Documentation by*lster.ORG*

xml22

Table of Contents
xml22

NAME

xml22 - search and edit XML files with PHP

VERSION

xml22 0.3.2

DESCRIPTION

Overview

This PHP4 code parses an XML document into a multidimensional array. The parser recognizes and processes almost all of XML, including processing instructions, general or external entities and notations. In addition many functions are provided to search and edit the document, build a new one and write it back to a string or to a file. The **xml22** code is aimed to provide a functionality comparable to the Document Object Model in a different manner, offering its own API.

Array Structure

The internal representation of the document in **xml22** differs from that of the Document Object Model. The user of the code is strongly encouraged to use the API functions when dealing with the document and *not* to touch the array directly. The internal structure will change in the future (see */IMPORTANT NOTE ABOUT THE ARRAY STRUCTURE*), but—of course—this will not affect the API.

For **xml22** an XML document is an array of tag arrays, or "tags" for short:

```
Array (
 [int] => array
 [...]
)
```

A tag array has this structure:

```
Array (
    [tag]        =>  string    // may be empty e.g. for comments
    [level]      =>  int       // level in XML tree
    [index]      =>  int       // in respect to the order of
                               // the document array
    [parindex]   =>  int       // points to the index of parent
                               // not set when no parent, i.e. when
                               // not below root
    [children]   => Array      // each element points to an immediate
                               // child
                               // not set when no children
           (
        [int] => int           // document index
        [int] => int           // document index
        )
        [attributes] => Array      // if there are any
           (
                [name]   => 'value'
                [name]   => 'value'
                )
        [content]    =>  string    // not set when no content enclosed
                               // by the tag
                               // and his closing tag
        ),
 [...]
)
```

There is one tag with a different structure, the doctype tag:

```
Array (
    [tag]        =>  string    // always set to "!DOCTYPE"
    [level]      =>  int       // level, should be 0
    [index]      =>  int       // in respect to the order of
```

```
                              // the document array
        [type]        =>  Array (
                                  [0] => string  // name of root
                                  [1] => string  // SYSTEM | PUBLIC
                                  [2] => string  // URI if SYSTEM, name if
                                                 // PUBLIC
                                  [3] => string  // URI if PUBLIC, not set
                                                 // if SYSTEM
                              )
        [entities]    =>  Array (   // not set if no <!ENTITY definition
                                  [<string>] => string // name => definition
                                  [...]
                              )
        [notations]   =>  Array (   // not set if no <!NOTATION definition
                                  [<string>] => string // name => definition
                                  [...]
                              )
        [comments]    =>  Array (   // all comments inside the <!DOCTYPE
                                    // not set if no comments
                                  [int] => string
                              )
        [pi]          =>  Array (   // processing instructions
                                    // inside <!DOCTYPE
                                    // not set if no PIs
                                   [int] => string
                              )
    )
```

IMPORTANT NOTE ABOUT THE ARRAY STRUCTURE

Until version 0.4.0 the array structure will change. This is needed for two reasons. First, the structure is insufficient to allow *complete* compliance to the XML standard. Second, another structure—mainly with additionally stored meta data—may lead to better performance in searching an editing. As long as another structure may prolong parsing time a little, persistent caching may help to save time in the large.
Again: Never touch the array by hand! Ever use the API functions!

Tags

What I call a "tag" here should better called a "node" but this term is not used to prevent confusion with the Document Object Model. The **xml22** code does not conform to the W3C recommendation. It cannot, at least because it is not written object oriented.
A "node" and a "tag" are nearly the same thing in this documentation. A "tag" is actually defined by two XML tags, one opening and one closing. Also a "text node" of the DOM is not considered as a separate "tag", it is considered as the "content" of a tag. In most cases this makes no difference, but there are some restrictions that result of this model (see */RESTRICTIONS*).
Any tag (or node) is represented by an array that may be multidimensional if there are any children of that tag in the XML tree or if there are attributes set for this tag. The array that represents the document is nothing else but a list, i.e. an integer indexed array of those tags (see */Array Structure*).
The order of the tags in the array represents the order of the nodes in the document, but the indices of the tags are not equal to their logical level in respect to the document. The root node of the document usually doesn‘t have the index of 0 in the document array. For well formed XML documents you will find at this point the XML version string. If you need the root node, use the function */xml22_get_root*.

Searching and Editing

Searching and editing can take place at three levels. You may access the document at the abstract XML level, at the tag level or even at the internal array level if you really need this. The latter is not recommended, and editing at the tag level should be done very carefully.
The main difference between XML level and tag level is that for the tag level you have to take care that some important fields of the tag array are defined and even initialized with the proper values. For the XML level you don‘t have to care about this.
For instance, if you need a particular tag ‘<wanted>’ with a special content, say ‘John Silver‘, you have two ways of searching. The first performs an XML level search:

```
$doc        = xml22_parse( 'pirate.xml' );
$firstresult = xml22_get_all_of_name( $doc, 'wanted' );
$nextresult  = xml22_get_all_of_content( $firstresult,
            '/John Silver/' );
$pirate      = array_shift( $nextresult ); // to get the
                                           // first occurance
```

This is the corresponding tag level search:

```
$doc          = xml22_parse( 'pirate.xml' );
$search       = array( 'tag'     => '/wanted/',
                       'content' => '/John Silver/' );
$result       = xml22_get_by_regex( $doc, $search, XML22_GET_FIRST );
$pirate       = array_shift( $result );
```

