supplied, file descriptors are annotated by attributes and/or child elements providing *file properties*. A file property is a value returned by a Foxpath expression evaluated in the context of the file URI. The value of $fileProperty consists of one or several pairs of items, where the first item specifies the property name and the second item the Foxpath expression returning the parameter value. Example:

('\*.dita @ti', {\\*\title},  
 '\*.dita terms/term?', {\\term => distinct-values() => sort()})

Optionally, the property name is preceded by a [Unified Filter Expression](#_Filter_syntax_1) followed by whitespace, restricting annotation to files with a matching name. If the property name is preceded by an @ character, the property is represented by an attribute, otherwise by child elements. If the property name has a suffix consisting of a slash and a second name, the property is represented by an element with the first name and child elements with the second name and containing a single item of the property value. If the property name ends with a question mark, no item is created if the property value is empty. If the property name ends with a \* character, one element per value item is created.The grammar of items specifying the name and representation of a property can be expressed as follows:

file-name-filter? ("@" pname "?"? | pname ("?"|"\*")? | pname "/" iname "?"?)

where file-name-filter is a [Unified Filter Expression](#_Filter_syntax_1), pname represents the property name, which is constrained to be an NCName, and iname represents the name of item elements. Note that no whitespace is allowed between pname, "/" and iname.

The following table compiles the alternative specification of property elements or attributes, omitting the optional [Unified Filter Expression](#_Filter_syntax_1) which may always precede the property name.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **Example** | **Meaning** |
| pname=expr | ('date', {file-date()}) | Single element |
| pname?=expr | ('ti?', {\\ti\string()}) | Single element; only if property value non-empty |
| pname\*=expr | ('href\*', {\\@href\string()}) | One element per value item |
| @pname=expr | ('@date', {file-date()}) | Attribute |
| @pname?=expr | ('@ti?', {\\ti\string()}) | Attribute; only if property value non-empty |
| pname/iname=expr | ('hrefs/href', {\\@href\string()}) | Wrapper element, containing one item element per value item |
| pname/iname?=expr | ('hrefs/href?', {\\@href\string()}) | Wrapper element, containing one item element per value item; only if property value non-empty |

In all cases the property name can be preceded by a file name selector, which is a [Unified Filter Expression](#_Filter_syntax_1), limiting annotation to files with a matching name. Examples using a file name selector:

('\*.dita @ti ', {\\ti\string()})

('\*.dita \*.learning\* ~\*internal\* hrefs/href', {\\@href\string()})

The parameter value can contain any number of item pairs.

***Parameters***

Described by the following table.

**Table**. Parameters of function ftree and ftree-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| dirs | NOTE: this parameter is only expected by function variant \*-ec.  The folders to be described. |
| fileProperties  … | Each parameter describes a file property. See text for syntax and semantics of the argument string.  Examples:  date=file-date()  \*.xml @size=file-size()  airports-\*.xml @countAirports?=\\airport => count()  airports-\*.xml codes/code=\\@iata\string() => sort()  airports-\*.xml codes/code?=\\@iata\string() => sort() |

**Examples**

Get a folder tree, that is, a tree representation of the folder "image-map".

fox "image-map/ftree()"

Get folder trees of folders matching "image-map\*". If there are several folders, the trees are wrapped in an ftrees element.

fox "image-map\* => ftree-ec()"

Get a folder tree in which every file is annotated with a @date attribute showing the file date. The whitespace after the equal sign is inserted in order to avoid string mangling caused by the command shell.

fox "image-map/ftree(('@date', {file-date()}))"

Get a folder tree in which every file is annotated with an @date attribute showing the file date and a @size attribute showing the file size. Use one parameter for each file property.

fox "image-map/ftree(('@date', {file-date()}, '@size', {file-size()}))"

Get a folder tree in which every .dita file is annotate with a @ti attribute showing the title. In order to annotated selected files, use a file name selector preceding the property name.

fox "image-map/ftree(('\*.dita @ti', {\\*\title\truncate(50)}))"

Get a folder tree in which every every .dita file has a href annotation, containing a space-separated list of referenced file names.

fox "image-map/ftree(('\*.dita href', {\\@href\substring-before(., '#')[string()] => distinct-values() => sort()}))"

Similar to the previous example, but writing one href element for each referenced file name. Use a \* after the property name in order to get one property element per value item.

fox "image-map/ftree(('\*.dita href\*', {\\@href\substring-before(., '#')[string()] => distinct-values() => sort()}))"

Similar to the previous example, but wrapping the href elements in a hrefs element. Specify two element names separated by a slash - first the wrapper element name, then the item element name.

fox "image-map/ftree(('\*.dita hrefs/href', {\\@href\substring-before(., '#')[string()] => distinct-values() => sort()}))"

As the previous example, but suppress the annotation if the property value is the empty sequence. Use a ? after the property name (or names) in order to suppress empty property items.

fox "image-map/ftree(('\*.dita hrefs/href?', {\\@href\substring-before(., '#')[string()] => distinct-values() => sort()}))"

### ftree-selective (\*-ec)

**ftree-selective**($fileNamesFilter as xs:string?,

$folderNamesFilter as xs:string?,

$fileProperties as item()\*)

as element()

**ftree-selective-ec**(

$dirs as item()\*,

$fileNamesFilter as xs:string?,

$folderNamesFilter as xs:string?,

$fileProperties as item()\*)

as element()

***Summary***

Returns a tree representation of selected folder contents.

***Details***

For each input folder, a filtered tree representation is generated, consisting of an <ftree> element with <fo> and <fi> descendants representing contained folders and files. The tree content may be filtered by name filters to be applied to folders and files, respectively. Using function variant ftree-selective-ec, input folder URIs are specified by the first parameter ($dirs); using function variant ftree-selective, the input folder URI is supplied by the context item. If several input folders are specified, the corresponding <ftree> elements are wrapped in an <ftrees> element; if a single folder is specified, the corresponding <ftree> element is returned without a wrapper element.

If a $fileProperties argument is supplied, file descriptors are annotated by attributes and/or child elements providing *file properties*. A file property is a value returned by a Foxpath expression evaluated in the context of the file URI. The value of $fileProperty consists of one or several pairs of items, where the first item specifies the property name and the second item the Foxpath expression returning the parameter value. Example:

('\*.dita @ti', {\\*\title},  
 '\*.dita terms/term?', {\\term => distinct-values() => sort()})

Optionally, the property name is preceded by a [Unified Filter Expression](#_Filter_syntax_1) followed by whitespace, restricting annotation to files with a matching name. If the property name is preceded by an @ character, the property is represented by an attribute, otherwise by child elements. If the property name has a suffix consisting of a slash and a second name, the property is represented by an element with the first name and child elements with the second name and containing a single item of the property value. If the property name ends with a question mark, no item is created if the property value is empty. If the property name ends with a \* character, one element per value item is created.The grammar of items specifying the name and representation of a property can be expressed as follows:

file-name-filter? ("@" pname "?"? | pname ("?"|"\*")? | pname "/" iname "?"?)

where file-name-filter is a [Unified Filter Expression](#_Filter_syntax_1), pname represents the property name, which is constrained to be an NCName, and iname represents the name of item elements. Note that no whitespace is allowed between pname, "/" and iname.

The following table compiles the alternative specification of property elements or attributes, omitting the optional [Unified Filter Expression](#_Filter_syntax_1) which may always precede the property name.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **Example** | **Meaning** |
| pname=expr | ('date', {file-date()}) | Single element |
| pname?=expr | ('ti?', {\\ti\string()}) | Single element; only if property value non-empty |
| pname\*=expr | ('href\*', {\\@href\string()}) | One element per value item |
| @pname=expr | ('@date', {file-date()}) | Attribute |
| @pname?=expr | ('@ti?', {\\ti\string()}) | Attribute; only if property value non-empty |
| pname/iname=expr | ('hrefs/href', {\\@href\string()}) | Wrapper element, containing one item element per value item |
| pname/iname?=expr | ('hrefs/href?', {\\@href\string()}) | Wrapper element, containing one item element per value item; only if property value non-empty |

In all cases the property name can be preceded by a file name selector, which is a [Unified Filter Expression](#_Filter_syntax_1), limiting annotation to files with a matching name. Examples using a file name selector:

('\*.dita @ti ', {\\ti\string()})

('\*.dita \*.learning\* ~\*internal\* hrefs/href', {\\@href\string()})

The parameter value can contain any number of item pairs.

***Parameters***

Described by the following table.

**Table**. Parameters of function ftree-selective and ftree-selective-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| Dirs | NOTE: this parameter is only expected by function variant \*-ec.  The folders to be described. |
| fileNamesFilter | Filters the descendant files by name. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| foldersNamesFilter | Filters the descendant folders by name. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| fileProperty | Specification of file properties. The parameter value consists of one or more pairs of items, where the first item specifies the property name and its representation by element or attribute, and the second item provides the Foxpath expression returning the property value.  Examples:  'date', {file-date()}  '\*.xml @size', {file-size()}  'airports-\*.xml @countAirports?', {\\airport => count()}  'airports-\*.xml codes/code' {\\@iata\string() => sort()}  'airports-\*.xml codes/code?', {\\@iata\string() => sort() |

***Examples***

Get a folder tree, that is, a tree representation of the folder image-map. Do not exclude any folders or files.

fox "image-map/ftree-selective()"

As the first example, but excluding any files which do not have the extension ".dita".

fox "image-map/ftree-selective('\*.dita')"

As the first example, but excluding any folders with a name matching \*parts\* or equal images.

fox "image-map/ftree-selective((), '~\*parts\* ~images')"

Get folder trees of folders matching image-map\*, including only files with extension .dita. If there are several folders matching image-map\*", the trees are wrapped in an ftrees element.

fox "image-map\* => ftree-selective-ec('\*.dita')"

Similar to the preceding example, but annotating every .dita file with a @ti attribute providing the title.

fox "image-map\* => ftree-selective-ec('\*.dita', (), ('\*.dita @ti', {\\*\title}))"

Similar to the preceding example, but annotating every .dita file also with a terms annotation containing all referenced terms.

fox "image-map\* => ftree-selective-ec('\*.dita', (),   
 ('\*.dita @ti', {\\*\title},  
 '\*.dita terms/term?', {\\term => distinct-values() => sort()}))"

For more examples how to annotate file elements, see function [ftree](#_ftree_(ftree-ec)).

### ftree-view

**ftree-view**($uris as xs:string\*,

$fileProperties as item()\*)

as element()

***Summary***

Returns a tree representation of a given set of resource URIs.

***Details***

The resource URIs are represented by an <ftree> element with <fo> and <fi> descendants representing the URIs arranged in a tree structure.

Note a difference between file trees returned by functions ftree() and ftree-selective() on the one hand, and ftree-view() on the other hand. The former functions consume one or several folder URIs as input URIs, each one of which is represented by a tree containing all descendant folders and files. Contrary to this, ftree-view() consumes URIs which may be folder and file URIs, and it returns a single tree containing only the folders and files corresponding to input URIs, without adding descendant resources.

If a $fileProperties argument is supplied, file descriptors are annotated by attributes and/or child elements providing *file properties*. A file property is a value returned by a Foxpath expression evaluated in the context of the file URI. The value of $fileProperty consists of one or several pairs of items, where the first item specifies the property name and the second item the Foxpath expression returning the parameter value. Example:

('\*.dita @ti', {\\*\title},  
 '\*.dita terms/term?', {\\term => distinct-values() => sort()})

Optionally, the property name is preceded by a [Unified Filter Expression](#_Filter_syntax_1) followed by whitespace, restricting annotation to files with a matching name. If the property name is preceded by an @ character, the property is represented by an attribute, otherwise by child elements. If the property name has a suffix consisting of a slash and a second name, the property is represented by an element with the first name and child elements with the second name and containing a single item of the property value. If the property name ends with a question mark, no item is created if the property value is empty. If the property name ends with a \* character, one element per value item is created.The grammar of items specifying the name and representation of a property can be expressed as follows:

file-name-filter? ("@" pname "?"? | pname ("?"|"\*")? | pname "/" iname "?"?)

where file-name-filter is a [Unified Filter Expression](#_Filter_syntax_1), pname represents the property name, which is constrained to be an NCName, and iname represents the name of item elements. Note that no whitespace is allowed between pname, "/" and iname.

The following table compiles the alternative specification of property elements or attributes, omitting the optional [Unified Filter Expression](#_Filter_syntax_1) which may always precede the property name.

|  |  |  |
| --- | --- | --- |
| **Pattern** | **Example** | **Meaning** |
| pname=expr | ('date', {file-date()}) | Single element |
| pname?=expr | ('ti?', {\\ti\string()}) | Single element; only if property value non-empty |
| pname\*=expr | ('href\*', {\\@href\string()}) | One element per value item |
| @pname=expr | ('@date', {file-date()}) | Attribute |
| @pname?=expr | ('@ti?', {\\ti\string()}) | Attribute; only if property value non-empty |
| pname/iname=expr | ('hrefs/href', {\\@href\string()}) | Wrapper element, containing one item element per value item |
| pname/iname?=expr | ('hrefs/href?', {\\@href\string()}) | Wrapper element, containing one item element per value item; only if property value non-empty |

In all cases the property name can be preceded by a file name selector, which is a [Unified Filter Expression](#_Filter_syntax_1), limiting annotation to files with a matching name. Examples using a file name selector:

('\*.dita @ti ', {\\ti\string()})

('\*.dita \*.learning\* ~\*internal\* hrefs/href', {\\@href\string()})

The parameter value can contain any number of item pairs.

***Parameters***

Described by the following table.

**Table**. Parameters of function ftree-selective and ftree-selective-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uris | File and folder URIs to be represented as a tree of folders and files |
| fileProperty  … | Specification of file properties. The parameter value consists of one or more pairs of items, where the first item specifies the property name and its representation by element or attribute, and the second item provides the Foxpath expression returning the property value.  Examples:  'date', {file-date()}  '\*.xml @size', {file-size()}  'airports-\*.xml @countAirports?', {\\airport => count()}  'airports-\*.xml codes/code' {\\@iata\string() => sort()}  'airports-\*.xml codes/code?', {\\@iata\string() => sort() |

***Examples***

Get a tree view of all .dita files with a "concept" root element.

fox ".//\*.dita[\concept] => ftree-view()"

As before, but annotate the file elements with an @ti annotation providing the title.

fox ".//\*.dita[\concept] => ftree-view(('@ti', {\\*\title\normalize-space(.)}))"

Add a second annotation, "terms", providing all terms contained. Create for each term a distinct child element "term".

fox ".//\*.dita[\concept] => ftree-view((  
 '@ti', {\\*\title\normalize-space(.)},   
 'terms/term?', {\\term[text()]\normalize-space(.) => distinct-values() => sort()}  
))"

For those files starting with "c\_mv\_" add a third annotation "enames" listing all element names used in the document. Create for each element name a distinct child element "ename".

fox ".//\*.dita[\concept] => ftree-view((  
 '@ti', {\\*\title},   
 'terms/term?', {\\term[text()]\normalize-space(.) => distinct-values() => sort()},   
 'c\_mv\_\* enames/ename', {\\\*\name() => distinct-values() => sort()}  
))"

## Exploration of node trees

The functions in this section support the exploration of node trees.

### child-name-seq (child-lname-seq, child-jname-seq) (\*-ec)

**child-name-seq**($nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

**child-name-seq**($nameFilter as xs:string?)

as xs:string\*

**child-name-seq**()

as xs:string\*

**child-name-seq-ec**($nodes as node()\*,

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

**child-lname-seq-ec**($nodes as node()\*,

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

**child-jname-seq-ec**($nodes as node()\*,

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

***Summary***

Returns for each input node the concatenated list of child element names, in order.

***Variation***

Dependent on the function name (\*-name-flow / \*-lname-flow / \*-jname-flow), the names returned are

* Lexical names (function \*-name-flow)
* Local names (function \*-lname-flow)
* JSON names (function \*-jname-flow)

(See [Node name types](#_Node_name_types) for details).

If the function name ends with -ec, the nodes to be reported are supplied by the first argument; otherwise the first argument is omitted and defaults to the context node.

***Details***

For each node to be analyzed, a string is returned which is the comma-separated list of child element names. The order of child elements is preserved, duplicates are retained.

***Parameters***

**Table**. Parameters of functions (ec-)?child-(names/lnames/jnames).

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| nodes | The nodes to be analyzed. Parameter only used by functions ec-\*. |
| nameFilter | Only names matching this name filter are reported. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| options | For future use. |

***Examples***

Report the child element names of <span> elements found in a document.

fox "https://www.w3.org/TR/xpath-functions-31/hdoc()\\\*:span\child-name-flow() => freq()"

.................................................................. (4167)

a ................................................................. (68)

a, a .............................................................. (3)

a, a, a, a, b ..................................................... (1)

a, a, a, span, a, a, b ............................................ (1)

a, a, b ........................................................... (4)

a, a, b, a, code, code, code, code, var, var ...................... (1)

a, a, span, a, a, b ............................................... (1)

a, a, span, b ..................................................... (1)

a, b .............................................................. (4)

a, b, a ........................................................... (2)

a, b, a, a, b, b, code, a ......................................... (1)

a, b, a, a, sup ................................................... (1)

a, b, a, a, sup, a, sup, a, sup, a, sup, a, sup, a, a, sup, a, sup (1)

a, b, a, code ..................................................... (1)

a, b, a, sup ...................................................... (1)

a, b, a, sup, a ................................................... (1)

a, b, code, a ..................................................... (1)

…

### child-names (-lnames, -jnames) (\*-ec)

**child-names**($nameFilter as xs:string?, $options as xs:string?)

as xs:string\*

**child-names**($nameFilter as xs:string?)

as xs:string\*

**child-names**()

as xs:string\*

**child-names-ec**($nodes as node()\*,

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

**child-names-ec**($nodes as node()\*,

$nameFilter as xs:string?)

as xs:string\*

**child-names-ec**($nodes as node()\*) as xs:string\*

***Summary***

Returns for each input node the concatenated list of distinct child element names.

***Variation***

Dependent on the function name (\*-names / \*-lnames / \*-jnames), the names returned are

* Lexical names (function \*-names)
* Local names (function \*-lnames)
* JSON names (function \*-jnames)

(See [Node name types](#_Node_name_types) for details).

If the function name ends with -ec, the nodes to be reported are supplied by the first argument, otherwise there is a single node to be reported, which is the context node.

***Details***

For each node to be analyzed, a string is returned which is the comma-separated list of deduplicated child names. By default, the child names are sorted lexicographically. Sorting is suppressed if option nosort is used.

***Parameters***

**Table**. Parameters of functions (ec-)?child-(names/lnames/jnames).

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| nodes | The nodes to be analyzed. Parameter only used by functions ec-\*. |
| nameFilter | Only names matching this name filter are reported. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| options | Options controlling the processing; values:  nosort – child names are not sorted, but concatenated in document order  duplicates – duplicates are not removed |

***Usage tips***

Option nosort can be useful when constructing a schema for a set of instance documents. Use child-names() with option nosort in order to explore the order of child elements.

***Examples***

Report the child element names of paragraph elements found in a set of documents.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:paragraph\**child-names()** => f()"

.................................... (20258)

l:bold .............................. (1740)

l:bold, l:br ........................ (83)

l:bold, l:br, l:subscript ........... (4)

l:bold, l:br, l:superscript ......... (4)

…

l:subscript, l:superscript .......... (6)

l:subscript, l:underline ............ (1)

l:superscript ....................... (683)

l:table ............................. (4)

l:underline ......................... (1868)

As before, but return the local names.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:paragraph\child-**l**names() => f()"

.............................. (20258)

bold .......................... (1740)

bold, br ...................... (83)

bold, br, subscript ........... (4)

bold, br, superscript ......... (4)

…

subscript, superscript ........ (6)

subscript, underline .......... (1)

superscript ................... (683)

table ......................... (4)

underline ..................... (1868)

As before, but consider only child element names matching one of bold, ital\*, under\*.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:paragraph\child-lnames(**'bold under\* ital\*'**)

=> f()"

................. (21726)

bold ............. (1851)

bold, italic ..... (8)

bold, underline .. (2)

italic ........... (1920)

italic, underline (10)

underline ........ (1899)

As before, but ignore child element names \*script.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:paragraph\child-lnames(**'~\*script'**) => f()"

..................... (21311)

bold ................. (1759)

bold, br ............. (91)

bold, figure ......... (1)

bold, italic ......... (8)

bold, underline ...... (2)

br ................... (366)

br, figure ........... (1)

br, italic ........... (9)

br, italic, underline (1)

br, underline ........ (30)

figure ............... (37)

footnote ............. (7)

italic ............... (1911)

italic, underline .... (9)

table ................ (4)

underline ............ (1869)

Equivalent to the first example, but using a single function call applied to all input nodes, rather than one function call for each input node.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:paragraph => **ec-**child-lnames() => f()"

.................................... (20258)

l:bold .............................. (1740)

l:bold, l:br ........................ (83)

l:bold, l:br, l:subscript ........... (4)

l:bold, l:br, l:superscript ......... (4)

…

l:subscript, l:superscript .......... (6)

l:subscript, l:underline ............ (1)

l:superscript ....................... (683)

l:table ............................. (4)

l:underline ......................... (1868)

### descendant-names (-lnames, -jnames) (\*-ec)

**descendant-names**($nameFilter as xs:string?, $options as xs:string?)

as xs:string\*

**descendant-names**($nameFilter as xs:string?)

as xs:string\*

**descendant-names**()

as xs:string\*

**descendant-names-ec**($nodes as node()\*,

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

**descendant-names-ec**($nodes as node()\*,

$nameFilter as xs:string?)

as xs:string\*

**descendant-names-ec**($nodes as node()\*)

as xs:string\*

***Summary***

Returns for each input node the concatenated list of distinct descendant element names.

