



Functions & Operators 4.0



Overview

- **EXPath** File and Binary modules adopted and refined
- ~90 new functions
- ~60 functions extended
- Options maps where appropriate
- Argument names are *now significant* (keyword calls)
- Consistency improved



Optional arguments

Multiple-arity functions now defined with optional arguments:

```
fn:deep-equal($parameter1 as item()*,  
               $parameter2 as item()*) as xs:boolean  
fn:deep-equal($parameter1 as item()*,  
               $parameter2 as item()*,  
               $collation as xs:string) as xs:boolean
```

```
fn:deep-equal(  
    $input1 as item()*,  
    $input2 as item()*,  
    $options as (xs:string | map(*))? := {}  
) as xs:boolean
```



F&O 4.0

Sequence position arguments in Higher-Order Functions

Many of the higher-order functions now have an optional sequence position:

```
fn:filter($input as item()*,  
          $predicate as fn(item(), xs:integer) as xs:boolean?  
        ) as item()*
```

Sequence position

```
('a','z') =!> string-to-codepoints()  
=> fn($t) { $t[1] to $t[2] }() =!> codepoints-to-string()  
=> filter(fn($item,$pos) {$pos mod 2 eq 0})
```



'b', 'd', 'f'... 'x', 'z'



F&O 4.0

Keyword arguments in static function calls

Argument
names now
significant:

```
fn:sort( $input as item()*,  
         $collation as xs:string? := fn:default-collation(),  
         $key as fn(item()) as xs:anyAtomicType* := fn:data#1  
       ) as item()*
```

Arguments by position

```
sort(//employee, key := fn($e) { xs:decimal($e/salary) })
```

Arguments by keyword (following)

```
fn:sort(//employee, fn:default-collation(),  
        fn($e) { xs:decimal($e/salary) })
```



Options as a record argument

Mostly on functions processing maps, arrays and resources:

```
fn:csv-to-arrays(  
    $value as xs:string?,  
    $options as map(*)? := {}  
) as array(xs:string)*
```

```
record(  
    field-delimiter? as xs:string,  
    row-delimiter? as xs:string,  
    quote-character? as xs:string,  
    trim-whitespace? as xs:boolean  
)
```



XPath workbench

- Select XPath/XQuery--
- General
 - argument keywords
- CSV Processing
 - csv1
 - csv2
 - csv3
- Higher-Order
 - filter() - positional
 - do-until() - Squares
 - do-until() - Fibonacci
 - partition() - group-together
- Problem: partition() - line-wrapping
- Solution: partition() - line-wrapping
- transitive-closure() - Reporting
- JSON
 - lookup
 - element-to-map()
 - element-to-map() - options
- Maps
 - map-build()

QT4 XPath Examples ▶ Help About

Line wrap a string using the `partition()` function. The input to `$line-wrap()` is supplied as a non-empty normalized string.

Code Choose file No file chosen Problem: partition() - line-wrapping Run

```
let $line-wrap := function($in,$length as xs:integer) {
    partition(
        tokenize($in),
        $inpartition, $next) {
            sum($partition, $next) ! string-length()) + count($partition) gt $length
        }
    ) => string-join(" ")
}
let $sentences := normalize-space(.)
    => tokenize("\.\s*")
    => filter(matches(?,"\$"))
return
    for $s in $sentences
        return $line-wrap($s,30)
```

INPUT

In the beginning was the Word, and the Word was with God, and the Word was God.
The same was in the beginning with God.
All things were made by him; and without him was not any thing made that was made.
In him was life; and the life was the light of men.
And the light shineth in darkness; and the darkness comprehended it not.
There was a man sent from God, whose name was John.

RESULT(s)

In the beginning was the Word,
and the Word was with God,
and the Word was God.

The same was in the beginning
with God.

All things were made by him;
and without him was not any
thing made that was made.

In him was life; and the life
was the light of men.

And the light shineth in
darkness; and the darkness
comprehended it not.

There was a man sent from God,
whose name was John.

Problem statement & hints

CODE

INPUT

RESULT(s)

CODE & INPUT
can be edited and loaded by:

- Example drop-down
- Local file chooser
- Drag-and-drop (not Firefox)

Run uses
BaseX Fiddle
to process the
expression
(Many thanks!)



