XML and JSON

Disclaimer: вы смотрите просто запись лекции, это **НЕ** специально подготовленный видеокурс!

Intro

XML and JSON are popular and widely used in PHP for miscellaneous reasons:

- intercomponent communication (including AJAX);
- configuration for applications and servers;
- communication with NoSQL data storages;
- etc…

Disclaimer

In this course we are not going to scrutinize XML and JSON, we are just going to explore basics enough to use these formats in everyday development.

About XML

XML is e**X**tensible **M**arkup **L**anguage.

- It is based on the Standard Generalized Markup Language (SGML).
- It is used to describe data.
- In XML tags are not defined initially, they must be defined separately.
- DTD (Document Type Definition) or XML Schema are used to describe XML data rules.
- XML data can be stored both in a separate file and inside HTML, which will be responsible only for the display format, but not for the data.
- XML can be used to exchange information between different (including incompatible) systems.
- XML is used both for storing data in the file system and for storing and retrieving information in a database.
- XML is actively used to exchange information in AJAX applications.

XML vs HTML

- The same characters (e.g., “<“, “>”, “&”) to denote language elements.
- The same comments (i.e., <!-- like this -->).

- Browsers “forgive” errors in HTML, and try to make “corrections” on-the-fly; XML parsers in case of any error stop processing.
- XML is a much more rigorous language, built around clear rules that must be followed strictly.
- XML does not remove extra whitespace at the beginning/middle/end of a string.
- In XML, newlines are denoted as \n (LF) (unlike <br> in HTML).

XML Pros and Cons

- XML allows one to create own structures to store information.
- The task of XML parsing is well defined and has many implementations.
- XML uses Unicode, which simplifies the development of international documents.
- XML parsers allow validation of document structure and data types.
- XML is a text format, which makes it easier to read, document, and debug.
- XML tools are available on all platforms.
- XML allows one to use the infrastructure built for HTML (including HTTP and some browsers).

- XML documents are bigger than similar binary formats.
- XML generates more traffic or more CPU usage (if compression is used).
- Parsing XML can be slower and more memory intensive than parsing optimized binary documents.

How does XML look like?

Here’s a simple XML document.

```xml
<?xml version="1.0"?>
<message>
    <from>John</from>
    <to>Jack</to>
    <header>Reminder</header>
    <text>Don't forget to call the Customer!</text>
</message>
```

Well-formed XML

A well-formed XML-document follows there rules

Only one root element

All tags are closed

Tags are case sensitive

Strict tags nesting

All attributes’ values are in “

Symbols <, >, &, ’, “ are represented as &lt;, &gt;, &amp;, &apos;, &quot;

All symbols are in the same (defined) encoding

Only one root element

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>John</givenName>
    <familyName>Smith</familyName>
</person>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>John</givenName>
    <familyName>Smith</familyName>
</person>
<person>
    <givenName>Jane</givenName>
    <familyName>Smith</familyName>
</person>
```

ERROR!

All tags are closed

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>John</givenName>
    <familyName>Smith</familyName>
</person>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>John</givenName>
    <familyName>Smith
</person>
```

ERROR!

Tags are case sensitive

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>John</givenName>
    <familyName>Smith</familyName>
</person>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>John</givenName>
    <familyName>Smith</FAMILYNAME>
</person>
```

ERROR!

Strict tags nesting

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>John</givenName>
    <familyName>Smith</familyName>
</person>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>John<familyName>
     </givenName>Smith</familyName>
</person>
```

ERROR!