It is up to you which way you prefer.
I prefer to edit at the abstract XML level and this is what **xml22** is really made for. The DOM on the other hand allows *only* editing at the XML level, so **xml22** may be considered a little more flexible though the code may look not that pretty.
In the function documentation below the functions are marked to belong to the XML level or to the tag level. Some of them belong to both because it is hard to decide wether they are those or that.
The performance of the searching functions differs considerably. Very fast are functions like */xml22_get_first_child*, */xml22_get_last_child*, */xml22_get_next_sibling*, */xml22_get_prev_sibling* or */xml22_get_parent* because they cost only some array operations. Slow are functions that search for specific values like */xml22_get_by_regex*, */xml22_get_all_of_content*, */xml22_get_all_of_name* or */xml22_get_by_id*. The performance of the others is somewhere in between.
Note: As long as **xml22** is still under development, from time to time you may find yourself in situations where you have to touch the tag arrays directly. This will be obsolete in the future as the code provides more and more functions that do this for you.

Functions Returning Tags

The **xml22** code includes a lot of functions dealing with single tags. Some editing functions return the edited documents but a lot of functions return single tags. They are devided into two classes.
Those that may return a collection of tags return them as an indexed array of tag arrays. If you need the particular tags you have to take them out of that array first. This may be done by `array_pop()`, `array_shift()` or in a `foreach` loop. You have to use `array_pop()` even if these functions return only *one* single tag. However, if there really is only one element in the returned array, i.e. the searching function got only one match, you may pass the returned array as is to the next searching or editing function you choose. The functions will do an `array_pop()` for you. But beware if you‘re not shure to got only one single tag. If there are more than one in the array and you pass this array to another function as a single tag, only the *last* of these elements will be recognized.
However, all this must *not* be done for functions that by definition return only one tag. Most of the time you may use functions of the second class. They seem to be more handy.
As of version 0.3.0 a returned array of tag arrays—class one—will be considered to be a document fragment (see */Document Fragments*). You can use */xml22_is_tag*, */xml22_is_fragment* or */xml22_is_document* to check which type of array you are about to use.
The members of the two classes are listed here:

- functions returning arrays of tags aka document fragments
  /xml22_get_by_regex, */xml22_get_all_descendants*, */xml22_get_all_ancestors*, */xml22_get_all_siblings*, */xml22_get_all_of_name*, */xml22_get_all_of_content*, */xml22_copy_fragment*
- functions returning single tags
  /xml22_create_tag, */xml22_add_attribute*, */xml22_delete_attribute*, */xml22_get_root*, */xml22_get_by_id*, */xml22_get_first_child*, */xml22_get_last_child*, */xml22_get_next_sibling*, */xml22_get_prev_sibling*

Document Fragments

A document fragment may be considered to be an XML document without the required `<?xml>` declaration at the start of it and without any DTD. If you created a fragment using */xml22_create_fragment* you may treat it as if it were a document—with the one exception, that you cannot parse a fragment after you wrote it using */xml22_write_document*. Of course, you will also fail if you try */xml22_get_version* or */xml22_get_doctype* with a fragment or if you hope to find a root node.

Even only one tag may form a document fragment. The difference to a single tag is—internally—that a fragment is represented by an array of tag arrays while a single tag is represented by a plain tag array. You may use */xml22_is_tag* or */xml22_is_fragment* to distinguish between the two.

Working with fragments may speed up your scripts a little. For an example look at this little code snippet. It prints the fragment of all the tags belonging to the first child of the root element, including that child itself:

```
$doc     = xml22_parse( 'file.xml' );
$parent  = xml22_get_first_child( $doc, xml22_get_root( $doc ) );
$offset  = count( xml22_get_all_descendants( $doc, $parent ));
$frament = xml22_copy_fragment( $doc, $parent, $offset );
print xml22_write_document_str( $fragment, XML22_BEAUTIFY );
```

Caching

Sometimes your script may try to read the same XML file over and over again, e.g. in case of concatenated XML files, one pointing to some others. To prevent multiple parsing, the parsing function tries to cache the already parsed data and reads them back out of the cache if additional parsing is requested. This gives a significant increase of performance. The cache is written to the file system, so the data do not influence the available memory of the scripting engine.

Exmaple:

```
$doc1 = xml22_parse( "file1.xml" ); // file1 is parsed and cached
// do something with $doc1
unset( $doc1 );
$doc2 = xml22_parse( "file2.xml" ); // file2 is parsed and cached
// do something with $doc2
unset( $doc2 );
$doc1 = xml22_parse( "file1.xml" ); // file1 is rebuild by the cache
                                    // no parsing will happen
// do something with $doc1
unset( $doc1 );
```

The caching mechanism checks if a document‘s file has been changed since it was parsed the last time, there should be no need to turn the caching off. In several cases, e.g. if the script has problems writing to the file system, you may want to turn it out nevertheless. This may be done for the whole runtime of the script using */xml22_setup* or for every single call to */xml22_parse* passing the argument `$usecache` to that function.

INSTALLATION

First you have to download the latest release.

Then type this sequence on your Unix/Linux box:

```
tar -xzf xml22-0_3_2.tar.gz
cd xml22-0_3_2
make install INSTALLDIR=<path>
```

If you want to uninstall the files type:

```
make uninstall INSTALLDIR=<path>
```

If your system is Windows you may use a tool like Winzip to unpack the archive, then copy all `*.inc`-files of the ***./xml22*** subdirectory to your prefered destination.

To use **xml22** in your own scripts, include the file ***xml22.inc***. If all you need is the parser, you may include only ***xml22-parser.inc***, which includes ***xml22-share.inc*** via *require_once()*. If you do not want to edit XML files, you may omit to include ***xml22-edit.inc***.