General Functions on Sequences

fn:foot()	The last of a sequence	<i>cf fn:head()</i>
fn:trunk()	All but the last	<i>cf fn:tail()</i>
fn:identity()	Passes argument unchanged	
fn:void()	Consumes the argument	
fn:items-at()	Items from selected positions	Can be any order and have duplicates
fn:replicate()	Multiple copies of a sequence	(1 to \$count) ! \$input
fn:sequence-join()	Insert separator between adjacent items	
fn:slice()	Subsequence with 'step/stride'	Can be in reverse <i>cf Javascript/Python</i>



Comparison Functions

fn:atomic-equal()	(Coerced) type/value equality	\equiv comparing map keys
fn:contains-subsequence()	One sequence contains a given contiguous subsequence	Used-suppliable comparison function
fn:duplicate-values()	Values that appear more than once	
fn:ends-with-subsequence()	One sequence finishes with a given contiguous subsequence	Used-suppliable comparison function
fn:starts-with-subsequence()	One sequence starts with a given contiguous subsequence	Used-suppliable comparison function
fn:all-equal()	All items have equal atomic value	Used-suppliable collation
fn:all-different()	No two items have equal atomic value	Used-suppliable collation



Basic Higher-Order Functions i

fn:do-until() fn:while-do()	Processes a value repeatedly while a condition is (false true)	Action & predicate functions, with <i>\$position</i>
fn:every() fn:some()	(Every at least one) item satisfies a predicate	Predicate function with <i>\$position</i>
fn:highest() fn:lowest()	Items which have the (highest lowest) sort key	Collation & key function
fn:sort-by()	Sort using several sort-key functions	Collations & sort-key value functions
fn:sort-with()	Sort using comparator functions	List of -ve 0 +ve comparison functions



Example – fn:do-until()

Starting value Current result

```
fn:do-until(  
    $input      as item()*  
    $action     as fn(item()*  
    $predicate   as fn(item()*  
) as item()*
```

Stop when Sequence position

The diagram illustrates the parameters of the `fn:do-until` function. It shows the starting value, current result, stop when condition, and sequence position. The parameters `$input`, `$action`, and `$predicate` are highlighted with red boxes and arrows. The `fn:do-until` function takes three parameters: `$input` (as `item()*`), `$action` (as `fn(item())*`), and `$predicate` (as `fn(item())*`). The `$predicate` parameter is further detailed as `xs:integer) as item()*, xs:integer) as xs:boolean?`. The `Sequence position` label points to the final part of the predicate.

Example
Higher-Order
do-until()



Basic Higher-Order Functions ii

fn:index-where()	Positions in input of items that match predicate	Predicate function with \$position
fn:subsequence-where()	Contiguous subsequence – start and end points determined by predicates	Start & end predicate functions with \$position
fn:take-while()	Items prior to the first failing predicate match	Predicate function with \$position
fn:partition()	Partition into arrays splitting when a condition is true. cf xsl:for-each-group[@split-when]	Split function(\$so-far, \$item, \$position).
fn:scan-left() fn:scan-right()	Array of successive partial results of fn:fold-left() fn:fold-right()	



Example – fn:partition()

```
fn:partition(  
    $input as item()*,  
    $split-when as fn(item()* item(), xs:integer) as xs:boolean?  
) as array(item())*
```

Partition so far

Next item

Sequence position

Example
Higher-Order
partition()
line wrapping



Basic Higher-Order Functions iii

fn:transitive-closure()	All GNodes reachable by repeated function application	Can be used on nested maps/arrays
fn:partial-apply()	Partial binding of function arguments	More flexible than <i>f:is-larger(?,0)</i>
fn:op()	Generates a function to apply a binary operator to two arguments	

Example
Higher-Order
transitive-closure()
checking ixml



Numeric/Math Functions

fn:divide-decimals()	Divide xs:decimals to defined precision returning quotient and remainder.	
fn:isNaN()	true if xs:float or xs:double NaN value	
fn:parse-integer()	String to integer with radix 2 to 36	
math:e() math:cosh() math:sinh() math:tanh()		



F&O 4.0

String Functions

fn:collation()	Constructs a collation URI with properties.	For Unicode Collation Algorithm
fn:collation-available()	Checks whether a collation URI with properties is recognized by the implementation	
fn:char()	A string containing a single character or glyph	
fn:characters()	Split into a sequence of single-character strings	string-to-codepoints(\$value) ! codepoints-to-string(.)
fn:graphemes()	Produces grapheme clusters	
fn:hash()	Hash, checksum or CRC function applied to string or binary input	



URI & QName Functions

fn:decode-from-uri()	Decodes URI-escaped characters in a string	
fn:parse-uri()	Parses a URI to a map of its parts	
fn:build-uri()	Constructs a URI string from its parts	
fn:parse-QName()	EQName → xs:QName	prefix:local or Q{URI}local
fn:expanded-QName()	xs:QName → EQName	Q{URI}local



