# Fox functions

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

*[Work in progress]*

*[template]*

## Concepts

Bla

### Unified Filter Expression

…

#### Complex filter

…

#### Pattern-or-regex

…

### Parameterized fox axes

* fshifted
* ftransferred
* fmirrored
* ftransferred-and-shifted
* fmirrored-and-shifted

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

Bla

### Function variants

Bla

#### ec – variant

bla

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

bla

## Functions supporting node tree navigation

The functions in this section support a more concise expression of complex node tree navigation.

### all-descendant (\*-ec)

**all-descendant**($namesFilter as xs:string?,

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

**all-descendant**($namesFilter as xs:string?,

$pselecter as xs:integer?)

as node()\*

**all-descendant**($namesFilter as xs:string?) as node()\*

**all-descendant**() as node()\*

**all-descendant-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\*')]"

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

**all-descendant-or-self**(

$namesFilter as xs:string?,

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

**all-descendant-or-self**(

$namesFilter as xs:string?,

$pselecter as xs:integer?)

as node()\*

**all-descendant-or-self**(

$namesFilter as xs:string?) as node()\*

**all-descendant-or-self**() as node()\*

**all-descendant-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 elements and their attributes, optionally filtered by name and/or position.

**Details**

Like function [all-descendant](#_all-descendant_(ec-*)), this function returns descendant elements and their attributes, but it also returns those input items themselves which are elements, along with their attributes. See function [all-descendant](#_all-descendant_(ec-*)) for all details, especially concerning the filtering of result nodes.

***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 descendant-or-self elements and their attributes. 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()"

### ancestor (\*-ec)

**ancestor**($namesFilter as xs:string?,

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

**ancestor**($namesFilter as xs:string?,

$pselecter as xs:integer?)

as node()\*

**ancestor**($namesFilter as xs:string?) as node()\*

**ancestor**() as node()\*

**ancestor-ec**(

$inputItems as item()\*,

$namesFilter as xs:string?,

$pselector as xs:integer?,

$options as xs:string?)

as node()\*

***Summary***

Returns ancestor nodes, optionally selected by local 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 ancestor nodes of the input items. The nodes returned by the function can be 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)).

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 (in reverse document order) backward, with -1, -2, … meaning the last item, second last item, etc.

*Options* are provided as option names separated by whitespace. Supported 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 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. |
| namesFilter | Name filter used for selecting result nodes. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). Note that use of this parameters implies that only element nodes are returned, as document nodes do not have a name. |
| 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***

[Under construction]

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

**ancestor-or-self**(

$namesFilter as xs:string?,

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

**ancestor-or-self**(

$namesFilter as xs:string?,

$pselecter as xs:integer?)

as node()\*

**ancestor-or-self**(

$namesFilter as xs:string?) as node()\*

**Ancestor-or-self**() as node()\*

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

$inputItems as item()\*,

$namesFilter as xs:string?,

$pselector as xs:integer?,

$options as xs:string?)

as node()\*

***Summary***

Returns ancestor-or-self nodes, optionally selected by local 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 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)).

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 (in reverse document order) backward, with -1, -2, … meaning the last item, second last item, etc.

*Options* are provided as option names separated by whitespace. Supported 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 ancestor-or-self and ancestor-or-self-ec.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| inputItems | NOTE: this parameter is only expected by function variant \*-ec.  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). Note that use of this parameters implies that only element nodes are returned, as document nodes do not have a name. |
| 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***

[Under construction]

### child (ec-\*)

**child**($namesFilter as xs:string?,

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

**child**($namesFilter as xs:string?,

$pselecter as xs:integer?)

as node()\*

**child**($namesFilter as xs:string?) as node()\*

**child**() as node()\*

**ec-child**(

$inputItems as item()\*,

$namesFilter as xs:string?,

$pselector as xs:integer?,

$options as xs:string?)

as node()\*

***Summary***

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

**Details**

Under construction.

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **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 must 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***

Under construction.

### attributes (ec-\*)

**attributes**($namesFilter as xs:string?,

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

**attributes**($namesFilter as xs:string?,

$pselecter as xs:integer?)

as node()\*

**attributes**($namesFilter as xs:string?) as node()\*

**attributes**() as node()\*

**ec-attributes**(

$inputItems as item()\*,

$namesFilter as xs:string?,

$pselector as xs:integer?,

$options as xs:string?)

as node()\*

***Summary***

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

**Details**

Under construction.

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **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 must 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***

Under construction.

