Plexippus XPath

May 16, 2008

Contents

1 The xpath package **2**

1.1 Using XPath 2
1.2 Compiling XPath dynamically 3
1.3 Type coercion 5
1.4 The dynamic environment 7
1.5 The run-time context 8
1.6 Node sets 11
1.7 Miscellaneous 17
1.8 Other functions 19
1.9 Other classes 20
1.10 Other variables 20

2 The xpath-sys package **21**

2.1 Pipes 21
2.2 Node sets 23
2.3 Implementing environments 24
2.4 Defining extension functions 25
2.5 Miscellaneous functions 29

3 The xpattern package **31**

3.1 Utilities powered by pattern matchers 31
3.2 Compiling pattern matchers dynamically 33
3.3 Applying pattern matchers 38
3.4 Other functions 39
3.5 Other variables 40

Chapter 1

The xpath package

Plexippus XPath is an XPath 1.0 implementation for Common Lisp.

1.1 Using XPath

In this section:

- evaluate
- xpath

Almost all uses of XPath involve the `evaluate` function, which can parse, compile, and invoke XPath expressions.

`evaluate` *xpath context* `&optional` *unordered-p* [*Function*]

Arguments

xpath — an XPath expression

context — an XPath context

unordered-p — specify true to get unordered node-set

Return Values

the result of evaluating `xpath` within the `context`

Details

Evaluates an XPath expression

`xpath` can be a string, a sexpr-based XPath epression or a compiled expression. The `context` can be obtained using `make-context` . As an alternative, a node can be specifed.

If `unordered-p` is false (default) and value being returned is a `node-set` , it will be sorted using `sort-node-set` so its nodes will be in document order. If `unordered-p` is true, the order of the nodes is unspecified. Unordered mode can be significantly faster in some cases (and never slower).

See also

- make-context
- node-set
- sort-node-set

`xpath` *form* [*Macro*]

Arguments

form — a sexpr-based XPath form

Return Values

a list consisting of symbol XPATH and the `form`

Details

This macro is used to specify sexpr-based XPath expression for `evaluate`

See also

- evaluate

1.2 Compiling XPath dynamically

In this section:

- parse-xpath
- compile-xpath

`compile-xpath` allows the compilation of XPath into a closure ahead of time, so that `evaluate` only needs to invoke that closure rather than having to recompile it first.

Although `evaluate` itself already performs caching of compiled closures, explicit precompilation can aid optimizations if one call site uses multiple XPath expressions.

Explicit compilation using `compile-xpath` is also required when using custom environment classes, since `evaluate` compiles expressions using the dynamic environment only.

`parse-xpath` can be used to translate the standard string representation of XPath into a Plexippus-specific sexp representation. Both `compile-xpath` and `evaluate` accept sexps instead of strings.

`parse-xpath` *str* [*Function*]

Arguments

str — a string

Return Values

a s-expression-based XPath expression

Details

Parses a string-based XPath expression into s-expression-based one.

`compile-xpath` *xpath* `&optional` [*Function*]
(*environment* `(make-dynamic-environment *dynamic-namespaces*)`)

Arguments

xpath — an XPath expression

Return Values

a compiled XPath expression

Details

Compiles an XPath expression

The `xpath` expression is compiled using current environment if it isn't compiled yet. `xpath` can be a string, a sexpr-based XPath epression or a compiled expression. In the latter case `xpath` argument value itself is returned.

1.3 Type coercion

In this section:

- boolean-value
- string-value
- number-value
- node-set-value

These correspond to the XPath functions boolean(), string(), and number(). In addition, `node-set-value` is provided, which turns nodes into node sets.

`boolean-value` *value* [*Function*]

Arguments

value — value of an XPath-supported type or an XML node

Return Values

an XPath boolean

Details

Returns the value of XPath boolean() function.

For XML nodes returns the value of XPath boolean() function applied to the result of calling `string-value` for the specified `value`.

See also

- string-value

`string-value` *value* [*Function*]

Arguments

value — value of an XPath-supported type or an XML node

Return Values

an XPath string

Details

Returns the value of XPath number() function.

