

# Generated Documentation



# Contents

<a href="#">Package default Classes</a>	1
<a href="#">Class xml_line</a>	1
<a href="#">Var \$action_func</a>	2
<a href="#">Var \$akt_name</a>	2
<a href="#">Var \$breakset</a>	2
<a href="#">Var \$csv_buffer</a>	2
<a href="#">Var \$datafound</a>	2
<a href="#">Var \$data_buffer</a>	2
<a href="#">Var \$elc</a>	2
<a href="#">Var \$elc_path_array</a>	3
<a href="#">Var \$el_counter</a>	3
<a href="#">Var \$error</a>	3
<a href="#">Var \$file</a>	3
<a href="#">Var \$fill</a>	3
<a href="#">Var \$fillattr</a>	4
<a href="#">Var \$found_data</a>	4
<a href="#">Var \$found_data_part</a>	4
<a href="#">Var \$func_counter</a>	4
<a href="#">Var \$highlight</a>	4
<a href="#">Var \$ignore_default_data</a>	4
<a href="#">Var \$last_nowrite_line</a>	5
<a href="#">Var \$level</a>	5
<a href="#">Var \$ls</a>	5
<a href="#">Var \$of</a>	5
<a href="#">Var \$outfile</a>	5
<a href="#">Var \$output_action</a>	6
<a href="#">Var \$parser</a>	6
<a href="#">Var \$path_array</a>	6
<a href="#">Var \$procdata</a>	6
<a href="#">Var \$table_result</a>	7
<a href="#">Var \$temp_attr_buffer</a>	7
<a href="#">Var \$testbuffer</a>	7
<a href="#">Var \$write_flag</a>	7
<a href="#">Var \$xml_buffer</a>	7
<a href="#">Constructor xml_line</a>	7
<a href="#">Method add_attributes</a>	8
<a href="#">Method append_data</a>	8
<a href="#">Method change_attribute_value</a>	9
<a href="#">Method change_data</a>	9
<a href="#">Method change_data_by_id</a>	10
<a href="#">Method defaultData</a>	10
<a href="#">Method delete_attribute</a>	10
<a href="#">Method delete_content</a>	11

<a href="#">Method delete element</a>	11
<a href="#">Method find data by id</a>	12
<a href="#">Method get attribute</a>	12
<a href="#">Method get data</a>	13
<a href="#">Method get element counter array</a>	14
<a href="#">Method get output</a>	14
<a href="#">Method get record</a>	14
<a href="#">Method insert before end</a>	15
<a href="#">Method insert element</a>	15
<a href="#">Method insert element after element</a>	16
<a href="#">Method replace attribute</a>	16
<a href="#">Method replace content</a>	17
<a href="#">Method replace element</a>	17
<a href="#">Method set action</a>	17
<a href="#">Method set element counter</a>	18
<a href="#">Method xml stream</a>	18
<b>Appendices</b>	19
<a href="#">Appendix A - Class Trees</a>	20
<a href="#">default</a>	20
<a href="#">Appendix D - Todo List</a>	21

# Package default Classes

## Class xml\_line

[line 30]

### XML-LINE Version 0.2.11 == 09.04.2004

PHP-Klasse zum Abfragen und Ändern von XML-Dateien (C) 2004 Peter Bieling, pb@media-palette.de Einige Codezeilen, für den Aufruf der Parserfunktionen sind dem PHP-Manual entnommen.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

- **Package** default
- **TODO** (!) Datei-Umbenennungsfunktionen, falls Zieldatei die Quelldatei ersetzen soll. -(!!!) Quell-XML wird nur aus einer Datei nicht aus String angenommen. -(!!!) Kopieren von ganzen XML-Abschnitten oder Elementen ist noch nicht möglich
- **Author** Peter Bieling < [pb@media-palette.de](mailto:pb@media-palette.de) >
- **Version** xml-line.php v 0.2.8

#### xml\_line::\$action\_func

*array = array() [line 154]*

**Name der aufzurufenden Funktion. Funktionen werden in einem Array gespeichert.**

`$action_func[0]` = Funktionsname    `$action_func[1]` = Parameterliste (array)  
`$action_func[2]` = Kommando    `$action_func[3]` = Ersetzungsstring