### descendant (ec-\*)

**descendant**($namesFilter as xs:string?,

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

**descendant**($namesFilter as xs:string?,

$pselecter as xs:integer?)

as node()\*

**descendant**($namesFilter as xs:string?) as node()\*

**descendant**() as node()\*

**ec-descendant**(

$inputItems as item()\*,

$namesFilter as xs:string?,

$pselector as xs:integer?,

$options as xs:string?)

as node()\*

**Summary**

Returns descendant 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 nodes which are descendant elements of the input items. The nodes returned by the function can be filtered by name and/or by position.

Function variant ec- descendant 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 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 descendant and ec-descendant.

|  |  |
| --- | --- |
| **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**

….

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

**descendant-or-self**($namesFilter as xs:string?,

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

**descendant-or-self**($namesFilter as xs:string?,

$pselecter as xs:integer?)

as node()\*

**descendant-or-self**($namesFilter as xs:string?) as node()\*

**descendant-or-self**()

as node()\*

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

$inputItems as item()\*,

$namesFilter as xs:string?,

$pselector as xs:integer?,

$options as xs:string?)

as node()\*

**Summary**

Returns descendant-or-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 nodes which are descendant-or-self elements of the input items. The nodes returned by the function can be filtered by name and/or by position.

Function variant ec-descendant-or-self 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)).

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 descendant-or-self and ec-descendant-or-self.

|  |  |
| --- | --- |
| **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**

….

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

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

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

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

$pselecter as xs:integer?)

as node()\*

**following-sibling**($namesFilter as xs:string?) as node()\*

**following-sibling**()

as node()\*

**ec-following-sibling**(

$inputItems as item()\*,

$namesFilter as xs:string?,

$pselector as xs:integer?,

$options as xs:string?)

as node()\*

**Summary**

Returns following-sibling 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 nodes which are following-sibling elements of the input items. The nodes returned by the function can be filtered by name and/or by position.

Function variant ec-following-sibling 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 ec-following-sibling.

|  |  |
| --- | --- |
| **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**

….

### parent (ec-\*)

**parent**($namesFilter as xs:string?,

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

**parent**($namesFilter as xs:string?,

$pselecter as xs:integer?)

as node()\*

**parent**($namesFilter as xs:string?) as node()\*

**parent**()

as node()\*

**ec-parent**(

$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. Atomic input items are interpreted as document URI and replaced with the corresponding document node. The function returns nodes which are following-sibling elements of the input items. The nodes returned by the function can be filtered by name and/or by position.

Function variant ec-parent 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 ec-parent.

|  |  |
| --- | --- |
| **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**

….

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

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

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

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

$pselecter as xs:integer?)

as node()\*

**preceding-sibling**($namesFilter as xs:string?) as node()\*

**preceding-sibling**()

as node()\*

**ec-preceding-sibling**(

$inputItems as item()\*,

$namesFilter as xs:string?,

$pselector as xs:integer?,

$options as xs:string?)

as node()\*

**Summary**

Returns preceding-sibling 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 nodes which are preceding-sibling elements of the input items. The nodes returned by the function can be filtered by name and/or by position.

Function variant ec-preceding-sibling 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 ec-preceding-sibling.

|  |  |
| --- | --- |
| **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 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**($namesFilter as xs:string?,

$pselecter as xs:integer?)

as node()\*

**self**($namesFilter as xs:string?) as node()\*

**self**()

as node()\*

**ec-self**(

$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 those nodes among the input items which are elements, optionally filtered by name and/or by position.

Function variant ec-self 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 ec-self.

|  |  |
| --- | --- |
| **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**

….

### sibling (ec:\*)

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

**sibling**($namesFilter as xs:string?,

$pselecter as xs:integer?,

$options as xs:string?)

as node()\*

**sibling**($namesFilter as xs:string?,

$pselecter as xs:integer?)

as node()\*

**sibling**($namesFilter as xs:string?) as node()\*

**sibling**()

as node()\*

**ec-sibling**(

$inputItems as item()\*,

$namesFilter as xs:string?,

$pselector as xs:integer?,

$options as xs:string?)

as node()\*

**Summary**

Returns sibling 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 nodes which are sibling elements of the input items. The nodes returned by the function can be filtered by name and/or by position.

Function variant ec- sibling 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 ec-sibling.

|  |  |
| --- | --- |
| **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**

….

## Functions supporting file system navigation

The functions in this section support a more concise expression of complex file system navigation.

### fancestor - ec-fancestor

**ec-fancestor**($uris as item()\*, $names as xs:string?, $namesExcluded as xs:string, $pselector as item()?, $caseSensitive as xs:boolean?) as node()\*

**ec-fancestor**($uris as item()\*, $names as xs:string?, $namesExcluded as xs:string, $pselector as item()?) as node()\*

**ec-fancestor**($uris as item()\*, $names as xs:string?, $namesExcluded as xs:string) as node()\*

**ec-fancestor**($uris as item()\*, $names as xs:string?) as node()\*

**ec-fancestor**($uris as item()\*) as node()\*

**ec-fancestor**() as node()\*

***Summary***

Returns ancestor URIs of given URIs, optionally selected by name and/or position.