For XML nodes returns the value of `xpath-protocol:node-text` applied to the specified `value`.

See also

- xpath-protocol:node-text

`number-value` *value* [*Function*]

Arguments

value — value of an XPath-supported type or an XML node

Return Values

an XPath number

Details

Returns the value of XPath number() function.

For XML nodes returns the value of XPath number() function applied to the result of calling `string-value` for the specified `value`.

See also

- string-value

`node-set-value` *value* [*Function*]

Arguments

value — value of an XPath-supported type or an XML node

Return Values

a node set

Details

Returns the value of XPath node-set() function.

For XML nodes returns a node set consisting of the single node specified by `value`.

1.4 The dynamic environment

In this section:

- with-namespaces
- with-variables

The default enviroment used by `evaluate` is the dynamic environment, backed by information bound in dynamic variables. The following macros are used to bind these variables. They have dynamic scope. (The dynamic environment is currently not capable of dynamic declarations for variables, but can be used with extension functions that are declared globally.)

(The XPATH-SYS defined an environment protocol for user-defined environment classes.)

`with-namespaces (`&*rest* `bindings)` &*body body* [*Macro*]

Arguments

bindings — bindings in the form (PREFIX VALUE). PREFIXes and VALUEs are evaluated

Return Values

the tresult of evaluating `body`

Details

Provides namespace bindings for XPath compilation

Namespace bindings are used for compilation of XPath expressions. nil is equivalent of "" prefix. Bindings provided by this macro have dynamic scope.

`with-variables (`&*rest* `bindings)` &*body body* [*Macro*]

Arguments

bindings — bindings in the form (QNAME VALUE). QNAMEs and VALUEs are evaluated

Return Values

the tresult of evaluating `body`

Details

Provides bindings for XPath variables

Variable bindings are used for evaluation of compiled XPath expressions. Bindings provided by this macro have dynamic scope.

1.5 The run-time context

In this section:

- context
- make-context
- context-node
- context-starting-node
- context-position
- context-size

Instead of passing a node to `evaluate`, user code can construct a full context object.

The context object specifies values to be returned by position(), current(), and last().

`context` [*Class*]

Superclasses

common-lisp:standard-object, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

None

Direct Slots

position —

node —

size —

starting-node —

Details

Represents XPath context

`make-context` *node* `&optional` (*size* `1`) (*position* `1`) (*starting-node* *node*) [*Function*]

Arguments

node — an XML node

size — context size, a non-negative integer or a function without arguments returning non-negative integer

position — context position, a positive integer

Details

Makes a `context` object.

See also

- context

`context-node` *context* [*Function*]

Arguments

context — an XPath context

Return Values

an XML node

Details

Returns the context node of the XPath `context`.

`context-starting-node` *context* [*Function*]

Arguments

context — an XPath context

Return Values

an XML node

Details

Returns the node for which the whole XPath expression is evaluated.

`context-position` *context* [*Function*]

Arguments

context — an XPath context

Return Values

a positive integer

Details

Returns the current position of the XPath `context`.

`context-size` *context* [*Function*]

Arguments

context — an XPath context

Return Values

a non-negative number

Details

Returns the size of `context` If the context size was specified as a function, the result of calling that function is returned.

1.6 Node sets

In this section:

- first-node
- all-nodes
- map-node-set
- map-node-set-¿list
- do-node-set
- make-node-set-iterator
- node-set-iterator-end-p
- node-set-iterator-next
- node-set-iterator-current
- node-set-p
- node-set-empty-p
-
- list-¿node-set
- sort-node-set

Node sets are the XPath data type used to represent the results of evaluations that select multiple nodes. As sets, they never contain duplicates.

In addition to the high-level functions defined here, the XPATH-SYS package defined several low-level node set functions. Please also refer to the description there for details on node set order.

`first-node` *node-set* [*Function*]

Arguments

node-set — a `node-set`

Return Values

a `node-set` or nil

Details

Returns the first node in the `node-set` or nil if it's empty.

