Методические указания по теме
“РАБОТА С ДАННЫМИ - ЧАСТЬ 2”

Работа с XML 2
- Структура XML ... 2
  - Достоинства XML ... 3
  - Недостатки XML ... 3
  - Правильно построенный XML ... 4
- Чтение, обработка и написание XML в PHP ... 4
  - Краткий обзор предпочтительных API ... 4
- SimpleXML ... 4
  - Работа с SimpleXML ... 4
- DOM ... 10

Работа с CSV ... 12
- Общие сведения ... 12
- Чтение CSV файлов ... 12
- Запись в CSV файл ... 13

Работа с .ini ... 14
- Общие сведения ... 14
- Чтение .ini файлов ... 14
- Запись .ini файлов ... 15

Работа с JSON данными ... 16
- Общие сведения ... 16
- Интерфейс для работы с форматом ... 16
  - json_encode ... 16
  - json_last_error ... 17

Работа с XML

Язык Extensible Markup Language (XML) можно назвать и языком разметки, и форматом хранения текстовых данных. Это подмножество языка Standard Generalized Markup Language (SGML); он предоставляет текстовые средства для описания древовидных структур и их применения к информации. XML служит основой для целого ряда языков и форматов, таких как Really Simple Syndication (RSS), Mozilla XML User Interface Language (XUL), Macromedia Maximum eXperience Markup Language (MXML), Microsoft eXtensible Application Markup Language (XAML) и open source-язык Java XML UI Markup Language (XAMJ).

Структура XML

Базовым блоком данных в XML является элемент. Элементы выделяются начальным тегом, таким как <book>, и конечным тегом, таким как </book>. Каждому начальному тегу должен соответствовать конечный тег. Если для какого-то начального тега отсутствует конечный тег, XML-документ оформлен неправильно, и синтаксический анализатор (парсер) не сможет проанализировать его надлежащим образом. Названия тегов обычно отражают тип элемента. Можно ожидать, что элемент book содержит название книги, например, «Большой американский роман». Текст, содержащийся между тегами, включая пробелы, называется символьными данными.

Пример XML документа

```
<books>
 <book>
  <title>Большой американский роман</title>
  <characters>
   <character>
    <name>Клифф</name>
    <desc>отличный парень</desc>
   </character>
   <character>
    <name>Миловидная Женщина</name>
    <desc>редкая красавица</desc>
   </character>
   <character>
    <name>Преданный Пес</name>
    <desc>любит поспать</desc>
   </character>
  </characters>
  <plot>
   Клифф встречает Миловидную Женщину. Преданный Пес спит,
   но просыпается, чтобы облаять почтальона.
  </plot>
  <success type="bestseller">4</success>
  <success type="bookclubs">9</success>
  </book>
</books>
```

Имена XML-элементов и атрибутов могут состоять из латинских букв верхнего (A-Z) и нижнего (a-z) регистров, цифр (0-9), некоторых специальных и неанглийских символов, а также трех знаков пунктуации: дефиса, знака подчеркивания и точки. Другие символы в именах не допускаются.

XML чувствителен к регистру. В приведенном примере <Book> и <book> описывают два разных элемента. Оба имени приемлемы. Однако описание двух разных элементов именами <Book> и <book> нельзя считать разумным решением ввиду высокой вероятности опечаток.

Каждый документ XML содержит один и только один корневой элемент. Корневой элемент — это единственный элемент XML-документа, для которого нет родительского элемента. В приведенном выше примере корневым является элемент <books>. Большинство XML-документов содержат родительские и дочерние элементы. Элемент <books> имеет один дочерний элемент <book>. У элемента <book> четыре дочерних элемента: <title>, <characters>, <plot> и <success>. У элемента <characters> три дочерних элемента, каждый из которых является элементом <character>. У каждого элемента <character> по два дочерних элемента, <name> и <desc>.

Кроме вложенных элементов, что создает отношения родительский-дочерний, XML-элементы могут иметь атрибуты. Это пары имя-значение, присоединенные к начальному тегу элемента. Имена отделяются от значений знаком равенства, =. Значения заключаются в одинарные или двойные кавычки. В листинге 1 элемент <success> имеет два атрибута: "bestseller" и "bookclubs". XML-разработчики практикуют разные подходы к использованию атрибутов. Большую часть информации, содержащейся в атрибуте, можно поместить в дочерний элемент. Некоторые разработчики настаивают на том, чтобы информация атрибутов состояла не из данных, а из метаданных, то есть сведений о данных. Сами данные должны содержаться в элементах. На самом деле решение о том, использовать ли атрибуты, зависит от природы данных и от того, как они извлекаются из XML.

