Глава 7. Комплексная обработка данных

WDDX

Данная группа функций позволяет работать с данными в формате WDDX (http://www.wddx.org/). WDDX (Web Distributed Data Exchange) – это технология для «Распределенного обмена данными в Web». Этот формат (производный от XML), предназначен для переноса данных из одной системы в другую (например, обмена данными между ASP, Perl, ColdFusion и PHP). Он позволяет сохранять не только значения, но и типы, и структуру сложных данных. Поэтому данные WDDX, сериаллизированные в одной системе могут использоваться в другой. Тип данных определяется автоматически, и приводится к одному из следующих значений.

- null — пустые значения.
- bool — булевские значения.
- number — числа (целые и дробные не различаются).
- string — строки.
- dateTime — значения даты и времени.
- array — нумерованные массивы.
- struct — ассоциативные массивы.
- recordset — наборы записей (подобие таблиц).
- binary — двоичные данные (в кодировке Base64).

В отличие от стандартных средств сериаллизации PHP, данные функции используют формат, поддерживаемый большинством программных сред.

```
<?php
$pi = 3.1415926;
$packet_id = wddx_packet_start("Комментарий пакета");
wddx_add_vars($packet_id, "pi");

/* допустим $cities получено из БД */
$cities = array("Austin", "Seattle");
wddx_add_vars($packet_id, "cities");

print $packet = wddx_packet_end($packet_id);
?>
```

Пример выведет:

```
<wddxPacket version='1.0'><header comment='Комментарий пакета'/>
<data><struct><var name='pi'><number>3.1415926</number></var>
<var name='cities'><array length='2'>
<string>Austin</string<string>Seattle</string></array>
</var></struct></data></wddxPacket>
```

wddx_serialize_value. Занесение одного значения в пакет WDDX

```
string wddx_serialize_value (mixed var [, string comment])
```

Создает пакет WDDX из значения одной переменной `var`, добавляя в заголовок пакета комментарий `comment`, и возвращает полученный пакет.

```
<?php
print wddx_serialize_value("WDDX packet example", "PHP packet");
?>
```

Пример выведет:

```
<wddxPacket version='1.0'><header comment='PHP packet'/><data>
<string>WDDX packet example</string></data></wddxPacket>
```

wddx_serialize_vars. Создание пакета WDDX из нескольких значений

```
string wddx_serialize_vars (mixed var_name [, mixed ...])
```

```
<?php
$a = 1;
$b = 5.5;
$c = array("blue", "orange", "violet");
$d = "colors";

$clvars = array("c", "d");
print wddx_serialize_vars("a", "b", $clvars);
?>
```

Пример выведет:

```
<wddxPacket version='1.0'><header/><data><struct>
<var name='a'><number>1</number></var>
<var name='b'><number>5.5</number></var>
<var name='c'><array length='3'><string>blue</string>
   <string>orange</string><string>violet</string></array></var>
<var name='d'><string>colors</string></var>
</struct></data></wddxPacket>
```

wddx_packet_start. Начать новый пакет WDDX

```
int wddx_packet_start ([string comment])
```

Используется для начального создания пакета WDDX с автоинкрементным добавлением переменных. Данные затем добавляются функцией `wddx_add_vars()`, после чего пакет должен быть завершен с помощью `wddx_packet_end()`.

wddx_packet_end. Завершить пакет WDDX

```
string wddx_packet_end (int packet_id)
```

Возвращает содержимое пакета.

wddx_add_vars. Добавление следующей переменной в пакет WDDX

```
wddx_add_vars (int packet_id, mixed name_var [, mixed ...])
```

Используется для последовательного добавления данных в пакет `packet_id`, созданный функцией `wddx_packet_start()`.

wddx_deserialize. Распаковка пакета WDDX

```
mixed wddx_deserialize (string packet)
```

Возвращает данные такого типа, которые были сериаллизированны.

DOM XML

Эти функции доступны при компиляции PHP с опцией --with-dom=[DIR], используя библиотеку GNOME xml (не ниже libxml-2.0.0).

Модуль определяет следующие константы (типы элементов XML):

- 1 — XML_ELEMENT_NODE.
- 2 — XML_ATTRIBUTE_NODE.
- 3 — XML_TEXT_NODE.
- 4 — XML_CDATA_SECTION_NODE.
- 5 — XML_ENTITY_REF_NODE.
- 6 — XML_ENTITY_NODE.
- 7 — XML_PI_NODE.
- 8 — XML_COMMENT_NODE.
- 9 — XML_DOCUMENT_NODE.
- 10 — XML_DOCUMENT_TYPE_NODE.
- 11 — XML_DOCUMENT_FRAG_NODE.
- 12 — XML_NOTATION_NODE.

Модуль использует следующие классы для работы с интересом DOM документов XML: `DomDocument`, `DomNode`, `Dtd`, `DomAttribute`, `DomNamespace`, `XpathContext`, `XpathObject`.

Рекомендуется использовать объектный интерфейс, но возможно использовать и функции с префиксом `domxml_`, как аналоги методов DOM объектов.

