XPath Reader

The XPathReader class, typed reads of an XML document selected by XPath, with the same methods as ArrayReader

Anonymous · 05/11/2026

XPathReader Class

`XPathReader` reads an XML document with types. It is to an XML document what `ArrayReader` is to an array: the same methods, with the same contract, and an XPath expression where `ArrayReader` takes a key.

```
use Derafu\Xml\XmlDocument;

$document = new XmlDocument();
$document->loadXml(<<<XML
<order>
    <header>
        <info>
            <docType>33</docType>
            <number>0021</number>
            <issued>2025-03-07</issued>
        </info>
        <seller>
            <taxId>TAX-1</taxId>
            <name>Seller name</name>
            <phone>111</phone>
            <phone>222</phone>
        </seller>
    </header>
    <line>
        <title>Item A</title>
        <qty>2</qty>
        <code><value>111</value></code>
        <code><value>222</value></code>
    </line>
    <line>
        <title>Item B</title>
        <qty>3</qty>
        <code><value>333</value></code>
    </line>
    <shipping>
        <customs/>
    </shipping>
```

```
</order>
XML);

$reader = $document->getReader();

$reader->requireString('//seller/name');   // 'Seller name'
$reader->getAsInt('//info/number');          // 21
$reader->getString('//info/number');         // '0021'
$reader->getString('//seller/branch');     // null: no node matches.
$reader->requireString('//seller/branch'); // throws XmlQueryException.
```

Getting a Reader

```
use Derafu\Xml\XPathReader;

$reader = $document->getReader();
$reader = XPathReader::fromDocument($document);

// With the namespaces that the XPath expressions use, and where the document
// comes from, which the errors say.
$reader = $document->getReader(['cbc' => 'urn:example:basic'], 'invoice.xml');
```

Without namespaces they are ignored, as in `XmlDocument::query()`: the names in the XPath match the elements by their local name. With them the XPath is used as it is written, with the prefixes that you give (see XPath Query).

The reader reads the document each time it is asked, so it sees the changes that are made to the document and the XML that is loaded in it again.

Strict and Converting Reads

The text of an XML node is always a string. The reader has two kinds of reads, as `ArrayReader` does:

- `getXxx()` is strict: it returns `null` when no node matches, and throws when what it finds is not of the type. `requireXxx()` is the same, and it also throws when no node matches.
- `getAsXxx()` converts the text to the type, with the conversions of Value: `'0021'` is `21`. `requireAsXxx()` throws when no node matches.

There are no strict getters for `int`, `float`, `number` or `bool`: they could never succeed with the text of a node.

Method	Returns
`getString()`, `requireString()`	`string`, the text as it is

Method	Returns
`getNonEmptyString()`, `requireNonEmptyString()`	`non-empty-string`; an empty element throws
`getAsString()`, `requireAsString()`	`string`
`getAsInt()`, `requireAsInt()`	`int`
`getAsFloat()`, `requireAsFloat()`	`float`
`getAsNumber()`, `requireAsNumber()`	`int` or `float`, as the text is
`getAsBool()`, `requireAsBool()`	`bool`; a text that is not a boolean throws
`getDateTime()`, `requireDateTime()`	`DateTimeInterface`, from a text that PHP reads as a date (see `ValueType::dateTime()`); another text throws
`getEnum()`, `requireEnum()`	the case of a backed enum with that value; another text throws

```
$reader->getString('//info/number');       // '0021': the text, as it is.
$reader->getAsInt('//info/number');        // 21
$reader->getAsFloat('//info/number');      // 21.0
$reader->getAsBool('//info/number');       // throws: it is not a boolean text.
```

The `getXxx()` methods take a default when no node matches, as `ArrayReader` does:

```
$reader->getString('//seller/branch', 'Main'); // 'Main'
$reader->getAsInt('//seller/branchCode', 0);           // 0
```

Any Type

`get()` and `require()` read what the XPath finds as any `ValueType`. All the other methods are made with them:

```
use Derafu\Value\ValueType;

$reader->require('//info/number', ValueType::toInt());    // 21
$reader->get('//info/issued', ValueType::dateTime());   // DateTimeImmutable
```

What they find is what `XPathQuery::get()` gives: `null` if no node matches, a string for a node with only text, an array for a node with child elements, and a list if several nodes match.

Presence

```
$reader->hasKey('//seller/name');  // true: a node matches.
$reader->hasValue('//seller/name'); // true: a node matches and it has a value.
```

One Node or Many

XML has no list type: an element that appears once is a value, and an element that appears several times is a list. The reads of a list take what the XPath finds, one node or many, as a list:

```
$reader->getStringList('//seller/phone');  // ['111', '222']
$reader->getListOf('//line/qty', ValueType::toInt()); // [2, 3]
$reader->getList('//header');              // [['info' => [...], 'seller' => [...]]]:
one node, a list of one.
```

Method	Returns
`getList()`, `requireList()`	`list<mixed>`
`getStringList()`, `requireStringList()`	`list<string>`
`getListOf($xpath, $type)`, `requireListOf()`	`list<T>`, each item is of the type

A list of nothing is `null` with `getXxx()` and an exception with `requireXxx()`, as in `ArrayReader`. `$reader->getStringList('//seller/phone') ?? []` is the list that is empty when there is none.