#### xml\_line::\$akt\_name

*string = "" [line 57]*

**Name des aktuellen Elements**

#### xml\_line::\$breakset

*boolean = false [line 284]*

**Internes Flag für die Highlight-Ausgabe. Merkt sich, ob vorher ein Zeilenumbruch gesetzt wurde.**

#### xml\_line::\$csv\_buffer

*string = "" [line 86]*

**Speichert die CSV-Ausgabe, sofern sie (statt XML)gefordert ist, bzw. nicht in eine Datei geschrieben wird. Experimentell!!!**

#### xml\_line::\$datafound

*boolean = false [line 277]*

**Interner Schalter für die Highlight-Ausgabe. Merkt sich, ob Daten ausgegeben wurden**

#### xml\_line::\$data\_buffer

*string = "" [line 135]*

**Expat liefert die Daten häufig in mehreren Stücken    Hier wird das Ergebnis zwischengespeichert, bis der Data-String komplett ist**

#### xml\_line::\$elc

*array = array(array()) [line 217]*

**element-level-counter, enthält die Zählerwerte, der aktuellen Elemente.**

**[level]["elementname"]**

**xml\_line::\$elc\_path\_array**

*array = array() [line 224]*

**Kurzform des aktuellen Elementpfads (3,2,8) ohne Bezug auf die Elemente**

**xml\_line::\$el\_counter**

*mixed = "" [line 210]*

**Element-Counter. Läuft nur mit, wenn er gesetzt wird. Kann für statistische Zwecke benutzt werden** `$el_counter[elname] = array($countall, $level => $countonthislevel`

- **Var** (string or array)

**xml\_line::\$error**

*array = array() [line 232]*

**Error - noch sträflich vernachlässigt ;-( Soll die Fehlermeldungen aufnehmen, die man dann abrufen kann. (noch in der Entwicklung)**

**xml\_line::\$file**

*string = [line 43]*

**Der Name der XML-Input-Datei**

**xml\_line::\$fill**

*array = array() [line 94]*

**Speichert Position als key und einen String als value für das Ersetzen oder Einsetzen von Text oder XML .Jede Funktion nutzt diesen Speicher selbst bei Bedarf.**

- **Var** `join("", $this->ls)=> Insertstring`

**xml\_line::\$fillattr**

*array = array() [line 102]*

**Speichert einzusetzende Attribute in einem Hash (name=>value), wenn ein oder mehrere Attribute hinzugefügt werden sollen**

**xml\_line::\$found\_data**

*array = array() [line 194]*

**Speichert die gefundenen Daten, Pfadinformationen und Attribute in einem Array. Art der Rückgabe kann sich noch ändern, am besten mit print\_r() selbst ansehen. ;-)**

**xml\_line::\$found\_data\_part**

*array = array() [line 202]*

**Zähler für Textstücke innerhalb eines Elements mit weiteren untergeordneten Elementen, läuft nur mit, wenn er benötigt wird und das passende Element gefunden wurde.**

- **Var** (key aus Prozessnr-pathstring-und-levelstring)

**xml\_line::\$func\_counter**

*integer = 0 [line 162]*

**Zählt die gesetzten Funktionen. Ermöglicht das Auseinanderhalten der verschiedenen Prozesse und die Zuordnung der Rückgabewerte.**

**xml\_line::\$highlight**

*boolean = false [line 270]*

**Interner Schalter für die Highlight-Ausgabe**

**xml\_line::\$ignore\_default\_data**

*boolean = false [line 263]*

## XML-Header und Kommentare ausgeben oder nicht. Wird über Methodenaufruf verändert

### xml\_line::\$last\_nowrite\_line

```
string = "" [line 71]
```

Markiert das Ende von Bereichen, die nicht geschrieben (also de facto gelöscht) werden.

- `Var join("", $this->Is);`

### xml\_line::\$level

```
integer = 0 [line 127]
```

Gibt den aktuellen Level der Elementhierarchie an

### xml\_line::\$Is

```
array = array('type' => "", // S (START) D = (DATA)  
              // E (END) EOS (End of Start) A = (Attribute) X=(Doctype und Kommentare)  
              'level' => 0, // level of XML-Tree  
              'pathstring' => "", // element1/element2  
              'attrn' => "", // attributename if type='A'  
              'data' => "" // data of element or attribute  
            ) [line 180]
```

Enthält die wichtigsten Informationen der aktuellen XML-Position.