All attributes’ values are in “

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName type="official">John</givenName>
    <familyName>Smith</familyName>
</person>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName type=official>John</givenName>
    <familyName>Smith</familyName>
</person>
```

ERROR!

Symbols <, >, &, ', " are represented as &lt;, &gt;, &amp;, &apos;, &quot;

```
<?xml version="1.0" encoding="UTF-8"?>
<description>
    <name>Peter &amp; Jane</name>
</description>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<description>
    <name>Peter & Jane</name>
</description>
```

ERROR!

&lt;	<
&gt;	>
&amp;	&
&apos;	'
&quot;	"

All symbols are in the same (defined) encoding

```
<?xml version="1.0" encoding="UTF-8"?>
<description>
    <data>Some information: Ў</data>
</description>
```

```
<?xml version="1.0" encoding="windows-1251"?>
<description>
    <data>Some information: Ў</data>
</description>
```

ERROR!

How to parse or create XML in PHP

To work with XML in PHP, you can use:

- DOM (http://www.php.net/manual/en/book.dom.php)
- SAX (http://php.net/manual/en/book.xml.php)
- SimpleXML (http://www.php.net/manual/en/book.simplexml.php) as the simplest.

Sometimes “looses” some elements in complex documents!

DOM vs SAX

DOM

- Reads the whole document and builds a DOM model.
- Provides API to navigate the model.
- Good for small documents.
- Good for multiple changes in a document.
- Needs a lot of memory.
- Difficult to control parsing progress.

SAX

- Reads the document “by portions”.
- Reacts (calls callbacks) on “events” (tag opened, tag closed, attribute found, etc.).
- Good for large documents.
- Good for single-pass processing.
- Needs a little of memory.
- Easy to control parsing progress.

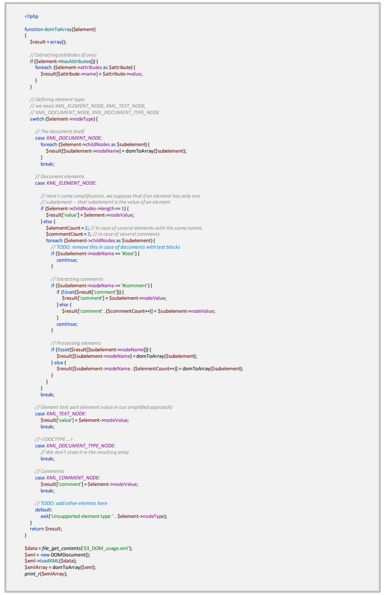
Parsing samples

Let’s start with the easiest approach and use SimpleXML to convert an XML document into a typical PHP array.

```
<?php

// Yes, it's that simple!

// WARNING! Sometimes SimpleXML “looses” some
// elements in complex documents!

$xmlData = simplexml_load_file('02_SimpleXML_usage.xml');
$jsonData = json_encode($xmlData);
$xmlArray = json_decode($jsonData, true);
print_r($xmlArray);
```

Parsing samples

Now let’s parse XML to array with DOM (see the code in the handouts).

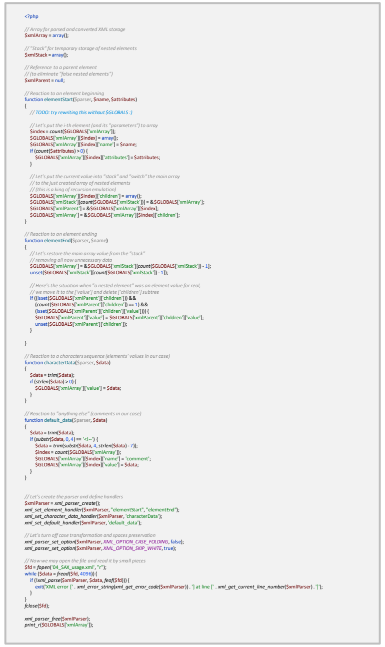


```php
<?php

function domToArray($element)
{
  $result = array();

  // Extracting attributes (if any)
  if ($element->hasAttributes()) {
    foreach ($element->attributes as $attribute) {
      $result[$attribute->name] = $attribute->value;
    }
  }

  // Defining element type:
  // we need XML_ELEMENT_NODE, XML_TEXT_NODE,
  // XML_DOCUMENT_NODE, XML_DOCUMENT_TYPE_NODE
  switch ($element->nodeType) {