***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.

For each input URI, ancestor URIs are returned in reverse document order. If $names and/or $namesExcluded are specified, only ancestors with a name matching a name or name pattern from $names, and not matching a name or name pattern from $namesExcluded are considered. Parameter $pselector selects an individual ancestor by position in reverse file system order, specified as an integer number or the string #last.

When there are several input URIs, the result sequence is composed of subsequences, each one of which contains the ancestors of an input URI in reverse file system order. In this case, duplicate URIs may occur, and the result as a whole may neither be in file system order nor in reverse file system order.

***Parameters***

Described by the following table.

**Table**. Parameters of function fox-ancestor.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| uris | URIs whose ancestor URIs will be returned |
| names | Whitespace-separated list of names or name patterns; only ancesters with a file name matching one of these will be returned |
| namesExcluded | Whitespace-separated list of names or name patterns; only ancesters with a file name matching none of these will be returned |
| pselector | An integer number or the token #last; for each input URI, only the ancestor at corresponding position will be returned |
| caseSensitive | If true, file names are compared with parameter values in a case-sensitive way; by default, names are compared in a case-insensitive way |

**Examples**

Get the first ancestor URIs belonging to a folder with name proj\*.

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

### fancestor-or-self - ec-fancestor-or-self

**fox-ancestor-or-self**($uris as item()\*, $names as xs:string?, $namesExcluded as xs:string, $pselector as item()?, $caseSensitive as xs:boolean?) as node()\*

**fox-ancestor-or-self** ($uris as item()\*, $names as xs:string?, $namesExcluded as xs:string, $pselector as item()?) as node()\*

**fox-ancestor-or-self** ($uris as item()\*, $names as xs:string?, $namesExcluded as xs:string) as node()\*

**fox-ancestor-or-self** ($uris as item()\*, $names as xs:string?) as node()\*

**fox-ancestor-or-self** ($uris as item()\*) as node()\*

**fox-ancestor-or-self** () as node()\*

***Summary***

Returns ancestor-or-self URIs of given URIs, optionally selected by local name and/or position.

***More information***

Function semantics are identical to the semantics of function [fox-ancestor](#_fox-ancestor), with the exception that fox-ancestor-self considers the ancestor-or-self URIs, whereas fox-ancestor considers only the ancestor URIs. See function [fox-ancestor](#_fox-ancestor) for details, a description of the function parameters and examples.

### fchild - ec-fchild

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

**fox-child**($names as xs:string?, $namesExcluded as xs:string?) as xs:string\*

***Summary***

Returns selected child URIs of the context URI.

***Details***

Returns the child URIs of the context URI with a name matching a name or name pattern from $names, and not matching a name or name pattern from $namesExcluded. The parameters $names and $namesExcluded are whitespace-separated lists of names or name patterns.

***Examples***

Get all child URIs with a name matching \*concur\* or \*fieldglass\*.

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

Get all child resources with a name matching \*concur\*, but not matching \*expense\* or ignore\*.

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

### fdescendant - ec-fdescendant

fox-descendant($names)

fox-descendant($names, $namesExcluded?)

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

### fdescendant-or-self - ec-fdescendant-or-self

fox-descendant-or-self($names)

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

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

### ffollowing-sibling - ec-ffollowing-sibling

**ec-ffollowing-sibling**(…)

…

### fparent - ec-fparent

**fparent**(…) as xs:string\*

***Summary***

…

### fparent-sibling - ec-fparent-sibling

**ec-fparent-sibling**(…)

…

### preceding-sibling - ec-fpreceding-sibling

**ec-fpreceding-sibling**(…)

…

### fself - ec-fself

**ec-fself**($uris as item()\*, $names as xs:string?, $namesExcluded as xs:string, $pselector as item()?, $caseSensitive as xs:boolean?) as node()\*

**ec-fself**($uris as item()\*, $names as xs:string?, $namesExcluded as xs:string, $pselector as item()?) as node()\*

**ec-fself**($uris as item()\*, $names as xs:string?, $namesExcluded as xs:string) as node()\*

**ec-fself**($uris as item()\*, $names as xs:string?) as node()\*

**ec-fself**($uris as item()\*) as node()\*

**ec-fself**() 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

fox-sibling($names)

fox-sibling($names, $namesExcluded?)

