NAME

XML::LibXML::XPathContext – XPath Evaluation

SYNOPSIS

```
my $xpc = XML::LibXML::XPathContext->new();
my $xpc = XML::LibXML::XPathContext->new($node);
$xpc->registerNs($prefix, $namespace_uri)
$xpc->unregisterNs($prefix)
$uri = $xpc->lookupNs($prefix)
$xpc->registerVarLookupFunc($callback, $data)
$data = $xpc->getVarLookupData();
$callback = $xpc->getVarLookupFunc();
$xpc->unregisterVarLookupFunc($name);
$xpc->registerFunctionNS($name, $uri, $callback)
$xpc->unregisterFunctionNS($name, $uri)
$xpc->registerFunction($name, $callback)
$xpc->unregisterFunction($name)
@nodes = $xpc->findnodes($xpath)
@nodes = $xpc->findnodes($xpath, $context_node )
$nodelist = $xpc->findnodes($xpath, $context_node )
$object = $xpc->find($xpath )
$object = $xpc->find($xpath, $context_node )
$value = $xpc->findvalue($xpath )
$value = $xpc->findvalue($xpath, $context_node )
$bool = $xpc->exists( $xpath_expression, $context_node );
$xpc->setContextNode($node)
my $node = $xpc->getContextNode;
$xpc->setContextPosition($position)
my $position = $xpc->getContextPosition;
$xpc->setContextSize($size)
my $size = $xpc->getContextSize;
$xpc->setContextNode($node)
```

DESCRIPTION

The XML::LibXML::XPathContext class provides an almost complete interface to libxml2's XPath implementation. With XML::LibXML::XPathContext, it is possible to evaluate XPath expressions in the context of arbitrary node, context size, and context position, with a user-defined namespace-prefix mapping, custom XPath functions written in Perl, and even a custom XPath variable resolver.

EXAMPLES

Namespaces

This example demonstrates `registerNs()` method. It finds all paragraph nodes in an XHTML document.

```
my $xc = XML::LibXML::XPathContext->new($xhtml_doc);
$xc->registerNs('xhtml', 'http://www.w3.org/1999/xhtml');
my @nodes = $xc->findnodes('//xhtml:p');
```

Custom XPath functions

This example demonstrates `registerFunction()` method by defining a function filtering nodes based on a Perl regular expression:

```
sub grep_nodes {
  my ($nodelist,$regexp) =  @_;
  my $result = XML::LibXML::NodeList->new;
  for my $node ($nodelist->get_nodelist()) {
    $result->push($node) if $node->textContent =~ $regexp;
  }
  return $result;
};

my $xc = XML::LibXML::XPathContext->new($node);
$xc->registerFunction('grep_nodes', \&grep_nodes);
my @nodes = $xc->findnodes('//section[grep_nodes(para,"\bsearch(ing|es)?\b")]');
```

Variables

This example demonstrates `registerVarLookup()` method. We use XPath variables to recycle results of previous evaluations:

```
sub var_lookup {
  my ($varname,$ns,$data)=@_;
  return $data->{$varname};
}

my $areas = XML::LibXML->new->parse_file('areas.xml');
my $empl = XML::LibXML->new->parse_file('employees.xml');

my $xc = XML::LibXML::XPathContext->new($empl);

my %variables = (
  A => $xc->find('/employees/employee[@salary>10000]'),
  B => $areas->find('/areas/area[district='Brooklyn']/street'),
);

# get names of employees from $A working in an area listed in $B
$xc->registerVarLookupFunc(\&var_lookup, \%variables);
my @nodes = $xc->findnodes('$A[work_area/street = $B]/name');
```

METHODS

new

```
my $xpc = XML::LibXML::XPathContext->new();
```

Creates a new XML::LibXML::XPathContext object without a context node.

```
my $xpc = XML::LibXML::XPathContext->new($node);
```

Creates a new XML::LibXML::XPathContext object with the context node set to `$node`.

registerNs

```
$xpc->registerNs($prefix, $namespace_uri)
```

Registers namespace `$prefix` to `$namespace_uri`.

unregisterNs

```
$xpc->unregisterNs($prefix)
```

Unregisters namespace `$prefix`.

lookupNs

```
$uri = $xpc->lookupNs($prefix)
```

Returns namespace URI registered with `$prefix`. If `$prefix` is not registered to any namespace URI returns `undef`.

registerVarLookupFunc

```
$xpc->registerVarLookupFunc($callback, $data)
```

Registers variable lookup function `$prefix`. The registered function is executed by the XPath engine each time an XPath variable is evaluated. It takes three arguments: `$data`, variable name, and variable ns-URI and must return one value: a number or string or any `XML::LibXML::` object that can be a result of findnodes: Boolean, Literal, Number, Node (e.g. Document, Element, etc.), or NodeList. For convenience, simple (non-blessed) array references containing only XML::LibXML::Node objects can be used instead of an XML::LibXML::NodeList.

getVarLookupData

```
$data = $xpc->getVarLookupData();
```

Returns the data that have been associated with a variable lookup function during a previous call to `registerVarLookupFunc`.

getVarLookupFunc

```
$callback = $xpc->getVarLookupFunc();
```

Returns the variable lookup function previously registered with `registerVarLookupFunc`.