FUNCTIONS

Parse

xml22_parse

```
array xml22_parse ( string $file,
                   [bool $usecache = XML22_CACHE] )
```

This function parses the XML document as given by `$file` and builds a multidimensional array representig this XML document. `$file` may either be a valid path of the local file system or a single string containing well formed XML. An XML string is assumed if the `$file` string starts with an `<?xml>` declaration. So, if you pass a huge string with a misspelled opening, the string will be treated as a file name. The function returns the document array or *false* on failure.
The parser changes the document a little, mainly in respect to the XML syntax. These changes are remaining when the document is written back to a file. Multiple declarations of the same entity or notation are discarded with an error message. Newlines inside of CDATA are replaced with spaces. Comments inside of the doctype declaration are stored in the array but when the document is written they are ignored (this is considered to be a bug).
As required by the XML specification, entities are resolved. Predefined entities are expanded by the parser and will be restored by */xml22_write_document*. All other entities are resolved while parsing, but they are not encoded again when the document is written. You may use */xml22_get_entity* to do this by yourself if needed.
By default the parser is non-validating, i.e. external entities are not resolved. You may set the setup option *XML22_OPT_EXTERNALS* to a value of *true* if you want external entities to be parsed (see */xml22_setup*). As stated for general entities, external entities are resolved but they cannot be restored by */xml22_write_document*. If the external entity is given as an URI with a ‘http://‘-prefix, the parser tries to GET it.
The `$usecache` parameter switches the internal caching mechanism. A value of *true*—which is the default—turns caching on, *false* turns it off (see */Caching*). Instead of *true* and *false* you may use XML22_CACHE or XML22_NOCACHE. If the cache encounters a problem, an error message will be stored (see */xml22_error*) and the file will be parsed again. Passed XML strings will not be cached.
The caching mechanism checks if a file has been changed since the last request for parsing. This is done reading the modification time of the file (using `filemtime()`). If the file has been changed, it is parsed again.
Note that caching still only works for the execution time of the script, not for the whole session. This may change in the future.

xml22_register_pi_func

```
bool xml22_register_pi_func( string $target, string $funcname )
```

The function may be used to register a callback function for a processing instruction. The function expects the case sensitive target name and the name of the callback function. **xml22_register_pi_func** should be called *before* */xml22_parse* is called, otherwise nothing will happen. All processing instructions are passed to the callback function as they are recognized by the parser. If the parser reads a PI with no registered function the PI will be silently ignored but still stored in the document array. You may register as many callback functions as needed.
If the callback function to register is not defined, **xml22_register_pi_func** returns *false* and sets an error message. *true* is returned on success.
The callback function will receive only one argument, the data of the processing instruction:

```
bool callback( string $data )
```

xml22_register_char_entity_set

```
array xml22_register_char_entity_set( array $match = null,
                                      array $replace = null )
```

By default, only predefined entities are encoded by */xml22_write_document* and */xml22_write_document_str*, though the parser resolves character entities as well. With this function it is possible to register a set of character entities to be encoded when a document is written. Two arguments should be passed, the array `$match` with a number of perl regular expressions and the array `$replace` build by the same number of character entities to replace the matches.

Write

xml22_write_document

```
bool xml22_write_document( array $document,
                           ressource $fd, [int $style = XML22_NORMALIZE] )
```

This function will write the document array as build by */xml22_parse* to the file descriptor `$fd` as a well formed XML document. Predefined entities are encoded. General or external entities are not encoded. Tags without any content will be written in short notation, like `<tag id="id"/>`. The function returns *true* on success or *false* on failure.
The parameter `$style` determines in which way the document is written. A value of XML22_NORMALIZE—which is the default—will cause the function to write the document in normalized form, i.e. without any spaces, tabs or newlines. You may set this variable to XML22_BEAUTIFY to write the document in human readable form.

xml22_write_document_str

```
string xml22_write_document_str( array &$document
                                 [, int $style = XML22_NORMALIZE] )
```

The function behaves the same like */xml22_write_document* but is not writing to a file descriptor but to a string which is returned on success. *false* will be returned on error.

Search

xml22_get_by_regex

tag level

```
array xml22_get_by_regex( array &$document,
                          array &$query,
                         [int $tokencount = XML22_GET_ALL,
                          int $startindex = 0,
                          int $offset     = 0] )
```

This function is used to search the document array for specific tags or values. It returns an array of any number of tag arrays that match the query as given by `$query`, *false* on failure.
The parameter `$tokencount` determines the number of found tags to be returned. If `$tokencount` is set to 0, the function will return all tags found. If it is greater than 0, the function returns only the `$tokencount` first. You may use XML22_GET_ALL or XML22_GET_FIRST instead of numerical arguments. The default is to return all found tags.
There are some parameters to influence the performance of the search a little. If you specify the `$startindex`, the search will start at the `$startindex`th element of the array passed to as `$document`.
As another option `$offset` may be set. Is it greater than 0, the search will end *at* the `($startindex + $offset)`th element. Is `$offset == 0`—which is the default—the search will continue down to the end of the document array. An `$offset` lesser than 0 will perform a backward search. `$startindex` is then counted from the last element of `$document`. If you need a backward search starting at the end of the document running up to the first element, you may set `$startindex = 0` and `$offset = -1`.
The structure of a query is

```
$query = Array( 'field' => '/regexp/', 'field' => '/regexp/' ... )
```

or

```
$query = Array( 'field' => '/regexp/', ...,
                'attribute' => array( 'name' => '/regexp/', ... ) )
```