Достоинства XML

Одно из достоинств XML состоит в его относительной простоте. XML-документ можно составить в простом текстовом редакторе или текстовом процессоре, не прибегая к специальным инструментам или ПО. Базовый синтаксис XML состоит из вложенных элементов, некоторые из которых имеют атрибуты и содержание. Обычно элемент начинается открывающим тегом <тег> и заканчивается соответствующим закрывающим тегом </тег >. XML чувствителен к регистру и не игнорирует пробелы и табуляции. Он очень похож на HTML, но, в отличие от HTML, позволяет присваивать тегам имена для лучшего описания свох данных. К числу преимуществ XML относится самодокументирование, читабельный для людей и компьютеров формат, поддержка Unicode, что позволяет создавать документы на разных языках, и простые требования к синтаксису и синтаксическому анализу. К сожалению, в PHP5 поддержка UTF-8 сопряжена с проблемами; это один из тех недостатков, которые привели к разработке PHP6.

Недостатки XML

XML многословен и избыточен, что порождает документы большого объема, занимающие много дискового пространства и сетевых ресурсов. Предполагается, что он должен быть читабелен для людей, но трудно представить себе человека, пытающегося прочесть файл XML с 7 млн. узлов. Простейшие синтаксические анализаторы функционально не способны поддерживать широкий набор типов данных; по этой причине редкие или необычные данные, каких бывает много, становятся серьезным источником затруднений.

Правильно построенный XML

XML-документ считается построенным правильно, если в нем соблюдены правила XML-синтаксиса. Неправильно построенный формат в техническом смысле не является XML-документом. Например, такой HTML-тег, как
, в XML неприемлем; соответствующий правильный тег выглядит как
. Корневой элемент можно представить себе как бесконечный шкаф с документами. У вас всего один шкаф, но почти нет ограничений на тип и количество его содержимого. В вашем шкафу помещается бесконечное количество ящиков и папок для документов.

Чтение, обработка и написание XML в PHP

SimpleXML, при необходимости в сочетании с DOM, — идеальный выбор для чтения, обработки и составления в PHP5 простых, предсказуемых и относительно компактных документов.

Краткий обзор предпочтительных API

Из множества API, присутствующих в PHP5, DOM — самый знакомый, а на SimpleXML проще всего программировать. В типичных ситуациях, таких как те, что мы здесь рассматриваем, они наиболее эффективны.

Расширение DOM

Document Object Model (DOM) — это принятый W3C стандартный набор объектов для документов HTML и XML, стандартная модель сочетания этих объектов и стандартный интерфейс для доступа к ним и манипуляций с ними. Многие поставщики поддерживают DOM в качестве интерфейса к своим специальным структурам данных и API, благодаря чему модель DOM знакома массе разработчиков. DOM легко освоить и применять, так как его структура в памяти напоминает исходный документ XML. Чтобы передать информацию приложению, DOM создает дерево объектов, которое в точности повторяет дерево элементов файла XML, так что каждый элемент XML служит узлом этого дерева. DOM — это парсер, основанный на древовидной структуре. Поскольку DOM строит дерево всего документа, он потребляет много ресурсов памяти и времени процессора. Поэтому анализ очень крупных документов посредством DOM непрактичен из-за проблем производительности. В контексте данной статьи расширение DOM используется главным образом из-за его способности импортировать формат SimpleXML и выводить XML в формате DOM, или, наоборот, для использования в качестве строковых данных или XML-файла.

SimpleXML

Расширение SimpleXML – это предпочтительный инструмент для синтаксического анализа XML. Оно взаимодействует с DOM при составлении XML-файлов и имеет встроенную поддержку XPath. SimpleXML лучше всего работает с несложными данными типа записей, такими как XML, передаваемый в виде документа или строки из другой части того же приложения. Если XML-документ не слишком сложный, не слишком глубокий и не содержит смешанного контента, для SimpleXML кодировать проще, чем для DOM, как и предполагает название. К тому же он надежнее при работе с известной структурой документа.

Работа с SimpleXML

Рассмотрим работу с данным интерфейсом на следующем примере.