See also

- node-set

`all-nodes` *node-set* [*Function*]

Arguments

node-set — a `node-set`

Return Values

a list of nodes

Details

Returns all nodes of the `node-set` as a list.

See also

- node-set

`map-node-set` *func node-set* [*Function*]

Arguments

func — a function

node-set — a `node-set`

Return Values

nil

Details

Calls `func` for each node in `node-set`

The operation is performed lazily, i.e. if it's terminated via a non-local exit it doesn't necessarily cause the XPath engine to find out all nodes in the `node-set` internally.

See also

- node-set

`map-node-set->list` *func node-set* [*Function*]

Arguments

func — a function

node-set — a `node-set`

Return Values

a list

Details

Calls `func` for each node in `node-set` and conses up a list of its return values

The operation is performed lazily, i.e. if it's terminated via a non-local exit it doesn't necessarily cause the XPath engine to find out all nodes in the `node-set` internally.

See also

- node-set

`do-node-set (`*var* `node-set &optional result)` *&body body* [*Macro*]

Arguments

var — symbol, a variable name

node-set — a `node-set`

result — a form

Return Values

the result of evaluating `result`

Details

Executes `body` with `var` bound to successive nodes in `node-set`

The operation is performed lazily, i.e. if it's terminated via a non-local exit it doesn't necessarily cause the XPath engine to find out all nodes in the `node-set` internally.

Returns nil if `result` form isn't specified.

See also

- node-set

`make-node-set-iterator` *node-set* [*Function*]

Arguments

node-set — a `node-set`

Return Values

a node-set iterator

Details

Creates a node set iterator for `node-set`

Node set iterators can be used to iterate over node-sets. This can be done without causing the XPath engine to find out all their nodes and using non-local exits.

See also

- node-set

`node-set-iterator-end-p` *iterator* [*Function*]

Arguments

iterator — a node-set iterator returned by `make-node-set-iterator`

Return Values

a generalized boolean

Details

Returns true if `iterator` points to the end of its node set

See also

- make-node-set-iterator

`node-set-iterator-next` *iterator* [*Function*]

Arguments

iterator — a node-set iterator returned by `make-node-set-iterator`

Return Values

the value of `iterator`

Details

Advances `iterator` if it's not at the end of its node set, does nothing otherwise.

See also

- make-node-set-iterator

`node-set-iterator-current` *iterator* [*Function*]

Arguments

iterator — a node-set iterator returned by `make-node-set-iterator`

Return Values

a node or nil

Details

Returns current node of `iterator` or nil if it's at the end of its node set.

See also

- make-node-set-iterator

`node-set-p` *object* [*Function*]

Arguments

object — a value of any type

Return Values

a generalized boolean

Details

Returns true if `object` is a `node-set`

See also

- node-set

`node-set-empty-p` *node-set* [*Function*]

Arguments

node-set — a node-set

Return Values

a generalized boolean

Details

Returns true if `node-set` is empty

`list->node-set` *list* [*Function*]

Arguments

list — a list of nodes

Return Values

a `node-set`

Details

Makes a `node-set` from the `list` of nodes.

See also

- node-set

`sort-node-set` *node-set* [*Function*]

Arguments

node-set — a node set

Return Values

a sorted version of `node-set`

Details

Sorts the `node-set` according to document order.

1.7 Miscellaneous

In this section:

- with-plx-extensions
- xpath-error

Other useful functions, variables, and classes:

`with-plx-extensions` &*body* *body* [*Macro*]

Details

Binds plx prefix to Plexippus XPath extensions namespace.

The following functions are currently available:

```
plx:matches(string, pattern, flags?)
```

Returns true if `string` is matched by regular expression `pattern`, false otherwise. Optional `flags` specify modifiers (i, m, s). CL-PPCRE is used as regular expression engine.

```
plx:replace(string, pattern, replacement, flags?)
```

Returns `string` with all matches of regular expression `pattern` replaced with `replacement`. Optional `flags` specify modifiers (i, m, s).