Класс **`DomDocument`**

- метод `root()` – аналог `domxml_root()`
- метод `children()` – аналог `domxml_children()`
- метод `add_root()` – аналог `domxml_add_root()`
- метод `dtd()` – аналог `domxml_intdtd()`
- метод `dumpmem()` – аналог `domxml()`
- метод `xpath_init()` – аналог `xpath_init()`
- метод `xpath_new_context()` – аналог `xpath_new_context()`
- метод `xptr_new_context()` – аналог `xptr_new_context()`
- свойство `doc` – сам объект класса DomDocument
- свойство `name` – наименование документа
- свойство `url` – адрес документа
- свойство `version` – версия of XML
- свойство `encoding` – название кодировки
- свойство `standalone` – 1, если файл одиночный
- свойство `type` – см. список констант
- свойство `compression`– 1, если файл сжатый
- свойство `charset` – номер кодировки

Класс **`DomNode`**

- метод `lastchild()` – аналог `domxml_last_child()`
- метод `children()` – аналог `domxml_children()`
- метод `parent()` – аналог `domxml_parent()`
- метод `new_child()` – аналог `domxml_new_child()`
- метод `get_attribute()` – аналог `domxml_get_attribute()`
- метод `set_attribute()` – аналог `domxml_set_attribute()`
- метод `attributes()` – аналог `domxml_attributes()`
- метод `node()` – аналог `domxml_node()`

- метод `set_content()` – аналог `domxml_set_content()`
- свойство `node` – сам объект класса DomNode
- свойство `type` – см. список констант
- свойство `name` string
- свойство `content` string

```
<?php
$doc = new_xmldoc("1.0");           // DomDocument
$root = $doc->add_root("HTML");      // DomNode
$head = $root->new_child("HEAD", "");
$head->new_child("TITLE", "Hier der Titel");
echo $doc->dumpmem();
?>
```

Скрипт выводит следующее:

```
<?xml version="1.0"?>
<HTML><HEAD><TITLE>Hier der Titel</TITLE></HEAD></HTML>
```

new_xmldoc. Создание нового документа DOM XML

`object` **`new_xmldoc`** `(string version)`

Возвращает объект класса «DomDocument».

Синоним функции `domxml_new_xmldoc()`.

xmldoc. Создание DOM объекта из документа XML

`object` **`xmldoc`** `(string strXML)`

Интерпретирует строку, содержащую документ XML и возвращает объект класса «DomDocument.

xmldocfile. Создание DOM объекта из файла XML

`object` **`xmldocfile`** `(string XMLfilename)`

Интерпретирует файл, содержащий документ XML и возвращает объект класса «DomDocument».

xmltree. Создание дерева объектов php из документа XML

`object` **`xmltree`** `(string strXML)`

```
$sXML='<?xml version="1.0"?>
<HTML><HEAD Language="ge"><TITLE>Title OF DOC</TITLE></HEAD></HTML>';
var_dump(xmltree($sXML));
```

Будет выведено следующее:

```
object(DomDocument)(5) {
  ["version"]=>       string(3) "1.0"
  ["standalone"]=>     int(-1)
  ["type"]=>        int(9)  XML_DOCUMENT_NODE
  ["children"]=>     array(1) {
    [0]=>       &object(DomNode)(5) {
      ["name"]=>      string(4) "HTML"
      ["type"]=>      int(1)
      ["content"]=>   string(12) "Title OF DOC"
      ["node"]=>      resource(6) of type (domxml node)
      ["children"]=>  array(1) {
        [0]=>        object(DomNode)(6) {
          ["name"]=>          string(4) "HEAD"
          ["type"]=>          int(1)
          ["content"]=>       string(12) "Title OF DOC"
          ["node"]=>          resource(7) of type (domxml node)
          ["attributes"]=>    array(1) {
            [0]=>             object(DomAttribute)(2) {
              ["name"]=>      string(8) "Language"
              ["children"]=>  array(1) {
                [0]=>         object(DomNode)(4) {
                  ["name"]=>      string(4) "text"
                  ["type"]=>      int(3)
                  ["content"]=>   string(2) "ge"
                  ["node"]=>      resource(8) of type (domxml node)
          ["children"]=>          array(1) {
            [0]=>            object(DomNode)(5) {
              ["name"]=>     string(5) "TITLE"
              ["type"]=>     int(1)
              ["content"]=>  string(12) "Title OF DOC"
              ["node"]=>     resource(9) of type (domxml node)
              ["children"]=> array(1) {
                [0]=>        object(DomNode)(4) {
                  ["name"]=>       string(4) "text"
                  ["type"]=>       int(3)
                  ["content"]=>    string(12) "Title OF DOC"
                  ["node"]=>       resource(10) of type (domxml node)
  ["root"]=>  &object(DomNode)(5) {
    ["name"]=>       string(4) "HTML"
    ["type"]=>       int(1)
    ["content"]=>       string(12) "Title OF DOC"
    ["node"]=>       resource(6) of type (domxml node)
    ["children"]=>      array(1) {
      [0]=>            object(DomNode)(6) {
        ["name"]=>        string(4) "HEAD"
        ["type"]=>        int(1)
        ["content"]=>     string(12) "Title OF DOC"
```

```
["node"]=>        resource(7) of type (domxml node)
["attributes"]=>  array(1) {
  [0]=>           object(DomAttribute)(2) {
    ["name"]=>       string(8) "Language"
    ["children"]=>    array(1) {
      [0]=>           object(DomNode)(4) {
        ["name"]=>    string(4) "text"
        ["type"]=>    int(3)
        ["content"]=> string(2) "ge"
        ["node"]=>    resource(8) of type (domxml node)
["children"]=>        array(1) {
  [0]=>           object(DomNode)(5) {
    ["name"]=>    string(5) "TITLE"
    ["type"]=>    int(1)
    ["content"]=> string(12) "Title OF DOC"
    ["node"]=>    resource(9) of type (domxml node)
    ["children"]=> array(1) {
      [0]=>           object(DomNode)(4) {
        ["name"]=>    string(4) "text"
        ["type"]=>    int(3)
        ["content"]=> string(12) "Title OF DOC"
        ["node"]=>    resource(10) of type (domxml node)
```