### xml\_line::\$of

```
integer = [line 246]
```

## Outputfilehandler

- `Var resource_id`

### xml\_line::\$outfile

```
string = "" [line 50]
```



## Der Name der Output-Datei

`xml_line::$output_action`

*mixed = "" [line 120]*

**Gibt an, ob bzw. was ausgegeben werden soll.**

xml, csv, hixml oder Leerstring

`xml_line::$parser`

*integer = [line 239]*

## Expat-Variable

- **Var** resource

`xml_line::$path_array`

*array = array() [line 143]*

**Enthält alle Elemente des aktuellen Elementpfads einzeln als Array-Werte.**

Damit kann dieser problemlos ergänzt oder gekürzt werden

- **Var** (root,level1,level2)

`xml_line::$procddata`

*array = array(array()) [line 173]*

**Speicher für zusätzliche Prozessinformationen, 'datapart' z.B. ([1]['datapart']) merkt sich bei gemischten Inhalten, welcher Datenabschnitt ausgegeben werden soll 'limit' [1]['limit']=3 bedeutet: 3 Treffer werden gesucht, danach steigt die Funktion aus. (runterzählen bis auf 0)**

#### xml\_line::\$table\_result

*mixed = array() [line 256]*

**Soll bei Datensatzabfrage, das Ergebnis speichern: \$table\_result[Funktionsnr][counter][key] => "value" key ist die "Spaltenüberschrift" = Name des Elements.**

Ist das Element untergeordnet ist key 'parent/child' Attribute werden ebenfalls in die Tabelle aufgenommen: 'element@attribut1'

#### xml\_line::\$temp\_attr\_buffer

*array = array() [line 112]*

**Speichert die Attributhashes solange das Element aktuell ist. (Key ist der "pathname") Achtung: Wird wegen neuer Funktion get\_record zur Zeit nicht gelöscht. (todo) Wichtig für Einfügeoperationen, damit auch beim Endelement noch bekannt ist, welche Attribute das Element hat.**

#### xml\_line::\$testbuffer

*string = [line 36]*

**Variable für Testausgaben.**

#### xml\_line::\$write\_flag

*boolean = true [line 64]*

**Ausgabe wird nur geschrieben, wenn true gesetzt ist.**

#### xml\_line::\$xml\_buffer

*string = "" [line 78]*

**Speichert die XML-Ausgabe, sofern sie gefordert ist, bzw. nicht in eine Datei geschrieben wird**

Constructor function xml\_line::xml\_line(\$file, [\$out = ""], [\$outfile = ""]) [line 294]

#### **Function Parameters:**

- *string* **\$file** XML-Quelldatei
- *string* **\$out** optional Ausgabeformat (z.B. xml oder hixml)

- *string* **\$outfile** optional Ausgabedatei

## Constructor.

- **Access** public

function xml\_line::add\_attributes(\$arg, \$insert) *[line 882]*

### **Function Parameters:**

- *array* **\$arg**
- *string* **\$insert**

**Fügt Attribute in einem Hash hinzu. Nutzt ebenfalls die Startelementmarke und nicht die Attributmarke, weil direkt auf das Expat-Attributarray zugegriffen wird, bevor dieses weiter verarbeitet wird.**

(Das muss man sich klar machen, wenn zunächst Attribute zugefügt und gleichzeitig nach gleichnamigen Attributen gesucht wird!)

- **See** [xml\\_line::get\\_data\(\)](#)
- **Access** public

function xml\_line::append\_data(\$arg, \$value) *[line 786]*

### **Function Parameters:**

- *array* **\$arg**
- *string* **\$value**

**Hängt den neuen Inhalt an den Wert an** Zusätzlich zur Parameterliste \$arg wird ein Wert übergeben, der den gefundenen String erweitert.