***Variation***

Dependent on the function name (\*-names / \*-lnames / \*-jnames), the names returned are

* Lexical names (function \*-names)
* Local names (function \*-lnames)
* JSON names (function \*-jnames)

(See [Node name types](#_Node_name_types) for details).

If the function name starts with ec-, the nodes to be reported are supplied by the first argument, otherwise there is a single node to be reported, which is the context node.

***Details***

For each node to be analyzed, a string is returned which is the comma-separated list of deduplicated descendant element names. By default, the descendant names are sorted lexicographilly. Sorting is suppressed if option nosort is used.

***Parameters***

**Table**. Parameters of functions (ec-)?descendant-(names/lnames/jnames).

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| nodes | The nodes to be analyzed. Parameter only used by functions ec-\*. |
| nameFilter | Only names matching this name filter are reported. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| options | Options controlling the processing; values:  nosort – descendant names are not sorted, but concatenated in document order |

***Examples***

Report the child element names of paragraph elements found in a set of documents.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:paragraph\**descendant-names**() => f()"

............................................................ (20258)

l:alternative-text, l:figure, l:image, l:paragraph .......... (6)

l:bold ...................................................... (1578)

l:bold, l:br ................................................ (79)

l:bold, l:br, l:italic, l:underline ......................... (1)

…

l:col, l:colgroup, l:paragraph, l:table, l:tbody, l:td, l:tr (4)

l:figure, l:image ........................................... (31)

l:footnote, l:paragraph ..................................... (7)

l:italic .................................................... (1502)

l:italic, l:subscript ....................................... (14)

l:italic, l:subscript, l:superscript ........................ (3)

l:italic, l:superscript ..................................... (60)

l:italic, l:superscript, l:underline ........................ (1)

l:italic, l:underline ....................................... (342)

l:subscript ................................................. (364)

l:subscript, l:superscript .................................. (6)

l:subscript, l:underline .................................... (1)

l:superscript ............................................... (683)

l:superscript, l:underline .................................. (12)

l:underline ................................................. (1853)

Equivalent to the first example, but using a single function call applied to all input nodes, rather than one function call for each input node.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:paragraph => **ec-**descendant-lnames() => f()"

.............................................. (20258)

alternative-text, figure, image, paragraph .... (6)

bold .......................................... (1578)

bold, br ...................................... (79)

bold, br, italic, underline ................... (1)

…

col, colgroup, paragraph, table, tbody, td, tr (4)

figure, image ................................. (31)

footnote, paragraph ........................... (7)

italic ........................................ (1502)

italic, subscript ............................. (14)

italic, subscript, superscript ................ (3)

italic, superscript ........................... (60)

italic, superscript, underline ................ (1)

italic, underline ............................. (342)

subscript ..................................... (364)

subscript, superscript ........................ (6)

subscript, underline .......................... (1)

superscript ................................... (683)

superscript, underline ........................ (12)

underline ..................................... (1853)

### att-names (-lnames, -jnames( (\*-ec)

**att-names**($nameFilter as xs:string?, $options as xs:string?)

as xs:string\*

**att-names**($nameFilter as xs:string?)

as xs:string\*

**att-names**()as xs:string\*

**att-names-ec**($nodes as node()\*,

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

**att-names-ec**( $nodes as node()\*,

$nameFilter as xs:string?)

as xs:string\*

**att-names-ec**($nodes as node()\*)

as xs:string\*

***Summary***

Returns for each input node the concatenated list of attribute names.

***Variation***

Dependent on the function name (\*-names / \*-lnames / \*-jnames), the names returned are

* Lexical names (function \*-names)
* Local names (function \*-lnames)
* JSON names (function \*-jnames)

(See [Node name types](#_Node_name_types) for details).

If the function name starts with ec-, the nodes to be reported are supplied by the first argument, otherwise there is a single node to be reported, which is the context node.

***Details***

For each node to be analyzed, a string is returned which is the comma-separated list of deduplicated attribute names. By default, the attribute names are sorted lexicographilly. Sorting is suppressed if option nosort is used.

***Parameters***

**Table**. Parameters of functions (ec-)?att-(names/lnames/jnames).

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| nodes | The nodes to be analyzed. Parameter only used by functions ec-\*. |
| nameFilter | Only names matching this name filter are reported. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| options | Options controlling the processing; values:  nosort – attribute names are not sorted, but concatenated in document order |

***Tips***

The use of a name filter can help you to focus on the information actually needed. To illustrate this, observe the following investigation yielding the frequencies of attribute name combinations, as encounted in table cell elements:

fox "../d2cx-mass//\*.xml\\td\att-names() => f()"

.......................................................................................... (110308)

border, border-tl2br ..................................................................... (2)

border, border-tl2br, border-tr2bl ....................................................... (2)

border, border-tl2br, border-tr2bl, colspan .............................................. (1)

border, colspan .......................................................................... (9951)

border, colspan, rowspan ................................................................. (42)

border, rowspan .......................................................................... (2479)

border-bottom, border-left, border-right, border-tl2br, border-top, border-tr2bl ......... (3)

border-bottom, border-left, border-right, border-tl2br, border-top, border-tr2bl, colspan (3)

border-bottom, border-left, border-right, border-top ..................................... (17651)

border-bottom, border-left, border-right, border-top, colspan ............................ (2168)

border-bottom, border-left, border-right, border-top, colspan, rowspan ................... (13)

border-bottom, border-left, border-right, border-top, rowspan ............................ (599)

Now suppose that attributes matching border\* are currently not of interest. The situation becomes much more transparent by excluding these attributes by a name filter.

fox "../d2cx-mass//\*.xml\\td\att-names('~border\*') => f()"

................ (127966)

colspan ......... (12123)

colspan, rowspan (55)

rowspan ......... (3078)

Note in particular that the use of a name filter may entail a different aggregation of results. The combination “with @colspan, without @rowspan” is represented by a single line:

colspan ......... (12123)

In the result obtained without name filter, the combination is represented by four different lines:

border, border-tl2br, border-tr2bl, colspan .............................................. (1)

border, colspan .......................................................................... (9951)

border-bottom, border-left, border-right, border-tl2br, border-top, border-tr2bl, colspan (3)

border-bottom, border-left, border-right, border-top, colspan ............................ (2168)

***Examples***

Report the attribute names of td elements found in a set of documents.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:td\att-names() => f()"

border-bottom, border-left, border-right, border-top .................. (7686)

border-bottom, border-left, border-right, border-top, colspan ......... (557)

border-bottom, border-left, border-right, border-top, colspan, rowspan (1)

border-bottom, border-left, border-right, border-top, rowspan ......... (152)

Equivalent to the first example, but using a single function call applied to all input nodes, rather than one function call for each input node.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:td => **ec-**att-names() => f()"

border-bottom, border-left, border-right, border-top .................. (7686)

border-bottom, border-left, border-right, border-top, colspan ......... (557)

border-bottom, border-left, border-right, border-top, colspan, rowspan (1)

border-bottom, border-left, border-right, border-top, rowspan ......... (152)

### content-names (-lnames, -jnames) (\*-ec)

**content-names**($nameFilter as xs:string?, $options as xs:string?)

as xs:string\*

**content-names**($nameFilter as xs:string?)

as xs:string\*

**content-names**()

as xs:string\*

**content-names-ec**($nodes as node()\*,

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

**content-names-ec**($nodes as node()\*,

$nameFilter as xs:string?)

as xs:string\*

**content-names-ec**($nodes as node()\*)

as xs:string\*

***Summary***

Returns for each input node the concatenated list of attribute names and child element names.

***Variation***

Dependent on the function name (\*-names / \*-lnames / \*-jnames), the names returned are

* Lexical names (function \*-names)
* Local names (function \*-lnames)
* JSON names (function \*-jnames)

(See [Node name types](#_Node_name_types) for details).

If the function name starts with ec-, the nodes to be reported are supplied by the first argument, otherwise there is a single node to be reported, which is the context node.

***Details***

For each node to be analyzed, a string is returned which is the comma-separated list of deduplicated attribute names, followed by deduplicated child element names. Attribute names are preceded by an @ character. By default, the attribute names and child element names are sorted lexicographically. Sorting is suppressed if option nosort is used.

***Parameters***

**Table**. Parameters of functions (ec-)?att-(names/lnames/jnames).

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| nodes | The nodes to be analyzed. Parameter only used by functions ec-\*. |
| nameFilter | Only names matching this name filter are reported. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| Options | Options controlling the processing; values:  nosort – attribute and child element names are not sorted, but concatenated in document order |

***Examples***

Report the attribute and child element names of td elements found in a set of documents.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:td\**content-names()** => f()"

@border-bottom, @border-left, @border-right, @border-top ................................. (1081)

@border-bottom, @border-left, @border-right, @border-top, @colspan ....................... (25)

@border-bottom, @border-left, @border-right, @border-top, @colspan, @rowspan, l:paragraph (1)

@border-bottom, @border-left, @border-right, @border-top, @colspan, l:list ............... (1)

@border-bottom, @border-left, @border-right, @border-top, @colspan, l:list, l:paragraph .. (3)

@border-bottom, @border-left, @border-right, @border-top, @colspan, l:paragraph .......... (528)

@border-bottom, @border-left, @border-right, @border-top, @rowspan ....................... (23)

@border-bottom, @border-left, @border-right, @border-top, @rowspan, l:paragraph .......... (129)

@border-bottom, @border-left, @border-right, @border-top, l:list ......................... (5)

@border-bottom, @border-left, @border-right, @border-top, l:list, l:paragraph ............ (5)

@border-bottom, @border-left, @border-right, @border-top, l:paragraph .................... (6595)

Equivalent to the first example, but using a single function call applied to all input nodes, rather than one function call for each input node.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:td => **ec-**content-names() => f()"

@border-bottom, @border-left, @border-right, @border-top ................................. (1081)

@border-bottom, @border-left, @border-right, @border-top, @colspan ....................... (25)

@border-bottom, @border-left, @border-right, @border-top, @colspan, @rowspan, l:paragraph (1)

@border-bottom, @border-left, @border-right, @border-top, @colspan, l:list ............... (1)

@border-bottom, @border-left, @border-right, @border-top, @colspan, l:list, l:paragraph .. (3)

@border-bottom, @border-left, @border-right, @border-top, @colspan, l:paragraph .......... (528)

@border-bottom, @border-left, @border-right, @border-top, @rowspan ....................... (23)

@border-bottom, @border-left, @border-right, @border-top, @rowspan, l:paragraph .......... (129)

@border-bottom, @border-left, @border-right, @border-top, l:list ......................... (5)

@border-bottom, @border-left, @border-right, @border-top, l:list, l:paragraph ............ (5)

@border-bottom, @border-left, @border-right, @border-top, l:paragraph .................... (6595)

### parent-name (-lname, -jname) (\*-ec)

**parent-name**($nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

**parent-name**($nameFilter as xs:string?)

as xs:string\*

**parent-name**()

as xs:string\*

**parent-name-ec**($nodes as node()\*,

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

**parent-name-ec**($nodes as node()\*,

$nameFilter as xs:string?)

as xs:string\*

**parent-name-ec**($nodes as node()\*)

as xs:string\*

***Summary***

Returns for each input node the parent node name.

***Variation***

Dependent on the function name (\*-names / \*-lnames / \*-jnames), the names returned are

* Lexical names (function \*-names)
* Local names (function \*-lnames)
* JSON names (function \*-jnames)

(See [Node name types](#_Node_name_types) for details).

If the function name starts with ec-, the nodes to be reported are supplied by the first argument, otherwise there is a single node to be reported, which is the context node.

***Details***

For each node to be analyzed, the parent node name is returned

***Parameters***

**Table**. Parameters of functions (ec-)?parent-(names/lnames/jnames).

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| nodes | The nodes to be analyzed. Parameter only used by functions ec-\*. |
| nameFilter | Only names matching this name filter are reported. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| options | Options controlling the processing; values: (currently not evaluated) |

***Examples***

Report the parent names of paragraph elements found in a set of documents.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:paragraph\**parent-name()** => f()"

a:textPrecedingHeader ............ (10)

fi:ability-drive ................. (120)

fi:authorisation-date ............ (137)

fi:clinical-particulars .......... (1)

fi:composition ................... (553)

…

fi:warnings ...................... (1854)

l:alternative-text ............... (6)

l:footnote ....................... (7)

l:listitem ....................... (1021)

l:section ........................ (54)

l:td ............................. (9115)

Equivalent to the first example, but using a single function call applied to all input nodes, rather than one function call for each input node.

fox "../output-convert-mass/\*01/\*fibook.xml\\\*:td => **ec-**att-names() => f()"

a:textPrecedingHeader ............ (10)

fi:ability-drive ................. (120)

fi:authorisation-date ............ (137)

fi:clinical-particulars .......... (1)

fi:composition ................... (553)

…

fi:warnings ...................... (1854)

l:alternative-text ............... (6)

l:footnote ....................... (7)

l:listitem ....................... (1021)

l:section ........................ (54)

l:td ............................. (9115)

### path-content (\*-ec)

**path-content** ($options as xs:string?)

as xs:string\*

**path-content-ec**(

$nodesOrUris as item()\*,

$options as xs:string?)

as xs:string\*

***Summary***

Returns for each input item (node or URI) the relative paths of its content nodes, together with its frequency.

***Details***

Lists the name paths of all elements and attributes directly or indirectly contained by input nodes. The paths are relative to the input nodes. Input nodes are specified explicitly (function variant -ec) or default to the context node (function variant without –ec).

***Parameters***

**Table**. Parameters of functions path-content, path-content-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| nodesOrUris | The nodes to be analyzed, provided as nodes or document URIs. This parameter is only used by function variant -ec. |
| Options | Options controlling the processing; values:  lname – the name path uses local names (default)  name – the name path uses lexical names  jname – the name path uses JSON names  Note: filter parameters refer to the name kind selected.  text - Return also the paths of text nodes  with-inner - Return also the paths of inner nodes (non-leaf nodes)  with-context – If the context node is an element, the first path step provides the name of the context node  textNN – pad the path string to a length of NN characters  xml – result as XML  json – result as JSON  csv – result as CSV |

***Examples***

Get the path content of simple type definitions:

fox "/programme/\*oxy\*/frame\*/docbook//\*.xsd\\*\xs:simpleType => path-content-ec()"

=== path-content ===============================

@name ........................................... (104)

restriction ..................................... (12)

restriction/@base ............................... (104)

restriction/enumeration/@value .................. (499)

restriction/enumeration/annotation/documentation (499)

================================================

As before, including inner nodes:

fox "/programme/\*oxy\*/frame\*/docbook//\*.xsd\\*\xs:simpleType =>

path-content-ec('with-inner')"

=== path-content ===============================

@name ........................................... (104)

restriction ..................................... (104)

restriction/@base ............................... (104)

restriction/enumeration ......................... (499)

restriction/enumeration/@value .................. (499)

restriction/enumeration/annotation .............. (499)

restriction/enumeration/annotation/documentation (499)

================================================

***Usage tip***

Option with-inner is well suited for comparing the frequencies of elements and their child elements or attributes. For example, if you want to check if every xs:restriction element has a @base attribute, you need the frequencies of *all* xs:restriction elements, rather than only those which have no child elements.

### path-content-filtered (\*-ec)

**path-content-filtered**(

$nameFilter as xs:string?,

$ancestorNameFilter as xs:string?,

$excludedAncestorNameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

**path-content-filtered**(

$nameFilter as xs:string?,

$ancestorNameFilter as xs:string?,

$excludedAncestorNameFilter as xs:string?)

as xs:string\*

**path-content-filtered**(

$nameFilter as xs:string?,

$ancestorNameFilter as xs:string?)

as xs:string\*

**path-content-filtered**(

$nameFilter as xs:string?)

as xs:string\*

**path-content-filtered-ec**(

$nodesOrUris as item()\*,

$nameFilter as xs:string?,

$ancestorNameFilter as xs:string?,

$excludedAncestorNameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

…

***Summary***

Returns for each input item (node or URI) the relative paths of its content nodes, filtered by various criteria.

***Details***

Lists the relative data path of all nodes directly or indirectly contained by a context node. The paths are relative to the implicit context node (function variant without –ec) or specified by parameter $nodesOrUris (function variant \*-ec).

By default, only leaf nodes are reported, defined as element nodes without child elements and attributes. To have also inner nodes reported, use option with-inner.

The content nodes to be reported can be filtered in various ways:

* Filter by node name $nameFilter
* Consider only nodes which have an ancestor matching $ancestorNameFilter
* Consider only nodes which do *not* have an ancestor matching $excludedAncestorNameFilter

***Parameters***

**Table**. Parameters of functions path-content-filtered, path-content-filtered-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| nodesOrUris | The nodes to be analyzed, provided as nodes or as document URI. This parameter is only used by function \*-ec. |
| nameFilter | Only content leaves with a matching name are reported. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| ancestorNameFilter | Ignore content nodes which do not have an ancestor node with a name matching the filter. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| excludedAncestorName-Filter | Ignore content nodes which have an ancestor node with a name matching the filter. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| Options | Options controlling the processing; values:  name – constructing paths and filtering: use lexical names  jname - constructing paths and filtering: use JSON names  lname - constructing paths and filtering: use lexical names (default)  text - Return also the paths of text nodes  with-inner - Return also the paths of inner nodes (non-leaf nodes)  with-context – If the context node is an element, the first path step provides the name of the context node  textNN – pad the path string to a length of NN characters  xml – result as XML  json – result as JSON  csv – result as CSV |

***Examples***

Get the path content of dita task elements, filtering for nodes with a name matching \*path\*:

fox "/programme/\*oxy\*/samples/dita//\*.dita\\task

=> path-content-filtered-ec('\*path\*')"

=== path-content ===============================

taskbody/example/p/filepath ............................... (1)

taskbody/steps/step/cmd/filepath .......................... (8)

taskbody/steps/step/stepresult/p/filepath ................. (1)

taskbody/steps/step/substeps/substep/info/ul/li/p/filepath (2)

taskbody/steps/step/substeps/substep/stepxmp/p/filepath ... (1)

================================================

As before, including the paths of inner nodes:

fox "/programme/\*oxy\*/samples/dita//\*.dita\\task =>

path-content-filtered-ec('\*path\*', (), (), 'with-inner')"

=== path-content ===============================

taskbody .................................................. (9)

taskbody/example .......................................... (1)

taskbody/example/p ........................................ (1)

taskbody/example/p/filepath ............................... (1)

taskbody/steps ............................................ (9)

taskbody/steps/step ....................................... (11)

taskbody/steps/step/cmd ................................... (8)

taskbody/steps/step/cmd/filepath .......................... (8)

…

================================================

***Usage tip***

Option with-inner is well suited for comparing the frequencies of elements and their child elements or attributes. For example, if you want to check if every step element has a cmd child, you need the frequencies of *all* step elements.

### name-content (\*-ec)

**name-content**(

$nameFilter as xs:string,

$options as xs:string?)

as xs:string\*

**name-content-ec**(

$uriOrNode as item()+,

$nameFilter as xs:string,

$options as xs:string?)

as xs:string\*

***Summary***

Returns the names and frequencies of nodes and optionally the names and frequencies of related nodes.

***Details***

Lists the names and frequencies of elements and attributes, optionally filtered by a name filter. Optionally also lists the names and frequencies of nodes related to the nodes with a given name: parents, attributes, children.

***Parameters***

Described by the following table.

**Table**. Parameters of function name-content and name-content-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uriOrNode | NOTE: this parameter is only expected by function variant \*-ec.  Input nodes the content of which shall be reported. An atomic item is interpreted as document URI and replaced with the corresponding document node. |
| namesFilter | Name filter used for selecting the content nodes to be reported. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| options | Options controling the function behaviour.  Options specifying which kinds of related nodes of the content nodes to be reported. These options can be combined:  a – report attributes of the content nodes  c – report child elements of the content nodes  p – report parent elements of the content nodes  Options selecting the kind of node name to be reported. These options are mutually exclusive.  name – lexical names (possibly containing a prefix)  lname – local names (default)  jname – JSON names |

***Examples***

Report all nodes, but do not report related nodes. The context items are interpreted as document URIs:

fox "output/\*01/\*fibook\* => **ec-node-names**()"

Report only the contents of <revision-date> elements. The context items are elements:

fox "output/\*fibook\*\\\*:revision-date => **ec-node-names**()"

Report the nodes selected by a name filter:

fox "output/\*fibook\* => **ec-node-names**('pharma\* ~\*form')"

Report also the parents:

fox "output/\*fibook\* => **ec-node-names**('pharma\* ~\*form', 'p')"

Report also the attributs and child nodes:

fox "output/\*fibook\* => **ec-node-names**('pharma\* ~\*form', 'pac')"

Filter by and display lexical names, rather than local names:

fox "output/\*01/\*fibook\* => **ec-node-names**('pharma\* ~\*form', 'pac', 'name')"

Use non-ec form:

fox "output/\*fibook\*[1]/**node-names**()"

## Exploration of node location

The functions in this section deal with the location of nodes.