`plx:current()`

Returns a `node-set` consisting of one node which was specifed as context node for expression evaluation. Analagous to `current()` function of XSLT.

`plx:generate-id(node-set?)`

Returns an alphanumeric string that uniquely identifies the first node of the `node-set` (or context node if `node-set` isn't specified) within its document. Analagous to `generate-id()` of XSLT.

See also

- node-set

navigator

`xpath-error` [*Class*]

Superclasses

common-lisp:simple-error, common-lisp:simple-condition, common-lisp:error, common-lisp:serious-condition, common-lisp:condition, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

None

Direct Slots

None

Details

The class of all xpath errors.

1.8 Other functions

`evaluate-compiled` *compiled-xpath context* `&optional` *unordered-p* [*Function*]

Arguments

compiled-xpath — a compiled XPath expression

context — an XPath context

unordered-p — specify true to get unordered node-set

Return Values

the result of evaluating `compiled-xpath` within the `context`

Details

Evaluates a compiled XPath expression returned by `compile-xpath`

The `context` can be obtained using `make-context` . As an alternative, a node can be specifed.

If `unordered-p` is false (default) and value being returned is a `node-set` , it will be sorted using `sort-node-set` so its nodes will be in document order. If `unordered-p` is true, the order of the nodes is unspecified. Unordered mode can be significantly faster in some cases (and never slower).

See also

- compile-xpath
- make-context
- node-set
- sort-node-set

`xpath-error` *fmt* `&rest` *args* [*Function*]

Arguments

fmt — format control string

args — format arguments

Details

Signals the `xpath-error` condition with specified message.

See also

- xpath-error

1.9 Other classes

`node-set` [*Class*]

Superclasses

common-lisp:standard-object, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

None

Direct Slots

ordering —

pipe —

Details

Represents an XPath node set

1.10 Other variables

`*navigator*` [*Variable*]

No documentation string. Possibly unimplemented or incomplete.

Chapter 2

The xpath-sys package

The XPATH-SYS package provides an API for extensions to Plexippus XPath.

2.1 Pipes

In this section:

- make-pipe
- pipe-head
- pipe-tail

Pipes are lazy lists, inspired by their implementation in Norvig's 'Paradigms of Artificial Intelligence Programming'.

`make-pipe` *head tail* [*Macro*]

Arguments

head — pipe head (evaluated)

tail — tail expression

Return Values

a pipe

Details

Constructs a pipe (lazy list).

The `head` expression is evaluated immediatelly. The value of `head` will be returned by `pipe-head` called for the pipe object returned by `make-pipe`. Evaluation of `tail` expression is delayed until `pipe-tail` is called for the pipe returned by this function. Evaluation of `tail` expression should produce a pipe or a list.

See also

- pipe-head
- pipe-tail

`pipe-head` *pipe* [*Function*]

Arguments

pipe — a pipe

Return Values

an object

Details

Returns the head of the `pipe`.

`pipe-tail` *pipe* [*Function*]

Arguments

pipe — a pipe

Return Values

an object

Details

Returns the tail of the list.

First time `pipe-tail` is called it causes pipe tail expression to be evaluated and remembered for later calls.

2.2 Node sets

In this section:

- make-node-set
- pipe-of

Node sets are the XPath data type used to represent the results of evaluations that select multiple nodes. As sets, they never contain duplicates. Conceptually, they are unordered, with the most important order defined on them being the document order.

As a data structure though, node sets are backed by a pipe, and the order of elements in that pipe is well-documented: By default, the pipe of returned node sets is sorted into document order. When unordered results are requested, the order is usually not specified, but in some cases, are already sorted according to the axis being queried, which is usually sorted either in document order,or in reverse document order. See `xpath:evaluate` for the `unordered` argument.

`make-node-set` *pipe* `&optional` (*ordering* *unordered*) [*Function*]

Arguments

pipe — a pipe

ordering — one of :document-order, :reverse-document-order, :unordered

Return Values

a node set

Details

Makes a node set containing nodes from the `pipe` with specified `ordering`.