- **Access public**
- See [xml\\_line::get\\_data\(\)](#)

function xml\_line::change\_attribute\_value(\$arg, \$value) [*line 653*]

**Function Parameters:**

- *array* **\$arg**
- *mixed* **\$value** (string or integer)

**Setzt eine Änderungsabfrage.** Funktioniert ähnlich wie `get_attribute`. Die dort übergebenen Parameter werden hier als Array übergeben. Als zweiter Parameter kommt der String oder die Zahl, die den alten Wert ersetzt. Die Funktion wird sooft angewendet, wie sie auf die Abfrage trifft. Der entfernte Wert wird nicht gespeichert.

- See [xml\\_line::get\\_attribute\(\)](#)
- **Access public**

function xml\_line::change\_data(\$arg, \$value) [*line 772*]

**Function Parameters:**

- *array* **\$arg**
- *string* **\$value**

**Ersetzt den gefundenen String durch einen neuen.**

Zusätzlich zur Parameterliste \$arg wird ein Wert übergeben, der den gefundenen String ersetzt.



- *array* **\$arg**

**Löscht das angegebene Attribut.**

- **Access** public
- **See** [xml\\_line::get\\_attribute\(\)](#)

`function xml_line::delete_content([$limit = 0], [$path = ""], [$pattern = ""], [$attrn = ""], [$elcount = ""]) [line 832]`  
**Function Parameters:**

- *integer* **\$limit** optional Anzahl der gesuchten Treffer. Danach wird die Funktion beendet
- *string* **\$path** optional root/element1/element2 (kann unvollständig sein, z.B. element2)
- *string* **\$pattern** optional String, der in den gesuchten Daten enthalten sein muss
- *mixed* **\$attrn** optional array(name=>value, name2=>value2) Attribut(e) müssen enthalten sein.
- *string* **\$elcount** optional "1-6-2" oder "1-4-6:2" oder "1-6-{3 to 5}" oder "1-6-{3,5,8}"

**Löscht den Inhalt des gesuchten Elements, indem die Ausgabe unterbrochen wird.**

- **See** [xml\\_line::get\\_attribute\(\)](#)
- **Access** public

`function xml_line::delete_element([$limit = 0], [$path = ""], [$pattern = ""], [$attrn = ""], [$elcount = ""]) [line 802]`  
**Function Parameters:**

- *integer* **\$limit** optional Anzahl der gesuchten Treffer. Danach wird die Funktion beendet

- *string* **\$path** optional root/element1/element2 (kann unvollständig sein, z.B. element2)
- *string* **\$pattern** optional String, der in den gesuchten Daten enthalten sein muss
- *mixed* **\$attrn** optional array(name=>value, name2=>value2) Attribut(e) müssen enthalten sein.
- *string* **\$elcount** optional "1-6-2" oder "1-4-6:2" oder "1-6-{3 to 5}" oder "1-6-{3,5,8}"

**Löscht das gesuchte Element und seinen Inhalt, indem die Ausgabe unterbrochen wird.**

- See [xml\\_line::get\\_attribute\(\)](#)
- **Access** public

```
function xml_line::find_data_by_id($value, [$idname = "id"]) [line 924]
```

**Function Parameters:**

- *string* **\$value**
- *string* **\$idname** optional (anderer Attributname, wenn nicht "id")

**Findet und liefert den Elementinhalt mit einem bestimmten id-Attribut, z.B. find\_data\_by\_id(15); <Kiste id="1">Schrauben</Kiste>**

Shorthand für get\_data(). Heißt "id" nicht "id" sondern "nr", kann der 2. Parameter gesetzt werden. Damit wird aus der Funktion eigentlich ein find\_data\_by\_attribute\_name\_value.

- **Access** public

```
function xml_line::get_attribute([$limit = 0], [$path = ""], [$pattern = ""], [$attrn = ""], [$elcount = ""]) [line 636]
```

**Function Parameters:**

- *integer* **\$limit** optional 0 bedeutet: kein Limit
- *string* **\$path** optional
- *string* **\$pattern** optional
- *string* **\$attrn** optional
- *string* **\$elcount** optional

**Setzt die Abfrage für die Werte des Attributs oder der Attribute, die der Vorgabe entsprechen. Ohne Parameter werden alle Werte in das Ergebnis-Array geschrieben.**

- **Access** public

```
function xml_line::get_data([$limit = 0], [$path = ""], [$pattern = ""], [$attrn = ""], [$elcount = ""]) [line 740]
```