unregisterVarLookupFunc

```
$xpc->unregisterVarLookupFunc($name);
```

Unregisters variable lookup function and the associated lookup data.

registerFunctionNS

```
$xpc->registerFunctionNS($name, $uri, $callback)
```

Registers an extension function `$name` in `$uri` namespace. `$callback` must be a CODE reference. The arguments of the callback function are either simple scalars or `XML::LibXML::*` objects depending on the XPath argument types. The function is responsible for checking the argument number and types. Result of the callback code must be a single value of the following types: a simple scalar (number, string) or an arbitrary `XML::LibXML::*` object that can be a result of findnodes: Boolean, Literal, Number, Node (e.g. Document, Element, etc.), or NodeList. For convenience, simple (non-blessed) array references containing only XML::LibXML::Node objects can be used instead of a XML::LibXML::NodeList.

unregisterFunctionNS

```
$xpc->unregisterFunctionNS($name, $uri)
```

Unregisters extension function `$name` in `$uri` namespace. Has the same effect as passing `undef` as `$callback` to registerFunctionNS.

registerFunction

```
$xpc->registerFunction($name, $callback)
```

Same as `registerFunctionNS` but without a namespace.

unregisterFunction

```
$xpc->unregisterFunction($name)
```

Same as `unregisterFunctionNS` but without a namespace.

findnodes

```
@nodes = $xpc->findnodes($xpath)

@nodes = $xpc->findnodes($xpath, $context_node )

$nodelist = $xpc->findnodes($xpath, $context_node )
```

Performs the xpath statement on the current node and returns the result as an array. In scalar context, returns an XML::LibXML::NodeList object. Optionally, a node may be passed as a second argument to set the context node for the query.

The xpath expression can be passed either as a string, or as a XML::LibXML::XPathExpression object.

find

```
$object = $xpc->find($xpath )

$object = $xpc->find($xpath, $context_node )
```

Performs the xpath expression using the current node as the context of the expression, and returns the result depending on what type of result the XPath expression had. For example, the XPath `1 * 3 + 52` results in an XML::LibXML::Number object being returned. Other expressions might return a XML::LibXML::Boolean object, or a XML::LibXML::Literal object (a string). Each of those objects uses Perl's overload feature to ''do the right thing'' in different contexts. Optionally, a node may be passed as a second argument to set the context node for the query.

The xpath expression can be passed either as a string, or as a XML::LibXML::XPathExpression object.

findvalue

```
$value = $xpc->findvalue($xpath )

$value = $xpc->findvalue($xpath, $context_node )
```

Is exactly equivalent to:

```
$xpc->find( $xpath, $context_node )->to_literal;
```

That is, it returns the literal value of the results. This enables you to ensure that you get a string back from your search, allowing certain shortcuts. This could be used as the equivalent of <xsl:value−of select=''some_xpath''/>. Optionally, a node may be passed in the second argument to set the context node for the query.

The xpath expression can be passed either as a string, or as a XML::LibXML::XPathExpression object.

exists

```
$bool = $xpc->exists( $xpath_expression, $context_node );
```

This method behaves like *findnodes*, except that it only returns a boolean value (1 if the expression matches a node, 0 otherwise) and may be faster than *findnodes*, because the XPath evaluation may stop early on the first match (this is true for libxml2 >= 2.6.27).

For XPath expressions that do not return node-set, the method returns true if the returned value is a non-zero number or a non-empty string.

setContextNode

```
$xpc->setContextNode($node)
```

Set the current context node.

getContextNode

```
my $node = $xpc->getContextNode;
```

Get the current context node.

setContextPosition

```
$xpc->setContextPosition($position)
```

Set the current context position. By default, this value is −1 (and evaluating XPath function `position()` in the initial context raises an XPath error), but can be set to any value up to context size. This usually only serves to cheat the XPath engine to return given position when `position()` XPath function is called. Setting this value to −1 restores the default behavior.

getContextPosition

```
my $position = $xpc->getContextPosition;
```

Get the current context position.

setContextSize

```
$xpc->setContextSize($size)
```

Set the current context size. By default, this value is −1 (and evaluating XPath function `last()` in the initial context raises an XPath error), but can be set to any non-negative value. This usually only serves to cheat the XPath engine to return the given value when `last()` XPath function is called. If context size is set to 0, position is automatically also set to 0. If context size is positive, position is automatically set to 1. Setting context size to −1 restores the default behavior.

getContextSize

```
my $size = $xpc->getContextSize;
```

Get the current context size.

setContextNode

```
$xpc->setContextNode($node)
```

Set the current context node.

BUGS AND CAVEATS

XML::LibXML::XPathContext objects *are* reentrant, meaning that you can call methods of an XML::LibXML::XPathContext even from XPath extension functions registered with the same object or from a variable lookup function. On the other hand, you should rather avoid registering new extension functions, namespaces and a variable lookup function from within extension functions and a variable lookup function, unless you want to experience untested behavior.

AUTHORS

Ilya Martynov and Petr Pajas, based on XML::LibXML and XML::LibXSLT code by Matt Sergeant and Christian Glahn.

HISTORICAL REMARK

Prior to XML::LibXML 1.61 this module was distributed separately for maintenance reasons.

AUTHORS

Matt Sergeant, Christian Glahn, Petr Pajas

VERSION

2.0134

COPYRIGHT

LICENSE