Structures

A node with child elements is a structure, an array. An element without text and without children is an empty structure: it is `[]` in the methods that read a structure.

```
$reader->getMap('//seller');           // ['taxId' => 'TAX-1', 'name' => 'Seller
name', 'phone' => ['111', '222']]
$reader->getMap('//customs');           // []: an empty element.
$reader->getMap('//seller/name');    // throws: it has text, it is not a structure.
```

Method	Returns
`getArray()`, `requireArray()`	`array`
`getMap()`, `requireMap()`	`array<string, mixed>`
`getMapOf($xpath, $type)`, `requireMapOf()`	`array<string, T>`
`getArrayOf($xpath, $type)`, `requireArrayOf()`	`array<array-key, T>`

An empty element is also `[]` when it is an item of a list of structures or a value of a structure of structures, which is when the type of the items is `map`, `array` or `list`:

```
// <r><item/><item><a>1</a></item></r>
$reader->getListOf('//item', ValueType::map());  // [[], ['a' => '1']]
```

```
// <r><text/><text>x</text></r>
$reader->getListOf('//text', ValueType::string()); // ['', 'x']: the items are not
structures.
```

`get()` and `require()` give an empty element as it is in `XPathQuery`, an empty string.

Readers of a Node

`getReader()` gives a reader of one node, and `getReaders()` a reader of each node that the XPath matches, one node or many. They are readers like the one that makes them, scoped to the node: all that is read from them is read in the same way, at any depth.

```
foreach ($reader->getReaders('//line') ?? [] as $item) {
    $item->requireString('title');           // 'Item A', then 'Item B'.
    $item->getAsInt('qty');                 // 2, then 3.

    // Inside of a node, one element or many.
    foreach ($item->getReaders('code') ?? [] as $codigo) {
        $codigo->requireString('value');   // '111', '222', then '333'.
    }
}

$seller = $reader->requireReader('//seller');
$seller->requireString('taxId');
```

Method	Returns
`getReader()`, `requireReader()`	`XPathReader`; it throws if the XPath matches more than one node
`getReaders()`, `requireReaders()`	`list<XPathReader>`

The reader of an empty element is a reader of `[]`: everything that it reads is absent. A node that is not an element or that has text is not a structure and has no reader.

An optional `ValueType` checks the structure of the node: `$reader->getReader('//seller', ValueType::map())`.

Relative XPath

The XPath expressions of a reader of a node are relative to it: `title` and `./title` are its child. A `//` still starts at the root of the document:

```
$item->requireString('//seller/name'); // 'Seller name': from the root.
```

The Whole Structure

The operations on the whole structure of the document, or of the node of a reader, are the ones of `ArrayReader`:

```
$reader->toArray();                      // What XmlDocument::toArray() gives.
$reader->toMap();                        // The structure, that must have only string
keys.
$reader->to(ValueType::array());         // Checks it against a type.
$seller->assertOnlyKeys('taxId', 'name', 'phone'); // An error for another child.
```

Their errors are `XmlQueryException` like the ones of the reads, with the XPath of the structure: `/` for the whole document, and for an element that is not allowed the XPath of the element (`//seller/branch`).

Errors

Every error is an `XmlQueryException` (see Exceptions) that has the XPath expression, the document and, when it was a type that failed, the cause:

```
use Derafu\Xml\Exception\XmlQueryException;

try {
    $reader->requireString('//seller/branch');
} catch (XmlQueryException $e) {
    $e->getMessage();         // The XPath "//seller/branch" is required: it matched
no node.
    $e->getXpathExpression(); // '//seller/branch'
    $e->getXmlDocument();     // The document.
}
```

The errors of a reader of a node have the full XPath of the node: `//seller/missing`, or `(//line)[2]/title` for the second of several. With a source, the message says it: `The XPath "//seller/branch" of invoice.xml is required: it matched no node.`

An XPath expression that is not valid throws the `XmlQueryException` of `XPathQuery`.

Methods

Kind	Methods
Presence	`hasKey()`, `hasValue()`
Any type	`get()`, `require()`

Kind	Methods
Text and values	`getString()`, `getNonEmptyString()`, `getDateTime()`, `getEnum()` and their `require` pairs
Conversions	`getAsString()`, `getAsInt()`, `getAsFloat()`, `getAsNumber()`, `getAsBool()` and their `requireAs` pairs
Structures	`getArray()`, `getMap()`, `getMapOf()`, `getArrayOf()` and their `require` pairs
Lists	`getList()`, `getStringList()`, `getListOf()` and their `require` pairs
Readers of a node	`getReader()`, `getReaders()` and their `require` pairs
The whole structure	`toArray()`, `to()`, `toMap()`, `toList()`, `assertOnlyKeys()`

Last updated on 05/11/2026