Представим себе, что вы хотите создать страницу PHP, которая конвертирует канал RSS в HTML код. RSS является основным форматом XML для публикации содержания, взятого из нескольких источников. Корневой элемент этого документа – rss, который содержит единственный элемент channel. Элемент channel содержит метаданные о содержимом, включая его заголовок, язык и URL. Он также содержит разнообразные текстовые элементы, вложенные в элементы item. Каждый элемент item имеет элемент link, содержащий URL или title, или description (обычно оба), в которых находится читаемый текст. Области имен не используются. Конечно, ещё много чего можно сказать об RSS, но для целей этой статьи информации достаточно. Пример кода ниже показывает типичный пример с парой информационных сообщений.

Пример RSS-канала:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="0.92">
<channel>
  <title>Mokka mit Schlag</title>
  <link>http://www.elharo.com/blog</link>
  <language>en</language>
  <item>
    <title>Penn Station: Gone but not Forgotten</title>
    <description>
     The old Penn Station in New York was torn down before I was born.
     Looking at these pictures, that feels like a mistake.  The current site is
     functional, but no more; really just some office towers and underground
corridors of no particular interest or beauty. The new Madison Square...
</description>
    <link>http://www.elharo.com/blog/new-york/2006/07/31/penn-station</link>
</item>
  <item>
    <title>Personal for Elliotte Harold</title>
    <description>Some people use very obnoxious spam filters that require you
     to type some random string in your subject such as E37T to get through.
     Needless to say neither I nor most other people bother to communicate with
     these paranoids. They are grossly overreacting to the spam problem.
     Personally I won't...</description>

    <link>http://www.elharo.com/blog/tech/2006/07/28/personal-for-elliotte-harold/</link>
</item>
</channel>
</rss>
```

Для того чтобы вывести на экран все названия элементов RSS канала, то можем использоваться следующий код:

```
// считываем данные из файла
$xml = simplexml_load_file('test1.xml');
```

```
// проходимся по каждому item и выводим на экран
foreach ($xml->channel->item as $item) {
   echo $item->title.'<br />';
}
```

Первый шаг – анализ XML-документа и его сохранение в переменной. Это требует написания всего лишь одной строки кода, которая передает URL функции simplexml_load_file().

Заметьте, что несмотря на имя simplexml_load_file(), этой функция может анализировать XML документ на удаленном сервере. Но это не единственная неожиданность в этой функции. Возвращаемое значение функции, которое здесь хранится в переменной $xml, не указывает на весь документ, как вы могли бы ожидать, исходя из опыта работы с другими интерфейсами API, такими, как, например, Объектная модель документа (DOM). Скорее оно указывает на корневой элемент документа. Контент, который находится в прологе и эпилоге документа, недоступен из SimpleXML.

Нахождение имени канала

Имя всего канала (в отличие от заголовков отдельных текстовых фрагментов этого канала) находится в дочернем элементе title от элемента channel, порожденного корневым элементом xml. Вы можете загрузить этот заголовок, как будто бы XML-документ был бы просто последовательной формой объекта класса xml с полем channel, которое в свою очередь имело бы поле title. Используя регулярный PHP-синтаксис ссылки на объект, этот оператор находит заголовок:

```
$title =  $xml->channel->title;
```

Найдя заголовок, вы должны добавить его к выходным данным HTML. Сделать это просто: повторите переменную $title:

```
<title><?php echo $title; ?></title>
```

Эта строка выводит строковое значение элемента, но не весь элемент. То есть текст записывается, а теги нет.

Вы можете даже полностью пропустить промежуточную переменную $title: <title><?php
echo $xml->channel->title; ?></title>

Потому что эта страница многократно использует это значение во многих местах, я нахожу более удобным хранить его как описательно озаглавленную переменную.

Обработка ошибок

Не все каналы RSS так хорошо сформированы, как это должно быть. Спецификация XML требует, чтобы процессоры прекратили обрабатывать документы, как только обнаружена формальная ошибка, а SimpleXML соответствовал программе обработки XML. Однако это вам особенно не поможет, когда будет обнаружена ошибка. Как правило, программа записывает предупреждение в файл php-ошибок (но без детального сообщения об ошибке), и функция simplexml-load-file() выдает ошибку. Если вы не уверены,

что файл, который вы анализируете хорошо выстроен, проверьте на наличие этой ошибки, перед тем как использовать данные файла, как это показано в примере ниже.