### name-path (\*-ec)

**name-path**($numberOfSteps as xs:integer?

$flags as xs:string?)

as xs:string

**name-path-ec**(

$nodes as node()\*,

$numberOfSteps as xs:integer?

$flags as xs:string?)

as xs:string

***Summary***

Returns the “name path” of a node, consisting of the names of the node and its ancestors.

***Details***

The name kind (local, lexical, JSON names) is determined by options, defaulting to local names.

Using parameter $numberOfSteps, only the trailing steps are returned.

Using options fname, fpath or rfpath, the file name or path is inserted before the path, separated from it by a # character.

Using option value, the paths of leaf nodes are augmented by the postfix value=…, where … is the string value of the node.

***Parameters***

Described by the following table.

**Table**. Parameters of functions name-path, name-path-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| nodes | Nodes to be reported. Parameter only used by functions name-path-ec. |
| numberOfSteps | If the path has more steps than the parameter value, it is truncated to this number of steps by removing leading steps. In other words, only the last … steps are shown. |
| options | lname – path steps are local names  name – path steps are lexical names  jname – path steps are JSON names  fname – the path is preceded by the file name, followed by the character #  fpath – the path is preceded by the file path, followed by the character #  rfpath – the path is preceded by the relative file path, followed by the character  #; the file path is relative to the current working directory  indexed – the element steps have a predicate indicating the position among all  siblings with the same node name  value – the path is followed by “=value” (only attributes, text nodes and  element nodes with text node children)  text – text nodes are represented by a step #text=(string value)  xsdcompname – steps corresponding to named XSD components have a suffix  “(“component-name”)”, e.g. “(PriceType)”.  with-context – if a context node is specified, the path starts with the context  node itself |
|  |  |

***Examples***

-to-be-added-

### name-path-attributed (\*-ec)

**name-path-attributed**(

$attFilter as xs:string?,

$numberOfSteps as xs:integer?,

$flags as xs:string?)

as xs:string

**name-path-attributed-ec**(

$nodes as node()\*,

$attFilter as xs:string?,

$numberOfSteps as xs:integer?

$flags as xs:string?)

as xs:string

***Summary***

Returns the “name path” of a node, with attribute value information appended to each step containing attributes matching an attribute name filter.

***Details***

-to-be-added-

***Parameters***

**Table**. Parameters of functions node-location, lnode-location, jnode-location.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| nodes | Nodes to be reported. Parameter only used by functions name-path-ec. |
| attFilter | Attribute name filter, supplied as a unified string expression. |
| numberOfSteps | If the path has more steps than the parameter value, it is truncated to this number of steps by removing leading steps. In other words, only the last … steps are shown. |
| options | fname – the path is preceded by the file name, followed by the character #  fpath – the path is preceded by the file path, followed by the character #  rfpath – the path is preceded by the relative file path, followed by the character  #; the file path is relative to the current working directory  indexed – the element steps have a predicate indicating the position among all  siblings with the same node name  value – the path is followed by “=value” (only attributes, text nodes and  element nodes with text node children)  text – text nodes are represented by a step #text  xsdcompname – steps corresponding to named XSD components have a suffix  “(“component-name”)”, e.g. “(PriceType)”. |

***Examples***

-to-be-added-

### truncate-name-path (\*-ec)

**truncate-name-path**( $elemFilter as xs:string?)

as xs:string

**truncate-name-path-ec**(

$paths as xs:string\*,

$elemFilter as xs:string?)

as xs:string

***Summary***

Truncates the trailing part of a name path. The truncation replaces the child elements of elements reached by a path step matching a name filter.

***Details***

-to-be-added-

***Parameters***

-to-be-added-

***Examples***

Get the path content of elements with a certain name, truncating the paths at step “beschreibung”.

fox "../standards/\*/model/\*.xml

\\\*:codeDatatype\path-content()

=> truncate-name-path-ec('description') => f()"

### node-location (lnode-location, jnode-location)

**node-location**($nodes as node()\*,

$flags as xs:string?)

as xs:string

**lnode-location**($nodes as node()\*,

$flags as xs:string?)

as xs:string

**jnode-location**($nodes as node()\*,

$flags as xs:string?)

as xs:string

Including variants:

* lnode-location() – consider local names, not lexical node names
* jnode-location() – consider JSON names, not lexical node names

***Summary***

Returns a hierarchical representation of the locations and (optionally) the text content of given nodes.

***Details***

Returns a hierarchical representation of the locations and (optionally) the text content of given nodes. When using flag f, the file name and a flag-controlled number of containing folder names are included:

* Flag f – include the file name
* Flag f2 – include the name of the containing folder and the file name
* Flag f3 – include the names of the two innermost containing folders and the file name
* Etc.

When using flag v, the distinct values of attribute nodes and simple element nodes are included in the report.

***Parameters***

Described by the following table.

**Table**. Parameters of functions node-location, lnode-location, jnode-location.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| nodes | Nodes to be reported. |
| flags | The characters of the string value control the processing as follows:  v – the report includes the distinct value of attributes nodes and simple element nodes  a – node paths are grouped by common ancestors; root element names are preceded by a slash  f – the report includes the file names  f… (… an integer number): the report includes the file names and the names of enclosing folders; if … > 1, the number of folder names is … - 1. Examples:  f1 – the report includes the file names (equivalent to f)  f2 – the report includes the name of the containing folder and the file name  f3 – the report includes the names of the two nearest containing folders and the file name |

***Examples***

Report the occurrence of XSD documentation containing specified text:

fox "/ps/p-foo//\*.xsd\\xs:documentation[contains-text('.\*hinweise')] => node-location('fv')"

==========================================================================================

File

. Name

. . Path

. . . Value

==========================================================================================

foo.xsd

. xs:documentation

. . /xs:schema/xs:element/xs:annotation/xs:documentation

. . . value: Element disposal - 6.6 Besondere Vorsichtsmaßnahmen für die Beseitigung

. . . value: Element warnings - 4.4 Besondere Warnhinweise für die Anwendung

bar.xsd

. xs:documentation

. . /xs:schema/xs:element/xs:annotation/xs:documentation

. . . value: Weitere Hinweise.

### jnode-location, jlocation

**jnode-location**($nodes, $numFolders?)

as xs:string\*

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)

## Evaluation of values

The functions in this section support the inspection and transformation of values.

### values-distinct

**values-distinct**($items as item()\*)

as xs:boolean

***Summary***

Returns true if given items are distinct.

***Details***

Returns true if given items are distinct, false otherwise. More precisely, the function returns true if count($items) is equal to count(distinct-values($items)). This implies that the function also returns true if a single item or an empty sequence is provided.

***Parameters***

**Table**. Parameters of function values-distinct.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | The items to be checked. |

***Examples***

Illustrative example, listing the items explicitly.

fox "(1,2,3,2,2,5,6,1,1,1,7) => non-distinct-values()"

1

2

### non-distinct-values , non-distinct

**non-distinct-values**(

$items as item()\*,

$ignoreCase as xs:boolean?)

as xs:boolean

***Summary***

Extracts from a sequence of items all items occurring more than once.

***Details***

Returns the items occurring in $items at least twice. If $ignoreCase is true, distinctness is checked ignoring case.

***Parameters***

**Table**. Parameters of function non-distinct-items.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | The items to be checked. |
| ignoreCase | If true, distinctness is checked ignoring case. |

Examples

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

### both-values (bvalues, value-intersect)

**both-values**($value1 as item()\*, $value2 as item()\*)

as item()\*

***Summary***

Returns the atomic items belonging to both of two given values.

### left-value-only (left-value; value-except)

**left-value**($value1 as item()\*, $value2 as item()\*)

as item()\*

***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***

Returns file names found in the first folder (at any depth), but not in the second (at any depth):

fox "output.20220419//\*.xml/fname() => **left-value**(output.20220420//\*.xml/fname())"

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\\_)]"

### right-value-only (right-value)

**right-value**($value1 as item()\*, $value2 as item()\*)

as item()\*

***Summary***

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

***Details***

…

***Parameters***

***…***

***Examples***

Returns file names found in the second folder (at any depth), but not in the first (at any depth):

fox "output.20220419//\*.xml/fname() => right-value(output.20220420//\*.xml/fname())"

## String matching and filtering

The functions in this section support the matching of strings against some conditions, and filtering of items based on such matching.

### matches-pattern (\*-ec)

**matches-pattern**($pattern as xs:string)

as xs:boolean

**matches-pattern-ec**($item as item(),

$pattern as xs:string)

as xs:boolean

***Summary***

Checks if an item matches a unified pattern expression.

**Details**

Under construction.

***Parameters***

Described by the following table.

**Table**. Parameters of function matches-pattern. And matches-pattern-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| item | NOTE: this parameter is only expected by function variant \*-ec.  The item to be checked. |
| pattern | The pattern to be matched. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |

***Examples***

Get the names of all elemenets containing a text node matching a unified pattern. As the function is called with a single argument, the item to be inspected is the context item.

fox "doc.xml\\text()[matches-pattern('asp\*')]\..\name()"

As before, but using a full text pattern.

fox "doc.xml\\text()[matches-pattern('morgens und abends#ft')]\..\name()"

### filter-items

**filter-items**($items as item()\*,

$pattern as xs:string)

as item()\*

***Summary***

Filters a sequence of items, retaining those with a string value matching a unified string pattern.

**Details**

Under construction.

***Parameters***

Described by the following table.

**Table**. Parameters of function filter-items.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | The items to be filtered. . |
| pattern | A [Unified Filter Expression](#_Filter_syntax_1). Only items with a matching string value are retained. |

***Examples***

Get the names of all elemenets containing a text node matching a unified pattern. As the function is called with a single argument, the item to be inspected is the context item.

fox "doc.xml\\text()[matches-pattern('asp\*')]\..\name()"

As before, but using a full text pattern.

fox "doc.xml\\text()[matches-pattern('morgens und abends#ft')]\..\name()"

## Processing strings

The functions in this section support the evaluation of strings.

### concat-values

**concat-values**($values as item()\*,

$sep as xs:string?)

as xs:string

***Summary***

Returns a concatenated list of values.

Using options distinct, the list contains only the distinct values. Using options sort or numsort, the list is sorted alphanumerically or numerically.

**Table**. Parameters of function concat-values.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| values | The values to be concatenated. |
| sep | The separator |
| options | Options controlling the processing:  distinct – consider only the distinct values  sort – sort values alphanumerically  numsort – sort values numerically |

### text-to-codepoints

**text-to-codepoints**($text as xs:string?)

as xs:string\*

**text-to-codepoints**()

as xs:string\*

***Summary***

Returns the characters of a text together with their unicode codepoint numbers.

***Details***

If the argument is omitted, it defaults to the context item (.). The behavior of the function if the argument is omitted is exactly the same as if the context item had been passed as the argument.

Maps each item of the input value to a pair of strings, the first containing each character of the original string, separated by 5 blanks, the second containing the unicode codepoints, padded to a string of 6 characters. Example:

'b!

=>

' b !  
 39 98 33

***Parameters***

Described by the following table.

**Table**. Parameters of function text-to-codepoints.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| text | String items |

***Examples***

Inspect the unicode codepoints of some non-letter characters.

fox "tokenize('a@b§c x?y=z') => **text-to-codepoints**()"

=>

a @ b § c

97 64 98 167 99

x ? y = z

120 63 121 61 122

Display the text of an element, associating each character with its unicode codepoint.

fox foo.xml\\font-size[starts-with(., 'Wechsel')]\**text-to-codepoints**()"

=>

W e c h s e l - w i r k u n g e n   
87 101 99 104 115 101 108 45 119 105 114 107 117 110 103 101 110

### truncate (trunc)

**truncate**($len as xs:integer,

$flags as xs:string?)

as xs:string?

**truncate**($len as xs:integer)

as xs:string?

**truncate**($len as xs:integer)

as xs:string?

**truncate-ec**($string as xs:string?,

$len as xs:integer,

$flags as xs:string?)

as xs:string?

**truncate-ec**($string as xs:string?,

$len as xs:integer)

as xs:string?

**truncate-ec**($len as xs:integer)

as xs:string?

***Summary***

Truncates a string, if longer than a maximum length, appending “ …”.

***Details***

If the input string has a length less than or equal to $len, it is returned without changes. Otherwise, a truncated value, with an indicator of truncation (“ …”) appended, is returned. Truncation occurs after $len characters, unless option e is used, mandating truncation after $len – 4 characters. Option e thus ensures that the return value including the indicator of truncation is not longer then $len characters.

***Parameters***

**Table**. Parameters of function truncate.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| string | NOTE: this parameter is only expected by function variant \*-ec.  The string to be truncated. |
| len | The maximum string length allowed without truncation |
| flags | Flags controlling the truncation; values:  e – “even length”, the truncated string consists of the first $len – 4 characters, followed by “…”; by default, the truncated string contains the first $len characters, followed by “ …”. |

***Examples***

When called with a single argument, truncation is applied to the context va; the argument is interpreted as the length allowed without trunction.

fox "'The quick brown fox jumps over the lazy dog'/truncate(20)"

The quick brown fox ...

When called with two arguments, the first argument supplies the string and the second argument the length.

fox "'The quick brown fox jumps over the lazy dog'/truncate(20)"

The quick brown fox ...

The third argument supplies flags. Currently, only one flag is defined, e. When used, truncation is applied after the first $len – 4 characters, ensuring that the truncated string including the indicator of truncation (“ …”) has length $len.

fox "'The quick brown fox jumps over the lazy dog'/truncate(., 20, 'e')"

The quick brown ...

### char-class-report (\*-ec)

**char-class-report**($classes as xs:string,

$options as xs:string?)

as xs:element

**char-class-report-ec**(

$text as xs:string\*,

$classes as xs:string,

$options as xs:string?)

as xs:element

***Summary***

Analyses the use of character classes in a given text.

***Details***

-to-be-added-

Examples

fox output-from-d2cx.finext4//\*fibook.xml\\* => char-class-report-ec()"

<charClassReport>  
 <classes>  
 <letters>  
 <chars n="67">  
 <char s="A" code="65" n="7831"/>  
 <char s="B" code="66" n="4168"/>  
 …  
 <char s="ζ" code="950" n="1"/>  
 <char s="μ" code="956" n="5"/>  
 </chars>  
 </letters>  
 <marks>  
 <chars n="0"/>  
 </marks>  
 <numbers>  
 <chars n="12">  
 <char s="0" code="48" n="5919"/>  
 <char s="1" code="49" n="6232"/>  
 …  
 <char s="9" code="57" n="1853"/>  
 <char s="²" code="178" n="3"/>  
 <char s="½" code="189" n="2"/>  
 </chars>  
 </numbers>  
 <punctuation>  
 <chars n="34">  
 <char s="!" code="33" n="9"/>  
 <char s="&quot;" code="34" n="2"/>  
 <char s="#" code="35" n="17"/>  
 <char s="%" code="37" n="2377"/>

…  
 <char s="”" code="8221" n="3"/>  
 <char s="„" code="8222" n="82"/>  
 <char s="†" code="8224" n="75"/>  
 <char s="‡" code="8225" n="11"/>  
 <char s="•" code="8226" n="32"/>  
 </chars>  
 </punctuation>  
 <separators>  
 <chars n="2">  
 <char s=" " code="32" n="128569"/>  
 <char s=" " code="160" n="5"/>  
 </chars>  
 </separators>  
 <symbols>  
 <chars n="17">  
 <char s="+" code="43" n="738"/>  
 <char s="&lt;" code="60" n="266"/>  
 <char s="=" code="61" n="546"/>  
 <char s="&gt;" code="62" n="60"/>  
 <char s="^" code="94" n="2"/>  
 <char s="®" code="174" n="12"/>  
 <char s="°" code="176" n="44"/>  
 <char s="±" code="177" n="54"/>  
 <char s="×" code="215" n="29"/>  
 <char s="™" code="8482" n="1"/>  
 <char s="→" code="8594" n="1"/>  
 <char s="−" code="8722" n="2"/>  
 <char s="∞" code="8734" n="2"/>  
 <char s="≤" code="8804" n="92"/>  
 <char s="≥" code="8805" n="269"/>  
 <char s="▼" code="9660" n="5"/>  
 <char s="●" code="9679" n="23"/>  
 </chars>  
 </symbols>  
 <other>  
 <chars n="2">  
 <char s="&#xA;" code="10" n="20"/>  
 <char s="­" code="173" n="100"/>  
 </chars>  
 </other>  
 </classes>  
</charClassReport>

### chars

**chars**($text as xs:string)

as xs:string\*

**chars**()

as xs:string\*

***Summary***

Maps a string into a sequence of single chars, represented by string with length one.

***Details***

The string to be processed can be provided as the first and only argument. If no argument is provided, the string is the string value of the current context.

## Processing characters

The functions in this section support the processing of strings.

### replace-mark-chars, replace-chars, mark-chars (\*-ec)

**replace-mark-chars**($item as item()?,

$mark as xs:string?,

$replace as xs:string?)

as node()?

**replace-chars**($item as item()?,

$replace as xs:string?)

as node()?

**mark-chars**($item as item()?,

$mark as xs:string?)

as node()?

**replace-mark-chars-ec**($item as item()?,

$mark as xs:string?,

$replace as xs:string?)

as node()?

**replace-chars-ec**($item as item()?,

$replace as xs:string?)

as node()?

**mark-chars-ec**($item as item()?,

$mark as xs:string?)

as node()?

***Summary***

Replaces characters and/or marks them by inserting unicode codepoint information immediately before them.

***Details***

Only text nodes are processed – attributes are not changed. Characters may be provided as literal string (e.g.: “°”) or as codepoints, using the syntax “#” + decimal codepoint (e.g. #160).-

***Parameters***

Described by the following table.

**Table**. Parameters of function replace-and-mark-chars.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | NOTE: this parameter is only expected by function variant \*-ec.  The document(s) to be analyzed. An atomic item is interpreted as document URI and replaced with the corresponding root element. A node is replaced with the corresponding root element. |
| replace | Character replacements. Whitespace-separated list of items: char1=char2. Example: “Ä=A Ö=O Ü=U #160=#32” |
| mark | Characters to be marked. Whitespace-separated list of characters. Example: “#160 Ä °” |

**Table**. Parameters of function mark-chars.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | NOTE: this parameter is only expected by function variant \*-ec.  The document(s) to be analyzed. An atomic item is interpreted as document URI and replaced with the corresponding root element. A node is replaced with the corresponding root element. |
| mark | Characters to be marked. Whitespace-separated list of characters. Example: “#160 Ä °” |

**Table**. Parameters of function replace-chars.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | NOTE: this parameter is only expected by function variant \*-ec.  The document(s) to be analyzed. An atomic item is interpreted as document URI and replaced with the corresponding root element. A node is replaced with the corresponding root element. |
| replace | Character replacements. Whitespace-separated list of items: char1=char2. Example: “Ä=A Ö=O Ü=U #160=#32” |

***Examples***

Replace “protected whitespace (#160) with blank (#32) and Microsoft Office tracemark (#61650) with the official character (#174).

fox –o doc.edited.xml "doc.xml/replace-chars( '#160=#32 #61650=#174' )"

Mark “protected whitespace (#160) and empty set sign (#8709).

fox –o doc.edited.xml "doc.xml/mark-chars( '#160 #8709' )"

Replace “protected whitespace (#160) with blank (#32), and mark Microsoft Office tracemark (#61650).

fox –o doc.edited.xml "doc.xml/replace-and-markchars( '#160=#32', '#61650' )"

## Full-text evaluation

The functions in this section support the evaluation of strings in accordance with full text concepts.

### contains-text (\*-ec)

**contains-text**(

$ftSearch as xs:string)

as xs:boolean

**contains-text-ec**(

$items as item()\*,

$ftSearch as xs:string)

as xs:boolean

***Summary***

Returns true if one of the supplied items matches a given full-text search.

***Details***

Returns true if one of the supplied items matches a given full-text search. The query string is governed by the full-text query syntax described in [Full-text query](#_Full-text_query). As the query string is treated as a full-text query, not as a [unified filter expression](#_Filter_syntax_1), the option ftext need not be used and is ignored.

***Parameters***

Described by the following table.