To examine the tags, you have to `array_shift()` them off of the returned array even if there is only one tag found (see */Functions Returning Tags*).
Note: From version 0.0.2 to 0.2.95 the internal implementation of */xml22_get_by_regex* changed a little. In particular the meaning of `$startindex` and `$offset` is different now.

xml22_get_all_siblings

XML level, tag level

```
array xml22_get_all_siblings( array &$document, array &$tag );
```

This function returns an array of all elements of the same XML level and with the same parent node as `$searchtag`, *false* on failure. The result *includes* `$searchtag`.

xml22_get_all_descendants

XML level, tag level

```
array xml22_get_all_descendants( array &$document, array &$tag );
```

This function returns an array of all subsequent elements of `$searchtag`, *false* on error. Not only the immediate children, but also the children‘s children and so forth are returned. If you need only a tag‘s immediate children, check out the ‘children‘-field of the tag.

xml22_get_all_ancestors

XML level, tag level

```
array xml22_get_all_ancestors( array &$document, array &$tag );
```

This fnction returns an array of all related parents and grandparents of `$searchtag` or *false* on failure. All parent tags up to the root node of the XML document are returned. The immediate parent will be at the start of the returned array, the root node will be at the end. If you need *only* the immediate parent of a tag use L/xml22_get_parent.

xml22_get_all_of_name

XML level

```
array xml22_get_all_of_name( array &$document, string $name, $mode = false )
```

The function returns an array of all tags of name `$name` that are found in the document array; *false* will be returned if no tag was found or if an error occured. The argument `$name` should be a plain string, no regular expression.
As of version 0.3.2 you may add an argument `$mode` to the function. If it is set to
XML22_START_ROOT the search will start at the root node of the document (which may be a little faster).

xml22_get_all_of_content

XML level

```
array xml22_get_all_of_content( array &$document, string $regex, $mode = false )
```

This function returns an array of all tags of which the ‘content’ field matches the perl compatible regular expression given by `$regex`. The function will retun *false* if no tag was found or on failure.
As of version 0.3.2 you may add an argument `$mode` to the function. If it is set to
XML22_START_ROOT the search will start at the root node of the document (which may be a little faster).

xml22_get_version

XML level

```
array xml22_get_version( array &$document )
```

This function returns an array containing information about the XML version of the document or *false* on failure. If the version string misses information, i.e. the `encoding` attribute, the returned array misses this field too.
The array structure looks like this:

```
Array( ’version’    => <string>,  // "1.0"
       ’standalone’ => <string>,  // "no" | "yes"
       ’encoding’   => <string> ) // "US-ASCII" or whatever
```

xml22_get_doctype

XML level

```
array xml22_get_doctype( array &$document )
```

The function returns an array containing information about the XML doctype of the document or *false* on failure. If the doctype string misses information, i.e. the name in case of a SYSTEM DTD, the returned array misses this field too.

```
Array( 'root'       => <string>,  // the name of the root element
       'SYSTEM'     => <string>,  // the location of the SYSTEM DTD
       'name'       => <string>,  // the name of the PUBLIC DTD
       'PUBLIC'     => <string> ) // the location of the PUBLIC DTD
```

xml22_get_root

XML level

```
array xml22_get_root( array &$document )
```

The function returns one array representig the root element of the document or *false* on failure.

xml22_has_children

XML level

```
int xml22_has_children( array &$tag )
```

The function returns the number of children if `$tag` has any or 0 if not, *false* on error.

xml22_get_first_child

XML level

```
array xml22_get_first_child( array &$document, array &$parent )
```

Returns the first immediate child of `$parent`, *false* if no child was found or on error.

xml22_get_last_child

XML level

```
array xml22_get_last_child( array &$document, array &$parent )
```

Returns the last immediate child of `$parent`, *false* if no child was found or on error.

xml22_get_next_sibling

XML level

```
array xml22_get_next_sibling( array &$document, array &$tag )
```

Returns the successor tag of the same level in the XML tree as `$tag`, *false* on failure or if no match was found.

xml22_get_prev_sibling

XML level

```
array xml22_get_prev_sibling( array &$document, array &$tag )
```

Returns the ancestor tag of the same level in the XML tree as `$tag`, *false* on failure or if no match was found.

xml22_get_parent

XML level

```
array xml22_get_parent( array &$document, array &$tag )
```

The function returns the parent tag of `$tag`, *false* on failure. This returns the parent in respect to the XML tree, not in respect to the document array, i.e. you cannot get a parent of the root tag.

xml22_get_by_id

XML level

```
array xml22_get_by_id( array &$document, string $id,
                      [, string $idname = ""] )
```

The function returns one array representing the tag which has an attribute of type ID with the value `$id`. If `$idname` is given, the attribute of `$idname` is searched for the given `$id`. Otherwise an attribute name that consists of or ends with the strings ‘id’ or ‘ID’ is searched (regex: `/.*(id|ID)/`). The function returns *false* if no match was found or an error occured.

xml22_get_name

XML level, tag level

```
string xml22_get_name( array &$tag )
```

The function returns the name of `$tag` or *false* if no name is set. This may be the case for comments, processing instructions or the XML version string.

xml22_get_content

XML level, tag level

```
string xml22_get_content( array &$tag,
                         [ bool $plain = true, string $rplc = '' ] )
```