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

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

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

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

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

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

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

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

### fparent-shifted – ec-fparent-shifted

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

$nameReplaceSubstring as xs:string?,

$nameReplaceWith as xs:string?) as xs:string\*

**ec-fparent-shifted**($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 | [Paremeter 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

## Functions creating 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)

**ftree**($dirs as item()\*,

$foldersExcluded as xs:string?,

$fileProperties as xs:string?, …)

as element()

***Summary***

Returns a tree representation of folder contents.

***Details***

For each folder in $dirs, a tree representation is generated, consisting of an <ftree> element with <fo> and <fi> descendants representing contained folders and files. If $dirs identifies several folders, the corresponding <ftree> elements are wrapped in an <ftrees> element; if $dirs identifies a single folder, the corresponding <ftree> element is returned without a wrapper element.

If $foldersExcluded is specified, all folders with a matching name are ignored.

If $fileProperties is specified, file descriptors are annotated by attributes and/or child elements providing file properties. Each $fileProperties argument specifies a property name and a Foxpath expression providing its value, separated by an equal sign. The Foxpath expression is evaluated in the context of the file URI. If the property name is preceded by an @ character, the property is represented by an attribute, otherwise by a child element. Optionally, the property name is preceded by one or several glob patterns selecting the file names of files to be annotated. File name patterns are separated by whitespace. File name pattern(s) and property name are also separated by whitespace.

***Parameters***

Described by the following table.

**Table**. Parameters of function annotate.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| dirs | The folders to be described. |
| foldersExcluded | Identifies folders to be ignored when describing folder contents. The parameter value is a whitespace-separated list of name patterns. |
| properties  … | Each parameter describes a file property, format: file-name-filter property-name=property-value-expression. Where  file-name-filter – optional glob pattern(s) selecting files to be annotated; patterns white-space separated  property-name – the property name, used as attribute or element name; if preceded by character @, the property is represented by an attribute, otherwise by a child element  property-value-expression – Foxpath expression providing the property value; expression is evaluated in the context of the file URI  Examples:  date=file-date()  \* date=file-date()  \*anchors\* @countMsgs=\\@msg => count()  \*anchors\* msgs=\\@msg\xelement('msg ', string()) |

**Examples**

Get a tree representation of the gi subfolder of the current work directory.

fox "gi/ftree()"

As before, but the tree representation omits any folders with a name matching the pattern \*-\*.

fox "gi/ftree(., '\*-\*')"

Get tree representations of all gi.\* descendant folders.

fox ".//gi.\* => ftree()"

Get tree representation with annotated file elements - @msg elements found in the file are mapped to <msg> child elements of the file descriptor.

fox "../output-convert-mass/ftree(., (), '\*anchor\* msgs=\\@msg\xelement(local-name(.), string())')"

Get tree representation with annotated file elements - @rootName attributes provide the local root name.

fox "../output-convert-mass/ftree(., (), '\*anchor\* @rootName=\\*\name()')"

Get tree representation with annotated file elements - @rootName attributes provide the local root name, an <elemNames> child element a list of all distinct element names.

fox "../output-convert-mass/ftree(., (), '\*anchor\* @rootName=\\*\name()', '\*anchor\* elemNames=\\\*\name() => distinct-values() => sort() => string-join(`"`", `"`")')"

Get tree representation in which XSDs are annotated with their target namespace (@tns).

fox "configs/ftree(., (), '\*.xsd @tns=\\*\@targetNamespace')"

Get tree representation in which XSDs are annotated with their target namespace (@tns) and one child element (ename) for each top-level element name, containing that name..

fox "configs/ftree(., (), '\*.xsd @tns=\\*\@targetNamespace', '\*.xsd enames=\\*\xs:element\@name => sort() => xitems(`"`"ename`"`")')"

Get tree representation annotated with the result of a full-text search. The property @super has the value true if the document contains the text “Dieses Arzneimittel unterliegt”. Note the escaping of double quotes: they are escaped by a backtick, as required by the Powershell, and doubled, as required by the DOS script called by the Powershell script.

fox "../output-convert-mass/ftree(., (), '\*fibook\* @super = \contains-text(`"`"Dieses Arzneimittel unterliegt`"`")')"

### ftree-selective (ec-ftree-selective)

**ftree-selective**($dirs as item()\*,

$descendantNamesFilter as xs:string?,

$folderNamesFilter as xs:string?,

$fileProperties as xs:string?,

…)

as element()

***Summary***

Returns a tree representation of selected folder contents.