`pipe-of` *node-set* [*Function*]

Arguments

node-set — a node set

Return Values

a pipe

Details

Returns the pipe that contains the elements of the `node-set`.

2.3 Implementing environments

In this section:

- environment-find-namespace
- environment-find-variable
- environment-find-function

Environments provide compilation-time configuration for XPath. An environment is a CLOS object, which is queried by the compiler using generic functions that users can implement on their own subclasses of `xpath::environment` .

The default environment class implements a 'dynamic' environment, backed by information bound in dynamic variables, so that typical uses of XPath work without special environment classes.

`environment-find-namespace` *environment prefix* [*Function*]

Arguments

environment — an XPath environment object

prefix — prefix part of a QName

Details

Returns namespace URI for specified `prefix`.

`environment-find-variable` *environment local-name uri* [*Function*]

Arguments

environment — an XPath environment object

local-name — local part of expanded-name of the function

uri — namespace URI of the function

Return Values

XPath variable "thunk"

Details

Finds an XPath variable by `local-name` and `uri`.

XPath variable is represented by a "thunk". A "thunk" is a function that takes an instance of `context` as its argument and returns the value of one of XPath types.

See also

- context

`environment-find-function` *environment local-name uri* [*Function*]

Arguments

environment — an XPath environment object

local-name — local part of expanded-name of the function

uri — namespace URI of the function

Return Values

an XPath function or nil if it cannot be found

Details

Finds an XPath function by `local-name` and `uri`.

XPath function is a Lisp function that takes zero or more "thunks" as its arguments (corresponding to XPath expressions passed as function arguments) and returns a new "thunk". A "thunk" is a function that takes an instance of `context` as its argument and returns the value of one of XPath types.

See also

- context

2.4 Defining extension functions

In this section:

- define-extension

- define-xpath-function/lazy
- define-xpath-function/eager
- define-xpath-function/single-type
- find-xpath-function

XPath defines built-in functions in the empty namespace. Using the extension API, user code can implement XPath functions addressed using other namespaces.

`define-extension` *name uri &optional documentation* [*Macro*]

Arguments

name — the name of XPath extension (a symbol)

uri — URI corresponding to XPath extension (a string)

documentation — documentation string for the XPath extension

Details

Defines an XPath extension with specified `name` and `uri`.

An XPath extension is a collection of XPath functions that are defined using one of `define-xpath-function/lazy` , `define-xpath-function/eager` or `define-xpath-function/single-type`

macros. In order to use the extension, one must bind a prefix string to its `uri` using `with-namespaces` macro.

Example:

```
(defparameter *my-namespace* "http://example.net/my-xpath-extension")
(xpath-sys:define-extension
    my-ext *my-namespace*
  "My Extension")
(xpath-sys:define-xpath-function/single-type my-ext add-quotes string (string)
  (concat "\"" string "\""))
(defun get-my-quoted-string(doc)
  (with-namespaces (("my" *my-namespace*))
    (evaluate "add-quotes(//some-element)" doc)))
```

See also

- define-xpath-function/lazy
- define-xpath-function/eager

- define-xpath-function/single-type
- with-namespaces

`define-xpath-function/lazy` *ext name args &body body* [*Macro*]

Arguments

ext — name of an XPath extension (a symbol)

name — XPath function name

args — XPath function arguments

Details

Defines an XPath function, "lazy" style.

The `body` is evaluated during compilation of XPath expressions each time the function being defined is referenced. It's passed a list of "thunks" corresponding to XPath function arguments and should return a new "thunk". A "thunk" is a function that takes an XPath `context` as argument and returns value of one of XPath types (string, boolean, number, node set).

Example:

```
(define-xpath-function/lazy my-ext my-if (v if-part else-part)
  #'(lambda (ctx)
      (if (boolean-value (funcall v ctx))
          (funcall if-part ctx)
          (funcall else-part ctx))))
```

See also

- context
- define-xpath-extension
- define-xpath-function/eager
- define-xpath-function/single-type

`define-xpath-function/eager` *ext name args &body body* [*Macro*]

Arguments

ext — name of an XPath extension (a symbol)

name — XPath function name

args — XPath function arguments

Details

Defines an XPath function, "eager" style.