```
<?php
$xml =  simplexml_load_file('http://www.cafeaulait.org/today.rss'); if
($rss) {
  foreach ($rss->xpath('//title') as $title) {
    echo "<h2>". $title. "</h2>";
  }
} else
{
  echo "Упс! Ввод деформирован!";
}
?>
```

Другая распространенная ошибка случается, когда документ хорошо сформатирован, но не содержит те элементы, которые вы ожидаете там, где вы ожидаете их найти. Что происходит, например, с таким выражением $doc->rss->channel->item->title, когда элементная группа не имеет заголовка (как это случается, по меньшей мере, с одним из ста наиболее частотных RSS-каналов)? Самый простой подход – всегда обращаться с возвращаемым функцией значением как с массивом данных и заключать его в цикл. В этом случае вы защищены от факта наличия большего или меньшего количества элементов, чем вы ожидали. Однако, если вы знаете, что вы хотите первый элемент в документе, даже если там имеется больше чем один, вы можете запросить его через индекс, начинающийся с нуля. Например, для запроса заголовка первой группы элементов, вы можете написать:

```
$doc->rss->channel->item[0]->title[0]
```

Если первая группа элементов отсутствует, или не имеет названия, она рассматривается так же, как и любой другой, находящийся за установленными рамками индекс, в массиве PHP. То есть результат равен нулю, который превращается в пустую строку, когда вы попытаетесь вставить его в выходной код HTML.

Распознавание и отклонение неожиданных форматов, с которыми вы не готовы работать, обычно является сферой деятельности проверяющего на правильность парсера XML. Однако SimpleXML не может проверить относительно шаблона DTD (DTD) или схемы данных. Он проверяет только на формальную правильность.

Как работать с пространством имен

Попимо RSS протокола существует протокол Atom. Ниже показан пример документа в Atom. По многим параметрам этот документ идентичен примеру с RSS. Однако здесь больше метаданных, а корневой элемент - feed, вместо - rss. Элемент feed имеет списки вместо элементов. Элемент content заменяет элемент description. Гораздо важнее то, что документ Atom использует пространство имен, в то время как RSS нет. Таким образом, документ Atom может выводить реальное, не урезанное содержание Extensible HTML (XHTML).

```
<?xml version="1.0"?>
<feed xmlns="http://www.w3.org/2005/Atom" xml:lang="en-US"
    xml:base="http://www.cafeconleche.org/today.atom">
```

```
  <updated>2006-08-04T16:00:04-04:00</updated>
  <id>http://www.cafeconleche.org/</id>
  <title>Cafe con Leche XML News and Resources</title>
  <link rel="self" type="application/atom+xml" href="/today.atom"/>
  <rights>Copyright 2006 Elliotte Rusty Harold</rights>
  <entry>
    <title>Steve Palmer has posted a beta of Vienna 2.1, an open source
RSS/Atom client for Mac OS X.
         </title>
    <content type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml"
         id="August_1_2006_25279" class="2006-08-01T07:01:19Z">

<p>
 Steve Palmer has posted a beta of <a shape="rect"
 href="http://www.opencommunity.co.uk/vienna21.php">Vienna
 2.1</a>, an open source RSS/Atom client for Mac OS X. Vienna
 is the first reader I've found acceptable for daily use; not
 great but good enough. (Of course my standards for "good
 enough" are pretty high.) 2.1 focuses on improving the user
 interface with a unified layout that lets you scroll through
 several articles, article filtering (e.g. read all articles
 since the last refresh), manual folder reordering, a new get
 info window, and an improved condensed layout.
</p>

</div>
    </content>
    <link href="/#August_1_2006_25279"/>
    <id>http://www.cafeconleche.org/#August_1_2006_25279</id>
    <updated>2006-08-01T07:01:19Z</updated>
  </entry>
  <entry>
    <title>Matt Mullenweg has released Wordpress 2.0.4,
          a blog engine based on PHP and MySQL.
         </title>
    <content type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml"
          id="August_1_2006_21750" class="2006-08-01T06:02:30Z">

<p>
 Matt Mullenweg has released <a shape="rect"
 href="http://wordpress.org/development/2006/07/wordpress-204
 /">Wordpress 2.0.4</a>, a blog engine based on PHP and
 MySQL. 2.0.4 plugs various security holes, mostly involving  plugins.
</p>
</div>
```

```
  </content>
  <link href="/#August_1_2006_21750"/>
  <id>http://www.cafeconleche.org/#August_1_2006_21750</id>
  <updated>2006-08-01T06:02:30Z</updated>
 </entry>