### ftree-view

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

$fileProperties as xs:string?,

…)

## Functions exploring node trees

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

### 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\*

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

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

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

$nameFilter as xs:string?)

as xs:string\*

**ec-child-names**($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 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 child names. By default, the child names are sorted lexicographilly. 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 |

***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\*

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

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

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

$nameFilter as xs:string?)

as xs:string\*

**ec-descendant-names**($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\*

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

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

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

$nameFilter as xs:string?)

as xs:string\*

**ec-att-names**($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 |

***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\*

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

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

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

$nameFilter as xs:string?)

as xs:string\*

**ec-content-names**($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\*

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

$nameFilter as xs:string?,

$options as xs:string?)

as xs:string\*

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

$nameFilter as xs:string?)

as xs:string\*

**ec-parent-name**($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 (lpath-, jpath-, ec-…)

**path-content**($descendantNamesFilter as xs:string,

$excludedInnerNamesFilter as xs:string?,

$options as xs:string?)

**path-content**($descendantNamesFilter as xs:string,

$excludedInnerNamesFilter as xs:string?)

**path-content**($descendantNamesFilter as xs:string)

**path-content**()

**lpath-content**(…)

**jpath-content**(…)

**ec-path-content**($nodesOrUris as item()+, …)

**ec-lpath-content**($nodesOrUris as item()+, …)

**ec-jpath-content**($nodesOrUris as item()+, …)

***Summary***

Returns for each input item (node or URI) the relative paths of its content nodes.

***Details***

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

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

***…***

***Parameters***

***…***

***Examples***

***…***

### name-content (lname-, jname-, ec-…)

**name-content**($nameFilter as xs:string, $flags as xs:string?) as xs:string\*

**lname-content**($nameFilter as xs:string, $flags as xs:string?) as xs:string\*

**jname-content**($nameFilter as xs:string, $flags as xs:string?) as xs:string\*

**node-names**($nameFilter as xs:string) as xs:string\*

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

**ec-node-names**(

$nodesOrUris as item()+,

$nameFilter as xs:string?,

$options as xs:string?) as xs:string\*

**ec-node-names**(

$nodesOrUris as item()+,

$nameFilter as xs:string?) as xs:string\*

**ec-node-names**(

$nodesOrUris as item()) 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 node-names.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| contextItems | The items to be evaluated. Atomic items are interpreted as document URI and replaced by the corresponding document node. |
| namesFilter | Name filter used for selecting the nodes to be reported. The parameter value is a [Unified Filter Expression](#_Filter_syntax_1). |
| flags | A string controlling which related nodes are reported:   * Contains character p – parent nodes of the reported nodes * Contains character a – attribute nodes of the reported nodes * Contains character c – child nodes of the reported nodes |
| options | Tokens controlling the execution:   * name – display lexical names (possibly containing a prefix) * lname – display local names * jname – display 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**()"

## Functions exploring node location

The functions in this section deal with the location of nodes.

### name-path (lname-path, jname-path)

**name-path**($nodes as node()\*, $numberOfSteps as xs:integer?, $flags as xs:string?)

as xs:string\*

**name-path**($nodes as node()\*, $numberOfSteps as xs:integer?) as xs:string\*

**name-path**($nodes as node()\*) as xs:string\*

**name-path**() as xs:string\*

**lname-path**($nodes as node()\*, $numberOfSteps as xs:integer?, $flags as xs:string?)

as xs:string\*

**lname-path**($nodes as node()\*, $numberOfSteps as xs:integer?) as xs:string\*

**lname-path**($nodes as node()\*) as xs:string\*

**lname-path**() as xs:string\*

**jname-path**($nodes as node()\*, $numberOfSteps as xs:integer?, $flags as xs:string?)

as xs:string\*

**jname-path**($nodes as node()\*, $numberOfSteps as xs:integer?) as xs:string\*

**jname-path**($nodes as node()\*) as xs:string\*

**jname-path**() as xs:string\*

Including variants:

* jname-path() – consider JSON names, not lexical node names
* lname-path() – consider local names, not lexical node names

***Summary***

Returns the name paths of given nodes.

***Details***

Under construction.

***Parameters***

Described by the following table.

**Table**. Parameters of functions name-path, lname-path, jname-path.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| Nodes | Nodes to be reported. |
| numberOfSteps | The number of trailing path steps to be included; by default, the complete path is constructed. Example: if equal 3, only the last three steps of the paths are included (e.g. c/d/e, rather than the complete path /a/b/c/d/e). |
| flags | Options controlling the processing:  N – node names are not concatenated into a path, but returned as a sequence, in document order |

***Examples***

Get the paths of all XSD annotation elements:

fox "/projects/project-foo/xsd/foo.xsd\\xs:annotation\name-path() => f()"

/xs:schema/xs:element/xs:annotation ............................ (60)

/xs:schema/xs:element/xs:complexType/xs:annotation ............. (1)

/xs:schema/xs:element/xs:complexType/xs:attribute/xs:annotation (1)

/xs:schema/xs:group/xs:annotation .............................. (3)

As before, but for each path only the last 3 steps:

fox "/projects/project-foo/xsd/foo.xsd\\xs:annotation\name-path(., 3) => f()"

xs:complexType/xs:attribute/xs:annotation (1)

xs:element/xs:complexType/xs:annotation .. (1)

xs:schema/xs:element/xs:annotation ....... (60)

xs:schema/xs:group/xs:annotation ......... (3)

### 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?)

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)

## Functions evaluating values

The functions in this section support the inspection and transformation of values.