The `body` is evaluated during evaluation of XPath expressions each time the function being defined is called. It's passed a list of values corresponding to XPath function arguments and should return a value of one of XPath types (string, boolean, number, node set).

Example:

```
(define-xpath-function/eager my-ext join (delim node-set)
  (reduce (lambda (a b) (concatenate 'string a delim b))
          (map-node-set->list #'string-value node-set)))
```

See also

- define-xpath-extension
- define-xpath-function/lazy
- define-xpath-function/single-type

`define-xpath-function/single-type` *ext name type args &body body* [*Macro*]

Arguments

ext — name of an XPath extension (a symbol)

name — XPath function name

args — XPath function arguments

Details

Defines an XPath function, "eager" style with automatic type conversion.

The `body` is evaluated during evaluation of XPath expressions each time the function being defined is called. It's passed a list of values corresponding to XPath function arguments and should return a value of one of XPath types (string, boolean, number, node set). Argument values are automatically converted to specified XPath `type`.

Example:

```
(xpath-sys:define-xpath-function/single-type my-ext add-quotes string (string)
  (concat "\"" string "\""))
```

See also

- define-xpath-extension
- define-xpath-function/lazy
- define-xpath-function/eager

`find-xpath-function` *local-name uri* [*Function*]

Arguments

local-name — local part of the function name

uri — namespace URI of the function

Return Values

an XPath function object

Details

Performs an XPath function lookup using standard lookup rules

All defined extensions for the namespace specified by `uri` are scanned for function with specified `local-name`.

2.5 Miscellaneous functions

In this section:

- get-node-id

Other useful functions:

`get-node-id` *node-or-node-set* [*Function*]

Arguments

node-or-node-set — a `node-set` or a single XML node

Return Values

an alphanumeric string

Details

Generates an unique identifier for the first node `node-set` (or, if a node is specified, for that node).

This function is similar to the generate-id() XSLT function, but its results are unique only within its document, whereas XSLT also prepends an ID designating the document.

See also

- node-set

Chapter 3

The xpattern package

The XPATTERN package implements pattern matching compatible with XSLT 1.0.

3.1 Utilities powered by pattern matchers

In this section:

- node-matches-p
- pattern-case
- pattern-ecase

The following convenience functions and macros use patterns. They are implemented using the lower-level functions listed below.

`node-matches-p` *node pattern-expression* [*Function*]

Arguments

node — any node implementing the XPath protocol

pattern-expression — a string or s-expression

Return Values

a boolean

Details

Determine whether `node` matches the pattern expression.

The expression is compiled using the dynamic environment.

See also

- with-namespaces
- with-variables
- pattern-case
- pattern-ecase

`pattern-case` *node &body clauses* [*Macro*]

Arguments

node — any node implementing the XPath protocol

clauses — cases of the form (expression &rest body)

Return Values

The value returned by the matching clause body, or nil.

Details

Match a node against static expressions.

Evaluates `node`, and matches them against the specified XSLT patterns. The first matching pattern will be chosen, i.e. earlier clauses have higher priority that later clauses.

Expressions are compiled using the dynamic environment.

As a special case, the last expression can be `t`, in which case it matches unconditionally.

See also

- with-namespaces
- pattern-ecase
- node-matches-p
- with-variables

`pattern-ecase` *node &rest clauses* [*Macro*]

Arguments

node — any node implementing the XPath protocol

clauses — cases of the form (expression &rest body)

Return Values

The value returned by the matching clause body.

Details

Match a node against static expressions.

Evaluates `node`, and matches them against the specified XSLT patterns. The first matching pattern will be chosen, i.e. earlier clauses have higher priority that later clauses.

Expressions are compiled using the dynamic environment.

If no clause matches, an error will be signalled.