**Function Parameters:**

- *integer* **\$limit** optional Anzahl der gesuchten Treffer. 0 = keine Einschränkung
- *string* **\$path** optional root/element1/element2 (kann unvollständig sein, z.B. element2)
- *string* **\$pattern** optional String, der in den gesuchten Daten enthalten sein muss
- *mixed* **\$attrn** optional array(name=>value, name2=>value2) Attribut(e) müssen enthalten sein.
- *string* **\$elcount** optional "1-6-2" oder "1-4-6:2" oder "1-6-{3 to 5}" oder "1-6-{3,5,8}"

**Ruft set\_action auf und übergibt das Kommando "find\_data" und die Suchkriterien als Parameter.**

set\_action wird bei fast jedem Event des Expat-Parsers aufgerufen, und ruft wiederum die funktion find\_data (und die übrigen gesetzten Funktionen) auf.



- **Access** public

function xml\_line::get\_element\_counter\_array() [*line 1359*]

**Experimentell.**

Liefert das Array zurück, in dem die Elemente gezählt werden. Kann für statistische Zwecke genutzt werden. Die Funktion setzt voraus, dass der Zähler gesetzt wurde. Anzeige im Browser z.B. mit `print_r(get_element_counter_array())`

- **Access** public

function xml\_line::get\_output() [*line 1339*]

**Liefert den Output als String, wenn diese Variante gewählt wurde.**

Es wird geprüft, ob entweder der XML- oder der CSV-Buffer Inhalt hat.

- **Access** public

function xml\_line::get\_record([\$limit = 0], [\$path = ""], [\$pattern = ""], [\$attrn = ""], [\$elcount = ""]) [*line 758*]

**Function Parameters:**

- *integer* **\$limit**
- *string* **\$path**
- *string* **\$pattern**
- *mixed* **\$attrn**
- *string* **\$elcount**

**Übergibt das Kommando, alle Elemente innerhalb dieses Elementes als Ergebnis zurückzuliefern.**

Alle enthaltenen Elemente werden geliefert, in dem ein Zeiger gesetzt wird, der erst

wieder entfernt wird, wenn das Schlusstag gefunden wurde.

- **Access** public
- **See** [xml\\_line::get\\_data\(\)](#)

function xml\_line::insert\_before\_end(\$arg, \$insert) [*line 908*]

**Function Parameters:**

- *array* **\$arg**
- *string* **\$insert**

**Fügt einen String vor dem End-Tag eines Elements ein.**

- **See** [xml\\_line::get\\_data\(\)](#)
- **Access** public

function xml\_line::insert\_element(\$arg, \$insert) [*line 866*]

**Function Parameters:**

- *array* **\$arg**
- *string* **\$insert**

**Fügt Element(e) oder XML-Code an der gesuchten Stelle ein.**

Der Inhalt wird direkt hinter dem Start-Tag eingesetzt. Man kann dadurch auch den Zwischenraum zwischen zwei Starttags füllen, selbst wenn dort kein Leerzeichen ist. **worttext**</p>. Eingefügt wird so: <p>neuer Text**worttext**</p> Es können auch leere Elemente gefüllt werden: <textarea></textarea> <textarea>Text</textarea>

- See [xml\\_line::get\\_data\(\)](#)
- Access public

function xml\_line::insert\_element\_after\_element(\$arg, \$insert) [*line 895*]

**Function Parameters:**

- array **\$arg**
- string **\$insert**

**Fügt XML-Code oder einen String hinter dem End-Tag eines Elements ein.**

- See [xml\\_line::get\\_data\(\)](#)
- Access public

function xml\_line::replace\_attribute(\$arg, \$newattr) [*line 678*]

**Function Parameters:**

- array **\$arg**
- array **\$newattr**

**Ersetzt das angegebene Attribut durch ein anderes.**

- Access public
- See [xml\\_line::get\\_attribute\(\)](#)

function xml\_line::replace\_content(\$arg, \$replace) *[line 847]*

**Function Parameters:**

- *array* **\$arg**
- *string* **\$replace**

**== replace\_content** Ist identisch mit **delete\_content**. Zusätzlich werden Daten in einen Speicher (**fill**) geschrieben und dieser beim Erreichen der Marke in die Ausgabe eingefügt.