**Table**. Parameters of function contains-text and contains-text-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | NOTE: this parameter is only expected by function variant \*-ec.  Input items to which the full text query is applied. |
| ftSearch | Full text query, with syntax described [here](#_Full-text_query) |
| options | Options control various details of the processing.    merge – each input item which is an element or document node is replaced with  the string obtained by concatenating all descendant text nodes  trace – output on stdout the full text expression generated from the full text query text |

**Examples**

Extract the text of paragraphs containing the phrase 'propagating few errors'.

fox "books/books.xml\\p[contains-text('propagating few errors')]\normalize-space()"

Extract the text of paragraphs starting with a particular phrase.

fox "books/books.xml\\p[contains-text('^The usability of a Web site')]\normalize-space()"

Extract the text of paragraphs ending with a particular phrase.

fox "books/books.xml\\p[contains-text('while propagating few errors`$')]\normalize-space()"

Extract the text of elements with a text consisting of a particular phrase.

fox "books/books.xml\\note[contains-text('^This book has been approved by the Web Site Users Association`$')]\normalize-space()"

Extract the text of all titles containing all words from a list of words:

fox "books/books.xml\\title[contains-text('usability testing improving#W')]\normalize-space()"

Extract the text of all titles containing at least one word from a list of words:

fox "books/books.xml\\title[contains-text('astronaut expert eccentric#w')]\normalize-space()"

Check if a document contains all words from a list of words. Note that they may be distributed

fox "books/books.xml\contains-text('association expert marigold usability users#W')"

Check if a document contains a phrase ('web site users association') as well as at least one word from a list of words ('marigold', 'armstrong').

fox "books/books.xml\contains-text('web site users association / marigold armstrong @w')"

Check if a document contains at least one of two phrases.

fox "books/books.xml\contains-text('final task completion | effective task completion')"

Check if a document contains a phrase treated fuzzily.

fox "books/books.xml\contains-text('how well the site suppports#f1')"

Increasing the fuzziness.

fox "books/books.xml\contains-text('how well the site supppports#f2')"

Check if a document contains a phrase containing a word wildcard.

fox "books/books.xml\contains-text('efficient xxx effective task#wild-xxx')"

Extract the text of paragraphs containing three words, with a maximum distance of two words between adjacent terms.

fox "books/books.xml\\p[contains-text('task completion efficient#W dist..2')]\normalize-space()"

Extract the text of paragraphs containing three words, in the given order.

fox "books/books.xml\\p[contains-text('site learning completion#Wo')]\normalize-space()"

### ft-tokenize (fttok)

**ft-tokenize**($text as item()\*, $options as xs:string?)

as xs:string\*

**ft-tokenize**($text as xs:string)

as xs:string\*

**ft-tokenize**()

as xs:string\*

***Summary***

Performs full-text tokenization of text.

***Details***

If arguments are omitted, they default to a single argument which is the context item (.). The behavior of the function if arguments are omitted is exactly the same as if the context item had been passed as the only argument.

Tokenization is applied to the result of atomizing the text items and concatenating them, using a space character as separator. Unless option M is used, concatenation also inserts space characters between consecutive text nodes, ensuring that node borders are also token borders.

By default, tokenization is diacritics insensitive, case insensitive and does not perform stemming. The behaviour can be controlled by options:

* M – when atomizing element or document nodes, do not insert space characters between

consecutive text nodes; a token may then comprise characters from several text

nodes

* c – case sensitive tokenization
* d – diacritics sensitive tokenization
* s – tokenization applies stemming, assuming the default language en
* s-… - tokenization applies stemming, assuming the language …; example: s-de

Tokenizing functionality is delegated to the BaseX extension function ft:tokenize.

***Parameters***

Described by the following table.

**Table**. Parameters of function row.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| text | String items to be tokenized |
| options | Options – see text |

**Examples**

A basic example.

fox "ft-tokenize('The quick brown fox - does it jump over the lazy dog?')"

the

quick

brown

fox

does

it

jump

over

the

lazy

dog

By default, tokenization is case insensitve and diacritics insensitive.

fox "ft-tokenize('Über all Maßen.')"

uber

all

masen

Options c and d mandate case sensitive and diacritics sensitive tokenization.

fox "ft-tokenize('Über all Maßen.', 'c d')"

Über

alle

Maßen

Use option s for stemming.

fox "ft-tokenize('Days and nights', 's')"

dai

and

night

Append the language with a hyphen, unless the default language en is appropriate.

fox "ft-tokenize('Zusammenhänge, Irrtümer, Einsichten', 's-de')"

zusammenhang

irrtum

einsich

Given the following document doc.xml:

<doc>  
 <text><b>Over</b>emphasized.<i>Try to avoid this.</i></text>  
</doc>

By default, consecutive text nodes are separated:

fox "doc.xml\\*\ft-tokenize()"

over

emphasized

try

to

avoid

this

This separation of text nodes can be switch off using option M; in this case, the string values of nodes are used, without inserting separating space characters:

fox "doc.xml\\*\ft-tokenize(., 'M')"

overemphasized

try

to

avoid

this

## Comparison of document contents

The functions in this section support the comparison of documents.

As a starting point, consider the standard function deep-equal() which compares two value and checks them for deep equality. Note that the values are not necessarily single items and are not necessarily nodes. Thus the function may, for example, be used for comparing two sequences of strings. Example:

fox "deep-equal((1,2,3), (1,2,3))"

This means that the comparison of documents requires node arguments, rather than URI arguments. Thus contrary to what you might expect, the following call does not perform a comparison of document contents, but returns false, as two URIs are compared:

fox "input/deep-equal(input1.xml, input2.xml)"

In order to compare the documents, you would need to provide the document nodes explicitly:

fox "input/deep-equal(input1.xml\\*, input2.xml\\*)"

When using the function call as a path step, this requires a somewhat awkward expression:

fox "input/input1.xml/deep-equal(\\*, ../input2.xml\\*)"

The comparison of more than two documents would be very difficult to express.

[To be continued]

### name-diff (\*-ec)

**name-diff**($uriOrNode2 as item(),

$options as xs:string?)

as element(nameDiff)

**name-diff-ec**(

$uriOrNode1 as item(),

$uriOrNode2 as item(),

$options as xs:string?)

as element(nameDiff)

***Summary***

Compares the item names found in two documents and reports the differences.

***Details***

Compares the item names found in two documents or document fractions and reports the differences. Input nodes can be supplied as nodes or document URIs, which are replaced with the corresponding document node. The name variant to be used can be controlled by options (see below, parameter table, options row).

The node name type, as well as various details of the report can be controled by options. In particular:

* Use options lname, name or jname in order to report [local names](#_Local_names), [lexical names](#_Lexical_names) or [JSON names](#_JSON_names), respectively (default: local names)
* Use options common, uncommon, only1, only2 in order to restrict the report to names found in both documents, in only one document, or only the first or only the second document, or use option all in order to exclude nothing
* Use option fname to label documents by file name, rather than document URI

Using function variant name-diff-ec, *both* input URIs or nodes are supplied by the first parameter. Function variant name-diff uses the context item of the function call as the first URI or node and the first parameter as the second URI or node.

If more than two input items are supplied, an error is thrown. This is the case when the first parameter of name-diff-ec has more than two items or the first parameter of name-diff has more than one item.

If less than two input items are supplied, or one of the items is a URI which cannot be parsed into a document node, the empty sequence is returned.

***Parameters***

Described by the following table.

**Table**. Parameters of function name-diff and name-diff-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uriOrNode1 | NOTE: this parameter is only expected by function variant \*-ec.  The node which to compare with another node, which is the value of the second parameter. An atomic item is interpreted as document URI and replaced with the corresponding document node. |
| uriOrNode2 | Another node with which to compare a given node, which is the value of the first parameter (function variant name-diff-ec) or the context node (function variant name-diff). An atomic item is interpreted as document URI and replaced with the corresponding document node |
| options | Whitespace-separated list of options. They control various aspects of the report.  *Options group 1* – node name type used:  lname – local names  jname – JSON names  name – lexical names  *Options group 2* – scope of the report:  only1 – report names occurring only in document 1  only2 – report names occurring only in document 2  uncommon – report names occurring only in document 1 or only in 2  common – report names occurring only in document 1 *and* 2  all – equivalent to specifying uncommon and common  Values from group 2 can be combined, e.g. only1 common  *Options group 3* – miscellaneous:  fname – the report includes file names, rather than file URIs |

***Examples***

Compare two sibling files.

fox "data/msg1.xml/name-diff(../msg2.xml)"

Equivalent to the first example. </description>

fox "data/(msg1.xml, msg2.xml) => name-diff-ec()"

Get a list of names used in both files.

fox "data/(msg1.xml, msg2.xml) => name-diff-ec('common')"

Get a list of names used in both files, as well as names used in only onefile.

fox "data/(msg1.xml, msg2.xml) => name-diff-ec('common uncommon')"

Get a list of names used only in the second file.

fox "data/(msg1.xml, msg2.xml) => name-diff-ec('only2')"

### name-multi-diff (\*-ec)

**name-multi-diff**(

$urisOrNodes as item()+,

$options as xs:string?)

as element()?

**name-multi-diff-ec**(

$urisOrNodes as item()+,

$options as xs:string?)

as element()?

***Summary***

Reports the item names contained in a set of documents or document fragments.

***Details***

Reports the item names contained in a set of documents of document fragments. Input items can be nodes and/or atomic items. Atomic input items are interpreted as document URI and replaced with the corresponding document node. The function reports the names of descendant elements and attributes of the input nodes. Attribute names are preceded by a @ character.

By default, the report comprises the following sections. To request a subset, use the corresponding options:

* The document URIs and fragment paths, when appropriate (option docs)
* The item names contained by all nodes (option common)
* The item names contained by some, but not all nodes (option uncommon)
* For each input node the item names contained by this, but not every other node (option details)

The [node name type](#_Node_name_types) reported can be controled by options name, jname and lname - lexical name, JSON name or local name (default).

In order to request a subset of all possible results, either use a subset of options in order to *include* corresponding sections - docs, common, uncommon, details; or use a subset of options *excluding* corresponding sections - ~docs, ~common, ~uncommon, ~details.

***Parameters***

Described by the following table.

**Table**. Parameters of function name-multi-diff and name-multi-diff-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| urisOrNodes | Function variant name-multi-diff-ec: two or more nodes which to compare.  Function variant name-multi-diff: also the context node is included.  Atomic items are interpreted as document URIs and replaced with the corresponding document node. |
| options | Options control various details of the processing.    Options group 1 – node name kind reported:  lname – local names  jname – JSON names  name – lexical names  Options group 2 – report sections included:  docs – include: document URIs and fragment paths, if appropriate  common – include: data paths contained by all input nodes  uncommon – include: data paths not contained by all input nodes  details – include: for each input node the uncommon data paths it contains  Options group 3 – report sections excluded:  ~docs – exclude: document URIs and fragment paths, if appropriate  ~common – exclude: data paths contained by all input nodes  ~uncommon – exclude: data paths not contained by all input nodes  ~details – exclude: for each input node the uncommon data paths it contains  Options group 4 – miscellaneous:  fname – the report contains file names, rather than file URIs |

***Examples***

Check a set of documents for common/uncommon item names.

fox "data/airports\*.xml => name-multi-diff()"

Restrict the report to names not occurring in all documents.

fox "data/airports\*.xml => name-multi-diff('uncommon')"

Restrict the report to details about the individual documents - the item names contained and not contained by all other documents.

fox "data/airports\*.xml => name-multi-diff('details')"

Get a complete report in which documents are described by file names, rather than URIs.

fox "data/airports\*.xml => name-multi-diff('fname')"

### path-diff (\*-ec)

**path-diff**($uriOrNode2 as item(),

$options as xs:string?)

as element(pathDiff)

**path-diff-ec**(

$uriOrNode1 as item(),

$uriOrNode2 as item(),

$options as xs:string?)

as element(pathDiff)

***Summary***

Compares the data paths found in two documents and reports the differences.

***Details***

Compares the data paths contained by two documents or document fractions and reports the differences. Input nodes can be supplied as nodes or document URIs, which are replaced with the corresponding document node. The function reports the data paths “contained” by the input nodes, more precisely: the data paths connecting the input nodes and the nodes which they contain.

The construction of data paths, as well as the details of comparison can be controled by options. In particular:

* Use options path, path-count, indexed, indexed-value in order to control the type of data path ([index-less](#_Index-less_paths) or [indexed](#_Indexed_paths)) and the type of comparison (whether to include path counts and data values)
* Use options lname, name or jname if data paths should use [local names](#_Local_names), [lexical names](#_Lexical_names) or [JSON names](#_JSON_names), respectively (default: local names)
* Use options common, uncommon in order to restrict the report to names found in both documents or in only one document, or use option all in order to exclude nothing
* Use option keep-ws in order to preserve [pretty print text nodes](#_Pretty_print_text), rather than discard them (only relevant when comparing data values, comparison type indexed-value)
* Use option fname to label documents by file name, rather than document URI

Function variant path-diff compares the context item and the node identified by the first function parameter. Function variant path-diff-ec, compares the node identified by the first parameter with the node identified by the second parameter.

***Parameters***

Described by the following table.

**Table**. Parameters of function path-diff and path-diff-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uriOrNode1 | NOTE: this parameter is only expected by function variant \*-ec.  The node which to compare with another node, which is the value of the second parameter. An atomic item is interpreted as document URI and replaced with the corresponding document node. |
| uriOrNode2 | Another node with which to compare a given node, which is the value of the first parameter (function variant path-diff-ec) or the context node (function variant path-diff). An atomic item is interpreted as document URI and replaced with the corresponding document node |
| options | Whitespace-separated list of options. They control how to construct path strings and how to compare them.  *Options group 1* – type of data path and type of comparison:  path – report [index-less paths](#_Index-less_paths) occurring in only one of the documents  path-count – report [index-less paths](#_Index-less_paths) occurring in only one of the  documents, or occurring a different number of times in the documents  indexed – report [indexed paths](#_Indexed_paths) occurring in only one of the documents  indexed-value – report [indexed paths](#_Indexed_paths) of simple-content items occurring  in only one of the documents, or containing a different string  *Options group 2* – node names used by the path steps:  lname – [local names](#_Local_names)  name – [lexical names](#_Lexical_names)  jname – [JSON names](#_JSON_names)  *Options group 3* – scope of the report:  uncommon – report paths occurring only in document 1 or only in 2  common – report paths occurring only in document 1 *and* 2  all – equivalent to specifying uncommon and common  *Options group 4* – miscellaneous:  keep-ws – do not remove [pretty print text nodes](#_Pretty_print_text) before comparing data  values (removal by default!)  fname – the report includes file names, rather than file URIs |

***Examples***

Check if two documents contain the same data paths, ignoring their frequencies and ignoring data.

fox "data/airports.xml/path-diff(../airports.renamed-elem.xml)"

Check if two documents contain the same data paths with the same frequencies.

fox "data/airports.xml/path-diff(../airports.repeated-elem.xml, 'path-count')"

Check if two documents contain the same indexed data paths.

fox "data/airports.xml/path-diff(../airports.repeated-elem.xml, 'indexed')"

Check if two documents contain the same indexed data paths and each path the same data value.

fox "data/airports.xml/path-diff(../airports.changed-value.xml, 'indexed-value')"

Check if two documents contain the same indexed data paths and each path the same data value. As you are expecting mixed content, do not ignore whitespace text nodes.

fox "data/gardenPreparation.dita/path-diff(../gardenPreparation.changed-ws.dita, 'indexed-value keep-ws')"

Check if two documents contain the same data paths, using lexical names rather than local names. Lexical names may contain a prefix. Note however, that in case of deep equality of the input nodes, no differences will be reported.

fox "data/doc-with-prefix-a.xml/path-diff(../doc-with-prefix-b.xml, 'name')"

Compare two documents; the report should describe the documents by their file names, rather than their document URIs.

fox "data/airports.xml/path-diff(../airports.new-elem.xml, 'fname')"

### path-multi-diff (\*-ec)

**path-multi-diff**(

$urisOrNodes as item()+,

$options as xs:string?)

as element()?

**path-multi-diff-ec**(

$urisOrNodes as item()+,

$options as xs:string?)

as element()?

***Summary***

Reports the data paths contained in a set of documents or document fragments.

***Details***

Reports the data paths contained in a set of documents or document fragments and reports the differences. Input nodes can be supplied as nodes or document URIs, which are replaced with the corresponding document node. The function reports the data paths “contained” by the input nodes, more precisely: the data paths connecting the input nodes and the nodes which they contain.

By default, the reported data paths are [index-less paths](#_Index-less_paths), rather than [indexed paths](#_Indexed_paths). Use option indexed in order to obtain indexed paths.

By default, the report comprises the following sections. To request a subset, use the corresponding options:

* The document URIs and fragment paths, when appropriate (option docs)
* The data paths contained by all nodes (option common)
* The data paths contained by some, but not all nodes (option uncommon)
* For each input node the paths contained by this, but not every other node (option details)

The [node name type](#_Node_name_types) used by the reported paths can be controled by options name, jname and lname - lexical name, JSON name or local name (default).

In order to request a subset of all possible results, either use a subset of options in order to *include* corresponding sections - docs, common, uncommon, details; or use a subset of options *excluding* corresponding sections - ~docs, ~common, ~uncommon, ~details.

***Parameters***

Described by the following table.

**Table**. Parameters of function path-multi-diff and path-multi-diff-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| urisOrNodes | Function variant path-multi-diff-ec: two or more nodes which to compare.  Function variant path-multi-diff: also the context node is included.  Atomic items are interpreted as document URIs and replaced with the corresponding document node. |
| options | Options control various details of the processing.    Options group 1 – node name kind used by data paths:  lname – local names  jname – JSON names  name – lexical names  Options group 2 – report sections included:  docs – include: document URIs and fragment paths, if appropriate  common – include: data paths contained by all input nodes  uncommon – include: data paths not contained by all input nodes  details – include: for each input node the uncommon data paths it contains  Options group 3 – report sections excluded:  ~docs – exclude: document URIs and fragment paths, if appropriate  ~common – exclude: data paths contained by all input nodes  ~uncommon – exclude: data paths not contained by all input nodes  ~details – exclude: for each input node the uncommon data paths it contains  Options group 4 – path kind:  indexed – use [indexed paths](#_Indexed_paths), rather than [index-less paths](#_Index-less_paths)  Options group 5 – miscellaneous:  fname – the report contains file names, rather than file URIs |

***Examples***

Check a set of documents for common/uncommon data paths.

fox "data/airports\*.xml => path-multi-diff()"

Restrict the report to paths not occurring in all documents.

fox "data/airports\*.xml => path-multi-diff('uncommon')"

Restrict the report to details about the individual documents - the paths contained and not contained by all other paths.

fox "data/airports\*.xml => path-multi-diff('details')"

Report indexed paths, rather than index-less paths.

fox "data/airports\*.xml => path-multi-diff('indexed uncommon')"

Get a complete report in which documents are described by file names, rather than URIs.

fox "data/airports\*.xml => path-multi-diff('fname')"

### node-deep-equal (\*-ec)

**node-deep-equal**($urisOrNodes as item()+)

as xs:boolean

**node-deep-equal-ec**($urisOrNodes as item()+)

as xs:boolean

***Summary***

Compares two or more nodes for deep equality.

***Details***

Checks if two or more nodes are [deep-equal](https://www.w3.org/TR/xpath-functions-31/#func-deep-equal). Input nodes can be supplied as nodes or document URIs, which are replaced with the corresponding document node. Atomic input items which cannot be parsed into a document node are silently ignored.

When there are less than two input nodes, the function returns the empty sequence.

Otherwise, the function returns true if all nodes are deep-equal.

***Parameters***

Described by the following table.

**Table**. Parameters of function node-deep-equal and node-deep-equal-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| urisOrNodes | Function variant node-deep-equal-ec: two or more nodes which to compare.  Function variant node-deep-equal: one or more nodes with which to compare the node given or identified by the context item.  An atomic item is interpreted as document URI and replaced with the corresponding document node. |

***Examples***

Compare two sibling files. The function treats the context item as an input item and the argument as other items with which to compare.

fox "msg1.xml/node-deep-equal(../msg2.xml)"

Equivalent to the preceding example. The \*-ec variant does not include the context item – all input items are taken from the argument, which here is the left-hand operand of the arrow operator.

fox "(msg1.xml, msg2.xml) => node-deep-equal-ec()"

Check a set of documents for deep equality. The function can compare any number of input items.

fox "msg\*.xml => node-deep-equal-ec()"

The selection of documents can be made more complex without compromising the simplicity of the expression as a whole – the selection is just an expression to the left of the arrow operator.

fox "(msg\*.xml[not(\\*\@deprecated)] except msg2.xml) => node-deep-equal-ec()"

Compare two elements contained by two documents. Approach: (1) navigate to one of the elements, (2) call the function and supply as argument a navigation to the other element.

fox "msg1.xml\\airport[@id = 612]\node-deep-equal(bsibling('msg2.xml')\\airport[@id = 612])"

If the elements are retrieved by the same expression, a more elegant alternative is available.

fox "(msg1.xml, msg2.xml)\\airport[@id = 612] => node-deep-equal-ec()"

Compare corresponding elements in a set of documents. The correspondance is established by the expression selecting the elements.

fox "msg\*.xml\\airport[@id = 611] => node-deep-equal-ec()"

The selection of documents can be made more complex without compromising the simplicity of the expression as a whole, and the same applies to the selection of an element. The overall structure is stable – (%select-document)\%select-elem => node-deep-equal-ec().

fox "(.//msgs-\*/msg\*.xml[not(\@deprecated)] except msg3.xml)\\city[. eq 'Gronholt']\.. => node-deep-equal-ec()"

Compare selected files with the corresponding file in a different folder.

fox "msgs-a/msg\*.xml/node-deep-equal(fparent-shifted(../../msgs-b))"

Compare elements at corresponding position in different documents.

fox "for `$i in 1 to 10 return (airports-denmark\*.xml\descendant::airport[`$i] => node-deep-equal-ec())"

### node-deep-similar (\*-ec)

**node-deep-similar**(

$urisOrNodes as item()+,

$excludeExpr as xs:string\*

…)

as xs:boolean

**node-deep-similar-ec**(

$urisOrNodes as item()+,

$excludeExprs as xs:string\*

…)

as xs:boolean

***Summary***

Compares two or more nodes for deep-similarity.