Интерфейс DOM

domxml_root. Получение корневого элемента документа XML

`object` **`domxml_root`** `(resource DomDocument)`

Возвращает объект класса «DomNode».

```
if($dom = xmldoc($xmlstr))    $root = $dom->root();
```

domxml_add_root. Создание корневого элемента DOM XML

`resource` **`domxml_add_root`** `(resource DomDocument, string name)`

Возвращает объект класса «DomNode», созданный внутри объекта `DomDocument`.

domxml_dumpmem. Создание XML документа из объекта DOM

`string` **`domxml_dumpmem`** `(resource DomDocument)`

Возвращает текстовое представление документа.

domxml_children. Получение массива вложенных объектов раздела

`array` **`domxml_children`** `(object node)`

Возвращает массив дочерних объектов объекта класса " DomNode».

```
$doc = new_xmldoc("1.0");            // DomDocument
$root = $doc->add_root("ML");          // DomNode <ML></ML>
$root->new_child("H1", "1212");      //  <H1>1212</H1>
$p=$root->new_child("Pp", "Tttt tt"); //  <Pp>Tttt tt</Pp>
$p->set_attribute("Id","8");         //  <Pp Id="8">

echo $doc->dumpmem();
$d=domxml_children($doc->root());      // $root, <ML></ML>
var_dump($d);
$d=domxml_children($d[1]);         // <Pp Id="8">Tttt tt</Pp>
var_dump($d);                     // "Tttt tt"
$d=domxml_children($d[0]);         // = bool(false)
```

Пример выводит следующее:

```
<?xml version="1.0"?>
<ML><H1>1212</H1><Pp Id="8">Tttt tt</Pp></ML>

array(2) {
  [0]=>  object(DomNode)(4) {
    ["name"]=>      string(2) "H1"
    ["content"]=> string(4) "1212"
    ["node"]=>    resource(6) of type (domxml node)
    ["type"]=>    int(1)
  }
  [1]=>  object(DomNode)(4) {
    ["name"]=>    string(2) "Pp"
    ["content"]=> string(7) "Tttt tt"
    ["node"]=>    resource(7) of type (domxml node)
    ["type"]=>    int(1)
  }
}

array(1) {  [0]=>  object(DomNode)(4) {
    ["name"]=>      string(4) "text"
    ["content"]=>   string(7) "Tttt tt"
    ["node"]=>      resource(8) of type (domxml node)
    ["type"]=>      int(3)
  }
}
```

domxml_new_child. Создание секции тега

`resource` **`domxml_new_child`** `(string name, string content)`

Возвращает объект класса «DomNode», созданный внутри объекта «DomNode».

domxml_attributes. Получение атрибутов узла

```
array domxml_attributes (resource node)
```

Возвращает объект класса «DomDocument».

```
$doc = new_xmldoc("1.0");            // DomDocument
$root = $doc->add_root("ML");        // DomNode
$h=$root->new_child("H1", "1212");
$p=$root->new_child("Pp", "Tttt tt");
$p->set_attribute("Id","8");
$p->set_attribute("Bold","");

echo $doc->dumpmem();
var_dump($p->attributes());
var_dump($h->attributes());      //
var_dump($p->getattr("Id"));
```

Пример выведет следующее:

```
<?xml version="1.0"?>
<ML><H1>1212</H1><Pp Id="8" Bold="">Tttt tt</Pp></ML>

array(2) {
  [0]=>  object(DomAttribute)(2) {
    ["name"]=>    string(2) "Id"
    ["children"]=>   array(1) {
      [0]=>      object(DomNode)(4) {
        ["name"]=>        string(4) "text"
        ["type"]=>        int(3)
        ["content"]=>     string(1) "8"
        ["node"]=>        resource(8) of type (domxml node)
      }
    }
  }
  [1]=>  object(DomAttribute)(1) {
    ["name"]=>    string(4) "Bold"
  }
}
bool(false)
string(1) "8"
```

domxml_getattr. Получение атрибута узла

```
object domxml_getattr (resource node, string name)
```

Возвращает значение атрибута узла. В документации функция почему-то значится под именем `domxml_get_attribute()`. См. пример выше.

domxml_set_attribute. Установка атрибута узла

```
object domxml_set_attribute (resource node, string name, string
value)
```

См. пример выше. Имена атрибутов учитывают регистр символов.

XML интерпретация

XML (eXtensible Markup Language) – формат обмена структурированными данными в пространстве Web; стандарт, определенный Консорциумом World Wide Web (W3C). Информацию о XML и связанных технологиях см. на http://www.w3.org/XML/. Для понимания материала главы необходимо разбираться в синтаксисе XML.

Для работы с документами XML используются два способа:

1. Использование объектной модели документа (DOM), которая удобна для работы с документом в целом, но расходует больше ресурсов.
2. Интерпретация документа на основе модели событий, удобная при однократном просмотре документа, и менее требовательная к ресурсам (что важно при работе с объемными документами).

Первый способ рассмотрен в предыдущей главе. Функции, описанные в данной главе, реализуют второй способ обработки. Событийная модель обработки (интерпретации) подразумевает следующую схему функционирования программы:

- Устанавливаются функции обработки элементов документа различного типа (например, маркеров, данных, инструкций, и т. д.)
- Инициализируется и запускается интерпретатор, который последовательно просматривает документ и запускает соответствующие обработчики данных.
- Функции-обработчики определенным образом обрабатывают передаваемые им фрагменты документа.

Заметьте, что интерпретатор предназначен для обработки документа, но не для его проверки; документ должен соответствовать грамматике XML (и правилам DTD).

В PHP интерпретатор XML использует библиотеку expat (ее также использует Apache-1.3.9 и последующие версии), см.: http://www.jclark.com/xml/. Скомпилировать PHP с поддержкой expat можно используя опцию --with-xml.