Date/Time Functions

fn:seconds()	A duration defined in seconds	
fn:unix-dateTime()	Conventional timezone offset for a given place and time	
fn:civil-timezone()	A string containing a single character or glyph	



Node Functions

fn:in-scope-namespaces()	A map of prefix → namespace URI in-scope for an element node	fn:namespace-uri-for-prefix() and fn:in-scope-prefixes() redefined in terms of this
fn:siblings()	Siblings of a GNode in document order	Works with map/array entries
fn:distinct-ordered-nodes()	Removes duplicate GNodes and sorts the input into document order	



F&O 4.0

Map Functions

map:build()	Build a map from a sequence of items	Key and value generating functions with <i>\$position</i>
map:empty()	true if a map has no entries	
map:entries()	Returns all the key-value pairs of a map as a sequence of single-entry maps	
map:filter()	Selects entries from a map, returning a new map	Predicate function(<i>\$key, \$value</i>)
map:items()	All the values of a map, as a sequence in order	
map:keys-where()	Selects keys from a map	Predicate function(<i>\$key, \$value</i>)



Example – map:build()

```
map:build(  
    $input  as item()*,  
    $key    as (fn($item as item(), $position as xs:integer)  
               as xs:anyAtomicType*)? := fn:identity#1,  
    $value  as (fn($item as item(), $position as xs:integer)  
               as item()*?)? := fn:identity#1,  
    $options as map(*)? := {}  
) as map(*)
```

Sequence
position

\$position as xs:integer

Example
Maps
map-build()
frequency analysis



Element → Map (for JSON)

fn:element-to-map()	Converts an element node into a map that is suitable for JSON serialization	Can use a pre-analysed plan
fn:element-to-map-plan()	Analyzes sample data to generate a conversion plan, organised by element and attribute name	Suitable for use by fn:element-to-map() Anticipated running on a sample set to then apply to a larger corpus

Example
JSON
element-to-map()-*



Array Functions - i

array:build()	Build an array from a sequence of items	Value generating function with <i>\$position</i>
array:empty()	true if an array has no entries	
array:foot()	The last member of an array	<i>cf fn:foot()</i>
array:trunk()	Remove the last member of an array	<i>cf fn:trunk()</i>
array:slice()	An array containing members of an input array selected by position	<i>\$start, \$end</i> and <i>\$step.</i> <i>cf fn:slice()</i>
array:index-of()	Returns positions of members equal to a target	Comparison using fn:deep-equal() with collation
array:index-where()	Returns positions of members which match a predicate	Predicate function(<i>\$value,\$position</i>)



Array Functions - ii

array:items()	All the values of an array, as a concatenated sequence	
array:members()	The contents of an array, as a sequence of value records	{'value': \$sequence}*
array:of-members()	Constructs an array from the contents of a sequence of value records	{'value': \$sequence}*
array:sort-by()	Sort using several sort-key functions	Collations & sort-key value functions <i>cf fn:sort-by()</i>
array:split()	The contents of an array, as a sequence of single entry arrays	



JNode Functions

fn:jtree()	Delivers a root JNode wrapping a map or array, enabling the use of lookup expressions to navigate a JTree rooted at that map or array	
fn:jnode-selector()	The selector property of a JNode	xs:integer ≥ 1 for an array JNode, xs:anyAtomicType for a map JNode
fn:jnode-content()	The content property of a JNode	Most coercion rules handle this automatically
fn:jnode-position()	The position property of a JNode	1 except for map/array multiple sequence values (never in JSON structures)



External Resources - i

fn:unparsed-binary()	Reads an external resource (e.g. a file) and returns its contents in binary	<i>cf file:read-binary()</i>
fn:xsd-validator()	Generates a function suitable for validating a document or element	
fn:parse-html()	Parses input as HTML returning a document node	
fn:html-doc()	Reads an external resource containing HTML and parses it	



External Resources - ii

fn:csv-to-arrays()	Parses CSV data supplied as a string, returning the results as a sequence of arrays of strings	
fn:parse-csv()	Parses CSV data, returning a record containing information about the names in the header, as well as the data itself	
fn:csv-doc()	Reads an external resource containing CSV and parses it	
fn:csv-to-xml()	Parses CSV data supplied as a string, returning the results as an XML document	
fn:invisible-xml()	Creates an Invisible XML parser for a grammar	