Write a datatype

The PHP side of a custom datatype - the eZDataType methods grouped by when the kernel calls them, registration with eZDataType::register(), storage in data_text, data_int and data_float or a table of your own, string...

Source file: doc/eng-us/help/php-developers/write-a-datatype.md Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/php-developers/write-a-datatype.md

This page is for PHP developers who write a new field type for content classes. A datatype is a class extending `eZDataType` (`kernel/classes/ezdatatype.php`) plus edit and view templates. The complete working example, with its templates and registration, is on Writing a custom datatype; this page maps the PHP methods to the moments the kernel calls them, so you know which to override.

Contents

1. Registration
2. The methods by moment
3. Storage
4. String, package and search forms
5. Class attribute settings
6. PHP points to watch
7. Testing a datatype

Registration

PHP

```php
class acmeColorType extends eZDataType
{
    const DATA_TYPE_STRING = 'acmecolor';

    public function __construct()
    {
        parent::__construct( self::DATA_TYPE_STRING, ezpI18n::tr( 'extension/acmereport', 'Color'
 ),
                             array( 'serialize_supported' => true ) );
    }
    // ... the methods below
}

eZDataType::register( acmeColorType::DATA_TYPE_STRING, 'acmeColorType' );
```

The file is `extension/acmereport/datatypes/acmecolor/acmecolortype.php`, found through `content.ini`:

INI

```
<?php /* #?ini charset="utf-8"?
[DataTypeSettings]
ExtensionDirectories[]=acmereport
AvailableDataTypes[]=acmecolor
*/ ?>
```

The type string is stored in every attribute of this type, forever: choose it once, prefixed, lower case.

The methods by moment

Moment	Methods
an editor submits the edit form	`validateObjectAttributeHTTPInput()`, then `fetchObjectAttributeHTTPInput()`; `customObjectAttributeHTTPAction()` for extra buttons
the attribute is stored	`storeObjectAttribute()`
a new version or translation is created	`initializeObjectAttribute()`, `postInitializeObjectAttribute()`
the object is published	`onPublish()`
templates read the value	`objectAttributeContent()` (`$attribute.content`), `hasObjectAttributeContent()` (`$attribute.has_content`), `title()`
the search index is built	`isIndexable()`, `metaData()`
lists are sorted by the attribute	`sortKey()`, `sortKeyType()` (`'string'` or `'int'`)
a version is removed, or the object goes to the trash	`deleteStoredObjectAttribute()`, `trashStoredObjectAttribute()`, `restoreTrashedObjectAttribute()`
export and import	`toString()`, `fromString()`, `serializeContentObjectAttribute()`, `unserializeContentObjectAttribute()`
the class is edited	`validateClassAttributeHTTPInput()`, `fetchClassAttributeHTTPInput()`, `initializeClassAttribute()`, `storeClassAttribute()`

`validate...` returns `eZInputValidator::STATE_ACCEPTED`, `STATE_INVALID` (after `setValidationError()`) or `STATE_INTERMEDIATE`. `fetch...` returns `true` when it stored a value. The life cycle in detail: The datatype life cycle.

Storage

Where	For
`data_text`, `data_int`, `data_float` of `ezcontentobject_attribute`	simple values, one per attribute version and language
`data_text1` ... `data_int1` ... of `ezcontentclass_attribute`	settings of the class attribute
a table of your own, keyed by attribute id and version	structured values; remove rows in `deleteStoredObjectAttribute()`

Every version and translation has its own attribute row, so a value stored elsewhere must be

keyed by attribute id **and** version, and copied in `initializeObjectAttribute()` when a new version starts from an old one.

String, package and search forms

- `toString()` and `fromString()` are the plain text form used by imports, the REST API and scripts; make them round trip: `fromString( toString( x ) )` gives `x`.
- `serializeContentObjectAttribute()` and `unserializeContentObjectAttribute()` write and read the value in content packages; set `'serialize_supported' => true` in the constructor when you implement them.
- `metaData()` is the text the search engine indexes; return the words a visitor would search for.

Class attribute settings

A datatype with options per class attribute (a default color, a list of allowed values) reads them from the class edit form in `fetchClassAttributeHTTPInput()`, stores them in `data_text1` and friends, and shows them in its class templates `class/datatype/edit/<type>.tpl` and `class/datatype/view/<type>.tpl`.

PHP points to watch

- **Match the parent's signatures exactly.** A parameter less, or a by-reference marker that the parent does not have, is a fatal error when the class loads, and every page that touches content breaks (PHP 8 migration notes for your code).
- **Declare properties** your subclass uses; PHP 8.2 deprecates dynamic properties.
- `null` **from the database.** `data_text` can be `null`; cast before string functions (`(string)`).
- **Escape in templates**, not in `objectAttributeContent()`: the content is data, `|wash` makes it safe in HTML.

Testing a datatype

1. DB-less: `toString()`/`fromString()` round trips and the validation rules in a separate validator class (PHPUnit without a database).
2. On an installation: add an attribute to a test class; create, edit, translate, copy, move to the trash, restore and remove objects; export and import a package.
3. Check the search index and sorting if you implemented them.

Related pages

- Datatypes (extension developers)
- eZDataType API
- Datatype pattern
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

Become an Exponential Partner

Partners share their information, provide funding, contribute back and develop the products with us, and that is how Exponential grows. A partner website and dashboard are planned for mid-2027: please become a partner today to speed up our growth rate and land new features faster.

Partners: start here | How to apply

Exponential Copyright © 1998-2026 Exponential Foundation and 7x
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
This documentation is free documentation, licensed under the GNU Free Documentation License 1.3 or later:
https://www.gnu.org/licenses/fdl-1.3.html.