XML Service

XmlService Class

Anonymous · 09/09/2026

XmlService Class

The `XmlService` class is the central component of the Derafu XML library, providing a unified interface for encoding, decoding, and validating XML documents. It follows a service-oriented architecture by delegating the actual work to specialized components.

Service Structure

The `XmlService` relies on three key components:

1. **XmlEncoder**: Converts PHP arrays to XML documents.
2. **XmlDecoder**: Converts XML documents to PHP arrays.
3. **XmlValidator**: Validates XML documents against XSD schemas.

Basic Usage

```
use Derafu\Xml\Service\XmlEncoder;
use Derafu\Xml\Service\XmlDecoder;
use Derafu\Xml\Service\XmlValidator;
use Derafu\Xml\Service\XmlService;

// Initialize the components.
$encoder = new XmlEncoder();
$decoder = new XmlDecoder();
$validator = new XmlValidator();

// Create the service.
$xmlService = new XmlService($encoder, $decoder, $validator);
```

Encoding (Array to XML)

The `encode()` method converts a PHP array to an XML document:

```
$data = [
    'invoice' => [
```

```
        '@attributes' => [
            'id' => '12345',
            'date' => '2025-03-05'
        ],
        'client' => [
            'name' => 'Acme Corporation',
            'tax_id' => '123456789'
        ],
        'items' => [
            'item' => [
                [
                    '@attributes' => ['id' => '1'],
                    'description' => 'Product A',
                    'quantity' => '2',
                    'price' => '19.99'
                ],
                [
                    '@attributes' => ['id' => '2'],
                    'description' => 'Product B',
                    'quantity' => '1',
                    'price' => '29.99'
                ]
            ]
        ],
        'total' => '69.97'
    ]
];

$xmlDocument = $xmlService->encode($data);
```

This produces:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<invoice id="12345" date="2025-03-05">
  <client>
    <name>Acme Corporation</name>
    <tax_id>123456789</tax_id>
  </client>
  <items>
    <item id="1">
      <description>Product A</description>
      <quantity>2</quantity>
      <price>19.99</price>
    </item>
    <item id="2">
      <description>Product B</description>
      <quantity>1</quantity>
      <price>29.99</price>
    </item>
  </items>
```

```
  <total>69.97</total>
</invoice>
```

Using Namespaces

You can specify XML namespaces during encoding:

```
$namespace = ['http://example.com/invoice', 'inv'];
$xmlDocument = $xmlService->encode($data, $namespace);
```

This generates:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<inv:invoice id="12345" date="2025-03-05" xmlns:inv="http://example.com/invoice">
  <!-- Content with namespace prefixes -->
</inv:invoice>
```

Decoding (XML to Array)

The `decode()` method converts an XML document to a PHP array:

```
// Assuming $xmlDocument is a Derafu\Xml\XmlDocument instance.
$array = $xmlService->decode($xmlDocument);

// Or starting from a DOMElement.
$element = $xmlDocument->getDocumentElement();
$array = $xmlService->decode($element);
```

Handling Repeated Elements

By default, elements with the same name are collected into arrays:

```
<root>
  <item>Value 1</item>
  <item>Value 2</item>
  <item>Value 3</item>
</root>
```

Becomes:

```
[
    'root' => [
        'item' => [
            'Value 1',
```

```
            'Value 2',
            'Value 3'
        ]
    ]
]
```

You can control this behavior with the `$twinsAsArray` parameter:

```
$array = $xmlService->decode($xmlDocument, null, twinsAsArray: true);
```

Validation

The `validate()` method checks an XML document against an XSD schema:

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

try {
    $xmlService->validate($xmlDocument, '/path/to/schema.xsd');
    echo "XML is valid!";
} catch (XmlException $e) {
    echo "Validation failed: " . $e->getMessage();

    // Get detailed error information.
    $errors = $e->getErrors();
    foreach ($errors as $error) {
        echo "- $error\n";
    }
}
```

Schema Auto-Detection

If the XML document includes a `schemaLocation` attribute, you can omit the schema path:

```
// XML with xsi:schemaLocation="http://example.com/ns schema.xsd"
try {
    $xmlService->validate($xmlDocument); // Will use schema.xsd
} catch (XmlException $e) {
    // Handle validation error.
}
```

Error Translations

The validator can translate technical libxml error messages into more user-friendly messages:

```
$translations = [
```

```
    'element1' => 'Customer Info',
    'element2' => 'Product Details'
];

try {
    $xmlService->validate($xmlDocument, '/path/to/schema.xsd', $translations);
} catch (XmlException $e) {
    // Error messages will use the translated element names.
}
```

Integration Example

This complete example shows how to use the XML service for a typical workflow:

```
// Initialize the service.
$encoder = new XmlEncoder();
$decoder = new XmlDecoder();
$validator = new XmlValidator();
$xmlService = new XmlService($encoder, $decoder, $validator);

// Create XML from array.
$data = ['root' => ['element' => 'value']];
$xmlDocument = $xmlService->encode($data);

// Save to a file.
file_put_contents('document.xml', $xmlDocument->saveXml());

// Load from a file.
$loadedXml = new \Derafu\Xml\XmlDocument();
$loadedXml->loadXml(file_get_contents('document.xml'));

// Validate.
try {
    $xmlService->validate($loadedXml, 'schema.xsd');

    // Convert back to array.
    $array = $xmlService->decode($loadedXml);

    // Process the data.
    // ...

} catch (\Derafu\Xml\Exception\XmlException $e) {
    // Handle validation errors.
}
```

Last updated on 09/09/2026