</feed>
```

Хотя имена элементов изменились, основной подход к работе с SimpleXML в документах в Atom такой же, как и с RSS. Единственная разница в том, что вам необходимо указать пространство имен, т.е. универсальный идентификатор ресурса (URI), когда вы запрашиваете элемент с именем, так же как и локальное имя. Это двухступенчатый процесс: во-первых, запросите дочерние элементы в данном пространстве имен, передав пространство имен URI функции children(). Затем запросите элементы с правильным местным именем в этом пространстве имени. Представьте, что вы сначала загрузили канал Atom в переменную $feed, следующим образом:

```
$feed = simplexml_load_file('http://www.cafeconleche.org/today.atom');
```

Эти две строки теперь находят элемент title:

```
$children =  $feed->children('http://www.w3.org/2005/Atom');  $title
= $children->title;
```

Вы можете сжать этот код в единое выражение, если хотите, хотя строка становится немного длинной.
Все другие элементы в пространствах имен должны быть проработаны подобным же образом. Пример ниже показывает полную страницу PHP, которая отображает заголовки из поименованного канала Atom.

```
<?php
$feed =  simplexml_load_file('http://www.cafeconleche.org/today.atom');
$children =  $feed->children('http://www.w3.org/2005/Atom'); $title
= $children->title;
?>
<html xml:lang="en" lang="en">
<head>
  <title><?php echo $title; ?></title>
</head>
<body>

<h1><?php echo $title; ?></h1>

<?php

$entries = $children->entry;
foreach ($entries as $entry) {
```

```
  $details = $entry->children('http://www.w3.org/2005/Atom');
  echo "<h2>". $details->title. "</h2>";
}
?>

</body>
</html>
```

XPath

Такие выражения как $rss->channel->item->title великолепны, только если вы точно знаете, какие элементы находятся в документе, и где точно они находятся. Однако вы далеко не всегда это знаете. Например, в XHTML элементы в заголовке (h1,h2,h3, и т.д.) могут быть дочерними от body, div, table и от нескольких других элементов. Более того, div, table, blockquote и другие элементы могут быть вложены друг в друга множество раз. Для многих менее определенных случаев использования, легче использовать выражения XPath, такие как //h1 или //h1[contains('Ben')]. SimpleXML имеет этот набор функциональных возможностей через функцию xpath().
Пример ниже показывает страницу PHP, которая содержит все заголовки в документе RSS – как заголовок самого канала, так и заголовки отдельных групп элементов.

```
<html xml:lang="en" lang="en">
<head>
  <title>XPath Example</title>
</head>
<body>

<?php
$rss =  simplexml_load_file('http://partners.userland.com/nytRss/nytHomepage.xml'); foreach
($rss->xpath('//title') as $title) {
  echo "<h2>". $title. "</h2>";
}
?>

</body>
</html>
```

SimpleXML поддерживает только строковые выражения XPath, задающие местонахождении файла и объединения этих выражений. Он не поддерживает выражения XPath, которые не возвращают множества узлов, таких как count(//para) или contains(title).

DOM

Модель DOM, реализованная в PHP, — это та же спецификация W3C DOM, с которой вы имеете дело в браузере и с которой работаете посредством JavaScript. Используются те же методы, так что способы кодирования покажутся вам знакомыми. Пример ниже иллюстрирует использование DOM для создания XML-строки и XML-документа, отформатированных в целях читабельности.

```
<?php

//Создает XML-строку и XML-документ при помощи DOM
$dom = new DomDocument('1.0', 'utf-8');

//добавление корня - <books>
$books = $dom->appendChild($dom->createElement('books'));

//добавление элемента <book> в <books>
$book = $books->appendChild($dom->createElement('book'));

// добавление элемента <title> в <book>
$title = $book->appendChild($dom->createElement('title'));

// добавление элемента текстового узла <title> в <title>
$title->appendChild(
    $dom->createTextNode('Moby-Dick'));

//генерация xml
$dom->formatOutput = true; // установка атрибута formatOutput
// domDocument в значение true
// save XML as string or file
$test1 = $dom->saveXML(); // передача строки в books
$dom->save('books.xml'); // сохранение файла
```

Это приводит к созданию выходного файла:

```
<?xml version="1.0" encoding="utf-8"?>
<books>
  <book>
    <title>Moby-Dick</title>
  </book>
</books>
```

Работа с CSV

Общие сведения

CSV - текстовый формат, предназначенный для представления табличных данных. Каждая строка файла — это одна строка таблицы. Значения отдельных колонок разделяются разделительным символом (delimiter) — запятой (,). Однако, большинство программ вольно трактует стандарт CSV и допускают использование иных символов в качестве разделителя. В частности в локалях, где десятичным разделителем является запятая, в качестве табличного разделителя, как правило, используется точка с запятой. Значения, содержащие зарезервированные символы (двойная кавычка, запятая, точка с запятой, новая строка) обрамляются двойными кавычками ("); если в значении встречаются кавычки — они представляются в файле в виде двух кавычек подряд.

Чтение CSV файлов

Предположим, у нас есть CSV-файл с данными:

```
"Иванов А.А.";"Программист компании OX2.ru";89255552332
"Сидоров А.Е.";"Дизайне компании OX2.ru";89161231212
"Пирожков А.Б.";"Арт-директор OX2.ru";84951232121 "Кулибин Б.А.";"Менеджер
OX2.ru";89031233333
```

Для вывода текста файла CSV на экран воспользуемся функцией :

fgetcsv - читает строку из файла и производит разбор данных CSV.

array **fgetcsv** (resource $handle [, int $length = 0 [, string $delimiter = ',' [, string $enclosure = '"' [, string $escape = '\\']]]]) - данная функция производит анализ строки на наличие записей в формате CSV и возвращает найденные поля в качестве массива.

```
<?php
$row = 1;
if (($handle = fopen("test.csv", "r")) !== FALSE) {
   while (($data = fgetcsv($handle, 1000, ",")) !== FALSE) {
      $num = count($data);
      echo "<p> $num полей в строке $row: <br /></p>\n";
$row++;
      for ($c=0; $c < $num; $c++) {
         echo $data[$c] . "<br />\n";
      }
   }
   fclose($handle);
}
```

Запись в CSV файл

Для записи данных в CSV файл необходимо использовать функцию **fputcsv**.

int **fputcsv** (resource $handle , array $fields [, string $delimiter = ',' [, string $enclosure = '"']])

fputcsv() форматирует строку (переданную в виде массива **fields**) в виде CSV и записывает её (заканчивая переводом строки) в указанный файл **handle** .

```
<?php

$list = array (
    array('aaa', 'bbb', 'ccc', 'dddd'),
    array('123', '456', '789'),
    array('"aaa"', '"bbb"')
);

$fp = fopen('file.csv', 'w');

foreach ($list as $fields) {
    fputcsv($fp, $fields);
}

fclose($fp);
```

Вышеуказаный пример запишет в файл file.csv следующее:

```
aaa,bbb,ccc,dddd 123,456,789
"""aaa""","""bbb"""
```

Работа с .ini

Общие сведения

ini-файл — это файл конфигурации, который содержит некоторые данные.
Синтаксис .ini файлов имеет всего три элемента:

1. Секция.
2. Параметр.
3. Значение.

Секция логически отделяет группу параметров от других групп. Название секции располагается в отдельной строке и указывается в квадратных скобках, например, [DB].
Параметр задает какой-то признак, который будет использоваться в программе после загрузки настроечного файла. Параметр всегда идет в паре со значением и стоит слева от знака ‘=’. Параметр должен быть уникальным в пределах одного файла.
Значение дается параметру с тем, чтобы оказать влияние на работу программы, загрузившей настроечный файл и считавшей этот параметр. Значение идет в паре со значением и стоит справа от знака ‘=’. В некоторых случаях значение может отсутствовать.

Пример кода, оформленного в правилах синтаксиса INI:

```
[Секция0]
Параметр01=Значение01
Параметр02=Другое значение 02
Параметр03=1

[Секция1]
Параметр11=
Параметр12=C:\
Параметр13=Значение параметра 13
```

Чтение .ini файлов

Предположим, что у нас есть .ini файл с данными.