### both-values, bvalues

**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

**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())"

## Functions 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

**matches-pattern**($item as item(), $pattern as xs:string) as xs:boolean

**matches-pattern**($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.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| item | The item to be checked.  NOTE: if the function is called with a single argument, this parameter defaults to the context item of the function call. |
| 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()"

## Functions processing strings

The functions in this section support the evaluation of strings.

### text-to-codepoints

**text-to-codepoints**($text as xs:string?) as xs:string\*

**text-to-codepoints**()

***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**($string as xs:string?, $len as xs:integer, $flags as xs:string?)

as xs:string?

**truncate**($string as xs:string?, $len as xs:integer) as xs:string?

**truncate**($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 | 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 ...

## Functions performing full-text evaluation

The functions in this section support the evaluation of strings in accordance with full text concepts.

### contains-text

**contains-text**($items as item()\*, $ftSearch as xs:string) as xs:boolean

**contains-text**($ftSearch as xs:string) as xs:boolean

***Summary***

Returns true if one of the supplied items matches a given full-text search.

### 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

## Functions comparing document contents

The functions in this section support the comparison of documents.

### path-compare (lpath-, jpath-, ec-…)

**path-compare**($otherNode as item(), $cmpType as xs:string?)

**lpath-compare**($otherNode as item(), $cmpType as xs:string?)

**jpath-compare**($otherNode as item(), $cmpType as xs:string?)

**ec-path-compare**($node as item(), $otherNode as item(), $cmpType as xs:string?)

**ec-lpath-compare**($node as item(), $otherNode as item(), $cmpType as xs:string?)

**ec-jpath-compare**($node as item(), $otherNode as item(), $cmpType as xs:string?)

***Summary***

...

***Details***

***…***

***Parameters***

Described by the following table.

**Table**. Parameters of function path-compare.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| otherNode | The node with which to compare, or the document URI identifying the node with which to compare |
| cmpType | Specifies how to compare:  plain – compare the disambiguated name paths  path-count – compare the disambiguated name paths, associated with the number of occurrences of the respective path |

***Examples***

Compare two sibling files, using plain path comparison.

fox "foo.xml/path-compare(../bar.xml)"

<deviations uri1="file:///C:/projects/a/foo.xml" uri2="file:///C:/projects/a/bar.xml">

<only2 count="1">

<loc path="/sections/l:section/l:paragraph/l:figure/l:image/@depthh"/>

</only2>

</deviations>

### node-deep-equal (ec-node-deep-equal)

**node-deep-equal**($otherNodes as item()+) as xs:boolean

**ec-node-deep-equal**($nodes as item()+, ) as xs:boolean

***Summary***

Compares two or more nodes for deep equality.

***Details***

Compares two or more nodes for deep equality. The nodes can be supplied as nodes or as document URI.

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| otherNodes | The nodes with which to compare the context node, or document URIs identifying the nodes with which to compare |
| nodes | The nodes which to compare among each other. |

***Examples***

Compare two sibling files, using plain path comparison. Node the parent step required to reach the sibling file.

fox "foo.xml/node-deep-equal(../bar.xml)"

true

Compare the context resource with (potentially) several siblings.

fox "foo.xml/node-deep-equal(../foo.sav\*.xml)"

false

Equivalent to the first example.

fox "foo.xml => node-deep-equal(bar.xml)"

true

### node-deep-similar (ec-node-deep-similar)

**node-deep-similar**($otherNodes as item()+, $excludeExprs as xs:string\*)

as xs:boolean

**ec-node-deep-similar**($nodes as item()+, $excludeExprs as xs:string\*)

as xs:boolean

***Summary***

Compares two or more nodes for deep similarity.

***Details***

Compares two or more nodes for deep similarity. The nodes can be supplied as nodes or as document URI.

Deep similarity means that after removing nodes selected by supplied expressions, the compared nodes are deep-equal.