See also

- with-namespaces
- pattern-case
- node-matches-p
- with-variables

3.2 Compiling pattern matchers dynamically

In this section:

- pattern
- pattern-value
- pattern-priority
- compute-patterns
- make-pattern-matcher
- make-pattern-matcher*

Pattern are represented as objects:

`pattern` [*Class*]

Superclasses

common-lisp:structure-object, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

None

Details

Represents a parsed XSLT pattern.

Returned by

- compute-patterns

Slot Access Functions

- pattern-value
- pattern-priority

See also

- make-pattern-matcher

`pattern-value` *instance* [*Function*]

Arguments

instance — a `pattern`

Return Values

an object

Details

Return the user-specified value that will be returned by pattern matchers if this pattern matches.

See also

- pattern
- matching-value
- matching-values

`pattern-priority` *instance* [*Function*]

Arguments

instance — a `pattern`

Return Values

an integer

Details

Return the priority of this pattern. When several patters would match the same node, the pattern matcher will only consider the patterns that have the highest priority.

See also

- pattern
- matching-value
- matching-values

Use `compute-patterns` to parse a pattern expression into pattern objects:

`compute-patterns` *expression priority value environment* [*Function*]

Arguments

expression — a string or s-expression

priority — an integer

value — an object

environment — an `environment`

Return Values

a list of `pattern` s

Details

Parse an XSLT pattern expression into one or more pattern objects.

Parses an expression, resolves its namespace-, variable-, and function-references using the specified `environment`, and creates a `pattern` object for the expression (if it does not use a union) or one `pattern` object for each sub-expression that is being joined into the union.

The specified `priority` is used as the `pattern-priority` , and the specified `value` is used as the `pattern-value` .

See also

- pattern
- pattern-priority
- pattern-value
- make-pattern-matcher*
- make-pattern-matcher

`make-pattern-matcher` builds a matcher functions from multiple pattern objects. The matcher will find the highest-priority match among them.

`make-pattern-matcher` *patterns* [*Function*]

Arguments

patterns — a list of `pattern` s

Return Values

the pattern matcher, a function

Details

Create a pattern matcher that distinguishes between multiple patterns.

This function combines several patterns, and creates a matcher function for use with `matching-value` or `matching-values` . The matcher function will compare a node against each pattern, and find the highest-priority pattern or patterns that match the node.

See also

- pattern
- matching-value
- matching-values
- compute-patterns

`make-pattern-matcher*` *expression environment* [*Function*]

Arguments

expression — a string or s-expression

environment — an `environment`

Return Values

the pattern matcher, a function

Details

Create a pattern matcher for a single pattern.

This function is a convenience wrapper around `compute-patterns` and `make-pattern-matcher` .

The resulting matcher will return T if the specified `expression` matches, or NIL if it doesn't.

See also

- compute-patterns
- make-pattern-matcher
- compute-patterns
- matching-value
- matching-values

3.3 Applying pattern matchers

In this section:

- matching-value
- matching-values

To invoke a matcher created by `make-pattern-matcher`, use `matching-value` or `matching-values`:

`matching-value` *matcher node* `&optional (`*default* `nil)` [*Function*]

Arguments

matcher — a pattern matching function

node — any node implementing the XPath protocol

default — an object

Return Values

an object

Details

Apply a pattern matcher to node, and return exactly one value.

For use with `matcher` functions that have been returned by `make-pattern-matcher` or a higher-level function like `make-pattern-matcher*` .

If exactly one pattern matches, or several patterns for the same value match, the user-specified values as determined by `pattern-value`

will be returned by this function.

If no pattern matches, `default` will be returned instead.

If more than one pattern of highest priority and different values match, an `xpath-error` will be signalled.

See also

- make-pattern-matcher
- make-pattern-matcher*
- pattern-value
- node-matches-p

- pattern-case
- pattern-ecase

`matching-values` *matcher node* [*Function*]

Arguments

matcher — a pattern matching function

node — any node implementing the XPath protocol

Return Values

an object

Details

Apply a pattern matcher to node, and return one or more matching values.