This function returns the string of the 'content' field of the tag `$tag` or *false* on failure. As of version 0.3.0 you may add the parameter `$plain`. If `$plain` is set to *false*, all '<![CDATA[' and ']]' strings will be deleted in the returned string, so you will get the content as it is meant rather than as it is stored. By default, both strings are replaced by the empty string. You may pass a third parameter `$rplc` to the function which changes the replacement string.

xml22_get_attribute

XML level

```
string xml22_get_attribute( array &$tag, string $attrname )
```

Get a string representing the value of the attribute `$attrname` of the tag `$tag`. The function returns *false* on failure.

xml22_get_entity

XML level

```
mixed xml22_get_entity( string &$doc [, string $name = ''] )
```

By default, the function returns an array containing pairs of entity names and definitions.

```
Array( [&name;] => "definition" )
```

If `$name` is specified, a string representing the definition of the entity given by `$name` is returned. A value of false will be returned on failure.
The entity must be defined by the *internal* DTD section of the document (see */RESTRICTIONS*).

xml22_get_notation

XML level

```
mixed xml22_get_notation( string &$doc [, string $name = ''] )
```

By default, the function returns an array containing pairs of notation names and definitions.

```
Array( [name] => "definition" )
```

If `$name` is specified, a string representing the definition of the notation given by `$name` is returned. A value of false will be returned on failure.
The notation must be defined by the *internal* DTD section of the document (see */RESTRICTIONS*).

xml22_is_tag

tag level

```
bool xml22_is_tag( array &$search );
```

Return *true* if `$search` is a tag array, *false* otherwise.

xml22_is_fragment

tag level

```
bool xml22_is_fragment( array &$search );
```

Return *true* if `$search` is an array representing a document fragment, *false* otherwise.

xml22_is_document

tag level

```
bool xml22_is_document( array &$search );
```

Return *true* if `$search` is a document array, *false* otherwise.

xml22_is_below_root

XML level

```
int xml22_is_below_root( array &$tag )
```

The function returns *1* if `$tag` is located below the root node of the document it belongs to, *0* if it is the root itself or *-1* if it is located above the root node. The returned value will be *false* if `$tag` is not valid (check for identity with ===, not just for equality).

Edit

xml22_create_document

XML level

```
array xml22_create_document( string $version
                            [, string $standalone = "no",
                               string $encoding = "" ] )
```

The function returns an array representing an empty XML document. The array actually consists of only one element containing the XML version string. The XML version will be set as given by `$version`. By default, the function creates a document with an external DTD, i.e. the attribute `standalone="no"`. You may change this to "yes" by the argument `$standalone`. In addition you can specify the encoding string with `$encoding`.

xml22_create_tag

tag level

```
array xml22_create_tag( string $name, [string $content = '',
                                       array $template = false,
                                       bool  $mode = XML22_BYSIBLING] )
```

Create a tag array with ‘tag‘-field set to `$name`. If `$content` is given, it will be written to a ‘content‘-field. The fields ‘index‘, ‘parindex’ and ‘level’ will be left *false* or will be filled with meaningful values if a `$template` tag array is given. The `$mode` marks the `$template` as a sibling or the parent of the tag to create. It may be set to XML22_BYSIBLING or XML22_BYPARENT. The created tag will have its ‘index‘-field set to the immediate descendant of `$template` in respect to the document array. The function returns the created tag array or *false* on failure.
The created tag does not appear in the document until you add it using */xml22_add_child*, */xml22_add_sibling* or */xml22_insert_tag*.

xml22_create_fragment

XML level, tag level

```
array xml22_create_fragment( array $firsttag )
```

The function returns an array representing a document fragment. The fragment will contain only one tag, `$firsttag`. The XML tree level of `$firsttag` will be reset to 0. If the returned value equals *false*, an error occured.

xml22_add_attribute

XML level

```
array xml22_add_attribute( array $tag, string $name, string $value )
```

This function adds an attribute `$name` with a value of `$value` to the tag given as `$tag`. If an attribute with name `$name` already exists, its value will be updated to `$value`. The function returns the edited tag array or *false* on failure.

xml22_delete_attribute

XML level

```
array xml22_delete_attribute( array $tag, string $name )
```

This function deletes an attribute `$name` from the tag given as `$tag`. If no attribute remains after this operation, the ‘attribute‘-field of `$tag` will also be deleted. The function returns the edited tag array or *false* on failure.

xml22_add_content

XML level, tag level

```
array xml22_add_content( array $tag, string $content
                         [, bool $action = XML22_OVWRT,
                            bool $cdata = false ]
```

By default, the function inserts the string `$content` to the ‘content’ field of the tag `$tag`, i.e. any content that already exists will be overwritten. The function concats the string to the ‘content’ field if the parameter `$action` is set to XML22_CAT.
The ‘content’ field will be written as an explicit `<![CDATA]]> declaration if the parameter $cdata is set to XML22_CDATA. Then the` *`whole`* `content will be enclosed in a single <![CDATA]]>—regardless if you set XML22_CAT—and all <![CDATA]]> that already exist will disappear. If you need mixed <![CDATA]]> and normal XML you should` *`not`* `use XML22_CDATA. You better specify the different <![CDATA]]> in the $content string one by one.`
The function returns the edited tag or *false* on failure.

xml22_delete_tag

XML level, tag level

```
array xml22_delete_tag( array $document, array $tag
                        [, bool $descend = false ] )
```

Delete the given tag. If you set the optional parameter `$descend` to *true*, all descendants of the tag will be deleted too. Instead of *true* you may use XML22_DEL_CHILDREN. The function returns the edited document array, *false* on failure.
Note that deleting a tag setting XML22_DEL_CHILDREN is not that much slower than deleting just a single tag.

