Introduction

Library for XML manipulation

Anonymous · 09/09/2026

Library for XML manipulation

A comprehensive PHP library for XML manipulation, providing robust tools for encoding, decoding, validating, and querying XML documents.

Features

- **Conversion**: Transform between XML and PHP arrays in both directions.
- **Validation**: Validate XML documents against XSD schemas.
- **Querying**: Powerful XPath querying with parameter support.
- **Canonicalization**: Support for C14N and ISO-8859-1 encoding.
- **Encoding**: Proper handling of character encoding between UTF-8 and ISO-8859-1.
- **Special Characters**: Automatic handling of XML special characters and entities.

Installation

```
composer require derafu/xml
```

Basic Usage

Creating XML from an Array

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

$encoder = new XmlEncoder();

// Create an array to convert to XML.
$data = [
    'root' => [
        'element1' => 'value1',
        'element2' => 'value2',
```

```
        'element3' => [
            '@attributes' => [
                'attr1' => 'attrValue',
            ],
            '@value' => 'value3',
        ],
        'repeatedElement' => ['value4', 'value5', 'value6'],
    ],
];

// Convert the array to XML.
$xmlDocument = $encoder->encode($data);

// Save as XML string.
$xmlString = $xmlDocument->saveXml();

// Get the XML without the XML declaration.
$xmlContent = $xmlDocument->getXml();

// Get a canonicalized version of the XML.
$c14nXml = $xmlDocument->C14N();

// Get a canonicalized version with ISO-8859-1 encoding.
$isoXml = $xmlDocument->C14NWithIso88591Encoding();
```

Converting XML to an Array

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

$decoder = new XmlDecoder();

// Load an existing XML string.
$xmlContent = '<?xml version="1.0"
encoding="UTF-8"?><root><element>value</element></root>';
$document = new XmlDocument();
$document->loadXml($xmlContent);

// Convert to an array.
$array = $decoder->decode($document);

// Now $array contains ['root' => ['element' => 'value']]
```

Working with XPath

```
use Derafu\Xml\XPathQuery;

$xmlContent = '<?xml version="1.0"?><root><item id="1">First</item><item
id="2">Second</item></root>';
```

```
// Create XPath query instance.
$query = new XPathQuery($xmlContent);

// Get a specific value.
$value = $query->getValue('/root/item[@id="2"]'); // "Second"

// Get multiple values.
$values = $query->getValues('/root/item'); // ["First", "Second"]

// Get structured array.
$array = $query->get('/root');
// Result: ['item' => ['First', 'Second']]
```

XML Validation

```
use Derafu\Xml\Exception\XmlException;
use Derafu\Xml\Service\XmlValidator;

$validator = new XmlValidator();

// Validate an XML document against a schema.
try {
    $validator->validate($xmlDocument, '/path/to/schema.xsd');
    echo "XML is valid!";
} catch (XmlException $e) {
    echo "Validation failed: " . $e->getMessage();
    $errors = $e->getErrors(); // Get detailed error information.
}
```

Advanced Usage

Working with Namespaces

```
// Create XPath query with namespace support.
$namespaces = ['ns' => 'http://example.com/namespace'];
$query = new XPathQuery($xmlContent, $namespaces);

// Query with namespace.
$result = $query->get('//ns:Element');
```

Array Structure for XML Creation

When creating XML from arrays, the structure follows these conventions:

- Simple key-value pairs become element nodes.
- Use `@attributes` for node attributes.

- Use `@value` for node text content when attributes are present.
- Arrays of values create repeated nodes with the same name.

Example:

```
$data = [
    'root' => [
        'element' => [
            '@attributes' => ['id' => '123'],
            '@value' => 'content',
        ],
        'items' => [
            'item' => ['value1', 'value2', 'value3'],
        ],
    ],
];
```

Produces:

```
<root>
  <element id="123">content</element>
  <items>
    <item>value1</item>
    <item>value2</item>
    <item>value3</item>
  </items>
</root>
```

Last updated on 09/09/2026