Для документов поддерживаются кодировки: US-ASCII, ISO-8859-1 (по ум.), UTF-8 (UTF-16 не поддерживается). Разделяют исходную и целевую кодировки. Исходная кодировка – это набор символов документа (ее можно изменять в процессе интерпретации). (Внутри PHP символы всегда хранятся в кодировке UTF-8, позволяя использовать символы размером до 21 бита). В функции-обработчики данные передаются в целевой кодировке (для всех типов данных). При нахождении в документе символа не соответствующего исходной кодировке выдается ошибка; а если символ не может быть представлен в целевой кодировке, то он заменяется на знак вопроса.

Интерпретация (по умолчанию) не учитывает регистр имен тегов, то есть в функции обработки символы передаются преобразованные к верхнему регистру. Для отключения этого используйте:

```
xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, 0);
```

Для интерпретатора можно определить следующие виды функций обработчиков:

`xml_set_element_handler()` – обработчики начальных и конечных тегов.

`xml_set_character_data_handler()` – текст между тегами (с учетом пробелов).

`xml_set_processing_instruction_handler()` – инструкции обработки. (например, `<?php ?>`, и подобные; но инструкция `<?xml ?>` зарезервирована).

`xml_set_default_handler()` – обработчик по умолчанию, используемый при невозможности использования иного обработчика.

`xml_set_unparsed_entity_decl_handler()` – обработчик необрабатываемых (NDATA) данных.

`xml_set_notation_decl_handler()` – обработчик нотаций.

`xml_set_external_entity_ref_handler()` – обработчик внешних ссылок.

Коды ошибок

Модулем интерпретатора XML определяются следующие константы кодов ошибок (возвращаемые функцией `xml_parse()`):

- `XML_ERROR_NONE`
- `XML_ERROR_NO_MEMORY`
- `XML_ERROR_SYNTAX`
- `XML_ERROR_NO_ELEMENTS`
- `XML_ERROR_INVALID_TOKEN`
- `XML_ERROR_UNCLOSED_TOKEN`
- `XML_ERROR_PARTIAL_CHAR`
- `XML_ERROR_TAG_MISMATCH`
- `XML_ERROR_DUPLICATE_ATTRIBUTE`
- `XML_ERROR_JUNK_AFTER_DOC_ELEMENT`
- `XML_ERROR_PARAM_ENTITY_REF`
- `XML_ERROR_UNDEFINED_ENTITY`
- `XML_ERROR_RECURSIVE_ENTITY_REF`
- `XML_ERROR_ASYNC_ENTITY`

- XML_ERROR_BAD_CHAR_REF
- XML_ERROR_BINARY_ENTITY_REF
- XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF
- XML_ERROR_MISPLACED_XML_PI
- XML_ERROR_UNKNOWN_ENCODING
- XML_ERROR_INCORRECT_ENCODING
- XML_ERROR_UNCLOSED_CDATA_SECTION
- XML_ERROR_EXTERNAL_ENTITY_HANDLING

Примеры

Следующие примеры демонстрируют возможности интерпретации документов XML.

Распечатка структуры XML документа

Скрипт выводит структуру на основе поиска открывающих и закрывающих тегов, используя в качестве форматирования отступы.

```
$file = "data.xml";
$depth = array();

function startElement($parser, $name, $attrs) {
    global $depth;
    for ($i = 0; $i < $depth[$parser]; $i++) {
        print "  ";
    }
    print "$name\n";
    $depth[$parser]++;
}

function endElement($parser, $name) {
    global $depth;
    $depth[$parser]--;
}

$xml_parser = xml_parser_create();
xml_set_element_handler($xml_parser, "startElement", "endElement");
if (!($fp = fopen($file, "r"))) {
    die("could not open XML input");
}

while ($data = fread($fp, 4096)) {
    if (!xml_parse($xml_parser, $data, feof($fp))) {
        die(sprintf("XML error: %s at line %d",
                    xml_error_string(xml_get_error_code($xml_parser)),
```

```
                    xml_get_current_line_number($xml_parser)));
    }
}
xml_parser_free($xml_parser);
```

Преобразование тегов XML в HTML

```
$file = "data.xml";
$map_array = array(
    "BOLD"     => "B",
    "EMPHASIS" => "I",
    "LITERAL"  => "TT"
);

function startElement($parser, $name, $attrs) {
    global $map_array;
    if ($htmltag = $map_array[$name]) {
        print "<$htmltag>";
    }
}
function endElement($parser, $name) {
    global $map_array;
    if ($htmltag = $map_array[$name]) {
        print "</$htmltag>";
    }
}
function characterData($parser, $data) {
    print $data;
}

$xml_parser = xml_parser_create();
// отключить чувствительность к регистру
xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, true);
xml_set_element_handler($xml_parser, "startElement", "endElement");
xml_set_character_data_handler($xml_parser, "characterData");
if (!($fp = fopen($file, "r"))) {
    die("could not open XML input");
}

while ($data = fread($fp, 4096)) {
    if (!xml_parse($xml_parser, $data, feof($fp))) {
        die(sprintf("XML error: %s at line %d",
                    xml_error_string(xml_get_error_code($xml_parser)),
                    xml_get_current_line_number($xml_parser)));
    }
}
```

```
xml_parser_free($xml_parser);
```