***Details***

Checks if two or more nodes are deep-similar. Deep similarity means that after removing content nodes selected by supplied expressions, the compared nodes are [deep-equal](https://www.w3.org/TR/xpath-functions-31/#func-deep-equal). Input nodes can be supplied as nodes or document URIs, which are replaced with the corresponding document node. Atomic input items which cannot be parsed into a document node are silently ignored.

When there are less than two input nodes, the function returns the empty sequence.

Otherwise, the function returns true if all nodes are deep-similar. The nodes to be removed are specified by expressions supplied by the second function argument and further arguments, one expression per argument.

***Parameters***

Described by the following table.

**Table**. Parameters of function node-deep-similar and node-deep-similar-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| urisOrNodes | Function variant node-deep-similar-ec: two or more nodes which to compare.  Function variant node-deep-similar: one or more nodes with which to compare the node identified or supplied by the context item.  Atomic items are interpreted as document URIs and replaced with the corresponding document node. |
| excludeExprs | A Foxpath expression selecting nodes in the content of the nodes to compare; the selected nodes are ignored when comparing. |
| … | The parameter excludeExprs can be repeated an arbitrary number of times, with each repetition supplying an expression. |

***Examples***

Compare two sibling files. The function treats the context item as an input item and the argument as other items with which to compare. As we do not yet exclude any nodes from the comparison, the call is equivalent to a call of function node-deep-equal.

fox "msgs/msg1.xml/node-deep-similar(../msg2.xml)"

Compare two sibling files. When comparing, ignore any @latitude attributes.

fox "msgs/msg1.xml/node-deep-similar(../msg2.xml, '\\@latitude')"

Compare two sibling files. When comparing, ignore any @latitude attributes, also ignore the 'airport' element with @icao equal 'EKAC'.

fox "msgs/msg1.xml/node-deep-similar(../msg3.xml, '\\@latitude', '\\airport[@icao eq """"EKAC""""]')"

Compare a set of documents, ignoring (a) airports/@variant, (b) airport/@source2, (c) all 'city' child elements of 'airport' except of the first one.

fox "airports/\*.xml => node-deep-similar-ec('\airports\@variant', '\\airport\source2', '\\airport\city[position() gt 1]')"

### content-deep-equal (\*-ec)

**content-deep-equal**(

$urisOrNodes as item()+,

$scope as xs:string?)

as xs:boolean

**content-deep-equal-ec**(

$urisOrNodes as item()+,

$scope as xs:string?)

as xs:boolean

***Summary***

Checks if two or more nodes have deep-equal content.

***Details***

Checks if the content of two or more nodes is [deep-equal](https://www.w3.org/TR/xpath-functions-31/#func-deep-equal). Input nodes can be supplied as nodes or document URIs, which are replaced with the corresponding document node. Atomic input items which cannot be parsed into a document node are silently ignored.

The considered content of the input nodes is controlled by the $scope parameter. Dependent on the parameter value, it can be restricted to the attributes, the child nodes or the union of attributes or child nodes. When the parameter value is empty (default), the input nodes themselves are checked, so that node names as well as node content are considered.

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| urisOrNodes | Function variant content-deep-equal-ec: two or more nodes which to compare.  Function variant content-deep-equal: one or more nodes with which to compare the context item.  Atomic items are interpreted as document URIs and replaced with the corresponding document node. |
| scope | Specifies what to compare:  c – compare content, that is, attributes and child nodes  n – compare child node content  a – compare attribute content  s – compare the nodes themselves, including their name |

***Examples***

Compare two documents with different root names for equal attribute and child node content.

fox "data/airports.xml/content-deep-equal(../airports-root-renamed.xml)"

Compare two root elements for equal attribute content, ignoring child nodes.

fox "data/airports.xml\\*\content-deep-equal(base-uri(.)/../airports-elems-renamed.xml, 'a')"

Check if every 'airport' element in one document has the same child node content as the corresponding 'airport' element in another document.

fox "every `$airport in data/airports.xml\\airport satisfies   
 content-deep-equal-ec(  
 (`$airport, data/airports-atts-removed.xml\\airport[@icao eq `$airport\@icao]),   
 'n')"

Check if every 'airport' element in one document has the attribute and child node content as the corresponding 'airport' element in another document. Note that the $scope parameter defaults to 'c' which means the attribute and child node content.

fox "every `$airport in data/airports.xml\\airport satisfies   
 content-deep-equal-ec(  
 (`$airport, data/airports-atts-removed.xml\\airport[@icao eq `$airport\@icao]))"

Compare a set of elements for equal child node content.

fox "data2/airports\*xml\\airport[@icao eq 'EKGH'] => content-deep-equal-ec('n')"

Compare a set of elements for equal attribute and child node content. As the $scope value 'c' is the default value, it might be omitted.

fox "data2/airports\*xml\\airport[@icao eq 'EKGH'] => content-deep-equal-ec('c')"

Compare a set of elements for equal names and equal content. This means checking for deep equality, so that also function nodes-deep-equal() might be used.

fox "data2/airports\*xml\\airport[@icao eq 'EKGH'] => content-deep-equal-ec('s')"

## Processing the file system

Bla.

### file-append-text

**file-append-text**(

$filePath as xs:string,

$data as item(),

$encoding as xs:string?)

as empty-sequence()

**file-append-text**($file as xs:string)

as empty-sequence()

***Summary***

Appends text to a file. If the file does not yet exist, it is created.

***Examples***

Appends the contents of a text file to a result file:

fox "/programme/\*oxygen\*//sound.properties/file-content()/**file-append-text**('copies.txt')"

### file-append-text-lines

**file-append-text-lines**(

$filePath as xs:string,

$data as item()\*,

$encoding as xs:string?)

as empty-sequence()

***Summary***

Appends text lines to a file. To each line, a newline character will be appended. If the file does not yet exist, it is created.

***Examples***

Write the file paths of XSLT stylesheets with version 1.0 into a file:

fox "/programme/\*oxygen\*//\*.xsl[\\*\@version eq '1.0']/**file-append-text-lines**('uris-xslt1.txt')"

Collect a set of .bat files into a single text file, where each file content is preceded by the file path and surrounded by empty lines:

fox "/programme/\*oxygen\*//\*.bat/**file-append-text-lines**('bats.txt', ('\*\*\* '||.||' \*\*\*', '', file-content(.), ''))"

### file-basename (file-bname, fbname)

**file-basename**($path as xs:string\*) as

xs:string

**file-basename**()

as xs:string\*

***Summary***

Extracts from a filepath or URI the file name without file name extension.

**Details**

Extracts from the input URIs or file paths the file base names. They are obtained by extracting the file name and removing the trailing substring starting with the last occurrence of a dot. If the file name does not contain a dot, the complete file name is returned.

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

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | A sequence of URIs or file paths. |

**Examples**

Get the distinct file base names of all files in the current work folder.

fox "\*[is-file()]/file-basename() => distinct-values()"

Geht the distinct file base names used by several files with different extensions.

fox "\*[is-file()]/fbname() => freq(2)"

### file-contains

**file-contains**($fileUri as xs:string,

$pattern xs:string,

$encoding as xs:string?)

as xs:boolean

**file-contains**($fileUri as xs:string,

$pattern xs:string)

as xs:boolean

**file-contains**($fileUri as xs:string)

as xs:boolean

***Summary***

Returns true if a given file contains a substring matching a given [pattern-or-regex](#_Pattern-or-regex).

***Details***

Returns true if the file identified by the URI or path exists and contains a substring matching a [pattern-or-regex](#_Pattern-or-regex). Returns false if the file exists, but does not contain a matching substring. Returns the empty sequence if no such file exists.

***Parameters***

Described by the following table.

**Table**. Parameters of function file-contains. If a single parameter is used, it is interpreted as $pattern and the $uri value is provided by the context item.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| $uri | URI or file path of the file to be analyzed. |
| $pattern | A string pattern or regular expression, optionally followed by flags. Flags seperated from the pattern by an unescaped # character. Literal backslash and # characters in the glob pattern must be escaped by a preceding backslash. Flags:  r – the pattern string is a regular expression, rathern than a glob pattern  c – perform case sensitive matching |
| $encoding | Encoding, e.g. “utf16”. Default: utf8. |

***Examples***

List all files in the working directory containing a substring “kapit\*”. Matching is case-insensitive.

fox "\*[is-file()][file-contains('kapit\*')]"

As before, but match case-sensitively.

fox "\*[is-file()][file-contains('kapit\*#c')]"

As before, but match a regular expression. Note the escaping of the literal # character.

fox "\*[is-file()][file-contains('\#d\d+#r')]"

### file-content (fcontent)

**file-content**($fileUri as xs:string?,

$encoding as xs:string?,

$start as xs:integer?,

$length asxs:string?)

as xs:string?

**file-content**($fileUri as xs:string?,

$encoding as xs:string?,

$start as xs:integer?)

as xs:string?

**file-content**($fileUri as xs:string?,

$encoding as xs:string?)

as xs:string?

**file-content**($fileUri as xs:string?) #

as xs:string?

**file-content**()

as xs:string?

***Summary***

Returns the text content of a given file.

***Details***

Returns the text content of a given file. A call without arguments is equivalent to a call with a single argument, which is the context item.

The encoding can be specified by a call parameter. Default encoding is UTF-8.

If parameter $start is used, only a substring starting at this position is returned. A negative parameter value is interpreted as “string-length + $start + 1”.

The optional parameter $length limits the returned string to a maximum length.

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| $uri | URI or file path of the file to be analyzed. Default value: context item. |
| $encoding | Encoding, e.g. “utf16”. Default: utf8. |
| $start | Return substring of the text content starting at this position. A negative value is interpreted as “string-length + $start + 1”. |
| $length | Maximum length of the string returned. |

***Examples***

Display file content.

fox "summary.txt/file-content()"

Display file content, encoding UTF-16.

fox "boing.txt/file-content(., 'utf16')"

Display file content, only the first 200 characters.

fox "log.txt/file-content(., (), (), 200)"

Display file content, only the last 1000 characters.

fox "log.txt/file-content(., (), -1000)"

### file-copy (fcopy)

**file-copy**($fileUris as xs:string\*,

$targetUri as xs:string,

$flags as xs:string?)

as empty-sequence()

**file-copy**($fileUris as xs:string\*,

$targetUri as xs:string)

as empty-sequence()

***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-tree-copy (ftcopy)

**file-tree-copy**($resources as item()\*,

$targetUri as xs:string,

$sourceContext as xs:string?,

$rename as xs:string?

$flags as xs:string?)

as empty-sequence()

***Summary***

Copies resources as a file tree, preserving the tree of containing folders.

***Details***

The resources may be files and folders. They are specified as resource URI, resource node or doc-resource.

Note that input doc-resources may be produced by a pipeline of document modifying functions.

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| resources | Resources to be copied, supplied as URIs or resource-docs |
| targetContext | The target file paths relative to $targetContext are equal to the source file paths relative to $sourceContext. |
| sourceContext | Context URI used when determining the relative source file paths which are appended to $targetContext in order to determine the target file paths. Default value: the deepest folder containing all resources to be copide. |
| rename | Optional renaming of files; syntax: from=to, where from and to are used as parameters 2 and 3 of standard function fn:replace. Examples: “$0=$0.sav”. |
| flags | Whitespace-separated list of flags. Supported item values:   * indent – copied node resources aare indented |

***Examples***

// Copies files into a target context

fox ../write-fibook-bulk/output//\*fibook\* => file-tree-copy('export ')

// Edits documents and copies the result documents into a target context

fox ../write-fibook-bulk/output//\*fibook\*/doc-resource()

/delete-nodes({\\\*:annotation})

=> file-tree-copy('export ')

### file-date (fdate)

**file-date**($fileUri)

as xs:dateTime

**file-date**()

as xs:dateTime

***Summary***

Returns the timestamp of the last modification of a file or folder.

***Details***

The timestamp is returned as an xs:dateTime value. Use function [file-date-string](#_file-date-string_,_fdates) in case you prefer a string result, e.g. in order to compare it with a date string like “2022-03”.

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

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| fileUri | File URIs or paths of the file or folder to be described |

***Examples***

// Returns the file date of a file specified explicitly

fox file-date(request.xml)

// Returns the names and file dates of all XML files in the current work folder which are older than one day

fox "\*.xml[file-date() < current-dateTime() - dayTimeDuration('P1D')]/file-name()"

### file-sdate (fsdate)

**file-sdate**($fileUri)

as xs:string

**file-sdate**()

as xs:string

***Summary***

Returns the string value of the timestamp of the last modification of a file or folder.

***Details***

The timestamp is returned as a string. Use function [file-date](#_file-date_,_fdate) in case you prefer the file date as an xs:dateTime value. The string value enables a simple comparison with a string, e.g. ".../file-date-string(.) lt '2022' ".

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

***Parameters***

Described by the following table.

**Table**. Parameters of function file-date-string.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| fileUri | File URIs or paths of the file or folder to be described |

***Examples***

// Returns the file date of a file specified explicitly

fox file-date-string(request.xml)

// Returns the names of all XML files in the current work folder with a file date less than “2022-04”.

fox "\*.xml[file-date-string() < '2022-04']/file-name()"

### file-exists

**file-exists**($fileUri as xsLstring)

as xs:boolean

**file-exists**()

as xs:boolean

***Summary***

Returns true or false, dependent on whether a file exists or not.

### file-extension (file-ext, fext)

**file-extension**($fileUri)

as xs:string

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**()

as:string

**file-info**($format)

as:string

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')"

### file-lines (flines)

**file-lines**($uri as xs:string, $line1 as xs:integer, $line2 as xs:integer,

$pattern as xs:string?)

as xs:string\*

**file-lines**($uri as xs:string, $line1 as xs:integer, $line2 as xs:integer)

as xs:string\*

**file-lines**($uri as xs:string, $line1 as xs:integer)

as xs:string\*

**file-lines**($uri as xs:string)

as xs:string\*

**file-lines**()

as xs:string\*

***Summary***

Returns all or selected lines from a file.

### file-name (fname)

**file-name**($uri as xs:string?)

as xs:string?

**file-name**()

as xs:string?

***Summary***

Returns the file name extracted from a URI.

***Details***

#Give a detailed description.

***Parameters***

Described by the following table.

**Table**. Parameters of function xyz.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| Uri | A URI |

**Examples**

Get the file names of the files contained by the current workfolder.

fox "\*[is-file()]/fname()"

### file-sdate (fsdate)

**file-sdate**($uri as xs:string?)

as xs:string?

**file-sdate**()

as xs:string?

***Summary***

Returns the date of last modification, as a string.

***Details***

#Give a detailed description.

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | A URI |

**Examples**

Get the file names of the files contained by the current workfolder with a last modification time greater than 6 PM.

fox "\*[is-file()][fsdate()/substring-after(., 'T') gt '18']"

### file-size (fsize)

**file-size**($uri as xs:string?)

as xs:integer?

**file-size**()

as xs:string?

***Summary***

Returns the size of a file, as number of bytes.

***Details***

If the argument is omitted, it defaults to the context item (.). The behavior of the function if the argument is omitted is exactly the same as if the context item had been passed as the argument.

Returns the size of a file, as number of bytes. Returns 0, if the file is a folder. Returns the empty sequence, if the file does not exist.

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | A URI |

**Examples**

Get the file paths of empty files.

fox "/products/x4//\*[is-file()][file-size() eq 0]"

### folder-size (fosize)

**folder-size**()

as xs:decimal?

**folder-size**($options as xs:string?)

as xs:decimal?

**folder-size-ec**($uri as xs:string)

as xs:decimal?

**folder-size-ec**($uri as xs:string, $options as xs:string?)

as xs:decimal?

***Summary***

Returns the size of a folder, or several folders, understood as the sum of the sizes of contained files.

***Details***

By default, files found at any depth are considered. Using option flat, only the files immediately contained by the input folders are considered.

By default, the size is returned as number of bytes. Use options mb or kb in order to get the result as number of megabytes or number of kilobytes.

Returns the empty sequence if the folders does not exist.

***Parameters***

Described by the following table.

**Table**. Parameters of function folder-size.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uris | NOTE: this parameter is only expected by function variant \*-ec. Folder URIs |
| options | Whitespace-separated list of options.  flat – consider only files immediately contained  deep – consider files found at any level under the input folders  mg – return the size in megabytes  kb – return the size in kilobytes |

**Examples**

Get the size of all files in order the curret working directory.

fox "fosize()"

Get a list of folders, annotated with the folder size in kilobytes.

fox "/programme/\*oxy\*25/samples/\*/annotate(fosize('kb'))"

Get the size of all files in a set of folders, in megatypes.

fox "/programme/\*oxy\*25/frameworks//dita-ot\* => fosize-ec('mb')"

## Creation of tables and hierarchical lists

Bla.

### table

**table**($rows as xs:string\*,

$header as xs:string\*,

$options as xs:string?)

as xs:string

**table**($rows as xs:string\*,

$header as xs:string\*)

as xs:string

**table**($rows as xs:string\*)

as xs:string

**table**()

as xs:string

***Summary***

Represents input data as a table.

***Details***

The data input consists of rows created using function row(). An optional second parameter specifies column headers. The headline is either a sequence of strings or a single string which is a concatenated list, using comma as a separator. Options are available for sorting table rows and discarding duplicate rows.

***Parameters***

Described by the following table.

**Table**. Parameters of function table.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| rows | Each item supplies the values of a single row. The items should be produced by function row(). |
| headers | Column headers. If the value is a single string, it is tokenized, splitting the string at each comma followed by optional whitespace. |
| options | Options controling details of function behaviour.  sort – table rows are sorted ascendingly  sortd – table rows are sorted descendingly  distinct – only distinct table rows are retained  xml – table is rendered as XML;  default element names: table, row, col1, col2, …  Use $headers in order to control element names:  table name: … from $headers item table=…,  row name: … from $headers item row=…,  column names: $headers items;  $headers example: name, meaning, table=elems, row=elem |

**Examples**

Write a two-column table describing DITA topic files - file name and title.   
fox "dita//\*.dita\\*\row(base-file-name(), string(title\normalize-space()/truncate(60))) => table('File name, Title')"

Write a table describing DITA files, with sorted rows.   
fox "dita//\*.dita\\*\row(base-file-name(), string(title\normalize-space()/truncate(60))) => table('File name, Title', 'sort')"

Write a table describing DITA files, with sorted rows and duplicate rows removed.   
fox "dita//\*.dita\\*\row(base-file-name(), string(title\normalize-space()/truncate(60))) => table('File name, Title', 'sort distinct')"

Write a table describing DITA files, ignoring DITA documents without title.   
fox "dita//\*.dita\\*[title]\row(base-file-name(), string(title\normalize-space()/truncate(60))) => table('File name, Title', 'sort distinct')"

Write a three-column table describing DITA topic files - file name, title and short description.  
fox "dita//\*.dita\\*\row(base-file-name(), string(title\normalize-space()/truncate(60)), string(shortdesc\normalize-space(.)/truncate(80))) => table('File name, Title, Short description')"

### hlist

**hlist**($hlistEntries as array(\*)\*,

$options as xs:string?)

as xs:string

***Summary***

Returns a hierarchical list, grouping input tuples from left to right.

***Details***

The input tuples must be created by functions tuple().

The tuples are grouped from left to right, that is:

* First-level grouping: by the first items of the tuples
* Second-level grouping: by the second items of the tuples
* …

Tuple items may also be multiple strings. Example:

tuple(base-dir-rel(), ., base-file-name(), ..\xs:element\@name => sort())

The grouping level is indicated by indentation. The indentation string consists of concatenated substrings, one substring per level. The substring is by default a dot followed by three blanks. The following strings are the indentation strings for the second, third, fourth level of grouping, respectively:

" . " , " . . ", " . . . "

The readability of the hlist can be enhanced using options emptylines=…, char=… or nochar.

Use option char=… in order to replace the dot with a different character, for example char=|. Use option nochar in order to replace the dot with a blank.

Use option emptylines=… in order to control the insertion of empty lines into the list. The option value is a sequence of digits: the first (second, third, …) digit is the number of empty lines inserted before each new value on the first (second, level, …) level. Example: emptylines=110 requests one empty line before each new value on the first and second level.

***Parameters***

Described by the following table.

