Array Structure

XML-Array Conversion Structure

Anonymous · 09/09/2026

XML-Array Conversion Structure

The Derafu XML library uses a specific convention for representing the structure of XML documents as PHP arrays and vice versa. Understanding this structure is essential for effectively using the encoding and decoding features.

Basic Structure

At its most basic level, an XML element with a value is represented as a key-value pair in the array:

XML:

```
<element>value</element>
```

Array:

```
['element' => 'value']
```

Nested Elements

Nested elements are represented as nested arrays:

XML:

```
<parent>
  <child>value</child>
</parent>
```

Array:

```
[
    'parent' => [
        'child' => 'value'
    ]
```

```
]
```

Attributes

XML attributes are handled using the special `@attributes` key:

XML:

```
<element id="123" type="example">value</element>
```

Array:

```
[
    'element' => [
        '@attributes' => [
            'id' => '123',
            'type' => 'example'
        ],
        '@value' => 'value'
    ]
]
```

Note the use of `@value` to hold the element's text content when attributes are present.

Repeated Elements

Elements with the same name are collected into arrays:

XML:

```
<parent>
  <child>value1</child>
  <child>value2</child>
  <child>value3</child>
</parent>
```

Array:

```
[
    'parent' => [
        'child' => [
            'value1',
            'value2',
            'value3'
```

```
        ]
    ]
]
```

Complex Structures

The conventions can be combined to represent complex XML structures:

XML:

```xml
<root>
  <items>
    <item id="1">
      <name>Item 1</name>
      <price currency="USD">19.99</price>
    </item>
    <item id="2">
      <name>Item 2</name>
      <price currency="EUR">29.99</price>
    </item>
  </items>
  <summary total="2">Multiple items</summary>
</root>
```

Array:

```
[
    'root' => [
        'items' => [
            'item' => [
                [
                    '@attributes' => ['id' => '1'],
                    'name' => 'Item 1',
                    'price' => [
                        '@attributes' => ['currency' => 'USD'],
                        '@value' => '19.99'
                    ]
                ],
                [
                    '@attributes' => ['id' => '2'],
                    'name' => 'Item 2',
                    'price' => [
                        '@attributes' => ['currency' => 'EUR'],
                        '@value' => '29.99'
                    ]
                ]
            ]
```

```
        ],
        'summary' => [
            '@attributes' => ['total' => '2'],
            '@value' => 'Multiple items'
        ]
    ]
]
```

Special Cases

Empty Elements

Empty elements are represented as `null` or empty strings:

XML:

```
<element></element>
```

Array:

```
['element' => null]
```

or when creating XML:

```
['element' => '']
```

Skipping Elements

When creating XML, certain values cause an element to be skipped:

```
[
    'included' => 'This will be in the XML',
    'excluded' => null,       // This will be skipped.
    'alsoExcluded' => false,  // This will be skipped.
    'emptyArraySkipped' => [] // This will be skipped.
]
```

Array to XML Encoding Rules

When using `XmlEncoder` to create XML from arrays, these rules apply:

1. Simple key-value pairs become elements with text content.
2. Nested arrays become nested elements.

3. The special key `@attributes` defines attributes for the parent element.
4. The special key `@value` defines the text content when attributes are present.
5. Arrays of values create multiple elements with the same name.
6. Null, false, and empty arrays are skipped (not included in the XML).
7. Empty strings (’’) create empty elements.

XML to Array Decoding Rules

When using `XmlDecoder` to convert XML to arrays, these rules apply:

1. Elements become keys in the array.
2. Text content becomes the value of the key.
3. Attributes are collected under the `@attributes` key.
4. When attributes are present, text content is stored under the `@value` key.
5. Multiple elements with the same name are collected into arrays.
6. Empty elements become null values.
7. Complex nested structures are preserved in the array hierarchy.

Usage Examples

Creating Complex XML

```
use Derafu\Xml\Service\XmlEncoder;

$data = [
    'invoice' => [
        '@attributes' => [
            'id' => 'INV-2025-001',
            'date' => '2025-03-05'
        ],
        'customer' => [
            'name' => 'Acme Inc.',
            'address' => [
                'street' => '123 Main St',
                'city' => 'Anytown',
                'zipcode' => '12345'
            ]
        ],
        'items' => [
            'item' => [
                [
                    '@attributes' => ['sku' => 'PROD1'],
                    'description' => 'Product One',
                    'quantity' => '2',
                    'price' => '10.99'
                ],
```

```
                [
                    '@attributes' => ['sku' => 'PROD2'],
                    'description' => 'Product Two',
                    'quantity' => '1',
                    'price' => '24.99'
                ]
            ]
        ],
        'total' => '46.97'
    ]
];

$encoder = new XmlEncoder();

$xmlDocument = $encoder->encode($data);
echo $xmlDocument->saveXml();
```

Processing XML into Arrays

```
use Derafu\Xml\Service\XmlDecoder;

$xmlString = '
<catalog>
    <book id="bk101">
        <author>Gambardella, Matthew</author>
        <title>XML Developer\'s Guide</title>
        <genre>Computer</genre>
        <price>44.95</price>
        <publish_date>2025-01-15</publish_date>
    </book>
    <book id="bk102">
        <author>Ralls, Kim</author>
        <title>Midnight Rain</title>
        <genre>Fantasy</genre>
        <price>5.95</price>
        <publish_date>2025-02-20</publish_date>
    </book>
</catalog>';

$document = new \Derafu\Xml\XmlDocument();
$document->loadXml($xmlString);

$decoder = new XmlDecoder();

$array = $decoder->decode($document);

// Access data directly.
$firstBookTitle = $array['catalog']['book'][0]['title']; // "XML Developer's Guide"
$secondBookPrice = $array['catalog']['book'][1]['price']; // "5.95"
```

Tips for Working with the Array Structure

1. **Plan Your Structure**: When creating XML, plan your array structure carefully to match the desired XML output.
2. **Handle Repeated Elements**: Always use numeric arrays for repeated elements with the same name.
3. **Use `@attributes` and `@value`**: Remember to use these special keys when working with elements that have both attributes and text content.
4. **Validate Your Output**: Always validate the generated XML against your schema if one exists.
5. **Check for Empty Values**: When processing arrays from XML, check for null values to handle empty elements properly.

Last updated on 09/09/2026