

Round Trip and Fidelity

What is kept when an XML becomes an array and an array becomes an XML, how it is tested and what to consider with other XML formats

Anonymous · 17/10/2026

Round Trip and Fidelity

The library converts XML to arrays and arrays to XML. A round trip is to do both: an XML becomes an array, and the array becomes an XML again. This page says what the XML that comes back has, how it is tested and what to consider when the XML is not the one of a document that the library was made for.

What Is Kept

When an XML is decoded and the array is encoded again, the XML that comes back has:

- The same elements, in the same namespaces, with the same names and prefixes.
- The same attributes, with their values and their namespaces (`xsi:schemaLocation` is not `schemaLocation`).
- The declarations of namespaces (`xmlns` and `xmlns:prefix`).
- The same text, exactly as it is: with its spaces, its line breaks and its special characters. The library does not trim it.
- The same comments inside of the document element.
- The same order of the child elements, when the elements that are repeated are together (what an XML schema asks for, in a sequence).
- The elements without content, as empty elements.

What is kept is the data of the XML, not the way it is written. The spaces that format the XML (the ones between elements), the quotes of the attributes, `<a/>` or `<a></a>` and the CDATA sections are not kept as they were written: they are the same data.

```
$document = new XmlDocument();
$document->loadXml($xml);

$array = $decoder->decode($document);
$result = $encoder->encode($array);

// What the XML has is the same: for example, its canonical XML.
// (Without the spaces that only format the XML.)
```

What Is Not Kept

An array of names can not say some things that an XML can: the order among child elements with different names when one of them is repeated, the place of the text and of the comments among the child elements, what is outside of the document element, the declaration of the XML and the entities that a document type defines. See [What the Array Does Not Keep](#).

The Encoding

The encoding is not in the array: it is a property of the document that is written. Whoever uses the decoder and the encoder keeps it, with `getEncoding()` and `setEncoding()`:

```
$document = new XmlDocument();  
$document->loadXml($xml); // For example, an XML that declares ISO-8859-1.  
$encoding = $document->getEncoding();  
  
$result = $encoder->encode($decoder->decode($document));  
$result->setEncoding($encoding);  
  
$result->saveXml(); // In ISO-8859-1. What it can not write is a character reference.
```

The Document That the Encoder Gives

The document that `XmlEncoder::encode()` gives has the namespaces that the array declares: its elements and its attributes are in them, as in a document that is loaded from the XML. So XPath queries with namespaces and `C14N()` work in it without writing the XML and loading it again.

How It Is Tested

The test suite makes the round trip, in both directions, with the XML of its fixtures (an example of an electronic tax document and documents of the UBL standard of versions 2.1 to 2.5) and with small XML that have what other XML have: namespaces declared in several places, attributes with prefix, comments, text with spaces, CDATA, entities, unicode and ISO-8859-1.

- **XML to array to XML:** the XML that comes back has the same elements, namespaces, attributes, text, comments and order. For the fixtures it is also the same canonical XML (C14N), compared with a method that does not depend on how the comparison above is made.
- **Array to XML to array:** the array that comes back is the same.
- **Stability:** one more round trip gives the same XML.
- **Encoder:** a test with arrays made at random checks that what is encoded is decoded as it was, that what means nothing (`null`, `false` and `[]`) does not change the XML, and that a number or `true` is the same as its text.

Using It with Other XML Formats

The library does not know the schema of the XML. It converts any XML that is data (elements, attributes and text) and keeps it, so it works with formats as UBL. What follows is what to consider when the array is used to read or to write.

One Element or a List

An element that appears once is its value. An element that is repeated is a list. The same element can be one or the other in different documents:

```
<Invoice><InvoiceLine>...</InvoiceLine></Invoice>           <!-- ['Invoice' =>
['InvoiceLine' => [...]] -->
<Invoice><InvoiceLine>...</InvoiceLine><InvoiceLine>...</InvoiceLine></Invoice>
                                                                <!-- ['Invoice' =>
['InvoiceLine' => [...], [...]] -->
```

Whoever reads an array has to consider both. If the schema says that an element can be repeated, a function that always gives a list makes the code simpler:

```
function asList(mixed $value): array
{
    return is_array($value) && array_is_list($value) ? $value : [$value];
}

foreach (asList($array['Invoice']['cac:InvoiceLine'] ?? []) as $line) {
    // ...
}
```

The Prefixes Are in the Keys

The names of the keys are the names of the XML, with the prefix that the document uses (cbc:ID). Two documents with the same data and other prefixes give arrays with other keys. To read a document that is not under your control, query it with XPath and the namespaces (see [XPath Query](#)), which do not depend on the prefixes.

Everything Is Text

The values are text: '1.50' is not a number, and a date is not a date. A layer that knows the schema converts the types.

The Order of the Array Is the Order of the XML

The encoder writes the elements in the order of the array. When an XML has to follow the order of a

schema, the array has to have the keys in that order.

Last updated on 17/10/2026