**Table**. Parameters of function row.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| tuples | A sequence of value tuples. All tuples should contain the same number of items. |
| headers | A comma-separated list of terms used as column headers for column 1, 2, … |
| options | Options controlling the processing:  emptylines=xyz – insert x (y, z) empty lines before every new item on level 1 (2, 3)  char=x – use charcter x within the indentation string as indicator of levels  nochar – do not write dots indicating the grouping levels |

***Examples***

Write a list of XSD names, grouped by directory path and target namespace. The directory path is relative to the current working dir. Column headers are “Dir”, “TNS” and “File”. Insert an empty line before each new directory name.

fox "frameworks/tei//\*.xsd\\*\@targetNamespace

\tuple(base-dir-rel(), ., base-file-name())

=> hlist('Dir, TNS, File','emptylines=110')"

=>

=========================================================================

Dir

. TNS

. . File

=========================================================================

frameworks/tei/xml/tei/custom/schema/xsd

. http://www.isocat.org/ns/dcr

. . tei\_all\_dcr.xsd

. . tei\_basic\_dcr.xsd

. . tei\_ms\_dcr.xsd

. . tei\_speech\_dcr.xsd

. http://www.tei-c.org/ns/1.0

. . tei\_all.xsd

. . tei\_bare.xsd

frameworks/tei/xml/tei/stylesheet/profiles/iso/schema

. http://relaxng.org/ns/compatibility/annotations/1.0

. . a.xsd

. http://relaxng.org/ns/structure/1.0

. . rng.xsd

. . structure.xsd

. http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing

. . w.xsd

…

Write a list of XSD top-level element names, grouped by directory path, target namespace and file name. The directory path is relative to the current working dir. Column headers are “Dir”, “TNS”, “File” and “Elem”. Insert an empty line before each new directory name and before each new target namespace. Use the pipe character as level-indicator.

> fox "frameworks/tei//\*.xsd\\*\@targetNamespace

\tuple(base-dir-rel(), ., base-file-name(), ..\xs:element\@name => sort())

=> hlist('Dir, TNS, File, Elems','emptylines=110 char=|')"

=============================================================  
Dir  
| TNS  
| | File  
| | | Elems  
=============================================================

frameworks/tei/xml/tei/stylesheet/profiles/iso/schema

| http://www.iso.org/ns/1.0  
| | ns2.xsd  
| | | wordObject  
  
| http://www.lisa.org/TBX-Specification.33.0.html  
| | t.xsd  
| | | admin  
| | | descrip  
| | | descripGrp  
| | | descripNote  
| | | hi  
| | | langSet  
| | | note  
| | | ntig  
| | | ref  
| | | term  
| | | termComp  
| | | termCompGrp  
| | | termCompList  
| | | termEntry  
| | | termGrp  
| | | termNote  
  
| http://www.oasis-open.org/specs/tm9901  
| | tm9901.xsd  
| | | colspec  
| | | entry  
| | | row  
| | | table  
| | | tbody  
| | | tgroup  
| | | thead  
| | | title

…

### tuple

**tuple**($value as item() …)

as array(\*)

***Summary***

Packs items into the internal representation of a row.

***Details***

Row content is extracted by functions using the row, like hlist() or table().

***Parameters***

Described by the following table.

**Table**. Parameters of function row.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| item, … | Every parameter is treated as a row column. |

**Examples**

Gets this or that.

## Editing documents

…

### doc-resource (\*-ec)

**doc-resource**()

as map(\*)?

**doc-resource-ec**(

$uriOrNode as item())

as map(\*)?

***Summary***

Creates a “doc-resource”, which is a map with maps containing the document URI and the root node.

***Details***

Bla.

***Parameters***

Described by the following table.

**Table**. Parameters of function doc-resourse and doc-resource-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| $uriOrNode as item() | NOTE: this parameter is only expected by function variant \*-ec.  The input document, provided as document URIs or as a node from its content. A node is replaced with the root node of the containing node tree. |

***Examples***

Creates doc-resources which are modified and copied into a file tree.

fox "output//\*fibook\*/doc-resource()

/delete-nodes({\\@pzn, \\\*:fi-stand})

=> file-tree-copy('tmp98')"

### insert-nodes-doc (\*-ec)

**insert-nodes**(

$insertWhereExpr as xs:string,

$insertValuesOrNodesExpr as xs:string,

$nodeName as xs:string?,

$options as xs:string?)

as node()?

**insert-nodes-ec**(

$docs as item()\*,

$insertWhereExpr as xs:string,

$insertValuesOrNodesExpr as xs:string,

$nodeName as xs:string\*,

$options as xs:string?)

as node()?

***Summary***

Inserts nodes into documents.

***Details***

Using function variant insert-nodes, the receiving document is identified or supplied by the context item, whereas using variant insert-nodes-ec the receiving documents are supplied by the first parameter. Input documents can be supplied as nodes or identified by document URIs.

Nodes are inserted at a location related to the nodes selected by parameter $insertWhereExpr. By default, new nodes are inserted as last child elements of these nodes. Use options first, before, after in order to insert new nodes as first child, or immediately before, or immediately after the nodes selected by $insertWhereExpr.

If parameter $nodeName is used, parameter $insertValuesOrNodesExpr is a Foxpath expression returning the *content* of the new node, which is wrapped in a node with the name given by $nodeName. The Foxpath expression is evaluated in the context of the current node selected by $insertWhereExpr. If option foreach is used, one node will be constructed for each item in the value of $insertValuesOrNodesExpr; otherwise, only one node is constructed, which is filled with the value of $insertValuesOrNodesExpr.

If parameter $nodeName is not used, the expression $insertValuesOrNodesExpr constructs the *nodes* to be inserted, and or strings which will automatically be wrapped in a new text node. In this case, option foreach is ignored.

***Parameters***

Described by the following table.

**Table**. Parameters of function insert-nodes and insert-nodes-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| docItems | NOTE: this parameter is only expected by function variant \*-ec.  The input documents, provided as document URIs or as nodes from their content. Nodes are replaced with the root node of the containing node tree. |
| insertWhereExpr | A Foxpath expression selection the nodes receiving new content. The expression is resolved in the context of the input node, or the document node of the input document specified by document URI. |
| insertValuesOr  NodesExpr | A Foxpath expression providing the content of inserted nodes (if parameter $nodeName is used), or the inserted nodes themselves (otherwise). |
| nodeName | If specified, the value returned by $insertValuesOrNodesExpr is wrapped in a node with this name. If the name is preceded by a @ character the node is an attribute, otherwise an element. |
| options | Options controling details of function behaviour.  Options group 1 – relative location of new nodes:  first – first child node of the current node selected by $insertWhereExpr  last – last child node of the current node selected by $insertWhereExpr  before – immediately before the current node selected by  $insertWhereExpr  after – immediately after the current node selected by $insertWhereExpr  Options group 2 - repetition  foreach – for each item in the value of $insertValuesOrNodesExpr a node with a name given by $nodeName is created; the option is ignored if $nodeName is not spefied  Options group 3 - miscellaneous  base – the result document has an @xml:base attribute inserted into the root element. This is especially useful if the result document shall be written into a file with the original file name or a name derived from the original file name. |

***Examples***

Insert into the root element a 'count' attribute containing the number of airports.

fox "airports/airports.xml/insert-nodes('\airports', 'count(airport)', '@count')"

Augment each 'geo' element with a further child element 'coordinates', containing the latitude and longitude rounded to 3 fractional digits.

fox "airports/airports.xml/insert-nodes('\\geo',

{round(latitude, 3)||'/'||round(longitude, 3)}, 'coordinates')/pretty-node()"

As the preceding example, but insert new elements as first child of their parent.

fox "airports/airports.xml/insert-nodes('\\geo',

{round(latitude, 3)||'/'||round(longitude, 3)}, 'coordinates', 'first')/pretty-node()"

Insert after every 'temporal' element copies of the child elements of 'temporal'.

fox "airports/airports.xml/insert-nodes('\\temporal', '\*', (), 'after')/pretty-node()"

Insert an element constructed using function xelem().

fox "airports/airports.xml/insert-nodes('\\*',

{xelem-ec(\\city, 'cities')}, (), 'first')/pretty-node()"

Chain two insertions: insert a 'cities' element, followed by an 'ids' element.

fox "airports/airports.xml  
 /insert-nodes('\\*', {xelem-ec(\\city, 'cities')}, (), 'first')  
 /insert-nodes('\\cities', {\\@id => string-join(' ')}, 'ids', 'after')  
 /pretty-node()"

Processing a set of documents, chaining two insertions, and write the results into an output folder.

fox "airports/\*.xml  
 /insert-nodes('\\*', {xelem-ec(\\city, 'cities')}, (), 'first')  
 /insert-nodes('\\cities', {\\@id => string-join(' ')}, 'ids', 'after')  
 /pretty-node()/write-doc('output')"

Equivalent to the preceding example, using the ec variant of the function which may consume multiple input nodes.

fox "airports/\*.xml  
 => insert-nodes-ec('\\*', {xelem-ec(\\city, 'cities')}, (), 'first')  
 => insert-nodes-ec('\\cities', {\\@id => string-join(' ')}, 'ids', 'after')  
 => pretty-node-ec() => write-doc-ec('output')"

### delete-nodes (\*-ec)

**delete-nodes**($excludeExprs as xs:string\*,

$options as xs:string?)

as node()?

**delete-nodes-ec**(

$uriOrNodes as item()\*,

$excludeExprs as xs:string\*,

$options as xs:string?)

as node()?

***Summary***

Creates reduced copies of documents, removing content nodes.

***Details***

Creates reduced copies of documents, removing content nodes. The documents are specified by document URI or supplied as a node. Document URIs are replaced with the document node, and non-root nodes are replaced with the root node of the containing node tree. Content nodes selected by the expressions $excludeExprs are removed. The expressions are evaluated in the context of the document root node.

Unless option keepws is used, existing [pretty print text nodes](#_Pretty_print_text) are removed and new ones are created, in order to construct a document which is regularly indented and does not contain “gaps” consisting of former pretty print text nodes surrounding a deleted element.

***Parameters***

Described by the following table.

**Table**. Parameters of function reduce-doc and reduce-doc-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| urisOrNode | NOTE: this parameter is only expected by function variant \*-ec.  The node to be reduced. An atomic item is interpreted as document URI and replaced with the corresponding document node. |
| excludeExprs | One or several expressions selecting nodes in the content of the nodes to compare; the selected nodes are ignored when comparing. Expressions are comma-separated. |
| options | Options controling the function behaviour.  base – add an @xml:id attribute to the root element  keepws – [pretty print text nodes](#_Pretty_print_text) are not reconstructed, so that the result  document is likely to contain irregular chunks of whitespace which  surrounded the deleted elements |

***Examples***

Remove source1 elements.

fox "airports/airports.xml/delete-nodes('\\source1')"

Remove source1 elements, as well as @id attributes.

fox "airports/airports.xml/delete-nodes('\\source1, \\@id')"

Remove source1 elements, as well as @id attributes; add an @xml:base attribute to the root element.

fox "airports/airports.xml/delete-nodes('\\source1, \\@id', 'base')"

Modify a set of documents and write the resulting documents into a folder.

fox "airports/\*.xml => delete-nodes-ec('\\source1, \\@id') => write-doc('output')"

Equivalent to the preceding example; using several expressions instead of a single sequence expression.

fox "airports/\*.xml => delete-nodes-ec(('\\source1', '\\@id')) => write-doc('output')"

### replace-values (\*-ec)

**replace-values**($targetNodesExpr as xs:string,

$valueExpr as xs:string,

$options as xs:string?)

as node()?

**replace-values-ec**(

$docItem as item(),

$targetNodesExpr as xs:string,

$valueExpr as xs:string,

$options as xs:string?)

as node()?

***Summary***

Replaces the values of selected nodes.

***Details***

[Under construction]

***Parameters***

Described by the following table.

**Table**. Parameters of function replace-values and replace-values-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| docItem | NOTE: this parameter is only expected by function variant \*-ec.  The input documents, provided as document URI or as a node from its content. |
| targetNodesExpr | A Foxpath expression selecting the nodes which to change. |
| valueExpr | A Foxpath expression returning the new value of the node; the expression is evaluated in the context of the node to be changed. |
| options | Options controling details of function behaviour.  base – the result document has an @xml:base attribute inserted into the root element. This is especially useful if the result document shall be written into a file with the original file name or a name derived from the original file name. |

***Examples***

Change @longitude and @latitude attributes, using decimal numbers with three decimal digits.

fox "airport\*.xml/replace-values('\\(@latitude, @longitude)', 'format-number(., """"##9.999"""")')"

As in the first example, but also add an @xml:base attribute to the root element and write the file into sub folder "edited", retaining the file name.

fox "airport\*.xml/replace-values('\\(@latitude, @longitude)', 'format-number(., """"##9.999"""")' , 'base') => write-files('edited')"

Chain various modifications - two value replacements, as well as the addition of a @count attribute and the deletion of 'source1' elements.

fox "airports/airport\*.xml/replace-values('\\(latitude, longitude)', 'format-number(., """"##9.999"""")' , 'base')/replace-values('\\dst', 'lower-case(.)')/insert-nodes('\airports', 'count(\*)', '@count')/delete-nodes('\\source1')/pretty-node() => write-files('output')"

### iexpand-nodes (\*-ec)

**iexpand-nodes**($targetNodesExpr as xs:string,

$grammar as xs:string,

$options as xs:string?)

as node()?

**iexpand-nodes-ec**(

$docItem as item(),

$targetNodesExpr as xs:string,

$grammar as xs:string,

$options as xs:string?)

as node()?

***Summary***

Expands attributes or text nodes using a grammar and writing the parse tree into a new child element of the parent of the target node.

***Details***

Target nodes may be text nodes, attributes or elements with simple content. The text value is parsed using the grammar identified by $grammar. A new element is inserted into the document, containing the parse tree. The element name is composed of the fox prefix and the local name of the target node (if element or attribute) or the parent node of the target node (if a text node).

***Parameters***

Described by the following table.

**Table**. Parameters of function iexpand-nodes and iexpand-nodes-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| docItem | NOTE: this parameter is only expected by function variant \*-ec.  The input documents, provided as document URI or as a node from its content. |
| targetNodesExpr | A Foxpath expression selecting the nodes to expand. |
| grammar | The name or resource URI of a grammar. If a name is supplied (example: #xpath31), it must be prefixed by a # character, followed by a grammar name found in a <grammar> element of the [Infospace definition](#_Unified_String_Expression). |
| Options | Options controling details of function behaviour.  pretty – remove from the result document pretty print nodes, thus enabling a clean indentation  base – the result document has an @xml:base attribute inserted into the root element. This is especially useful if the result document shall be written into a file with the original file name or a name derived from the original file name. |

***Examples***

Expand @match attributes of an XSLT stylesheet.

fox "frameworks/tei/descendant~::\*.xsl[1]/iexpand-nodes({\\@match}, '#xpath31')"

### rename-nodes (\*-ec)

**rename-nodes**($targetNodesExpr as xs:string,

$nameExpr as xs:string,

$options as xs:string?)

as node()?

**rename-nodes-ec**(

$docItems as item()\*,

$targetNodesExpr as xs:string,

$nameExpr as xs:string,

$options as xs:string?)

as node()?

***Summary***

Modifies the input documents, renaming selected nodes.

***Details***

[Under construction]

***Parameters***

Described by the following table.

**Table**. Parameters of function rename-nodes and rename-nodes-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| docItems | NOTE: this parameter is only expected by function variant \*-ec.  The input documents, provided as document URI or as a node from its content. |
| targetNodesExpr | A Foxpath expression selecting the nodes to be renamed. |
| nameExpr | A Foxpath expression returning the new name of the node; the expression is evaluated in the context of the node to be renamed. |
| options | Options controling details of function behaviour.  base – the result document has an @xml:base attribute inserted into the root element. This is especially useful if the result document shall be written into a file with the original file name or a name derived from the original file name. |

***Examples***

Rename 'source1' elements to 'source'.

fox "airports/airports.xml/rename-nodes('\\source1', '""""source""""')"

Rename all elements, changing the first character into uppercase.

fox "airports/airports.xml/rename-nodes('\\\*', 'concat(upper-case(substring(local-name(.), 1, 1)), substring(local-name(.), 2))')"

## Writing files

…

### write-doc

**write-doc**(

$folderPath as xs:string,

$options as xs:string?)

as empty-sequence()

**write-doc**(

$folderPath as xs:string)

as empty-sequence()

**write-doc-ec**(

$urisOrNodes as item()\*,

$folderPath as xs:string,

$options as xs:string?)

as empty-sequence()

***Summary***

Writes documents or document fragments into a folder.

***Details***

Documents can be supplied as document nodes or identified by document URIs. Document fragments are supplied as nodes.

Documents and fragments are written into the specified folder, with a file name extracted from the document URI or the base URI of the supplied node. If the output folder does not exist, it is created.

Output files are indented node serializations ([indentation](https://www.w3.org/TR/xslt-xquery-serialization-31/#xml-indent)), unless option noindent is used. Option unbase is used in order to remove any @xml:base attribute from the root element of the document or fragment.

***Parameters***

Described by the following table.

**Table**. Parameters of function write-doc and write-doc-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| urisOrNodes | NOTE: this parameter is only expected by function variant \*-ec.  The documents or document fragments to be written. An atomic item is interpreted as document URI and replaced with the corresponding document node. Non-root nodes are treated as fragment root nodes. |
| folderPath | The name or path of the output folder. If the folder does not exist, it is created. |
| Options | Options controlling the function behaviour.  docbase – the output folder path is resolved against the base URI of the  document to be written, rather than the current working directory noindent – the serialized nodes to be written are not indented  unbase – remove any @xml:id attribute from the root element |

***Examples***

Write selected documents into a folder. The written files are indented, but indentation may be irregular, if the input files have an irregular indentation.

fox "data\*/airports\*.xml/write-doc('output')"

Write selected documents into a folder, enforce regular indentation. This is achieved by submitting the input documents to function 'pretty-node', which removes any existing pretty-print nodes and thus enforces a complete re-indentation.

fox "data\*/airports\*.xml/pretty-node()/write-doc('output')"

Write selected documents into a folder, with any indentation removed. Function 'pretty-node' removes any pretty-print nodes, and option 'noindent' prevents the construction of new ones.

fox "data\*/airports\*.xml/pretty-node()/write-doc('output', 'noindent')"

Write selected documents, using the ec variant of write-doc.

fox "data/\*/airports\*.xml/pretty-node() => write-doc-ec('output')"

Write selected documents into folders derived from the document's base URI. This means that the documents may be written into different folders. As always, a non-existing output folder is created.

fox "data\*/airport\*.xml/pretty-node()/write-doc('output', 'docbase')"

Modifies documents and writes the results into a folder. The output documents have normalized indentation, as function 'delete-nodes' performs such normalization by default (unless option 'keepws' is used).

fox "data\*/airports\*.xml/delete-nodes('\\dst, \\source1')/write-doc('output')"

Writes a document fragment. The fragment is the 'airport' element with a particular ICAO code. The file name of the fragment is the file name of the containing document, what might be confuging. Use functions write-named-doc, write-renamed-doc, write-exnamed-doc in order to write to a different file name.

fox "data\*/airports\*.xml\\*\\*[@icao eq 'EFEU']/write-doc('output')"

## Node names

…

### aclark-name (acname)

**alark-name**($node as node())

as xs:string

**aclark-name**()

as xs:string

***Summary***

Returns the Clark name of a node, preceded by character @ if the node is an attribute name.

***Details***

Returns the Clark name of a node, preceded by character @ if the node is an attribute name. The Clark name of a node without namespace URI is equal to the local name of the node. The Clark name of a a node with namespace URI is Q{uri}lname, where uri is the namespace URI and lname the local name.

***Parameters***

Described by the following table.

**Table**. Parameters of function aname.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| node | A node. The default value is the context item. |

***Examples***

Get the distinct Clark names of all elements and attributes occurring in a set of documents.

fox \*.xml\\(\*,@\*)\acname() => freq()

### alname

**alname**($node as node())

as xs:string

**alname**()

as xs:string

***Summary***

Returns the local name of a node, preceded by character @ if the node is an attribute name.

***Details***

If the node is not an attribute name, the result is the same as the result of standard function local-name(); otherwise, the result of function local-name() is preceded by the character @.

***Parameters***

Described by the following table.

**Table**. Parameters of function aname.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| node | A node. The default value is the context item. |

***Examples***

Get the distinct local names of all elements and attributes occurring in a set of documents.

fox \*.xml\\(\*,@\*)\alname() => freq()

### aname

**aname**($node as node())

as xs:string

**aname**()

as xs:string

***Summary***

Returns the lexical name of a node, preceded by character @ if the node is an attribute name.

***Details***

If the node is not an attribute name, the result is the same as the result of standard function name(); otherwise, the result of function name() is preceded by the character @.

***Parameters***

Described by the following table.

**Table**. Parameters of function aname.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| node | A node. The default value is the context item. |

***Examples***

Get the distinct names of all elements and attributes occurring in a set of documents.

fox \*.xml\\(\*,@\*)\aname() => freq()

### clark-name (cname)

**clark-name**($node as node())

as xs:string

**clark-name**()

as xs:string

***Summary***

Returns the Clark name of a node.

***Details***

The Clark name of of a node without namespace URI is equal to the local name of the node. The Clark name of a a node with namespace URI is Q{uri}lname, where uri is the namespace URI and lname the local name.

***Parameters***

Described by the following table.

**Table**. Parameters of function aname.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| node | A node. The default value is the context item. |

***Examples***

Get the distinct Clark names of all elements and attributes occurring in a set of documents.

fox \*.xml\\(\*,@\*)\clark-name() => freq()

### check-unused-namespaces (\*-ec)

**check-unused-namespaces**()

as xs:string\*

**check-unused-namespaces**($items as item()\*)

as xs:string\*

***Summary***

Returns namespace bindings declared in the input but not used for element or attribute names.