- See [xml\\_line::get\\_data\(\)](#)
- Access public

function xml\_line::replace\_element(\$arg, \$replace) *[line 816]*

**Function Parameters:**

- *array* **\$arg**
- *string* **\$replace**

**Ist identisch mit delete\_element**. Zusätzlich werden Daten in einen Speicher (**fill**) geschrieben und dieser beim Erreichen der Marke in die Ausgabe eingefügt.

- See [xml\\_line::get\\_data\(\)](#)
- Access public

function xml\_line::set\_action(\$func, [\$arg = ""], [\$command = ""], [\$value = ""]) *[line 323]*

**Function Parameters:**

- *\$func* **\$func** string
- *\$arg* **\$arg** mixed (string or array)
- *\$command* **\$command** string
- *\$value* **\$value** mixed (string or array)

**Setzt den Namen der Funktion, die auf den Stream zugreift und übergibt eine Parameterliste oder -string**

function xml\_line::set\_element\_counter([\$value = true]) [*line 346*]

**Function Parameters:**

- *boolean* **\$value** optional

**Setzt den Elementcounter. Dadurch ist die Adressierung paralleler Elemente möglich.**

- **Access** public

array function xml\_line::xml\_stream() [*line 396*]

**Startet den Parser und gibt die gefundenen Daten oder false zurück.**

- **Access** public

# Appendices

# Appendix A - Class Trees

Package default

xml\_line

- [xml\\_line](#)

# Appendix D - Todo List

## In Package default

In [xml\\_line](#)

- (!) Datei-Umbenennungsfunktionen, falls Zielfile die Quelldatei ersetzen soll. -(!!!) Quell-XML wird nur aus einer Datei nicht aus String angenommen. -(!!!) Kopieren von ganzen XML-Abschnitten oder Elementen ist noch nicht möglich



# Index

## C

[constructor xml\\_line::xml\\_line\(\)](#) . . . . . 7  
*Constructor.*

## X

[xml\\_line::defaultData\(\)](#) . . . . . 10  
*Sorgt für die Ausgabe von Kommentaren und Doctype.*

[xml\\_line::change\\_data\\_by\\_id\(\)](#) . . . . . 10  
*Ersetzt den Elementwert eines Elements mit einem bestimmten id-Attribut,  
z.B. `change_data_by_id(15,"Nägel");` <Kiste  
`id="1">Schrauben</Kiste>`*

[xml\\_line::delete\\_attribute\(\)](#) . . . . . 10  
*Löscht das angegebene Attribut.*

[xml\\_line::delete\\_content\(\)](#) . . . . . 11  
*Löscht den Inhalt des gesuchten Elements, indem die Ausgabe unterbrochen wird.*

[xml\\_line::find\\_data\\_by\\_id\(\)](#) . . . . . 12  
*Findet und liefert den Elementinhalt mit einem bestimmten id-Attribut,  
z.B. `find_data_by_id(15);` <Kiste `id="1">Schrauben</Kiste>`*

[xml\\_line::delete\\_element\(\)](#) . . . . . 11  
*Löscht das gesuchte Element und seinen Inhalt, indem die Ausgabe unterbrochen wird.*

[xml\\_line::change\\_data\(\)](#) . . . . . 9  
*Ersetzt den gefundenen String durch einen neuen.*

[xml\\_line::change\\_attribute\\_value\(\)](#) . . . . . 9  
*Setzt eine Änderungsabfrage. Funktioniert ähnlich wie `get_attribute`. Die dort übergebenen  
Parameter werden hier als Array übergeben. Als zweiter Parameter kommt der String oder  
die Zahl, die den alten Wert ersetzt. Die Funktion wird sooft angewendet, wie sie auf  
die Abfrage trifft. Der entfernte Wert wird nicht gespeichert.*

[xml\\_line::\\$write\\_flag](#) . . . . . 7  
*Ausgabe wird nur geschrieben, wenn true gesetzt ist.*

[xml\\_line::\\$testbuffer](#) . . . . . 7  
*Variable für Testausgaben.*

[xml\\_line::\\$xml\\_buffer](#) . . . . . 7  
*Speichert die XML-Ausgabe, sofern sie gefordert ist, bzw. nicht in eine Datei geschrieben wird*

