Tokenizer

Introduction

The tokenizer functions provide an interface to the PHP tokenizer embedded in the Zend Engine. Using these functions you may write your own PHP source analyzing or modification tools without having to deal with the language specification at the lexical level.

See also the appendix about tokens.

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Installation

Beginning with PHP 4.3.0 these functions are enabled by default. For older versions you have to configure and compile PHP with *--enable-tokenizer*. You can disable tokenizer support with *--disable-tokenizer*.

The Windows version of PHP has built-in support for this extension. You do not need to load any additional extensions in order to use these functions.

Note
Built-in support for tokenizer is available as of PHP 4.3.0.

Runtime Configuration

This extension has no configuration directives defined in *php.ini*.

Resource Types

This extension has no resource types defined.

Predefined Constants

When the extension has either been compiled into PHP or dynamically loaded at runtime, the tokens listed in List of Parser Tokens are defined as constants.

Examples

Here is a simple example PHP scripts using the tokenizer that will read in a PHP file, strip all comments from the source and print the pure code only.

Example #1 - Strip comments with the tokenizer

```
<?php
/*
* T_ML_COMMENT does not exist in PHP 5.
* The following three lines define it in order to
* preserve backwards compatibility.
*
* The next two lines define the PHP 5 only T_DOC_COMMENT,
* which we will mask as T_ML_COMMENT for PHP 4.
*/
if (!defined('T_ML_COMMENT')) {
  define('T_ML_COMMENT', T_COMMENT);
} else {
  define('T_DOC_COMMENT', T_ML_COMMENT);
}

$source = file_get_contents('example.php');
$tokens = token_get_all($source);

foreach ($tokens as $token) {
  if (is_string($token)) {
      // simple 1-character token
      echo $token;
  } else {
      // token array
      list($id, $text) = $token;

      switch ($id) {
          case T_COMMENT:
          case T_ML_COMMENT: // we've defined this
          case T_DOC_COMMENT: // and this
              // no action on comments
              break;

          default:
              // anything else -> output "as is"
              echo $text;
              break;
      }
  }
}
?>
```

Tokenizer Functions

token_get_all

token_get_all -- Split given source into PHP tokens

Description

array **token_get_all** (string `$source`)

token_get_all() parses the given *source* string into PHP language tokens using the Zend engine's lexical scanner.

For a list of parser tokens, see List of Parser Tokens, or use token_name() to translate a token value into its string representation.

Parameters

source
The PHP source to parse.

Return Values

An array of token identifiers. Each individual token identifier is either a single character (i.e.:;,., >, !, etc...), or a three element array containing the token index in element 0, the string content of the original token in element 1 and the line number in element 2.

Examples

Example #2 - token_get_all() examples

```
<?php
$tokens = token_get_all('<?php echo; ?>'); /* => array(
                                                array(T_OPEN_TAG, '<?php'),

                                                array(T_ECHO, 'echo'),
                                                ';',
                                                array(T_CLOSE_TAG, '?>') );
*/

/* Note in the following example that the string is parsed as T_INLINE_HTML
  rather than the otherwise expected T_COMMENT (T_ML_COMMENT in PHP <5).
  This is because no open/close tags were used in the "code" provided.
  This would be equivalent to putting a comment outside of <?php ?> tags in
a normal file. */
$tokens = token_get_all('/* comment */'); // => array(array(T_INLINE_HTML,
'/* comment */'));
?>
```

ChangeLog

Version	Description
5.2.2	Line numbers are returned in element 2

token_name

token_name -- Get the symbolic name of a given PHP token

Description

string **token_name** (int `$token`)

token_name() gets the symbolic name for a PHP *`token`* value.

Parameters

`token`
The token value.

Return Values

The symbolic name of the given *`token`*. The returned name returned matches the name of the matching token constant.

Examples

Example #3 - token_name() example

```
<?php
// 260 is the token value for the T_REQUIRE token
echo token_name(260);        // -> "T_REQUIRE"

// a token constant maps to its own name
echo token_name(T_FUNCTION); // -> "T_FUNCTION"
?>
```

See Also

- List of Parser Tokens