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| otherNodes | The nodes with which to compare the context node, or document URIs identifying the nodes with which to compare |
| nodes | The nodes which to compare among each other. |

***Examples***

Compare two sibling files, using plain path comparison. Before comparing, any @creationDate attributes are removed. Node the parent step required to reach the sibling file.

fox "foo.xml/node-deep-similar(../bar.xml, '\\@creationDate')"

true

Compare the context resource with (potentially) several siblings.

fox "foo.xml/node-deep-similar(../foo.sav\*.xml, '\\@creationDate')"

false

Equivalent to the first example.

fox "foo.xml => node-deep-similar(bar.xml, '\\@creationDate')"

true

## Functions 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-date , fdate

file-date($fileUri)

file-date()

***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-date-string , fdates

file-date-string($fileUri)

file-dates()

***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-info , finfo

file-info()

file-info($format)

Returns a string describing the context resource.

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

Item kind:

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

Display format:

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

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

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

Default display: #nsd

Examples:

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

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

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

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

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

### 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 selectede 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]"

## Functions creating tables and hierarchical lists

Bla.

### table

**table**($rows as xs:string\*,

$header as xs:string\*) as xs:string

**table**($rows as xs:string\*) as xs:string

***Summary***

Returns the string representation of a table.

***Details***

The data input consists of rows created using function row().

***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. |

**Examples**

Gets this or that.

### hlist

hlist($hlistEntries, $emptyLinesSpec)

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

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

Example: hlist($entries, '010')

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

### row

row($value …)

***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.

## annotate

**annotate**($item as item(),

$annotation as xs:string) as xs:string

**annotate**($annotation as xs:string) as xs:string

***Summary***

Returns the string value of an item, with the annotation in parentheses appended.

***Details***

The item is atomized. The annotation is surrounded by parentheses. Item and annotation are separated by a blank.

If the function is called with a single parameter, the parameter is interpreted as annotation, and the item to be annotated is the context item.

***Parameters***

Described by the following table.

**Table**. Parameters of function annotate.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| item | The item to be annotated. |
| annotation | The annotation to be appended. |

**Examples**

Get a list of folders contained by the current work folder, annotated with the number of XSDs directly or indirectly contained.

fox "\*/annotate(count(.//\*.xsd))"

## file-extension , file-ext, fext

file-extension($fileUri)

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

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

Example: frequency distribution of all file extensions

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

## 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

***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

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

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

**base-file-name**($node as item()) as xs:string

**base-file-name**() as xs:string

***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**

#Gets this or that.

## base-uri-relative

**base-uri-relative**($context as xs:string) as xs:string

***Summary***

#Returns this or that.

## content-deep-equal

***Usage***

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

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

***Summary***

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

***Details***

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

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

***Parameters***

Described by the following table.

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

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

***Examples***

Comparing three documents specified by URI:

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

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

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

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

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

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

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

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

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

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

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

Given the following files:

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

Several Foxpath expressions yield values as shown below:

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

## count-chars

**count-chars**($string 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.

## csv-doc, cdoc

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

csv-doc()

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

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

Parameters:

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

Example: to parse the following CSV document:

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

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

data.csv/csv-doc()

csv-doc(data.csv)

Example: to parse the following CSV document:

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

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

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

## dcat

**dcat**($uris as xs:string\*, $onlyDocAvailable as xs:boolean?) as element(dcat)

**dcat**($uris as xs:string\*) as element(dcat)

***Summary***

Creates a catalog of document URIs. Such a catalog can be processed by other programs, e.g. XSLT stylesheets.

## dir-name, dname

**dir-name**($uri as xs:string) as xs:string

**dir-name**() as string

***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.

## 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')]"

## 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 | |

#----------------------------------------#

## frequencies (freq, f)

**frequencies**($values as item()\*,

$minFreq as xs:integer?,

$maxFreq as xs:integer?,

$order as xs:string?,

$format as xs:string?)

as item()

**frequencies**($values as item()\*,

$minFreq as xs:integer?,

$maxFreq as xs:integer?,

$order as xs:string?)

as item()

**frequencies**($values as item()\*,

$minFreq as xs:integer?,

$maxFreq as xs:integer?)

as item()

**frequencies**($values as item()\*,

$minFreq as xs:integer?)

as item()

**frequencies**($values as item()\*)

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. Values can be filtered by a minimum and/or maximum frequency. The sort order of the returned items, as well as the result format (text, CSV, JSON, XML) can be controlled – see *Parameters*.

***Parameters***

Described by the following table.

