NAME

XML::LibXML::Text – XML::LibXML Class for Text Nodes

SYNOPSIS

```
use XML::LibXML;
# Only methods specific to Text nodes are listed here,
# see the XML::LibXML::Node manpage for other methods

$text = XML::LibXML::Text->new( $content );
$nodedata = $text->data;
$text->setData( $text_content );
$text->substringData($offset, $length);
$text->appendData( $somedata );
$text->insertData($offset, $string);
$text->deleteData($offset, $length);
$text->deleteDataString($remstring, $all);
$text->replaceData($offset, $length, $string);
$text->replaceDataString($old, $new, $flag);
$text->replaceDataRegEx( $search_cond, $replace_cond, $reflags );
```

DESCRIPTION

Unlike the DOM specification, XML::LibXML implements the text node as the base class of all character data node. Therefore there exists no CharacterData class. This allows one to apply methods of text nodes also to Comments and CDATA-sections.

METHODS

The class inherits from XML::LibXML::Node. The documentation for Inherited methods is not listed here.

Many functions listed here are extensively documented in the DOM Level 3 specification (<http://www.w3.org/TR/DOM–Level–3–Core/>). Please refer to the specification for extensive documentation.

new

```
$text = XML::LibXML::Text->new( $content );
```

The constructor of the class. It creates an unbound text node.

data

```
$nodedata = $text->data;
```

Although there exists the `nodeValue` attribute in the Node class, the DOM specification defines data as a separate attribute. `XML::LibXML` implements these two attributes not as different attributes, but as aliases, such as `libxml2` does. Therefore

```
$text->data;
```

and

```
$text->nodeValue;
```

will have the same result and are not different entities.

setData($string)

```
$text->setData( $text_content );
```

This function sets or replaces text content to a node. The node has to be of the type "text", "cdata" or "comment".

substringData($offset,$length)

```
$text->substringData($offset, $length);
```

Extracts a range of data from the node. (DOM Spec) This function takes the two parameters `$offset` and `$length` and returns the sub-string, if available.

If the node contains no data or `$offset` refers to an non-existing string index, this function will return *undef*. If `$length` is out of range `substringData` will return the data starting at `$offset` instead of causing an error.

appendData($string)

```
$text->appendData( $somedata );
```

Appends a string to the end of the existing data. If the current text node contains no data, this function has the same effect as `setData`.

insertData($offset,$string)

```
$text->insertData($offset, $string);
```

Inserts the parameter `$string` at the given `$offset` of the existing data of the node. This operation will not remove existing data, but change the order of the existing data.

The `$offset` has to be a positive value. If `$offset` is out of range, `insertData` will have the same behaviour as `appendData`.

deleteData($offset, `$length`)

```
$text->deleteData($offset, $length);
```

This method removes a chunk from the existing node data at the given offset. The `$length` parameter tells, how many characters should be removed from the string.

deleteDataString($string, [$all])

```
$text->deleteDataString($remstring, $all);
```

This method removes a chunk from the existing node data. Since the DOM spec is quite unhandy if you already know `which` string to remove from a text node, this method allows more perlish code :)

The functions takes two parameters: *`$string`* and optional the *`$all`* flag. If `$all` is not set, *undef* or *0*, `deleteDataString` will remove only the first occurrence of `$string`. If `$all` is *TRUE*`deleteDataString` will remove all occurrences of *`$string`* from the node data.

replaceData($offset, `$length`, `$string`)

```
$text->replaceData($offset, $length, $string);
```

The DOM style version to replace node data.

replaceDataString($oldstring, `$newstring`, [$all])

```
$text->replaceDataString($old, $new, $flag);
```

The more programmer friendly version of **replaceData()** :)

Instead of giving offsets and length one can specify the exact string (*`$oldstring`*) to be replaced. Additionally the *`$all`* flag allows one to replace all occurrences of *`$oldstring`*.

replaceDataRegEx(`$search_cond`, `$replace_cond`, `$reflags`)

```
$text->replaceDataRegEx( $search_cond, $replace_cond, $reflags );
```

This method replaces the node's data by a `simple` regular expression. Optional, this function allows one to pass some flags that will be added as flag to the replace statement.

NOTE: This is a shortcut for

```
my $datastr = $node->getData();
 $datastr =~ s/somecond/replacement/g; # 'g' is just an example for any flag
 $node->setData( $datastr );
```

This function can make things easier to read for simple replacements. For more complex variants it is recommended to use the code snippet above.

AUTHORS

Matt Sergeant, Christian Glahn, Petr Pajas

VERSION

2.0134

COPYRIGHT

LICENSE