Обработка специальных конструкций XML

Пример демонстрирует обработку подключенных внешних файлов и инструкций обработки. Использует при выводе разметку HTML.

```
$file = "xmltest.xml";

function trustedFile($file) {
    // исполнять код можно только в собственных файлах
    if (!eregi("^([a-z]+)://", $file)
        && fileowner($file) == getmyuid()) {
            return true;
    }
    return false;
}

function startElement($parser, $name, $attribs) {
    print "&lt;<font color=\"#0000cc\">$name</font>";
    if (sizeof($attribs)) {
        while (list($k, $v) = each($attribs)) {
            print " <font color=\"#009900\">$k</font>=\"<font
                   color=\"#990000\">$v</font>\"";
        }
    }
    print "&gt;";
}

function endElement($parser, $name) {
    print "&lt;/<font color=\"#0000cc\">$name</font>&gt;";
}

function characterData($parser, $data) {
    print "<b>$data</b>";
}

function PIHandler($parser, $target, $data) {
    switch (strtolower($target)) {
        case "php":
            global $parser_file;
            // Проверим допустимо ли выполнить код PHP из документа
            // или ограничиться его простым отображением
            if (trustedFile($parser_file[$parser]))
                eval($data);
            else
                printf("Код PHP: <i>%s</i>",htmlspecialchars($data));
```

```
            break;
    }
}

function defaultHandler($parser, $data) {
    if (substr($data, 0, 1) == "&" && substr($data, -1, 1) == ";") {
        printf('<font color="#aa00aa">%s</font>',
                htmlspecialchars($data));
    } else {
        printf('<font size="-1">%s</font>',
                htmlspecialchars($data));
    }
}

function externalEntityRefHandler($parser, $openEntityNames,
         $base, $systemId, $publicId) {
    if ($systemId) {
        if (!list($parser, $fp) = new_xml_parser($systemId)) {
            printf("Необрабатываемая секция %s в позиции %s\n",
                  $openEntityNames, $systemId);
            return false;
        }
        while ($data = fread($fp, 4096)) {
           if (!xml_parse($parser, $data, feof($fp))) {
               printf("XML error: %s в стороке %d '%s' \n",
                xml_error_string(xml_get_error_code($parser)),
                xml_get_current_line_number($parser), $openEntityNames);
               xml_parser_free($parser);
               return false;
            }
        }
        xml_parser_free($parser);
        return true;
    }
    return false;
}

function new_xml_parser($file) {
    global $parser_file;

    $xml_parser = xml_parser_create();
    xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, 1);
    xml_set_element_handler($xml_parser, "startElement", "endElement");
    xml_set_character_data_handler($xml_parser, "characterData");
    xml_set_processing_instruction_handler($xml_parser, "PIHandler");
    xml_set_default_handler($xml_parser, "defaultHandler");
    xml_set_external_entity_ref_handler($xml_parser,
```

```
                        "externalEntityRefHandler");

    if (!($fp = @fopen($file, "r"))) {
        return false;
    }
    if (!is_array($parser_file)) {
        settype($parser_file, "array");
    }
    $parser_file[$xml_parser] = $file;
    return array($xml_parser, $fp);
}
//---------------------------------------------------------
if (!(list($xml_parser, $fp) = new_xml_parser($file))) {
    die("could not open XML input");
}

print "<pre>";
while ($data = fread($fp, 4096)) {
    if (!xml_parse($xml_parser, $data, feof($fp))) {
        die(sprintf("XML error: %s at line %d\n",
                    xml_error_string(xml_get_error_code($xml_parser)),
                    xml_get_current_line_number($xml_parser)));
    }
}
print "</pre>";
print "parse complete\n";
xml_parser_free($xml_parser);
```

Содержимое файла xmltest.xml

```
<?xml version='1.0'?>
<!DOCTYPE chapter SYSTEM "/just/a/test.dtd" [
<!ENTITY plainEntity "FOO entity">
<!ENTITY systemEntity SYSTEM "xmltest2.xml">
]>
<chapter>
 <TITLE>Title &plainEntity;</TITLE>
 <para>
  <informaltable>
   <tgroup cols="3">
    <tbody>
     <row><entry>a1</entry><entry morerows="1">b1</entry>
            <entry>c1</entry></row>
     <row><entry>a2</entry><entry>c2</entry></row>
     <row><entry>a3</entry><entry>b3</entry><entry>c3</entry></row>
    </tbody>
   </tgroup>
  </informaltable>
```

```
 </para>
 &systemEntity;
 <sect1 id="about">
  <title>About this Document</title>
  <para>
   <!-- this is a comment -->
   <?php print 'Hi! PHP version '.phpversion(); ?>
  </para>
 </sect1>
</chapter>
```

Подключаемый файл xmltest2.xml:

```
<?xml version="1.0"?>
<!DOCTYPE foo [
<!ENTITY testEnt "test entity">
]>
<foo>
   <element attrib="value"/>
   &testEnt;
   <?php print "This is PHP code."; ?>
</foo>
```

xml_parser_create. Инициализация интерпретатора XML

```
int xml_parser_create ([string encoding])
```

Необязательным аргументом можно указать кодировку, которую следует использовать: ISO-8859-1 (по ум.), US-ASCII, UTF-8.

Возвращает дескриптор созданного интерпретатора (используемый последующими функциями), или `false` при ошибке.

xml_set_object. Разрешение использования интерпретатора XML внутри объекта

```
void xml_set_object (int parser, object &object)
```

Функция позволяет использовать в качестве всех функций обработчиков методы объекта `object`.

```
<?php
class xml  {
var $parser;

function xml() {  // конструктор
    $this->parser = xml_parser_create();
    xml_set_object($this->parser,&$this);
    xml_set_element_handler($this->parser,"tag_open","tag_close");
```

```
    xml_set_character_data_handler($this->parser,"cdata");
}

function parse($data) {
    xml_parse($this->parser,$data);
}

function tag_open($parser,$tag,$attributes) {
    echo "** Tag open: ";    var_dump($tag,$attributes);
}

function cdata($parser,$cdata) {
    echo "  ++ Data: ";       var_dump($cdata);
}

function tag_close($parser,$tag) {
     echo "** Tag close: ";      var_dump($tag);
}

} // end of class xml

$xml_parser = new xml();
$xml_parser->parse('<A ID="99">PHP <z a="d"/>aaa</A>');
?>
```

При запуске пример выведет:

```
** Tag open: string(1) "A" array(1) { ["ID"]=> string(2) "99"}
++ Data: string(4) "PHP "
** Tag open: string(1) "Z" array(1) { ["A"]=> string(1) "d" }
** Tag close: string(1) "Z"
++ Data: string(3) "aaa"
** Tag close: string(1) "A"
```

xml_set_element_handler. Назначение обработчиков открывающего и закрывающего тега

`int` **`xml_set_element_handler`** `(int parser, string startElementHandler, string endElementHandler)`

В аргументах `startElementHandler` и `endElementHandler` указываются имена пользовательских функций, вызываемых во время интерпретации (при вызове `xml_parse()`), когда интерпретатор встречает открывающие и закрывающие теги.