***Details***

Each binding is returned as a string containing prefix=uri, example:

axf=http://www.antennahouse.com/names/XSL/Extensions

***Parameters***

Described by the following table.

**Table**. Parameters of function aname.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | NOTE: this parameter is only expected by function variant \*-ec.  The document(s) to be analyzed. An atomic item is interpreted as document URI and replaced with the corresponding root element. A node is replaced with the corresponding root element. |

***Examples***

Get the unused namespace bindings found in the deep content of a folder.

fox "../../octopus.xml-framework/steps//\*.xsl => check-unused-namespaces-ec() => freq()"

aid5=http://ns.adobe.com/AdobeInDesign/5.0/ .................................... (1)

aid=http://ns.adobe.com/AdobeInDesign/4.0/ ..................................... (1)

at=http://www.antennahouse.com/names/XSL/AreaTree .............................. (1)

bits=BITS ...................................................................... (1)

cfg=http://www.data2type.de/OctopusFramework/config ............................ (1)

css2cals=http://www.iso.org/ns/css2cals ........................................ (1)

d2t-config=http://www.data2type.de/OctopusFramework/config ..................... (1)

d2t-ditalt=http://www.data2type.de/ditaltframework ............................. (1)

d2t=http://www.data2type.de .................................................... (1)

…

## Statistics

The functions in this section provide statistical evaluation.

### frequencies (freq, f)

**frequencies**($values as item()\*,

$options as xs:string := ()

) as item()

***Summary***

Returns distinct values and their frequencies.

***Details***

Input items can be nodes or atoms. The function returns their distinct string values and their frequencies. Options support …

* Representation of the frequency as count, fraction or percent
* Format (text, csv, xml, json)
* Padding width (in case of text format)
* Filtering by minimum / maximum frequency
* Sort order

***Parameters***

Described by the following table.

**Table**. Parameters of function frequencies.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| values | The values to be analyzed |
| options | Options controlling the processing; whitespace-separated list of option names or assignments (option-name=option-value). See [Function options](#_Function_options) for syntactic details.  xml - output format XML  json - output format JSON  txt - output format text  lines - output format is a sequence of text lines    freq=(count|fraction|percent)  Representation of the frequency, default: count   * count – the number * fraction – the fraction (0 <= fraction <= 1) * Percent – fraction as percent   width=#number  Text output is padded to thie length.  This option is ignored unless the format is text.  min=#number  Display only items with a minimum frequency  max=#number  Ddisplay only items with a maximum frequency  order=(a|d|n|N|f|F)  Sort order, default: a   * a – item string, ascending * d – item string, descending * n – numeric item value, ascending * N –numeric item value, descending * f – frequency, ascending * F – frequency, descending |

***Examples***

Example 1: Get all attribute names found in TEI files.

fox "frameworks/tei//\*.xml\tei:\*\\@\*\name() => freq()"

Example 2: Get the attribute names found in TEI files at least 50 times.

fox "frameworks/tei//\*.xml\tei:\*\\@\*\name() => freq('min=100')"

Example 3: Get the attribute names found in TEI files at most twice.

fox "frameworks/tei//\*.xml\tei:\*\\@\*\name() => freq('max=2')"

Example 4: Get the 20 most frequent attribute names, sorted by descending frequency.

fox "frameworks/tei//\*.xml\tei:\*\\@\*\name() => freq('lines order=F') => subsequence(1, 20)"

***Tips***

* Use option lines in order to get a subset of entries depending on sort order. See example 4.

### fractions (frac)

**fractions**($values as item()\*,

$compareWith as item()+,

$comparison as xs:string,

$valueFormat as xs:string?,

$compareAs as xs:string?)

as item()

**fractions**($values as item()\*, $compareWith as item()+, $comparison as xs:string,

$valueFormat as xs:string?)

as item()

**fractions**($values as item()\*, $compareWith as item()+, $comparison as xs:string)

as item()

**fractions**($values as item()\*, $compareWith as item()+, $comparison as xs:string)

as item()

***Summary***

Reports fractions of values satisfying certain conditions.

***Details***

[UNDER CONSTRUCTION]

***Parameters***

Described by the following table.

**Table**. Parameters of function fractions.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| values | Input values; nodes will be atomized |
| compareWith | Values with which to compare; special semantics if a single item with the pattern $start; $end; $step, e.g. 0;1000;200. In this case the substrings replacing $start and $end are the first and last values with which to compare, and further values are obtained by the iteration $start + k \* step ( where k = 0, …, floor((end – start) / step). For $start and $end, the special value \* represents the minimum (maximum) of the values. Examples:  0;1010;200 => 0, 200, 400, 600, 800, 1000, 1200 |
| comparison | Specifies how to compare the values with the values of $compareWith:   * lt – less than * le – less than or equal * gt – greater than * ge – greater than or equal * eq – equal * ne – not equal * be – between   The comparison be means fractions between two values:   * the first fraction comprises all values less than the first value from $compareWith * the n-th fraction comprises all values >= the (n – 1)th value from $compareWith and < the nth value from $compareWith * the last fraction comprises all values >= the last value from $compareWith |
| valueFormat | Specifies the representation of fractions:  c|count – number of items  f|fraction – fraction of all values (0 <= fraction <= 1)  p|percent – percent of all values (0 <= percent <= 100)  If the value has a suffix colnn (where nn is an integer number), the fractions are also visualized by horizontal columns with a maximum width of nn characters; examples: pcol100, percentcol100, fcol40, ccol50 |
| compareAs | Specifies the type to be assumed when comparing values:  decimal – xs:decimal  string – xs:string  date – xs:date  Default value: decimal |

***Examples***

Get the number of values less than 10:

fox "(1,4,6,7,7,7,8,9,10,17,18,18,18,19,25,30) => frac(**10**, '**lt**')"

8

Get the fraction of values greater than or equal 18, in percent

fox "(1,4,6,7,7,7,8,9,10,17,18,18,18,19,25,30) => frac(**18**, '**ge**', '**p**')"

37.5

Get the fraction of values greater than or equal 18, as a fraction of all values

fox "(1,4,6,7,7,7,8,9,10,17,18,18,18,19,25,30) => frac(18, 'ge', '**f**')"

0.38

Get the fractions of values less than 10, 20, 30, 40, in percent

fox "(1,4,6,7,7,7,8,9,10,17,18,18,18,19,25,30) => frac((**10, 20, 30, 40**), 'lt', '**p**')"

lt 10 50.0

lt 20 87.5

lt 30 93.8

lt 40 100.0

Get the fractions of values less than 10, 20, 30, 40, in percent and as columns

fox "(1,4,6,7,7,7,8,9,10,17,18,18,18,19,25,30) => frac((10, 20, 30, 40), 'lt', **'pcol40'**)"

#----------------------------------------#

lt 10 50.0 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

lt 20 87.5 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

lt 30 93.8 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

lt 40 100.0 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|

#----------------------------------------#

Specify the values to compare with using “minimum;maximum;step width”:

fox "(1,4,6,7,7,7,8,9,10,17,18,18,18,19,25,30) => frac('**10;40;10**', 'lt', 'pcol40')"

#----------------------------------------#

lt 10 50.0 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

lt 20 87.5 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

lt 30 93.8 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

lt 40 100.0 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|

#----------------------------------------#

Specify the values to compare with using minimum and maximum as derived from the actual values:

fox "(1,4,6,7,7,7,8,9,10,17,18,18,18,19,25,30) => frac(**'\*;\*;10'**, 'lt', 'pcol40')"

#----------------------------------------#

lt 0 0.0 | |

lt 10 50.0 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

lt 20 87.5 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

lt 30 93.8 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

lt 40 100.0 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|

#----------------------------------------#

Get the fractions within intervals of width 5:

fox "(1,4,6,7,7,7,8,9,10,17,18,18,18,19,25,30) => frac('**\*;\*;5**', **'be'**, 'pcol40')"

#----------------------------------------#

[ 0 | |

[) 5 12.5 |\*\*\*\*\*\*\*\*\*\*\*\*\* |

[) 10 37.5 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|

[) 15 6.2 |\*\*\*\*\*\*\* |

[) 20 31.2 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

[) 25 0.0 | |

[) 30 6.2 |\*\*\*\*\*\*\* |

[) 35 6.2 |\*\*\*\*\*\*\* |

>= 35 0.0 | |

#----------------------------------------#

Get the numbers of files with a last modification date in intervals of width 90 days:

fox "../bin//\*.xqm/file-date() => frac(**'\*;\*;90'**, 'be', 'countcol30', **'date'**)"

#------------------------------#

[ 2018-02-22Z | |

[) 2018-05-23Z 1 |\* |

[) 2018-08-21Z 1 |\* |

[) 2018-11-19Z 0 | |

[) 2019-02-17Z 16 |\*\*\*\*\*\*\*\*\*\* |

[) 2019-05-18Z 0 | |

[) 2019-08-16Z 0 | |

[) 2019-11-14Z 0 | |

[) 2020-02-12Z 0 | |

[) 2020-05-12Z 1 |\* |

[) 2020-08-10Z 0 | |

[) 2020-11-08Z 0 | |

[) 2021-02-06Z 0 | |

[) 2021-05-07Z 0 | |

[) 2021-08-05Z 0 | |

[) 2021-11-03Z 0 | |

[) 2022-02-01Z 0 | |

>= 2022-02-01Z 48 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|

#------------------------------#

Get for a set of error files a distribution of the number of errors, using intervals of width 2:

fox "../output-convert-mass//\*error\*.xml/count(\\error) => frac('**0;\*;2**', 'be', 'pcol40')"

#----------------------------------------#

[ 0 | |

[) 2 57.8 |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|

[) 4 17.8 |\*\*\*\*\*\*\*\*\*\*\*\* |

[) 6 4.4 |\*\*\* |

[) 8 2.2 |\*\* |

[) 10 2.2 |\*\* |

[) 12 15.6 |\*\*\*\*\*\*\*\*\*\*\* |

>= 12 0.0 | |

#----------------------------------------#

### percent

**percent**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### median

**median**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## Miscellaneous functions

This section describes functions which do not fit into the function groups described by the previous sections.

Note: the grouping of functions is work in progress. It is planned to create new groups which will take up a large part of the functions currently found in the “miscellaneous” group.

### annotate

**annotate**($annotation as item()?,

$prefix as xs:string?,

$postfix as xs:string?)

as xs:string

**annotate**($annotation as item()?,

$prefix as xs:string?)

as xs:string

**annotate**($annotation as item()?)

as xs:string

**annotate-ec**(

$value as item()?,

$annotation as item()?,

$prefix as xs:string?,

$postfix as xs:string?)

as xs:string

…

***Summary***

Returns the string value of an item, with an annotation appended.

***Details***

The value to be annotated is either the first argument (function variant annnotate-ec) or the context item (function variant annotate). The value is atomized and the annotation is appended, preceded by the prefix and followed by the suffix. By default, prefix is “ (“ and suffix is “)”. These default values can be overridden by parameters $prefix and $suffix.

***Parameters***

Described by the following table.

**Table**. Parameters of function annotate.and annotate-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| value | NOTE: this parameter is only expected by function variant \*-ec.  The value to be annotated. The value must be a single item which can be a node or an atomic item. If the value has several items, only the first one is considered. |
| annotation | The annotation to be appended. |
| prefix | String inserted between value and annotation |
| postfix | String appended to the postfix |

**Examples**

Get file URIs annotated with the number of contained "airport" elements.

fox "airports\*.xml/annotate(\\airport => count())"

Get file names annotated with the number of contained "airport" elements.

fox "airports\*.xml/annotate-ec(file-name(.), \\airport => count())"

As the preceding example, but pad file names and use " airports" as postfix.

fox "airports\*.xml/annotate-ec(file-name(.)/rpad(., 20), \\airport => count(), (), ' airports)')"

### atts

**atts**($nodes as item()\*)

as node()\*

**atts**()

as node()\*

***Summary***

Writes a set of standard attributes.

***Details***

Give a detailed description.

***Parameters***

Described by the following table.

**Table**. Parameters of function xyz.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| Flags | Flags indicating which attributes to write. |

**Examples**

Gets this or that.

fox "\*[is-dir()]/row(., .//\*.xsd => count()) => table('Folder, Count XSDs')"

### back-slash (bslash)

**bslash**($string as xs:string)

as xs:string

*Abbreviations - t*he function name can be appreviated:

**bslash**()

***Summary***

Edits a string, replacing slash characters with backslash.

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

**base-dir-name**($node as item())

as xs:string

**base-dir-name**()

as xs:string

*Abbreviations - t*he function name can be appreviated:

**base-dname**()

**bdname**()

***Summary***

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

***Details***

If the argument is omitted, it defaults to the context item (.). The behavior of the function if the argument is omitted is exactly the same as if the context item had been passed as the argument.

If the argument is a node, the base URI is determined and the folder name is extracted from it.

If the argument is not a node, it is interpreted as a document URI and an attempt is made to parse the document. If parsing fails, the empty sequence is returned. Otherwise, the base URI is determined and the folder name is extracted from it.

Extraction of the folder name from the base URI is equivalent to applying the function call replace($baseURI, '.\*/(.+?)/.\*', '$1').

***Parameters***

Described by the following table.

**Table**. Parameters of function base-dir-name.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| node | A node or a string interpreted as document URI. |

**Example**

Returns the folder containing the document containing a given element. Strictly speaking, the folder is extracted from the base URI. If the element or an ancestor element has an @xml:base attribute, the folder is extracted from the URI specified by the attribute.

fox "study.xml\descendant::price\bdname()"

Returns the folder containing the document located by the path expression. Note that an @xml:base attribute on the root element has no effect, as it does not effect the base URI of the document node.

fox ".//study.xml/bdname()"

**Tip**

Use as component of row() when composing an hlist (hierarchical list). *(To be elaborated.)*

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

**base-file-name**($node as item())

as xs:string

**base-file-name**()

as xs:string

*Abbreviations - t*he function name can be appreviated:

**base-fname**()

**bfname**()

***Summary***

Returns the file name of the base URI.

***Details***

#Give a detailed description.

***Parameters***

Described by the following table.

**Table**. Parameters of function base-file-name.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| node | A node or a string interpreted as document URI. |

**Example**

(under construction)

### base-uri-relative (buri-relative, burirel)

**base-uri-relative**($context as xs:string)

as xs:string

*Abbreviations - t*he function name can be appreviated:

**buri-relative**()

**burirel**()

***Summary***

Returns the base URI as relative URI, in the context of an ancestor name specified by match pattern.

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### content-deep-equal

***Usage***

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

as xs:boolean?

**content-deep-equal**($items as item()+)

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 as xs:string, $char as xs:string)

as xs:integer

**count-chars**($char as xs:string)

as xs:integer

***Summary***

Counts occurrences of a character in a string.

***Details***

Returns the number of occurrences of a given character in a given 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(',')]"

### create-dir

**create-dir**($nodes as item()\*)

as empty-sequence()

**create-dir**()

as empty-sequence()

***Summary***

Creates a directory. Also creates all required ancestor directories.

### dcat

**dcat**($uris as xs:string\*, $basePath as xs:string?)

as element(dcat)

**dcat**($uris as xs:string\*)

as element(dcat)

***Summary***

Creates a catalog of document URIs. If $basePath is specified, the catalog contains relative paths, relative to the resolved path given by $basePath.

### depth (\*-ec)

**depth**()

as xs:integer?

**depth-ec**($item as item()?)

as xs:integer?

***Summary***

Returns the hierarchical depth of a node within the containing document.

***Details***

Returns the hierarchical depth of a node within the containing document. The root element has depth 1, its child ellements have depth 2, etc. Document nodes have depth 0, non-elemenet nodes have the depth of their parent element. If the input item is not a node, the empty sequence is returned.

Formatlly, the depth of a node is the number of ancestor-or-self element nodes.

Function variant depth-ec receives the input item as the value of the first parameter. Function variant depth processes the context item (for more information see [ec – variant](#_ec_–_variant)).

***Parameters***

Described by the following table.

**Table**. Parameters of function depth and depth-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| item | NOTE: this parameter is only expected by function variant \*-ec.  The input item. |

***Examples***

Compare two sibling files.

### dir-name (dname)

**dir-name**($uri as xs:string)

as xs:string

**dir-name**()

as string

*Abbreviations - t*he function name can be appreviated:

**dname**()

***Summary***

Extracts from a URI the name of the containing folder.

### distinct

**distinct**($items as item()\*)

as xs:boolean

***Summary***

Returns true or false, indicating that the input items are all distinct, or not.

### docx-ccount

**docx-ccount**($uri as xs:string)

as document-node()?

***Summary***

Returns the number of characters of a .docx document.

**Details**

Returns the number of characters of a .docx document.

***Parameters***

Described by the following table.

**Table**. Parameters of function docx-ccount.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| Uri | URI of the .docx file. Defaults to the context item. |

***Examples***

…

Returns the number of characters contained by the .docx document:

fox "\*.docx/docx-ccount()"

### docx-mcount

**docx-mcount**($uri as xs:string)

as document-node()?

***Summary***

Returns the number of media objects of a .docx document.

**Details**

Returns the number of media objects of a .docx document.

***Parameters***

Described by the following table.

**Table**. Parameters of function docx-mcount.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| Uri | URI of the .docx file. Defaults to the context item. |

***Examples***

…

Returns the number of media objects contained by the .docx document:

fox "\*.docx/docx-mcount()"

### docx-msize

**docx-msize**($uri as xs:string)

as document-node()?

***Summary***

Returns the total compressed size of media objects of a .docx document.

**Details**

Returns the total compressed size of media objects of a .docx document.

***Parameters***

Described by the following table.

**Table**. Parameters of function docx-mcount.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| Uri | URI of the .docx file. Defaults to the context item. |

***Examples***

…

Returns the total compressed size of media objects contained by the .docx document:

fox "\*.docx/docx-mcount()"

### echo

**echo**($value as item()\*)

as item()\*

***Summary***

Returns the input value.

### filter-regex (fregex)

**filter-regex**($items as item()\*,

$regex as xs:string+,

$flags as xs:string?)

as item()\*

**filter-regex**($items as item()\*,

$regex as xs:string+)

as item()\*

***Summary***

Filters items, retaining those matching a regular expression.

***Details***

Retained items match at least one of the regular expressions provided by $regex. When matching, the provided flags are applied.

*Note*. This function is a convenience function, meant to support concise specification of filtering. The expression

$items => filter-regex($regex, $flags)

is equivalent to:

$items[some $r in $regex satisfies matches(., $r, string($flags))]

***Parameters***

Described by the following table.

**Table**. Parameters of function filter-regex.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | Items to be filtered. May be atomic or nodes. |
| regex | One or several regular expressions used as filter criterion – only items matching at least one of them are retained. |
| flags | Flags applied when matching, with semantics as defined by standard function fn:matches(). |

***Examples***

Filter documents, retaining only those with a least one @msg attribute matching the given regular expression.

fox "../output-convert98/\*anchor\*[\\@msg => fregex('zusätzlich')]"

As before, but matching is case insensitive.

fox "../output-convert98/\*anchor\*[\\@msg => fregex('zusätzlich', 'i')]"

As before, but specifying three regular expressions, at least one of which must be matched.

fox "../output-convert98/\*anchor\*[\\@msg => fregex(('zusätzlich', 'kapitel', 'über'))]"

Combining multiple regular expressions with case insensitive matching.

fox "../output-convert98/\*anchor\*[\\@msg => fregex(('zusätzlich', 'kapitel', 'über'), 'i')]"

### group-items

**grep**($items as item()\*,,

$groupKeyExpr as xs:string?,

$groupProcExpr as xs:string?,

$groupWhereExpr as xs:string?,

$groupElemNameSpec as xs:string?,

$keyName as xs:string?,

$wrapperName as xs:string?,

$orderBy as xs:string?,

$options as xs:string?)

***Summary***

Maps a sequence of items to a sequence of elements representing groups of items.

***Details***

Input items are grouped by a key obtained from a Foxpath expression ($groupKeyExpr). The expression is resolved in a context using the item as context item.

The group is represented by an element with content obtained by resolving a Foxpath expression ($groupProcExpr). The expression is resolved in a context binding the group members to a variable $items.

The element name is specified by parameter $groupElemNameSpec, which is either a literal name or a Foxpath expression. The expression is resolved in a context binding the group members to a variable $items. The default name is group.

An optional where condition is defined by a Foxpath expression ($groupWhereExpr). The expression is resolved in a context binding the group members to a variable $items.

The elements representing a group have an attribute containing the grouping key. The attribute name is the value of parameter $keyName. The default name is key.

Parameter $orderBy controls the order of groups. Values s and n mandate an ordering by string value and numeric value, respectively.

***Parameters***

Described by the following table.