xml22_insert_tag

XML level, tag level

```
array xml22_insert_tag( array $document, array $tag
                        [, mixed $ancestor = 0] )
```

Insert the given tag at the position below `$ancestor`. The `$ancestor` may be a tag array or an integer representing the ‘index’ field of the ancestor tag. If `$ancestor` is omitted, the ancestor element is determined by the ‘index’ field of the tag which should be greater than 0. `$tag` will be updated and may be reused. The function returns the edited document array or *false* on failure.

xml22_replace_tag

XML level, tag level

```
array xml22_replace_tag( array $document, array $oldtag, array $newtag )
```

Replace `$oldtag` with `$newtag` and return the edited `$document`, *false* on failure.
This function may also be used to simply change a tag. It is much faster to use this than to first deleting a tag and after that inserting a new one at the same place. `$newtag` will be updated and may be reused.

xml22_move_tag

XML level, tag level

```
array xml22_move_tag( array $document, array $tag, mixed $newancestor )
```

This function moves `$tag` from its current position to the position below `$newancestor`. The argument `$newancestor` may be a tag array or an integer representing the ‘index’ field of the newancestor tag. `$tag` will be updated and may be reused. The function returns the edited document or *false* on failure.
Note: Currently this function is implemented by simply calling */xml22_delete_tag* followed by an */xml22_insert_tag*. This may change in the future.

xml22_copy_fragment

XML level

```
array xml22_copy_fragment( array &$document, array $start, int $offset )
```

This function returns an array representing a copy of the document fragment starting with tag `$start` and ending `$offset` tags below `$start`. The internal indices of the array are normalized to the base of 0. The function returns *false* on error.

xml22_delete_fragment

XML level

```
array xml22_delete_fragment( array $document, array $start, int $offset )
```

The function deletes the document fragment starting at the tag `$start` and ending `$offset` tags below `$start`. If `$offset` exceeds the document, all until the end of the document will be deleted. The edited document array is returned or *false* on error.

xml22_insert_fragment

XML level

```
array xml22_insert_fragment( array $document, array $fragment,
                             array &$ancestor )
```

The function will insert the document fragment `$fragment` below the tag given by `$ancestor`. If `$ancestor` has any descendants, the fragment will be inserted below the last of its descendants. The edited document array is returned or *false* on failure.

xml22_replace_fragment

XML level

```
array xml22_replace_fragment( array $document, array $fragment,
                              array $start )
```

The function replaces in `$document` a document fragment of length `count($fragment)`, beginning at `$starttag` with the document fragment given with `$fragment`. The edited document is returned or *false* on failure.
Note: The function does *not* check if the logical—or XML—structure of the fragment to replace is equivalent to the structure of the replacement.

xml22_move_fragment

XML level

```
array xml22_move_fragment( array &$document, array $start,
                           int $offset, array $newancestor )
```

This function will wove the document fragment that starts at tag `$start` and ends at `$ofset` tags below `$start` below `$newancestor`. If `$newancestor` has any descendants, the fragment will be inserted below the last of its descendants. The edited document array is returned or *false* on failure.

xml22_add_doctype

XML level

```
array xml22_add_doctype( array $document, array $doctype )
```

A proper doctype declaration as specified by the array `$doctype` will be inserted into the document `$document` at the position immediately below the XML version string. If a doctype declaration already exists it will be overwritten. The function returns the edited document or *false* on failure.
The `$doctype` array can have two variants as described here:

```
Array( 'root'   => <string>,  // name of the root element
       'SYSTEM' => <string> ) // location of the SYSTEM DTD

Array( 'root'   => <string>,  // name of the root element
       'name'   => <string>,  // name of the PUBLIC DTD
       'PUBLIC' => <string> ) // location of the PUBLIC DTD
```

xml22_add_root

XML level

```
array array xml22_add_root( array $document, string $name,
                            string $namespace )
```

The function adds a root tag to the document array `$document`. The name of the tag is given by `$name`. The parameter `$namespace` will be used to set an attribute ‘xmlns’ of the root tag. If a root tag already exists in the document it will be overwridden.
The function returns the edited document or *false* on failure.

xml22_add_child

XML level

```
array xml22_add_child( array $document, array $tag, array $parent )
```

The function adds the tag `$tag` to the document array `$document`, assuming that `$tag` is a child of `$parent`. If the parent already has any children, the new tag will be added below the last of these children or below the last of the subsequent children. `$tag` will be updated and may be reused. The edited document array is returned or *false* on failure.

xml22_add_sibling

XML level

```
array xml22_add_sibling( array $document, array $tag,
                         array $ancestor )
```

This function adds the tag `$tag` to the document array `$document`, assuming that `$tag` is a sibling of `$ancestor`. The new tag will be added immediately below the ancestor or below the last of the ancestors children if there are any. `$tag` will be updated and may be reused. The edited document array is returend or *false* on failure.

xml22_add_comment

XML level

```
array xml22_add_comment( array $document, string $comment,
                         mixed $ancestor )
```

The function adds a comment as given by `$comment` to the document `$document` at the position below `$ancestor`. The argument `$ancestor` may be a tag array or an integer representing the ‘index’ field of the ancestor tag. The function returns the edited document or *false* on failure.