    // The document itself
    case XML_DOCUMENT_NODE:
      foreach ($element->childNodes as $subelement) {
        $result[$subelement->nodeName] = domToArray($subelement);
      }
      break;

    // Document elements
    case XML_ELEMENT_NODE:

      // Here's some simplification, we suppose that if an element has only one
      // subelement -- that subelement is the value of an element
      if ($element->childNodes->length == 1) {
        $result['value'] = $element->nodeValue;
      } else {
        $elementCount = 2; // In case of several elements with the same names
        $commentCount = 2; // in case of several comments
        foreach ($element->childNodes as $subelement) {
          // TODO: remove this in case of documents with text blocks
          if ($subelement->nodeName == '#text') {
            continue;
          }

          // Extracting comments
          if ($subelement->nodeName == '#comment') {
            if (!isset($result['comment'])) {
              $result['comment'] = $subelement->nodeValue;
            } else {
              $result['comment' . ($commentCount++)] = $subelement->nodeValue;
            }
            continue;
          }

          // Processing elements
          if (!isset($result[$subelement->nodeName])) {
            $result[$subelement->nodeName] = domToArray($subelement);
          } else {
            $result[$subelement->nodeName . ($elementCount++)] = domToArray($subelement);
          }
        }
      }
      break;

    // Element text part (element value in our simplified approach)
    case XML_TEXT_NODE:
      $result['value'] = $element->nodeValue;
      break;

    // <!DOCTYPE ...>
    case XML_DOCUMENT_TYPE_NODE:
      // We don't store it in the resulting array
      break;

    // Comments
    case XML_COMMENT_NODE:
      $result['comment'] = $element->nodeValue;
      break;

    // TODO: add other elemnts here
    default:
      exit('Unsupported element type: ' . $element->nodeType);
  }
  return $result;
}

$data = file_get_contents('03_DOM_usage.xml');
$xml = new DOMDocument();
$xml->loadXML($data);
$xmlArray = domToArray($xml);
print_r($xmlArray);
```

This code here is just for the reference (in case you don’t have the handouts; otherwise – see the file in the handouts ☺).

Parsing samples

Now let’s parse XML to array with SAX (see the code in the handouts).



```php
<?php

// Array for parsed and converted XML storage
$xmlArray = array();

// "Stack" for temporary storage of nested elements
$xmlStack = array();

// Reference  to a parent element
// (to eliminate "false nested elements")
$xmlParent = null;

// Reaction to an element beginning
function elementStart($parser, $name, $attributes)
{
    // TODO: try rewriting this without $GLOBALS :)

    // Let's put the i-th element (and its "parameters") to array
    $index = count($GLOBALS['xmlArray']);
    $GLOBALS['xmlArray'][$index] = array();
    $GLOBALS['xmlArray'][$index]['name'] = $name;
    if (count($attributes) > 0) {
        $GLOBALS['xmlArray'][$index]['attributes'] = $attributes;
    }

    // Let's put the current value into "stack" and "switch" the main array
    // to the just created array of nested elements
    // (this is a king of recursion emulation)
    $GLOBALS['xmlArray'][$index]['children'] = array();
    $GLOBALS['xmlStack'][count($GLOBALS['xmlStack'])] = &$GLOBALS['xmlArray'];
    $GLOBALS['xmlParent'] = &$GLOBALS['xmlArray'][$index];
    $GLOBALS['xmlArray'] = &$GLOBALS['xmlArray'][$index]['children'];
}

// Reaction to an element ending
function elementEnd($parser, $name)
{
    // Let's restore the main array value from the "stack"
    // removing all now unnecessary data
    $GLOBALS['xmlArray'] = &$GLOBALS['xmlStack'][count($GLOBALS['xmlStack']) - 1];
    unset($GLOBALS['xmlStack'][count($GLOBALS['xmlStack']) - 1]);

