

Introduction

Typed PHP model of UBL 2.1 (Universal Business Language) to read, write and validate business documents against the official XSD

Anonymous · 12/10/2026

One Language for Every Business Document

github	derafu/ubl	last commit	today	CI	passing	code size	3.8 MiB	open issues	0	downloads	1
downloads	1 this month										

Derafu UBL is a typed PHP model of [UBL](#) (Universal Business Language): one PHP class per document type and per building block defined by the official UBL 2.1 XSD, with services to read and write those documents in multiple formats and validate them against the real schema.

It is deliberately generic: nothing in it knows about any one business (no fixed seller, no fixed tax rate, no fixed currency) and nothing about any e-invoicing profile (no Peppol rules, no country-specific mandatory fields or codelists). That kind of logic belongs in a package built on top of this one, for whichever network or country it targets.

Features

- **One class per UBL 2.1 type:** every document (`Quotation`, `Invoice`, `CreditNote`, ...) and every shared building block (`Party`, `MonetaryTotal`, `TaxTotal`, every `*Line` type) as a plain, typed PHP class.
- **Three wire formats:** serialize and deserialize the same document as `json`, `yaml` or `xml`, through one consistent API.
- **Schema validation:** validate a document's XML against the official OASIS UBL 2.1 XSD bundled with the package.
- **Array/JSON native:** every class also has `toArray()/fromArray()/jsonSerialize()`, for working with a document as plain data with no XML involved at all.

Installation

```
composer require derafu/ubl
```

Applying the Bundled Patch to xterr/php-ubl

`derafu/ubl` wraps `xterr/php-ubl` for `xml`, and ships a fix for a few of its known defects as a script that

patches it directly in `vendor/` (`tools/patch-xterr-ubl.php`). That script is wired as `derafu/ubl`'s own Composer `post-install-cmd/post-update-cmd`, so it reapplies automatically whenever `derafu/ubl` itself is installed or updated as the root project.

Composer does not run a dependency's own scripts for the project that requires it - only the root project's own `composer.json` scripts run. A project that depends on `derafu/ubl` must add the same command to its own `composer.json` to actually get the patch; otherwise it never runs, and `xterr/php-ubl` keeps its original, unpatched behavior silently.

`post-install-cmd/post-update-cmd` each accept either a single command or an array of them - use the array form, so this is added alongside whatever else is already there instead of replacing it:

```
{
  "scripts": {
    "post-install-cmd": [
      "php vendor/derafu/ubl/tools/patch-xterr-ubl.php"
    ],
    "post-update-cmd": [
      "php vendor/derafu/ubl/tools/patch-xterr-ubl.php"
    ]
  }
}
```

Run `composer install` (or `composer update`) once more after adding it, so the patch also applies to the `xterr/php-ubl` already sitting in `vendor/`.

Basic Usage

Building a Document

Using an object-oriented API:

```
use Derafu\Ubl\Oasis\Cac\Item;
use Derafu\Ubl\Oasis\Cac\LineItem;
use Derafu\Ubl\Oasis\Cac\MonetaryTotal;
use Derafu\Ubl\Oasis\Cac\Party;
use Derafu\Ubl\Oasis\Cac\PartyName;
use Derafu\Ubl\Oasis\Cac\Price;
use Derafu\Ubl\Oasis\Cac\QuotationLine;
use Derafu\Ubl\Oasis\Cac\SupplierParty;
use Derafu\Ubl\Oasis\Cbc\Amount;
use Derafu\Ubl\Oasis\Cbc\Identifier;
use Derafu\Ubl\Oasis\Cbc\Text;
use Derafu\Ubl\Oasis\Doc\Quotation;

$quotation = (new Quotation())
```

```

->setId((new Identifier())->setValue('Q-001'))
->setIssueDate(new DateTimeImmutable('2026-01-15'))
->setSellerSupplierParty(
    (new SupplierParty())->setParty(
        (new Party())->setPartyNames([
            (new PartyName())->setName((new Text())->setValue('Acme Corp')),
        ])
    )
)
->setQuotedMonetaryTotal(
    (new MonetaryTotal())->setPayableAmount(
        (new Amount())->setValue('100')->setCurrencyID('EUR')
    )
)
->setQuotationLines([
    (new QuotationLine())
        ->setId((new Identifier())->setValue('1'))
        ->setLineItem(
            (new LineItem())
                ->setId((new Identifier())->setValue('1'))
                ->setPrice(
                    (new Price())->setPriceAmount(
                        (new Amount())->setValue('100')->setCurrencyID('EUR')
                    )
                )
                ->setItem((new Item())->setName((new Text())->setValue('Widget')))
        ),
]);

```

Only `ID`, `IssueDate`, `SellerSupplierParty`, `QuotedMonetaryTotal` and one `QuotationLine` (with `LineItem.ID` and `LineItem.Item`) are actually required: `true` in UBL - an `Item` with no `Name`, and a line with no `Price`, are schema-valid too. The seller's and item's names, and the price, are kept here only so the example reads like a real quotation, the same choice behind this package's own committed `Quotation` fixture.

Or using a plain PHP array - the exact same shape `toArray()/jsonSerialize()` produce, and `json/yaml` wrap under the document type's name (see `Formats`):

```

use Derafu\Ubl\Oasis\Doc\Quotation;

$quotation = Quotation::fromArray([
    'ID' => 'Q-001',
    'IssueDate' => '2026-01-15',
    'SellerSupplierParty' => [
        'Party' => [
            'PartyName' => [
                ['Name' => 'Acme Corp'],
            ],
        ],
    ],
]);

```

```

],
'QuotedMonetaryTotal' => [
    'PayableAmount' => [
        '@value' => '100',
        '@attributes' => ['currencyID' => 'EUR'],
    ],
],
'QuotationLine' => [
    [
        'ID' => '1',
        'LineItem' => [
            'ID' => '1',
            'Price' => [
                'PriceAmount' => [
                    '@value' => '100',
                    '@attributes' => ['currencyID' => 'EUR'],
                ],
            ],
            'Item' => [
                'Name' => 'Widget',
            ],
        ],
    ],
],
]);

```

Serializing and Deserializing

```

use Derafu\Ubl\Service\Deserializer;
use Derafu\Ubl\Service\Serializer;

$serializer = new Serializer();
$deserializer = new Deserializer();

$yaml = $serializer->serialize($quotation, 'yaml');
$json = $serializer->serialize($quotation, 'json');
$xml = $serializer->serialize($quotation, 'xml');

$sameQuotation = $deserializer->deserialize($yaml, 'yaml');

```

See **Formats** for the exact shape of each one.

Validating Against the Official Schema

```

use Derafu\Ubl\Exception\ValidationException;
use Derafu\Ubl\Service\SchemaValidator;

try {
    (new SchemaValidator())->validate($xml);
}

```

```
} catch (ValidationException $e) {  
    foreach ($e->getErrors() as $error) {  
        echo $error->message, "\n";  
    }  
}
```

See **Validation** for how the document type is resolved and what else can be inspected on a failure.

What This Package Is Not

- **Not a profile.** PEPPOL BIS, EN 16931, and any country's CIUS (Core Invoice Usage Specification) restrict or extend this base syntax with their own mandatory fields, codelists and Schematron business rules. Those are separate packages that depend on this one, never the other way around.
- **Not a mapping layer.** Turning your own application's data (prices, customers, products) into one of these documents is the job of your application, not of this package.

Last updated on 12/10/2026