Miscellaneous

xml22_setup

```
bool xml22_setup( array &$options )
```

You may use this function to customize the behavior of the **xml22** code. It is intended to set some values for the whole runtime of the script. Once an options is set via */xml22_setup* it cannot be changed later on. For instance, if you want to switch caching off and on several times, don‘t use this function but the paramater `$usecache` of the */xml22_parse* function.
The array `$options` consists of pairs of option names and values:

```
$options = Array( ’XML22_OPT_<name>’ => <value>, ... )
```

Possible options and their values are:

- XML22_OPT_CACHE
  Switch caching on or off (see */xml22_parse*).
  ```
  XML22_OPT_CACHE = XML22_CACHE | XML22_NOCACHE
  ```
- XML22_OPT_EXTERNALS
  Switch parsing of external entities on and off (see */xml22_parse*).
  ```
  XML22_OPT_EXTERNALS = true | false
  ```
- XML22_OPT_WRITESTYLE
  Set the style for writing a document (see */xml22_write_document*).
  ```
  XML22_OPT_WRITESTYLE = XML22_NORMALIZE | XML22_BEAUTIFY
  ```
- XML22_OPT_TABSIZE

Set the tabsize in characters for indentation of the lines if the document is written in style XML22_BEAUTIFY (see */xml22_write_document*). This value defaults to 3.

```
XML22_OPT_TABSIZE = <int>
```

An error will be reported and the function will return *false* if any option doesn‘t match the range of possible values. Unknown options will be silently ignored.
The function should be called only once and *before* you call any other function of **xml22**. If you try to call it twice, the function will return *false* and an error will be set. You may check this via */xml22_error*. If you run it once but too late, no error will be reported but your code may not behave as expected.

xml22_error

```
array xml22_error()
```

Returns an array of all error messages. The last error is additionaly written to a field ‘last’ of the returned array. Every call to */xml22_error* will unset this field. You may check if this field is set to check if an error occured during the last function call.

FAQ

Why doesn‘t PHP find the include files?

Possibly you work with a different directory structure as stated in the */Installation* section. If this is what you want, you have to adjust the `require_once` statements at the start of the **xml22** include files by your own, or—simple—the include path via *ini_set()*.

Why does my parsing always fail?

If you pass a string, not a filename to */xml22_parse*, remember that it must contain *well formed* XML, i.e. the string has to start with at least this statement: `<?xml version="1.0"?>`.

When I parse an XML file, the script seems to wait forever. Why?

Possibly the parser tries to read an external entity via HTTP GET, but the network or the server is unreachable.

Why can‘t I find any ‘tag’ field in the matched element?

If you work with the returned data of searching functions that may return more than one match, remember that the returned array is not a plain tag array but an array of tag arrays. If you need the tag array by itself, you have to apply `array_shift()` or `array_pop()` first, even if there is only one match returned (see */Functions Returning Tags*).
Another reason may be that the element represents an XML comment, the `<?xml> declaration or a processing instruction.`
Usually you don‘t need to touch the tag array directly. Use */xml22_get_name* or */xml22_get_content*.

Why does my search always fail?

Check the regular expressions. Are there valid delimiters (e.g. "/^[fF]ine [rR]egex$/"—note the slashes)? Did you use variables inside the regular expressions, but single instead of double quotes (wrong: ‘/^$wonder$/’ right: "/^$wonder$/")?
If you use */xml22_get_by_regex* search for attributes by an additional array *inside* the array that represents the query.
As of version 0.0.2 you may pass tags to the searching functions either by an array as it is returned by */xml22_get_by_regex* or you may—better—search by the tag array itself as it results from an `array_pop()` of the latter. This has been introduced for robustness but you are on the safe side if you do the `array_pop()` by yourself (see */Functions Returning Tags*).
As of version 0.2.95 you may need */xml22_get_by_regex* only in some very special cases. Now **xml22** contains a lot of specialized searching functions (see */Search*).

Why does my editing always fail?

Especially if you try to insert or to replace a tag ensure that the given tag arrays include at least the fields ‘index‘, ‘level’ and ‘parindex’ i.e. you must tell the functions where to place the tags. The fields must be set (defined), they do not necessarily have to have a true value—though you should fill one in to prevent confusion.
Best practice is to get a sibling first and work like this:

```
$search  = array('tag'     => '/name/',
                 'content' => '/John Silver/');
$sibling = array_shift( xml22_get_by_regex( $doc,
                         $search, XML22_GET_FIRST ) );
$new = xml22_create_tag( 'position', 'chairman', $sibling,
                         XML22_BYSIBLING );
$doc = xml22_insert_tag( $doc, $new );
```

This may be done similar for a given parent using XML22_BYPARENT. Much better is to use */xml22_add_sibling* or */xml22_add_child*.

When I delete a parent, it‘s children remaining. What now?

You should not delete a parent before you have deleted all of it‘s children.
As of version 0.3.0 all you have to do is to add an additional parameter to */xml22_delete_tag*:

```
$doc = xml22_delete_tag( $doc, $tag, XML22_DEL_CHILDREN );
```

The old way to delete root safely looks like this:

```
function my_delete_fragment( $doc, $parent ) {
  $descarray = array_reverse(xml22_get_all_descendants($doc, $parent));
  foreach( $descarray as $child ) {
    if ( xml22_get_first_child( $doc, $child ) ) {
      $doc = my_delete_fragment( $doc, $child );
    }
    $doc = xml22_delete_tag( $doc, $child );
  }
  $doc = xml22_delete_tag( $doc, $parent );
  return $doc;
}

$doc  = xml22_parse('myfile.xml');
$root = xml22_get_root( $doc );
$doc  = my_delete_fragment( $doc, $root );
```

What means "Cannot pass parameter x by reference"?