    // Here's the situation when "a nested element" was an element value for real,
    // we move it to the ['value'] and delete ['children'] subtree
    if ((isset($GLOBALS['xmlParent']['children'])) &&
        (count($GLOBALS['xmlParent']['children']) == 1) &&
        (isset($GLOBALS['xmlParent']['children']['value']))) {
        $GLOBALS['xmlParent']['value'] = $GLOBALS['xmlParent']['children']['value'];
        unset($GLOBALS['xmlParent']['children']);
    }

}

// Reaction to a characters sequence (elements' values in our case)
function characterData($parser, $data)
{
    $data = trim($data);
    if (strlen($data) > 0) {
        $GLOBALS['xmlArray']['value'] = $data;
    }
}

// Reaction to "anything else" (comments in our case)
function default_data($parser, $data)
{
    $data = trim($data);
    if (substr($data, 0, 4) == '<!--') {
        $data = trim(substr($data, 4, strlen($data) - 7));
        $index = count($GLOBALS['xmlArray']);
        $GLOBALS['xmlArray'][$index]['name'] = 'comment';
        $GLOBALS['xmlArray'][$index]['value'] = $data;
    }
}


// Let's create the parser and define handlers
$xmlParser = xml_parser_create();
xml_set_element_handler($xmlParser, "elementStart", "elementEnd");
xml_set_character_data_handler($xmlParser, 'characterData');
xml_set_default_handler($xmlParser, 'default_data');

// Let's turn off case transformation and spaces preservation
xml_parser_set_option($xmlParser, XML_OPTION_CASE_FOLDING, false);
xml_parser_set_option($xmlParser, XML_OPTION_SKIP_WHITE, true);

// Now we may open the file  and read it by small pieces
$fd = fopen('04_SAX_usage.xml', "r");
while ($data = fread($fd, 4096)) {
    if (!xml_parse($xmlParser, $data, feof($fd))) {
        exit('XML error [' . xml_error_string(xml_get_error_code($xmlParser)) . '] at line [' . xml_get_current_line_number($xmlParser) . ']');
    }
}
fclose($fd);

xml_parser_free($xmlParser);
print_r($GLOBALS['xmlArray']);
```

This code here is just for the reference (in case you don’t have the handouts; otherwise – see the file in the handouts ☺).

About JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format.

- It is based on the JavaScript Programming Language Standard ECMA-262 3rd Edition.
- It is used to describe data.
- It is easy for humans to read and write.
- It is easy for software to parse and generate.
- It is built on two structures:
    - A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
    - An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence

JSON Pros and Cons (vs XML)

- Easier to read (for humans).
- Usually have a smaller size.
- Easier to parse.
- De-facto is a standard for AJAX and/or similar cases.
- Works perfectly for small documents and frequent data-exchange interactions.

- There are multiple incompatible JSON versions, path expressions, query languages, and transformation tools.
- No ability to add comments or attribute tags.
- Difficult to avoid naming collisions when merging data from several sources.

How does JSON look like?

Here’s a simple JSON document.

```
{
  "message": {
    "from": "John",
    "to": "Jack",
    "header": "Reminder",
    "text": "Don't forget to call the Customer!"
  }
}
```

XML for comparison

```
<?xml version="1.0"?>
<message>
    <from>John</from>
    <to>Jack</to>
    <header>Reminder</header>
    <text>Don't forget to call the Customer!</text>
</message>
```

Well-formed JSON

There are no explicit set of rules, still…

Data is “name/value” pairs

Data is separated by commas

Curly braces hold objects

Square brackets hold arrays

Symbols {, }, [,], ’, “ are represented as corresponding &sequences; or escaped

Strict nesting rules

Data is “name/value” pairs

```
{
  "message": {
    "from": "John",
    "to": "Jack",
    "header": "Reminder",
    "text": "Don't forget to call the Customer!"
  }
}
```

```
{
  "message": {
    "from": "John",
    "to": ,
    "header": "Reminder",
    "text": "Don't forget to call the Customer!"
  }
}
```

ERROR!

Data is separated by commas

```
{
  "message": {
    "from": "John",
    "to": "Jack",
    "header": "Reminder",
    "text": "Don't forget to call the Customer!"
  }
}
```