**Table**. Parameters of function group-items.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | The items to be grouped. |
| groupKeyExpr | Foxpath expression returning the item grouping key. The expression is evaluated in the context of the item. |
| groupProcExpr | Foxpath expression returning the group content. The expression is evaluated in a context binding the group members to variable $items. If not specified, group content defaults to the group members. |
| groupWhereExpr | Foxpath expression returning the where condition filtering the groups. The expression is evaluated in a context binding the group mebers to variable $items. If not specified, the groups are not filtered. |
| groupElemNameSpec | Specifies the name of the elements representing groups. Either a literal string or a Foxpath expression. The expression is evaluated in a context binding the group members to variable $items. The default name is group. |
| keyName | The name of the attribute containing the grouping key. The default name is key. |
| wrapperName | The name of the wrapper element containing the group elements. The default name is groups. |
| order by | If equal “s”, the groups are ordered by the string values of the grouping keys. If Equal “n”, the groups are ordered by the numeric values of the grouping keys. By default, the groups are not ordered. |

***Examples***

Example 1: Extract all element names and for each name the names of the containing elements.

fox "/gi-testframe-works/data/output//\*gibook\*\\@\* => group-items(

{name()}, {`$items\..\name() => distinct-values() => sort() => xatt('parentElems')}, (),

'name', 'att', 'atts', 's')"

=>

<atts count="19">

<att name="border-bottom" parentElems="l:td"/>

<att name="border-left" parentElems="l:td"/>

<att name="border-right" parentElems="l:td"/>

<att name="border-top" parentElems="l:td"/>

<att name="colspan" parentElems="l:td"/>

<att name="depth" parentElems="l:image"/>

<att name="fileref" parentElems="l:image"/>

<att name="format" parentElems="l:image"/>

<att name="name" parentElems="gi:darreichungsform gi:packung gi:praeparat gi:wirkstoff"/>

<att name="num" parentElems="l:col"/>

<att name="produkttyp" parentElems="gi:meta-daten"/>

<att name="pzn" parentElems="gi:packung"/>

<att name="sortid" parentElems="gi:anwendung-dosierung gi:aufbewahrung gi:gi-stand … />

<att name="span" parentElems="l:col"/>

<att name="status" parentElems="gi:meta-daten"/>

<att name="tocid" parentElems="gi:anwendung-dosierung gi:aufbewahrung gi:entry … />

<att name="type" parentElems="l:list"/>

<att name="version" parentElems="l:image"/>

<att name="width" parentElems="l:col l:image"/>

</atts>

### grep

**grep**($uris as xs:string\*,

$stringFilter as xs:string?,

$flags as xs:string\*)

as item()\*

**grep**($uris as xs:string\*,

$stringFilter as xs:string?)

as item()\*

***Summary***

Returns for each input file a representation of text lines matching given filters.

***Details***

If the function is called with a single argument, a single input file is considered, with a URI given by the context item. Otherwise, the files with URIs given by the first argument are considered.

The function selects all text lines matching a pattern from $patterns and not matching a pattern from $patternsExcluded. By default, patterns are interpreted as Glob patterns, which a substring of the text line must match, ignoring case. Using flag r, the patterns are interpreted as regular expressions, rather than Glob patterns, and flag c signals that matching is case-sensitive. When using flag a, the pattern must be matched by the complete text line, rather than an arbitrary substring.

By default, the function returns all matching text lines, and the matches from a single input file are preceded by an additional line containing the file path framed by the substring " ##### ". When using flag n, for each input file only the number of matching line is returned.

***Parameters***

Described by the following table.

**Table**. Parameters of function row.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uris | The URIs of the text files to be analyzed |
| patterns | Only text lines matching one of these patterns are considered. By default a pattern is interpreted as Glob pattern, which a substring of the text line must match. This interpretation can be modified by flags. |
| patternsExcluded | Only text lines matching none of these patterns are considered. By default a pattern is interpreted as Glob pattern, which a substring of the text line must match. This interpretation can be modified by flags. |
| flags | String of characters interpreted as follows:  c – matching is case-sensitive  a – anchors are added, representing the begin and the end of the string  r – the pattern is interpreted as a regular expression, not as a Glob pattern  n – for each input URI, return the number of matching lines, not the lines |

***Examples***

Perform file system navigation and show for each result file all text lines containing the string millicent, case insensitively.

fox "\*/grep('millicent')"

Same as preceding example, making input explicit.

grep(\*, 'millicent')"

Same as preceding example, more elegantly.

fox "\* => grep('millicent')"

Same as first example, but excluding lines containing the string <author. Note the ~ indicating an exclusive filter item.

fox "\*/grep('millicent ~<author')"

Patterns must be matched by the complete text line, rather than only a substring.

fox "\*/grep(., '\*millicent\*<author\*#a')"

Select lines by regular expression, rather than by Glob pattern.

fox "\*/grep(., '\s\*<author#r')"

Get a list of file paths, each one annotated with the number of matching text lines.

fox "\*/annotate(grep(., '<author', 'n'))"

Filter using a fulltext query – the words “enthält” and “Adalimumab” in this order, seperated by at most two words. Note the pseudo-option ftext indicating that the filter is a fulltext query:

fox "\*fibook.xml => grep('enthält Adalimumab#ftext phrase2')"

### indent

**indent**($width as xs:integer?,

$char as xs:string?)

as xs:string\*

**indent**($width as xs:integer?)

as xs:string\*

**indent**()

as xs:string\*

**indent-ec**(

$text as xs:string\*,

$width as xs:integer?,

$char as xs:string?)

as xs:string\*

…

***Summary***

Returns input strings, indented as specified.

***Details***

The input strings are “indented” by inserting strings (a) after every linefeed character, (b) before the first character. By default, the inserted strings consist of four blanks. Use the optional parameter $indentString in order to use a self-defined indentation string.

***Parameters***

Described by the following table.

**Table**. Parameters of function path-diff and path-diff-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| text | NOTE: this parameter is only expected by function variant \*-ec.  Text items which to indent. |
| indentString | The indentation string. Default: four blanks. |
| options | Options controling details of function behaviour.  skip1 – do not indent the first line of each text item |

***Examples***

Create a list of file names, where each name is followed by indented details.

fox "data/\*.xml/(fname(), \\city\indent())"

As the preceding example, but use a self-defined indentation string, rather than four blanks.

fox "data/\*.xml/(fname(), \\city\indent(' -> '))"

List file contents, indenting all lines except for the first.

fox "data/cities.txt/file-content()/indent(' ', 'skip1')"

### in-scope-namespaces

**in-scope-namespaces**(…)

as …

***Summary***

Returns the namespace bindings of input elements as “prefix=namespace-uri” items.

***Details***

***…***

### in-scope-namespaces-descriptor

**in-scope-namespaces-descriptor**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

...

***Examples***

...

### is-dir

**is-dir**($uri as xs:stinrg?)

as xs:bolean

***Summary***

Returns true if a given URI points to a folder resource, false otherwise.

***Details***

Returns true if a given URI points to a folder resource, false otherwise. Note in particular that a return value false does not imply that the resource exists. Use file-exists() in order to check the existence of a resource.

***Parameters***

Described by the following table.

**Table**. Parameters of function is-dir.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | URI or file path to be inspected. |

***Examples***

Count folder descendants of current workdir:

fox ".//\*[is-dir()] => count()"

List empty folder descendants of current workdir:

fox ".//\*[is-dir()][not(\*)]"

### is-file

**is-file**($uri as xs:stinrg?)

as xs:boolean

***Summary***

Returns true if a given URI points to a file resource, false otherwise.

***Details***

Returns true if a given URI points to a file resource, false otherwise. Note in particular that a return value false does not imply that the resource exists. Use file-exists() in order to check the existence of a resource.

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | URI or file path to be inspected. |

***Examples***

Count file descendants of current workdir:

fox ".//\*[is-file()] => count()"

### is-xml

**is-xml**(xs:string)

as xs:boolean

***Summary***

...

***Details***

***…***

### jschema-keywords, jskeywords

**jschema-keywords**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### jsoncat

**jsoncat**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### json-text

**json-text**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### json-doc-available, jdoc-available, is-json

**json-doc-available**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### json-effective-value

**json-effective-value**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### json-name (jname)

**json-name**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### json-parse (jparse)

**json-parse**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### linefeed

**linefeed**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### lines

**lines**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### lpad

**lpad**(…)

as xs:string

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### map-items

**map-items**($items as item()+,

$expr as xs:string)

as item()\*

***Summary***

Applies a Foxpath expression to every item of a value, returning the concatenated result sequences.

***Details***

The expression is evaluated once for each item in $items, using that item as context item. The result sequences are concatenated in order.

***Parameters***

Described by the following table.

**Table**. Parameters of function map-items.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | An XDM value, that is, a sequence of items. |
| expr | A Foxpath expression. |

***Examples***

Get a sorted list of distinct QNames used in a set of documents, inserting a slash between local name and namespace URI. The four-fold quoting is necessary when entering the expression on the command-line.

fox "doc\*.xml\\\*\node-name() => distinct-values() => map-items('local-name-from-QName(.)||"""" / """"||namespace-uri-from-QName(.)') => sort()"

Example output:

a / http://example.org

b / http://example2.org

b / http://example3.org

c / http://example.org

d / http://example3.org

doc / <http://example.org>

### nname

**nname**($nodes as node()\*)

as xs:string\*

**nname**()

as xs:string\*

***Summary***

Returns the names of given nodes. The names of attribute nodes is preceded by a “@” character.

***Details***

***…***

***Parameters***

***…***

***Examples***

List the names of all attributes and elements contained by a document.

fox "../output-convert-mass/\*01/\*fibook.xml/all-descendant()/nname() =>f()"

@context (1)

@countFi (1)

@countFo (1)

@name ... (9)

@skipdir (1)

fo ...... (9)

ftree ... (1)

### non-distinct-file-names (non-distinct-fnames)

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

as …

***Summary***

...

***Details***

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.

***Parameters***

***…***

***Examples***

***…***

### oas-jschema-keywords

**oas-jschema-keywords**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### oas-keywords

**oas-keywords**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### oas-msg-schemas (oasmsgs)

**oas-msg-schemas**(…)

as …

***Summary***

...

***Details***

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()"

***…***

***Parameters***

***…***

$nodes – nodes from OpenAPI documents

***Examples***

***…***

### order-diff

**order-diff**($value1 as item()\*,

$value21 as item()\*,

$reportType as xs:string?)

as xs:item\*

***Summary***

Compares the item order of two values and reports differences.

***Details***

The item order of two values differs if an item in the atomized value of $value1 is followed by an item which in the atomized value of $value2 precedes the other item. Note that a difference can only occur if both values have at least two items. The return value depends on $reportType:

* $reportType equal boolean – the Boolean value true if there is no difference, false otherwise
* $reportType equal backsteps – for each backstep item in $value1 the backstep item, preceded by the two items preceding it in $value1, separated by " # ". If the backstep item is the second item of $value1, only two, rather than three items are returned.
* $reportType equal backstep – like backsteps, but only the first backstep item is considered

The term “backstep item” denotes an item from $value1 which is preceded by an item which in $value2 follows it, directly or indirectly.

***Parameters***

Described by the following table.

**Table**. Parameters of function same-order.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| value1 | The first value to be compared |
| value2 | The second value to be compared |
| reportType | Identifies the way how differences of item order are reported |

**Examples**

Returns true.

fox "order-diff((2, 4, 5, 6), 1 to 6, 'boolean')"

Returns true – repetition cannot create a difference of item order.

fox "order-diff((2, 4, 5, 5), 1 to 6, 'boolean')"

Returns true – omission cannot create a difference of item order.

fox "order-diff((2, 5), 1 to 6, 'boolean')"

Returns true – if one of the values has a single item, there cannot be a difference.

fox "order-diff(2, 1 to 6, 'boolean')"

Returns true – if one of the values is empty, there cannot be a difference.

fox "order-diff((), 1 to 6, 'boolean')"

Two backsteps are reported: item “1” is preceded by item “2”, which in $value2 follows it; and item “4” is preceded by item “5”, which in $value2 follows it. As the first backstep item is preceded by only one item, it is reported by a pair of items, rather than three items.

fox "order-diff((2, 1, 5, 4), 1 to 6, 'backsteps')"

=>

2 # 1

1 # 5 # 4

One backstep are reported: item “Details” is preceded by item “AdditionalDetails”, which in $value2 follows it.

fox "order-diff(('Introduction', 'Summary', 'AdditionalDetails', 'Details'),

('Introduction', 'Summary', 'Details', 'AdditionalDetails'),

'backsteps'

=>

Summary # AdditionalDetails # Details

Only the first backstep item is reported, as the report type is backstep1.

fox "order-diff((('Summary', 'Conclusion', 'Introduction', 'AdditionalDetails', 'Details'),

('Introduction', 'Summary', 'Details', 'AdditionalDetails', 'Conclusion'),

'backsteps')

=>

Summary # Conclusion # Introduction

### pads

**pads**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### pretty-node (\*-ec)

**pretty-node**($options as xs:string?)

as node()\*

**pretty-node-ec**(

$uriOrNode as item()+,

$options as xs:string?)

as node()\*

***Summary***

Removes whitespace-only text nodes with element siblings.

***Details***

Input nodes are supplied as nodes or as document URIs which are replaced with the corresponding document node. The function removes all descendant text nodes with an element sibling and containing only whitespace. If option weak is used, removal is not performed if the text node has a sibling text node containing non-whitespace.

***Parameters***

Described by the following table.

**Table**. Parameters of function rpretty-node and pretty-node-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uriOrNode | NOTE: this parameter is only expected by function variant \*-ec.  The nodes to be edited. An atomic item is interpreted as document URI and replaced with the corresponding document node. |
| options | Options controling the function behaviour.  weak – whitespace-only text nodes are not removed if a sibling text node contains non-whitespace |

***Examples***

Bla

### ps.copy

**ps.copy**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### rcat

**rcat**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### relevant-xsds (rxsds)

**relevant-xsds**($docs as item()\*,

$xsds as item()\*)

as element(docs)

***Summary***

Reports which XSDs can be used for validating given documents.

***Details***

#Give a detailed description.

***Parameters***

Described by the following table.

**Table**. Parameters of function relevant-xsds.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| docs | A set of documents, supplied as URIs or nodes. |
| Xsds | A set of XSDs, supplied as URIs or nodes. |

**Examples**

Report for all .xml within a folder the relevant XSDs found within that folder.

fox "keycloak//\*.xml => **rxsds**(keycloak//\*.xsd)"

### rel-path

**rel-path**($uri as xs:string,

$referenceUri as xs:string)

as xs:string?

**rel-path**($referenceUri as xs:string)

as xs:string?

***Summary***

Calculates for a given absolute URI the relative URI leading from a reference URI ot the given URI.

***Details***

#Give a detailed description.

***Parameters***

Described by the following table.

**Table**. Parameters of function rel-path.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | An absolute URI |
| referenceURI | Another absolute URI |

**Examples**

Get the relative path leading from the second argument URI to the first argument URI.

fox "rel-path('/a/b', '/a/b/c/d')"

c/d

### remove-prefix

**remove-prefix**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### repeat

**repeat**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### resolve-json-allof (jallof)

**resolve-json-allof**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### resolve-json-anyof, janyof

**resolve-json-anyof**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### resolve-json-one of, joneof

**resolve-json-oneof**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### resolve-json-ref ; jsonref ; jref

**resolve-json-ref**($ref, $mode)

as …

**resolve-json-ref**($ref)

as …

**resolve-json-ref**()

as…

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### resolve-link (\*-ec)

**resolve-link**()

**resolve-link**($options as xs:string?)

as item()?

**resolve-link-ec**($node as node())

as item()?

**resolve-link-ec**($node as node(),

$options as xs:string?)

as item()?

***Summary***

Resolves links and returns the target resource URI. Unless option $ign-nofind is used, the empty sequence is returned if the target URI cannot be resolved to a file.

***Details***

The string value of the input node is resolved against the base URI of the containing document. Options:

* xml – the target resource is returned as an XML document
* ignore-nofind – the target resource URI is returned, regardless if it points to an existing file

The function can be used for checking document contents for valid links. The following expression

/path/to/docx.xml[\\@href[not(resolve-link())]]

Returns all documents containing @href attribute with links which cannot be resolved to a file.

***Parameters***

***…***

***Examples***

***…***

### resolve-path

**resolve-path**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### resolve-xsdtype-ref (typeref)

**resolve-xsdtype-ref**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### rpad

**rpad**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### serialize

**serialize**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### shift-uri

**shift-uri**($uri as xs:string,

$referenceUri as xs:string,

$targetReferenceUri)

as xs:string?

**shift-uri**($referenceUri as xs:string,

$targetReferenceUri)

as xs:string?

***Summary***

Maps a URI to another URI obtained by applying the relative path connecting a reference URI and the given URI to a different reference URI.

***Details***

#Give a detailed description.

***Parameters***

Described by the following table.

**Table**. Parameters of function shift-uri.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uri | An absolute URI |
| referenceURI | Another absolute URI, used for obtaining a relative path to the given URI |
| targetReferenceURI | Another absolute URI, to which the relative path between $referenceURI and $uri will be applied |

**Examples**

Get the relative path leading from the second argument URI to the first argument URI.

fox "keycloak//schema/application\_9.xsd =>

shift-uri(keycloak//schema, wildfly/docs/schema)"

C:/products/x4/x4/wildfly/docs/schema/application\_9.xsd

### subset-fraction

**subset-fraction**($values as item()\*,

$filterExpr as xs:string,

$valueFormat as xs:string?)

as item()?

**subset-fraction**($values as item()\*,

$filterExpr as xs:string)

as item()?

***Summary***

Returns the size of a subset of items meeting a filter conditions.

***Details***

[UNDER CONSTRUCTION]

***Parameters***

Described by the following table.

**Table**. Parameters of function subset-fraction.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| values | The values to be analyzed – may be nodes or atoms |
| filterExpr | Foxpath expression to be evaluated in the context of each item in $values |
| valueFormat | Specifies the representation of fractions:  c|count – number of items  f|fraction – fraction of all values (0 <= fraction <= 1)  p|percent – percent of all values (0 <= percent <= 100) |

***Examples***

Get the number of values less than 10:

### unescape-json-name

**unescape-json-name**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### value

**value**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### win.copy

**win.copy**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### win.delete

**win.delete**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### write-file

**write-file**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### write-files

**write-files**(

$items as item()\*,

$dir as xs:string,

$fileNameExpr as xs:string?,

$encoding as xs:string?,

$options as xs:string?)

as document-node()

***Summary***

Frites files into a folder.

***Details***

***…***

***Details***

[Under construction]

***Parameters***

Described by the following table.

**Table**. Parameters of function augment-doc and augment-doc-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | Each item will be written into a separate file. Items can be nodes or atomic values. |
| dir | The folder into which to write the files. If the folder does not yet exist, it is created. |
| fileNameExpr | A Foxpath expression returning a file name. The expression is evaluated in the context of the item to be written into the file. |
| encoding | Optional specification of the encoding. It is only evaluated if the item is an atomic item. |
| options | Options controling details of the function behaviour.  noindent – serialization of XML content without indentation |

***Examples***

Add an attribute to the root element, providing information about the content.

***…***

### write-json-docs

**write-json-docs**(…)

as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

### xatt

**xatt**( $items as item(),

$elemName as xs:string,

$options as xs:string?)

as element()\*

***Summary***

Creates an XML attribut.

***Details***

***…***

### Xelem (\*-ec)

**xelem**($elemName as xs:string,

$options as xs:string?)

as element()\*

**xelem**($elemName as xs:string)

as element()\*

**xelem-ec**(

$items as item(),

$elemName as xs:string,

$options as xs:string?)

as element()\*

***Summary***

Creates an XML element.

***Details***

***…***

***Parameters***

Described by the following table.

**Table**. Parameters of function xelem and xelem-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | NOTE: this parameter is only expected by function variant \*-ec.  The input items. |
| elemName | Element name to be used. |
| options | Options controling function behaviour. Not yet evaluted. |

***Examples***

***…***

### xelems

**xelems**($items as item(),

$elemName as xs:string,

$options as xs:string?)

as element()\*

***Summary***

Creates one or several XML elements.

***Details***

Each input item is wrapped in an XML element.

***Parameters***

Described by the following table.

**Table**. Parameters of function xelems.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | The input items. |
| elemName | Element name to be used. |
| options | Options controling function behaviour. Not yet evaluted. |

***Examples***

***…***

### xitems

**xitems**($items as item()\*,

$elemName as xs:string?,

$options as xs:string?)

as element()\*

***Summary***

Maps each item in a given sequence of items to an element representing the element.

***Details***

Each item in $items is mapped to an XML element. The element content is the item, which may be a node or an atom. The element name can be specified explicitly ($name), or it is implied: if the item is an XML element or attribute, the item name is used, otherwise the name item is used.

The mapping of items to elements can be influenced by options ($options). Currently, only one option is supported: string. When this option is specified, node items are mapped to elements containing their string content, rather than the node item – for example the attribute value, rather than an attribute.

***Parameters***

Described by the following table.

**Table**. Parameters of function xitems.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| items | Items to be mapped. |
| name | Name of the mapping element. |
| options | Whitespace-separated list of tokens representing options. Currently, only one option is supported:  string – the output elements contain the string values of the input items, rather than the items |

***Examples***

***…***

### xwrap

**xwrap**($values as item()\*,

$rootName as xs:string,

$flags as xs:string?,

$itemName as xs:string?)

as element()

as element()

***Summary***

Transforms a sequence of values into an XML document.

***Details***

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

B – add an attribute @xml:base, giving the base URI of the item as a relative URI in the context of the current working directory

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

*Further options*

P – the result document is pretty-printed

Examples

Wrap strings in elements with a specified name (p):

fox "boo.xml\\file\@name\string() => xwrap('fileNames', 'c', 'file-name')"