For use with `matcher` functions that have been returned by `make-pattern-matcher` or a higher-level function like `make-pattern-matcher*` .

The resulting list will contain the user-specified values as returned by `pattern-value` on the patterns for this matcher, in any order. Duplicates under `eql` will have been removed from the list.

See also

- make-pattern-matcher
- make-pattern-matcher*
- pattern-value
- node-matches-p
- pattern-case
- pattern-ecase

3.4 Other functions

`parse-pattern-expression` *str* [*Function*]

Arguments

str — a string

Return Values

a s-expression-based pattern expression

Details

Parses an XSLT pattern into an s-expression.

3.5 Other variables

`*allow-variables-in-patterns*` [*Variable*]

Details

If set to T, predicates in patterns are permitted to reference variables using $var syntax. If set to NIL, such variable references signal a compilation-time error. The default is T. Bind this variable to NIL to enable compatibility with XSLT 1.0.

See also

- compute-patterns

Index

allow-variables-in-patterns `*allow-variables-in-patterns*` variable, 40
navigator `*navigator*` variable, 20
all-nodes `all-nodes` function, 12
boolean-value `boolean-value` function, 5
compile-xpath `compile-xpath` function, 4
compute-patterns `compute-patterns` function, 35
context `context` class, 8
context-node `context-node` function, 9
context-position `context-position` function, 10
context-size `context-size` function, 10
context-starting-node `context-starting-node` function, 10
define-extension `define-extension` macro, 26
define-xpath-function/eager `define-xpath-function/eager` macro, 27
define-xpath-function/lazy `define-xpath-function/lazy` macro, 27
define-xpath-function/single-type `define-xpath-function/single-type` macro, 28
do-node-set `do-node-set` macro, 13
environment-find-function `environment-find-function` function, 25
environment-find-namespace `environment-find-namespace` function, 24
environment-find-variable `environment-find-variable` function, 24
evaluate `evaluate` function, 2
evaluate-compiled `evaluate-compiled` function, 19
find-xpath-function `find-xpath-function` function, 29
first-node `first-node` function, 11
get-node-id `get-node-id` function, 29
list-¿node-set `list-¿node-set` function, 16
make-context `make-context` function, 9
make-node-set `make-node-set` function, 23
make-node-set-iterator `make-node-set-iterator` function, 14
make-pattern-matcher `make-pattern-matcher` function, 36
make-pattern-matcher* `make-pattern-matcher*` function, 37
make-pipe `make-pipe` macro, 21
map-node-set `map-node-set` function, 12
map-node-set-¿list `map-node-set->list` function, 13
matching-value `matching-value` function, 38
matching-values `matching-values` function, 39
node-matches-p `node-matches-p` function, 37
node-set `node-set` class, 20
node-set-empty-p `node-set-empty-p` function, 16
node-set-iterator-current `node-set-iterator-current` function, 15
node-set-iterator-end-p `node-set-iterator-end-p` function, 14
node-set-iterator-next `node-set-iterator-next` function, 15
node-set-p `node-set-p` function, 15
node-set-value `node-set-value` function, 6
number-value `number-value` function, 6
parse-pattern-expression `parse-pattern-expression` function, 39
parse-xpath `parse-xpath` function, 4
pattern `pattern` class, 34

pattern-case `pattern-case` macro, 32
pattern-ecase `pattern-ecase` macro, 33
pattern-priority `pattern-priority` function, 35
pattern-value `pattern-value` function, 34
pipe-head `pipe-head` function, 22
pipe-of `pipe-of` function, 23
pipe-tail `pipe-tail` function, 22
sort-node-set `sort-node-set` function, 17
string-value `string-value` function, 5
with-namespaces `with-namespaces` macro, 7
with-plx-extensions `with-plx-extensions` macro, 17
with-variables `with-variables` macro, 7
xpath `xpath` macro, 3
xpath-error `xpath-error` class, 18
xpath-error `xpath-error` function, 19