As of version 0.2.95 the internal interface of the searching functions changed to improve performance. The arrays are no longer passed by value. Instead they are passed by reference. That means that you cannot write your searching arrays for */xml22_get_by_regex* inlined anymore. If you wrote:

```
$tag = xml22_get_by_regex( $doc, array('tag' => 'node' ) );
```

you must write now:

```
$search = array('tag' => 'node' );
$tag    = xml22_get_by_regex( $doc, $search );
```

Remember that you usually don‘t need */xml22_get_by_regex*. There are a lot of specialised searching functions available now.

Is there any way to mix (X)HTML into my XML?

Use an explicit CDATA declaration like this:

```
<node><![CDATA[<b>A <i>very important</i> notice.</b>]]></node>
```

This will result in an array element like this:

```
Array (
  [tag] => node,
  [...]
  [content] => <b>A <i>very important</i> notice.</b>
)
```

If you use */xml22_add_content* you can instruct the function to do this for you.

Does caching work persistent?

Not yet. It works only for the runtime of the script.

What about DOM?

This code is absolutely *not* conforming to DOM, though it is aimed to provide a similar functionality. If you have support of DOM compiled into your PHP module (support of *libxml2*), use that. This code may only be used if you have *no* DOM support for your PHP module.

Writing a class conforming to the specification of DOM Core Level 1 completely in PHP may become another story—though it may seem to be useless. As of PHP4 the scripting engine has the possibility to support DOM if it is compiled with support for *libxml2*. Unfortunately, not every user has control over the PHP module and the libraries installed on the system where (s)he runs her/his code. DOM support in PHP is still marked as "experimental" and some providers do not want to provide experimental stuff due to security reasons.

Last but not least, not everybody likes OOP. This one is for those who think of the old times as good times ;)

REQUIREMENTS

PHP = 4.0.0, XML support (libexpat), Perl compatible regular expressions.

RESTRICTIONS

Do not use these functions for *very* huge XML files. The whole file will be read into memory at once and the multidimensional array build up by the parsing function will consume obviously *more* memory than the plain file does. You better split your files into pieces and handle them using the caching mechanism (see */Caching*).

There are some restrictions to the parser. Currently not supported features are: namespaces.

External DTDs are not parsed, i.e. the parser is non-validating. This is a restriction of the XML parser of PHP4 which is actually an interface to `libexpat`.

Currently there is no unicode support.

The code may work for correctly nested data. It may *not* be useful for data including the XML data type ANY i.e. not for tags that are embedded in CDATA (you cannot use this code to process (X)HTML). This may change in the future.

TO DO

- change structure of the array to improve performance and compatibility to standards
- unicode support
- optimize performance of editing functions
- enable persistent caching

CHANGELOG

0.3.4

- Function */xml22_has_children* return the number of children now (added by request).

0.3.3

- Some bugs fixed.

0.3.2

- Extended functionality of */xml22_get_all_of_name* and */xml22_get_all_of_content*.
- Some optimizations, e.g. */xml22_get_by_regex* runs about two times faster now which effects a number of other functions too.
- Fixed bug in */xml22_get_all_of_name*.
- Test scripts may be used at command line now.
- Fixed bug in the profiling function of test scripts.
- Makefiles

0.3.1

- New function */xml22_register_char_entity_set*.
- Minor bugfixes.

0.3.0

- The parser accepts a filename as well as a single string containing well formed XML.
- General Entities are resolved now. The parser may be configured to parse external entities as well (see */xml22_setup*).
- Support for processing instructions, see */xml22_register_pi_func*.
- A new function */xml22_write_document_str* is introduced to write an XML document to a string.
- */xml22_delete_tag* is now able to delete a tag and all of its descendants in one task.
- Support for document fragments.
- Some optimizations in pattern matching.
- For better maintanance, the include file is splitted into five pieces. You still include simply ***xml22.inc*** that manages the additional includes for you. See */Installation*.

0.2.96

- Now */xml22_write_document* encodes predefined entities.

0.2.95

- Now the parser recognizes entities as well as `<!ENTITY> and <!NOTATION> definitions or processing instructions.`
- Fixed some hidden bugs in the parser.
- Added several functions to get particular nodes or information about nodes.
- Added several new editing functions.
- Added */xml22_setup* function.
- Caching now checks if the cached document‘s file has been changed.
- It should be safe now to pass the results of searching functions that return arrays of tags back to those functions. To make this possible some changes to */xml22_get_by_regex* were necessary. The meaning of `$startindex` and `$offset` changed a little.
- Changed pass-by-value to pass-by-reference internally where it makes sense. This influences the API at only one point (see */What means "Cannot pass parameter x by reference"?*).
- The tests run now with the (extended) W3C ‘staff.xml’.
- A new and more common version numbering. The code is still considered developmental, so the major version number is 0. Now the minor number marks the level of development and the third number marks the bugfix level.

0.0.2

- Added editing functions for single tags.
- Added function to write a document.
- Added internal checks to improve robustness.
- Changed the structure of the document array by adding a new element ‘index’.
- Enhanced error reporting. */xml22_parse_error* is obsolete. Now use */xml22_error* which returns an array containing all errors.

- Added basic support for XML comments.

0.0.1

- Parsing of a document into a multidimensional array.
- Basic searching functions.

LICENSE

REPORTING BUGS

If you want to report bugs or if you have any suggestions feel free to send a message to the author. You will find the current e-mail address at http://www.ister.org. Please add to your report the version of **xml22** you use, the version of your PHP module and the type of system it runs on.

AUTHOR

Ingo Schramm
http://code.ister.org