Функции должны принимать следующие аргументы:

`startElementHandler (int parser, string name, array attribs)`

`endElementHandler (int parser, string name)`

Аргумент `name` содержит имя тега, `attribs` – ассоциативный массив, содержащий атрибуты тега (если они имеются).

xml_set_character_data_handler. Назначение обработчика данных

```
int xml_set_character_data_handler (int parser, string handler)
```

Устанавливает пользовательскую функцию с именем `handler`, как обработчик данных документа. Данными считается все то, что находится между тегами, включая пробелы. Эта функция будет вызываться во время интерпретации (при вызове `xml_parse()`). Она должна соответствовать прототипу:

```
handler (int parser, string data)
```

В аргументе `data` функция получает текущий блок данных.

xml_set_processing_instruction_handler. Назначение обработчика инструкций обработки

```
int xml_set_processing_instruction_handler (int parser, string handler)
```

Инструкции обработки имеют следующий формат:

```
<?target data... ?>
```

Используя этот формат можно вставлять в XML документы PHP код, но последовательность, обозначающая конечный тег (?>) не должна встречаться в середине кода, иначе оставшаяся часть кода будет рассматриваться интерпретатором как обычные данные. Функция, устанавливаемая в качестве обработчика должна соответствовать следующему прототипу:

```
handler (int parser, string target, string data)
```

В аргументе `target` функция получает маркер, определяющий тип кода (это может быть не только «php», но и другой, кроме зарезервированного типа «xml»). В аргументе `data` передается текст всего кода, который содержится внутри тега.

xml_set_default_handler. Установка обработчика по умолчанию

```
int xml_set_default_handler (int parser, string handler)
```

Устанавливаемым обработчиком будут обрабатываться все данные, которые не могут быть обработаны иным зарегистрированным обработчиком. Функция, обработчика должна соответствовать следующему прототипу:

```
handler (int parser, string data)
```

В аргументе `data` функция получает блок данных для обработки.

xml_set_unparsed_entity_decl_handler. Установка обработчика необрабатываемых данных

```
int xml_set_unparsed_entity_decl_handler (int parser, string handler)
```

Данные такого типа (NDATA) определяются спецификацией XML 1.0 (раздел 4.2.2), и имеют формат подобный следующему:

```
<!ENTITY name {publicId | systemId} NDATA notationName>
```

Функция, обработчика должна соответствовать следующему прототипу:

```
handler (int parser, string entityName, string base, string systemId, string publicId, string notationName)
```

В аргументе `entityName` функция получает тип тега, в `base` в настоящее время всегда содержится пустая строка. В аргументах `systemId` и `publicId` содержаться соответственно системный и публичный внешние идентификаторы. Аргумент `notationName` содержит имя нотации (см. функцию `xml_set_notation_decl_handler()`).

xml_set_notation_decl_handler. Установка обработчика объявлений нотаций

```
int xml_set_notation_decl_handler (int parser, string handler)
```

Нотации (являющиеся частью документов DTD) описаны в спецификации XML 1.0 (раздел 4.7), имеют следующий формат:

```
<!NOTATION name {systemId | publicId}>
```

Функция, обработчика должна соответствовать следующему прототипу:

```
handler (int parser, string notationName, string base, string systemId, string publicId)
```

В аргументе `notationName` функция получает имя нотации, в `base` в настоящее время всегда содержится пустая строка. В аргументах `systemId` и `publicId` содержаться соответственно системный и публичный внешние идентификаторы.

xml_set_external_entity_ref_handler. Установка обработчика внешних ссылок

```
int xml_set_external_entity_ref_handler (int parser, string handler)
```

Функция, обработчика нотаций, содержащих внешние ссылки, должна возвращать целочисленное значение и соответствовать следующему прототипу:

```
int handler (int parser, string openEntityNames, string base, string systemId, string publicId)
```

Если обработчик возвращает значение `false` (или не возвращает никакого), интерпретатор XML прекращает интерпретацию и функция `xml_get_error_code()` возвращает значение `XML_ERROR_EXTERNAL_ENTITY_HANDLING`.

В аргументе `openEntityNames`, функция получает список имен открываемых для рекурсивной интерпретации (в виде строки, где разделителями являются пробелы), в `base` в настоящее время всегда содержится пустая строка. В аргументах `systemId`

и publicId содержаться соответственно системный и публичный внешние идентификаторы.

См. пример этой главы «Обработка специальных конструкций XML».

xml_parse. Начать интерпретацию документа XML

```
int xml_parse (int parser, string data [, int isFinal])
```

Функция позволяет обрабатывать документ XML по частям (многократно вызывая данную функцию и передавая каждый блок данных в аргументе data), тогда при обработке последней части документа в аргументе isFinal следует передать значение true.