[xml\\_line::add\\_attributes\(\)](#) . . . . . 8  
*Fügt Attribute in einem Hash hinzu. Nutzt ebenfalls die Startelementmarke und nicht die  
Attributmarke,  
weil direkt auf das Expat-Attributarray zugegriffen wird, bevor dieses weiter verarbeitet wird.*

[xml\\_line::append\\_data\(\)](#) . . . . . 8  
*Hängt den neuen Inhalt an den Wert an  
Zusätzlich zur Parameterliste `$arg` wird ein Wert übergeben, der den gefundenen String  
erweitert.*

[xml\\_line::get\\_attribute\(\)](#) . . . . . 12  
*Setzt die Abfrage für die Werte des Attributs oder der Attribute, die der Vorgabe*

	entsprechen. Ohne Parameter werden alle Werte in das Ergebnis-Array geschrieben.	
<a href="#">xml_line::get_data()</a>	Ruft <code>set_action</code> auf und übergibt das Kommando "find_data" und die Suchkriterien als Parameter.	13
<a href="#">xml_line::replace_element()</a>	Ist identisch mit <code>delete_element</code> . Zusätzlich werden Daten in einen Speicher (fill) geschrieben und dieser beim Erreichen der Marke in die Ausgabe eingefügt.	17
<a href="#">xml_line::replace_content()</a>	= <code>replace_content</code> Ist identisch mit <code>delete_content</code> . Zusätzlich werden Daten in einen Speicher (fill) geschrieben und dieser beim Erreichen der Marke in die Ausgabe eingefügt.	17
<a href="#">xml_line::set_action()</a>	Setzt den Namen der Funktion, die auf den Stream zugreift und übergibt eine Parameterliste oder -string	17
<a href="#">xml_line::set_element_counter()</a>	Setzt den Elementcounter. Dadurch ist die Adressierung paralleler Elemente möglich.	18
<a href="#">xml_line::xml_stream()</a>	Startet den Parser und gibt die gefundenen Daten oder false zurück.	18
<a href="#">xml_line::replace_attribute()</a>	Ersetzt das angegebene Attribut durch ein anderes.	16
<a href="#">xml_line::insert_element_after_element()</a>	Fügt XML-Code oder einen String hinter dem End-Tag eines Elements ein.	16
<a href="#">xml_line::get_output()</a>	Liefert den Output als String, wenn diese Variante gewählt wurde.	14
<a href="#">xml_line::get_element_counter_array()</a>	Experimentell.	14
<a href="#">xml_line::get_record()</a>	Übergibt das Kommando, alle Elemente innerhalb dieses Elementes als Ergebnis zurückzuliefern.	14
<a href="#">xml_line::insert_before_end()</a>	Fügt einen String vor dem End-Tag eines Elements ein.	15
<a href="#">xml_line::insert_element()</a>	Fügt Element(e) oder XML-Code an der gesuchten Stelle ein.	15
<a href="#">xml_line::\$temp_attr_buffer</a>	Speichert die Attributhashes solange das Element aktuell ist. (Key ist der "pathname") Achtung: Wird wegen neuer Funktion <code>get_record</code> zur Zeit nicht gelöscht. (todo) Wichtig für Einfügeoperationen, damit auch beim Endelement noch bekannt ist, welche Attribute das Element hat.	7
<a href="#">xml_line::\$table_result</a>	Soll bei Datensatzabfrage, das Ergebnis speichern: <code>\$table_result[Funktionsnr][counter]['key'] =&gt; "value"</code> key ist die "Spaltenüberschrift" = Name des Elements.	7
<a href="#">xml_line::\$el_counter</a>	Element-Counter. Läuft nur mit, wenn er gesetzt wird. Kann für statistische Zwecke benutzt werden <code>\$el_counter[elname] = array(\$countall, \$level =&gt; \$countonthislevel)</code>	3
<a href="#">xml_line::\$elc_path_array</a>	Kurzform des aktuellen Elementpfads (3,2,8) ohne Bezug auf die Elemente	3
<a href="#">xml_line::\$error</a>	Error - noch sträflich vernachlässigt ;-( Soll die Fehlermeldungen aufnehmen, die man dann abrufen kann. (noch in der Entwicklung)	3