```
[db_settings]
host=localhost
user=root
password="my secret password"
dbname="mydatabse"

[site_settings]
title=”My Site”
```

Для чтения данных из .ini файла воспользуемся функцией **parse_ini_file**
array parse_ini_file (string $filename [, bool $process_sections = false [, int $scanner_mode = INI_SCANNER_NORMAL]])

parse_ini_file() загружает ini-файл, указанный в аргументе filename, и возвращает его настройки в виде ассоциативного массива

Параметры

filename

Имя обрабатываемого ini-файла.

process_sections

Установив параметр process_sections в TRUE, вы получаете многомерный массив, который включает как название отдельных настроек, так и секции. По умолчанию process_sections равен FALSE

scanner_mode

Может принимать следующие значения: INI_SCANNER_NORMAL (по умолчанию) или INI_SCANNER_RAW. Если указано значение INI_SCANNER_RAW, то значения опций не будут обрабатываться.

```php
<?php
header('Content-Type: text/html; charset=utf-8');

$filepath = './settings.ini';
$settings = parse_ini_file($filepath, 1);
echo '<pre>';
print_r($settings);
```

Запись .ini файлов

Встроенной функции для записи .ini файлов в php нет.

Для записи можно использовать обычные средства для работы с файлами, например file_put_contents.

```php
<?php
$section_name = 'users';
$table_name = 'users';
$id = 'id';
$login = 'login';
$pass = 'pass';
$str = "[".$section_name."]\r\n";
$str .="table = "'.$table_name.'"\r\n";
$str .="id = "'.$table_name.'"\r\n";
$str .="login = "'.$login.'"\r\n";
$str .="pass = "'.$pass.'"\r\n";

file_put_contents('table_setting.ini', $str);
```

После выполнения данного кода запишется такой .ini файл.

[users]
table = 'users'
id = 'id'
login = 'login'
pass = 'pass'

Работа с JSON данными

Общие сведения

JSON — текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком. Как и многие другие текстовые форматы, JSON легко читается людьми. Формат JSON был разработан Дугласом Крокфордом.

Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

За счёт своей лаконичности по сравнению с XML, формат JSON может быть более подходящим для сериализации сложных структур. Если говорить о веб-приложениях, в таком ключе он уместен в задачах обмена данными как между браузером и сервером (AJAX), так и между самими серверами (программные HTTP-интерфейсы).

Поскольку формат JSON является подмножеством синтаксиса языка JavaScript, то он может быть быстро десериализован встроенной функцией eval(). Кроме того, возможна вставка вполне работоспособных JavaScript-функций. В языке PHP, начиная с версии 5.2.0, поддержка JSON включена в ядро в виде функций json_decode() и json_encode(), которые сами преобразуют типы данных JSON в соответствующие типы PHP и наоборот.

Интерфейс для работы с форматом

json_encode

json_encode (mixed $value [, int $options = 0])

Возвращает строку, содержащую JSON-представление *value* .

value value - значение, которое будет закодировано. Может быть любого типа за исключением resource.

Функция работает только с кодированными в UTF-8 данными. **options**

Битовая маска составляемая из значений **JSON_HEX_QUOT, JSON_HEX_TAG,**

JSON_HEX_AMP, JSON_HEX_APOS, JSON_NUMERIC_CHECK, JSON_BIGINT_AS_STRING, JSON_PRETTY_PRINT, JSON_UNESCAPED_SLASHES, JSON_FORCE_OBJECT, JSON_UNESCAPED_UNICODE.

Пример:

```
$a = array('<foo>','"bar"','"baz"','&blong&', "\xc3\xa9");
echo json_encode($a);

// Результат
["<foo>","\"bar\"","\"baz\"","&blong&","\u00e9"]
```

json_decode

json_decode (string $json [, bool $assoc = false]) - принимает закодированную в JSON строку и преобразует ее в переменную PHP.

json json строка (string) для декодирования. Функция работает только с кодированными в UTF-8 данными. **assoc** если TRUE, возвращаемые объекты будут преобразованы в ассоциативные массивы.

Пример:

```
$json = '{"foo-bar": 12345}'; $obj
= json_decode($json);
print $obj->{'foo-bar'}; // 12345
```

json_last_error

json_last_error — возвращает последнюю ошибку.

Возвращает целочисленное значение, которое может быть одной из следующих констант:

JSON_ERROR_NONE - Ошибок нет

J**SON_ERROR_DEPTH** - Достигнута максимальная глубина стека

JSON_ERROR_STATE_MISMATCH - Неверный или не корректный JSON

JSON_ERROR_CTRL_CHAR - Ошибка управляющего символа, возможно неверная кодировка

JSON_ERROR_SYNTAX - Синтаксическая ошибка

JSON_ERROR_UTF8 - Некорректные символы UTF-8, возможно неверная кодировка