До начала интерпретации следует инициализировать интерпретатор parser и установить все функции обработчики (которые будет вызывать данная функция). Функция возвращает значение true, если интерпретация блока документа прошла успешно. В случае ошибки возвращается значение false, и затем может быть получена информация об ошибке с помощью функций xml_get_error_code(), xml_error_string(), xml_get_current_line_number(), xml_get_current_column_number(), xml_get_current_byte_index().

xml_get_error_code. Получение кода ошибки интерпретатора XML

```
int xml_get_error_code (int parser)
```

Перечисление возвращаемых кодов ошибок (констант) см. в начале главы.

xml_error_string. Получение описания ошибки по ее коду

```
string xml_error_string (int code)
```

Функция позволяет получить в строке описание кода ошибки (возвращаемое функцией xml_get_error_code()). Если код недействителен, возвращается false.

xml_get_current_line_number. Номер текущей интерпретируемой строки документа

```
int xml_get_current_line_number (int parser)
```

xml_get_current_column_number. Номер обрабатываемого байта в текущей интерпретируемой строке документа

```
int xml_get_current_column_number (int parser)
```

xml_get_current_byte_index. Получение позиции текущего байта документа

```
int xml_get_current_byte_index (int parser)
```

Нумерация байтов начинается с 0.

xml_parse_into_struct. Занесение документа XML в структурированный массив

`int` **`xml_parse_into_struct`** `(int parser, string data, array &values, array &index)`

Функция заносит XML документ в 2 параллельных структуры: массив `values` содержит структуру тегов и их содержимое, а `index` содержит вспомогательные индексы для облегчения нахождения начальных и конечных тегов в первом массиве (массивы следует передавать по ссылке).

```
$simple = "<para><note>simple note</note></para>";
$p = xml_parser_create();
xml_parse_into_struct($p,$simple,&$vals,&$index);
xml_parser_free($p);
echo "Index ";
print_r($index);
echo "\nVals ";
print_r($vals);
```

При запуске скрипт выводит:

```
Index Array(
[PARA] => Array (   [0] => 0 [1] => 2 )
[NOTE] => Array (   [0] => 1 )
)

Vals Array(
[0] => Array (   [tag] => PARA
          [type] => open
          [level] => 1
)
[1] => Array (   [tag] => NOTE
                  [type] => complete
                  [level] => 2
                  [value] => simple note
)
[2] => Array (   [tag] => PARA
           [type] => close
              [level] => 1
)
)
```

Интерпретация сложных документов становится громоздкой при использовании обработчиков. Хотя эта функция и не создает объект подобный DOM, она генерирует структуру, которую легко преобразовать в древовидную. Можно, например, создавать объекты представляющие данные. Следующий пример демонстрирует создание объектов из xml базы данных химических элементов:

Файл. moldb.xml

```
<?xml version="1.0"?>
<moldb>
<molecule>
<name>Alanine</name>
<symbol>ala</symbol>
<code>A</code>
<type>hydrophobic</type>
</molecule>
<molecule>
<name>Lysine</name>
<symbol>lys</symbol>
<code>K</code>
<type>charged</type>
</molecule>
</moldb>
```

Скрипт parsemoldb.php, интерпретатор файла moldb.xml:

```
<?php

class AminoAcid {    // класс аминокислот
var $name;    // aa имя
var $symbol;       // трех буквенный символ
var $code;       // одно буквенный код
var $type;       // свойства

function AminoAcid ($aa) {
foreach ($aa as $k=>$v)
$this->$k = $aa[$k];
}
}

function readDatabase($filename) {
// read the xml database of aminoacids
$data = implode("",file($filename));
$parser = xml_parser_create();
xml_parser_set_option($parser,XML_OPTION_CASE_FOLDING,0);
xml_parser_set_option($parser,XML_OPTION_SKIP_WHITE,1);
xml_parse_into_struct($parser,$data,&$values,&$tags);
xml_parser_free($parser);

// loop through the structures
foreach ($tags as $key=>$val) {
if ($key == "molecule") {
$molranges = $val;
for ($i=0; $i < count($molranges); $i+=2) {
```

```
$offset = $molranges[$i] + 1;
$len = $molranges[$i + 1] - $offset;
$tdb[] = parseMol(array_slice($values, $offset, $len));
}
} else {
continue;
}
}
return $tdb;
}

function parseMol($mvalues) {
for ($i=0; $i < count($mvalues); $i++)
$mol[$mvalues[$i]["tag"]] = $mvalues[$i]["value"];
return new AminoAcid($mol);
}

$db = readDatabase("moldb.xml");
echo "** Database of AminoAcid objects: ";
print_r($db);
?>
```

После запуска скрипта, переменная `$db` будет содержать массив AminoAcid объектов, и будет выведено:

```
** Database of AminoAcid objects: Array
(
[0] => aminoacid Object
(
[name] => Alanine
[symbol] => ala
[code] => A
[type] => hydrophobic
)

[1] => aminoacid Object
(
[name] => Lysine
[symbol] => lys
[code] => K
[type] => charged
)
)
```

xml_parser_free. Закрытие интерпретатора XML

`string` **`xml_parser_free`** `(int parser)`

xml_parser_set_option. Установка опции XML интерпретатора

`int` **`xml_parser_set_option`** `(int parser, int option, mixed value)`

В аргументе `option` константой задается устанавливаемая опция, а в `value` – ее новое значение. Возможные опции:

XML_OPTION_CASE_FOLDING (тип значения опции – integer)

Управляет, преобразованием имен тегов в верхний регистр (по ум. 1).

XML_OPTION_TARGET_ENCODING (тип значения опции – string)

Целевая кодировка: «ISO-8859-1», «US-ASCII» или «UTF-8». По умолчанию такая же как и установлена функцией `xml_parser_create()`.