<a href="#">xml_line::\$file</a>	Der Name der XML-Input-Datei	3
<a href="#">xml_line::\$fillattr</a>	Speichert einzusetzende Attribute in einem Hash (name=>value), wenn ein oder mehrere Attribute hinzugefügt werden sollen	4
<a href="#">xml_line::\$fill</a>	Speichert Position als key und einen String als value für das Ersetzen oder Einsetzen von Text oder XML. Jede Funktion nutzt diesen Speicher selbst bei Bedarf.	3
<a href="#">xml_line::\$elc</a>	element-level-counter, enthält die Zählerwerte, der aktuellen Elemente. [level][["elementname"]]	2
<a href="#">xml_line::\$data_buffer</a>	Expat liefert die Daten häufig in mehreren Stücken Hier wird das Ergebnis zwischengespeichert, bis der Data-String komplett ist	2
<a href="#">xml_line::\$akt_name</a>	Name des aktuellen Elements	2
<a href="#">xml_line::\$action_func</a>	Name der aufzurufenden Funktion. Funktionen werden in einem Array gespeichert.	2
<a href="#">xml_line::\$breakset</a>	Internes Flag für die Highlight-Ausgabe. Merkt sich, ob vorher ein Zeilenumbruch gesetzt wurde.	2
<a href="#">xml_line::\$csv_buffer</a>	Speichert die CSV-Ausgabe, sofern sie (statt XML)gefordert ist, bzw. nicht in eine Datei geschrieben wird. Experimentell!!!	2
<a href="#">xml_line::\$datafound</a>	Interner Schalter für die Highlight-Ausgabe. Merkt sich, ob Daten ausgegeben wurden	2
<a href="#">xml_line::\$found_data</a>	Speichert die gefundenen Daten, Pfadinformationen und Attribute in einem Array Art der Rückgabe kann sich noch ändern, am besten mit print_r() selbst ansehen. ;-)	4
<a href="#">xml_line::\$found_data_part</a>	Zähler für Textstücke innerhalb eines Elements mit weiteren untergeordneten Elementen, läuft nur mit, wenn er benötigt wird und das passende Element gefunden wurde.	4
<a href="#">xml_line::\$output_action</a>	Gibt an, ob bzw. was ausgegeben werden soll.	6
<a href="#">xml_line::\$outfile</a>	Der Name der Output-Datei	5
<a href="#">xml_line::\$parser</a>	Expat-Variable	6
<a href="#">xml_line::\$path_array</a>	Enthält alle Elemente des aktuellen Elementpfads einzeln als Array-Werte.	6
<a href="#">xml_line::\$procdata</a>	Speicher für zusätzliche Prozessinformationen, 'datapart' z.B. ([1][['datapart']]) merkt sich bei gemischten Inhalten, welcher Datenabschnitt ausgegeben werden soll 'limit' [1][['limit']]=3 bedeutet: 3 Treffer werden gesucht, danach steigt die Funktion aus.	6
<a href="#">xml_line::\$of</a>	Outputfilehandler	5
<a href="#">xml_line::\$ls</a>	Enthält die wichtigsten Informationen der aktuellen XML-Position.	5
<a href="#">xml_line::\$highlight</a>	Interner Schalter für die Highlight-Ausgabe	4
<a href="#">xml_line::\$func_counter</a>	Zählt die gesetzten Funktionen. Ermöglicht das Auseinanderhalten der verschiedenen Prozesse und die Zuordnung der Rückgabewerte.	4

<a href="#">xml_line::\$ignore_default_data</a>	4
<i>XML-Header und Kommentare ausgeben oder nicht. Wird über Methodenaufruf verändert</i>	
<a href="#">xml_line::\$last_nowrite_line</a>	5
<i>Markiert das Ende von Bereichen, die nicht geschrieben (also de facto gelöscht) werden.</i>	
<a href="#">xml_line::\$level</a>	5
<i>Gibt den aktuellen Level der Elementhierarchie an</i>	
<a href="#">xml_line</a>	1
<i>XML-LINE Version 0.2.11 == 09.04.2004</i>	