Functions & Operators 4.0

F&O 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

F&O 4.0

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 are 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

Arguments by keyword (following)

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

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

F&O 4.0

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
)
```

F&O 4.0

General Functions on Sequences

`fn:foot()`	The last of a sequence	*cf* **`fn:head()`**
`fn:trunk()`	All but the last	*cf* **`fn:tail()`**
`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 *cf* Javascript/Python

F&O 4.0

Comparison Functions

`fn:atomic-equal()`	(Coerced) type/value equality	≡ 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

F&O 4.0

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 *$position*
`fn:every()|` **`fn:some()`**	(Every | at least one) item satisfies a predicate	Predicate function with *$position*
`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

F&O 4.0

Example – fn:do-until()

Starting value

Current result

```
fn:do-until(
        $input     as item()*,
        $action    as fn(item()*, xs:integer) as item()*,
        $predicate as fn(item()*, xs:integer) as xs:boolean?
) as item()*
```

Stop when

Sequence position

Example
Higher-Order
do-until()

F&O 4.0

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()`**	

F&O 4.0

Example – fn:partition()

Next item

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

Partition so far

Sequence position

Example
Higher-Order
partition()
line wrapping

F&O 4.0

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 *f:is-larger(?,0)*
`fn:op()`	Generates a function to apply a binary operator to two arguments	

F&O 4.0

Numeric/Math Functions

`fn:divide-decimals()`	Divide **`xs:decimal`**s 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	

F&O 4.0

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

F&O 4.0

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	

F&O 4.0

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 *$position*
`map:empty()`	**`true`** if a map has no entries	
`map:entries()`	Returns all the key-value pairs of a map as a sequence of single-entry maps	
`map:filter()`	Selects entries from a map, returning a new map	Predicate function(*$key*, *$value*)
`map:items()`	All the values of a map, as a sequence in order	
`map:keys-where()`	Selects keys from a map	Predicate function(*$key*, *$value*)

F&O 4.0

Example – map:build()

Sequence position

```
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(*)
```

Example
Maps
map-build()

F&O 4.0

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()-*

F&O 4.0

Array Functions - i

`array:build()`	Build an array from a sequence of items	Value generating function with *$position*
`array:empty()`	**`true`** if an array has no entries	
`array:foot()`	The last member of an array	*cf* **`fn:foot()`**
`array:trunk()`	Remove the last member of an array	*cf* **`fn:trunk()`**
`array:slice()`	An array containing members of an input array selected by position	*$start*, *$end* and *$step*. *cf* **`fn:slice()`**
`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(*$value,$position*)

F&O 4.0

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 *cf* **`fn:sort-by()`**
`array:split()`	The contents of an array, as a sequence of single entry arrays	

F&O 4.0

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)

F&O 4.0

External Resources - i

`fn:unparsed-binary()`	Reads an external resource (e.g. a file) and returns its contents in binary	*cf* **`file:read-binary()`**
`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	

F&O 4.0

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	