См. также информацию в начале главы.

xml_parser_get_option. Получение опции интерпретатора XML

`mixed` **`xml_parser_get_option`** `(int parser, int option)`

См. также: `xml_parser_set_option()`.

utf8_decode. Преобразование строки UTF-8 в ISO-8859-1

`string` **`utf8_decode`** `(string data)`

См. также: `utf8_encode()`.

utf8_encode. Кодирование строки ISO-8859-1 в UTF-8

`string` **`utf8_encode`** `(string data)`

Возвращает строку в формате UTF-8. UTF-8 – это формат представления текста Unicode, позволяющий представить большее количество символов, чем позволяет кодировка ASCII. Первые 127 символов ASCII в кодировке UTF-8 имеют идентичное представление, а последующие (обычно это символы национальных языков) кодируются следующими способами:

Таблица. Кодировка UTF-8

Размер символа в байтах	Число битов символа	Кодировка
1	7	0bbbbbbb
2	11	110bbbbb 10bbbbbb
3	16	1110bbbb 10bbbbbb 10bbbbbb
4	21	11110bbb 10bbbbbb 10bbbbbb 10bbbbbb

Как видно, размер символа может быть от 1 до 4 байт; b представляет один бит данных.

XSLT

XSLT (Extensible Stylesheet Language (XSL) Transformations) – язык преобразований XML документов в другие XML документы; стандарт определенный Консорциумом World Wide Web (W3C). См. документацию: http://www.w3.org/TR/xslt.

Технология предназначена для разделения содержания и формы представления документов с помощью шаблонов преобразования XSL. Обычно документы XML преобразовывают в формат HTML.

Это расширение использует Sabloton и expat (http://www.gingerall.com/). В UNIX, при компиляции PHP запустите configure с ключом --with-sablot.

xslt_create. Инициализация нового процессора XSL

```
resource xslt_create(void);
```

Функция возвращает дескриптор, используемый последующими XSL функциями.

xslt_free. Закрытие процессора XSLT

```
void xslt_free (resource xh)
```

xslt_openlog. Назначение журнала сообщений процессора XSLT

```
bool xslt_openlog ([resource xh, string logfile, int loglevel])
```

В указанный файл `logfile` будут записываться все сообщения об ошибках.

xslt_closelog. Закрытие и очистка журнала сообщений процессора XSLT

```
bool xslt_closelog (resource xh)
```

xslt_errno. Получение номера текущей ошибки процессора XSLT

```
int xslt_errno ([int xh])
```

xslt_error. Получение описания текущей ошибки

```
mixed xslt_error ([int xh])
```

xslt_fetch_result. Получение преобразованных данных из буфера

```
string xslt_fetch_result ([int xh, string result_name])
```

Если имя буфера не указывается (в `result_name`), то оно подразумевается "/_result».

xslt_output_begintransform. Начать преобразование XSLTс выводом

`void` **`xslt_output_begintransform`** `(string xslt_filename)`

Функция заставляет преобразовывать все выводимые данные с учетом шаблона из файла `xslt_filename`. Затем следует вызвать функцию `xslt_output_endtransform()`.

```
<?php
$xsl_file = "article.xsl";
xslt_output_begintransform($xsl_file);

$doc = new_xmldoc('1.0');
$article = $doc->new_root('article');

$article->new_child('title', 'The History');
$article->new_child('author', 'Peter');
$article->new_child('body', 'Nothing interesting has happened');

echo $doc->dumpmem();

xslt_output_endtransform();
?>
```

xslt_output_endtransform. Завершить преобразование начатое xslt_output_begintransform

`void` **`xslt_output_endtransform`** `(void);`

Эту функцию следует вызвать для того, чтобы вывести преобразованный документ.

xslt_process. Преобразование документа XML с помощью шаблона XSL

`bool` **`xslt_process`** `(string xsl_data, string xml_data, string result)`

В аргументе `xsl_data` указывается текст шаблона XSLT, а в `xml_data` – текст документа XML. Возвращает `true`, или `false` при ошибке. См. также: `xslt_errno()` и `xslt_error()` для получения кодов ошибки.

```
<?php

$xslData = '
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="article">
<table border="1" cellpadding="2" cellspacing="1">
<tr>
<td width="800">
<h2><xsl:value-of select="title"></h2>
```

```
<h3><xsl:value-of select="author"></h3>
<br><xsl:value-of select="body">
</td>
</tr>
</table>
</xsl:template>

</xsl:stylesheet>';

$xmlData = '
<?xml version="1.0"?>
<article>
<title>Learning Language</title>
<author>I. Myself</author>
<body>
Essential phrases:
<br>
<br>
Hello.
       Good day.<br>
</body>
</article>';

if (xslt_process($xslData, $xmlData, $result)) {
echo "Here is the article:<br>\n<br>", $result;
} else {
echo "Ошибка преобразования XSL. \n\tНомер: " . xslt_errno() .
            "\n\tОписание: " . xslt_error() . "\n";
exit;
}
?>
```

xslt_run. Применение XSLT к файлу

```
bool xslt_run ([resource xh, string xslt_file, string xml_data_file, string result, array xslt_params, array xslt_args]]])
```

Преобразовывает файл `xml_data_file`, применяя к нему шаблон из файла `xslt_file`. Шаблон получает доступ к массиву параметров `xslt_params` и аргументов `xslt_args`. Результат XSLT трансформации заносится в именованный буфер (по ум. "/_result»).

xslt_set_sax_handler. Установка обработчиков SAX для процессора XSLT

```
bool xslt_set_sax_handler (resource xh, array handlers)
```

xslt_transform. Выполнение трансформации XSLT

```
bool xslt_transform (string xsl, string xml, string result,
string params, string args, string resultBuffer)
```

Предоставляет расширенный интерфейс библиотеки Sablotron без необходимости использования ресурсов API.