```
{
  "message": {
    "from": "John",
    "to": "Jack"
    "header": "Reminder",
    "text": "Don't forget to call the Customer!"
  }
}
```

ERROR!

Curly braces hold objects

```
{
  "message": {
    "from": "John",
    "to": "Jack",
    "header": "Reminder",
    "text": "Don't forget to call the Customer!"
  }
}
```

```
{
  "message": [
    "from": "John",
    "to": "Jack"
    "header": "Reminder",
    "text": "Don't forget to call the Customer!"
  ]
}
```

ERROR!

Square brackets hold arrays

```
{
  "message": {
    "from": "John",
    "to": ["Jack", "Jane"],
    "header": "Reminder",
    "text": "Don't forget to call the Customer!"
  }
}
```

```
{
  "message": {
    "from": "John",
    "to": {"Jack", "Jane"},
    "header": "Reminder",
    "text": "Don't forget to call the Customer!"
  }
}
```

ERROR!

Symbols {, }, [,], ’, “ are represented as &sequences; or escaped

```
{
  "message": {
    "from": "John",
    "to": ["Jack", "Jane"],
    "header": "Reminder",
    "text": "Don't forget to call the &quot;Big &quot; Customer!"
  }
}
```

```
{
  "message": {
    "from": "John",
    "to": ["Jack", "Jane"],
    "header": "Reminder",
    "text": "Don't forget to call the "Big" Customer!"
  }
}
```

ERROR!

Strict nesting rules

```
{
  "message": {
    "from": "John",
    "to": ["Jack", "Jane"],
    "header": "Reminder",
    "text": "Don't forget to call the &quot;Big &quot; Customer!"
  }
}
```

```
{
  "message": {
    "from": "John",
    "to": ["Jack", "Jane"],
    "header": "Reminder"}
    "text": "Don't forget to call the "Big" Customer!"
  }
}
```

ERROR!

Encoding and decoding sample

In PHP there are some inbuilt JSON-related functions.
See details: https://www.php.net/manual/en/ref.json.php

```php
<?php

// Encoding
$data = array(
    'MP123456' => array('givenName' => 'John', 'familyName' => 'Smith'),
    'MP567890' => array('givenName' => 'Jane', 'familyName' => 'Smith')
);
echo json_encode($data) . "\n\n";

// Encoding again
$message = array(
    'message' => array('from' => 'John', 'to' => 'Jack',
    'header' => 'Reminder', 'text' => 'Don\'t forget to call the Customer!')
);
echo json_encode($message) . "\n\n";

// Decoding
$json = file_get_contents("05_JSON_encoding_decoding.json");
print_r(json_decode($json, true));
```

Finally…

Let’s look at a simple “server” and “client” that interact with XML and JSON.

Server

```
<?php

// This "server" returns a
// table with random columns and rows count
// as XML or as JSON

$tree = new SimpleXMLElement('<table/>');

$rows = mt_rand(1, 5);
$cols = mt_rand(1, 5);

for ($i = 0; $i < $rows; $i++) {
    $row = $tree->addChild('tr');
    for ($j = 0; $j < $cols; $j++) {
        $row->addChild('td', mt_rand(1, 100));
    }
}

if (isset($_GET['xml'])) {
    header("Content-type: application/xml");
    echo $tree->asXML();
} else {
    header("Content-type: text/plain");
    echo json_encode($tree);
}
```

Client

```
<?php

// Get XML data, parse it to a PHP array
$xmlData =
simplexml_load_file('http://127.0.0.1/06_server.php?xml=true');
$jsonData = json_encode($xmlData);
$xmlArray = json_decode($jsonData, true);
print_r($xmlArray);

// Get JSON data, parse it to a PHP array
$json = file_get_contents('http://127.0.0.1/06_server.php');
print_r(json_decode($json, true));
```

XML and JSON

Disclaimer: вы смотрите просто запись лекции,
это **НЕ** специально подготовленный видеокурс!