**Table**. Parameters of function frequencies.

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| values | The values to be analyzed |
| minFreq | Report only values with a frequence >= the parameter value |
| maxFreq | Report only values with a frequence <= the parameter value |
| order | Sort order:  a – order by ascending frequence  b – order by descending frequence  t – order alphabetically, ascending (default)  T – order alphabetically, descending  n – order numerically, ascending  N – order numerically, descending |
| format | Result format:  text – text file;  each distinct value a line, values not padded (default);  text\* – text file;  each distinct value a line, value padded to the length of the longest value string  textNN – text file;  each distinct value a line, value padded to a length of NN characters (e.g. text40)  csv – CSV document  json – JSON document  xml – XML document (root element name values, child element name value) |

***Examples***

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

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

## 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')"

## html-doc, hdoc

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

**html-doc**() as document-node()?

***Summary***

Parses the file found at the input URI into a node tree, treating file content as an HTML document.

***Details***

…

## indent

**indent**($lines as xs:string\*, $width as xs:integer) as xs:string\*

**indent**($lines as xs:string\*) as xs:string\*

**indent**() as xs:string\*

***Summary***

Returns input strings, prefixed with a substring supposed to represent an indentation.

***Details***

***…***

***Parameters***

***…***

***Examples***

…

## in-scope-namespaces

**in-scope-namespaces**(…) as …

***Summary***

...

***Details***

***…***

## in-scope-namespaces-descriptor

**in-scope-namespaces-descriptor**(…) as …

***Summary***

...

***Details***

***…***

## 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:bolean

***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**(…) as …

***Summary***

...

***Details***

***…***

## jpath-compare; jpathcmp

jpath-compare($nodes, options)

## jschema-keywords, jskeywords

**jschema-keywords**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## jsoncat

**jsoncat**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## json-doc, jdoc

**json-doc**(…) 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 …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## lpath-compare, lpathcmp

**lpath-compare**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## median

**median**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## 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)

***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***

***…***

## non-distinct-values , non-distinct

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

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

Example: return all OAS

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

Response:

AlertNotification

. cf\_configuration\_api.json

. . create, delete, get, getAll, update

. neo\_configuration\_api.json

. . create, delete, get, getAll, update

NFEAPIS

. nfe\_authorize.json

. . downloadNFe

SAPCustomerDataCloud

. GigyaAPI\_accounts\_b2b\_registerOrganization.json

. . accounts.b2b.registerOrganization

## oas-jschema-keywords

**oas-jschema-keywords**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## oas-keywords

**oas-keywords**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## oas-msg-schemas, oasmsgs

**oas-msg-schemas**(…)

***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**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## percent

**percent**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## pfrequencies, pfreq, pf

**pfrequencies**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## ps.copy

**ps.copy**(…)

***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**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## repeat

**repeat**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## resolve-json-allof, jallof

**resolve-json-allof**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## resolve-json-anyof, janyof

**resolve-json-anyof**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## resolve-json-one of, joneof

**resolve-json-oneof**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## resolve-json-ref ; jsonref ; jref

**resolve-json-ref**($ref, $mode)

**resolve-json-ref**($ref)

**resolve-json-ref**()

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## resolve-link

**resolve-link**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## resolve-path

**resolve-path**(…)

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## resolve-xsdtype-ref, typeref

**resolve-xsdtype-ref**(…)

***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***

***…***

## values-distinct

values-distinct($values)

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

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

## win.copy

**win.copy**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## win.delete

**win.delete**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## write-doc

**write-doc**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## write-file

**write-file**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## write-files

**write-files**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## write-json-docs

**write-json-docs**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## xelement

**xelement**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***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***

***…***

## xroot-matches, xroot

**xroot-matches**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## xroot-name

**xroot-name**(…) as …

***Summary***

...

***Details***

***…***

***Parameters***

***…***

***Examples***

***…***

## xsd-validate, xval

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

***Summary***

Validates all documents against schemas selected from the supplied schemas.

***Details***

Give a detailed description.

***Parameters***

Described by the following table.

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

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| docs | 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. Warning - if supplied as nodes, line numbers referring to erroneous schema content may be incorrect. |

**Examples**

Gets this or that.

## xwrap

**xwrap**($values as item()\*,

$rootName as xs:string,

$flags as xs:string?,

$itemName as xs:string?)

***Summary***

Transforms a sequence of values into an XML document.

Options:

*Options available for node items*

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

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

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

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

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

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

*Options available for atomic items*

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

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

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

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

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

Examples

Wrap strings in elements with a specified name (p):

fox "boo.xml\\file\@name\string() => xwrap('fileNames', 'c', 'file-name')"

## PROJECT: path-template ; ptemplate

path-template($paths, $tvarSpec …)

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

$tvarSpec:

number whitespace varName

| negnumber whitespace varName

| pathPattern:

examples:

paths/{path}

{path}/get

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

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

foo\*/{varname}/bar\*

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

Leading / - pattern anchored

Trailing / - pattern anchored

Steps can contain wildcards

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

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

{varname: (a\*, b\*, c\*)} – …s