Content classes and attributes

For CMS developers, every property of a content class and of a class attribute, what each one changes on the site, and the rules for identifiers, name patterns, containers and changing a class that already has content.

Source file: `doc/eng-us/help/cms-developers/content-classes-and-attributes.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/cms-developers/content-classes-and-attributes.md

This page is for developers who define the content types of a site. It goes through the properties of a class and of its attributes one by one and says what each changes for editors, templates, URLs and search. The click path for creating a class is in Creating a content class.

Contents

1. Class properties
2. Attribute properties
3. Identifiers are a contract
4. Name patterns
5. Class groups
6. Changing a class that has content
7. Classes in version control

Class properties

Property	Stored as	What it changes
Name, description	`serialized_name_list`, translatable	what editors see in the create menu and lists
Identifier	`identifier`	what templates, overrides, fetch filters and imports use
Object name pattern	`contentobject_name`	the name of every object: in the tree, in lists, in the default URL
URL alias pattern	`url_alias_name`	the URL part of each node, when it should differ from the name
Container	`is_container`	whether objects of the class may have children
Default sorting	`sort_field`, `sort_order`	the order of children of new nodes
Default object availability	`always_available`	whether objects are shown in languages they have no translation for
Remote ID	`remote_id`	a stable ID for the class itself, used by packages

The columns are defined in `kernel/classes/ezcontentclass.php`. A class is edited as a temporary version and only replaces the defined class when you press **OK**, so a half-finished

edit never reaches editors.

Attribute properties

Property	Stored as	What it changes
Name, description	translatable	the label and help text in the edit form
Identifier	`identifier`	`$node.data_map.<identifier>` in templates, `attributes` keys in PHP
Datatype	`data_type_string`	storage, validation, edit form, view template, search
Required	`is_required`	the object cannot be published without a value
Searchable	`is_searchable`	whether the value goes into the search index
Translatable	`can_translate`	whether each translation has its own value, or all share the main language's
Information collector	`is_information_collector`	visitors fill this field on the site, as a form (Forms and information collection)
Category	`category`	the group the field is shown in when editing; `content.ini [ClassAttributeSettings] CategoryList[]` ships `content` and `meta`
Placement	`placement`	the order of fields in the edit form and in `data_map` loops

Most datatypes add settings of their own: a maximum length for `ezstring`, the allowed classes and the default location for `ezobjectrelationlist`, the options of `ezselection`, the number of rows of `eztext`. These are stored in the class attribute's data fields and documented with each datatype (Datatypes and their templates).

Identifiers are a contract

Templates, override rules, fetches, imports, packages and the REST interface all refer to class and attribute identifiers. Treat them like a public API:

- lower case letters, digits and underscores, English words that say what the field is (`title`, `intro`, `body`, `image`, `start_date`);
- the same identifier for the same meaning in every class, so one template part can show the `image` of an article, an event and a product;
- never rename an identifier that code uses; rename the human name instead, which is free;
- prefix the identifiers of classes that belong to your project when they could clash with shipped ones (`exp_event` rather than a generic name already used by a site package).

The builder's chapter says the same from the template side: Content classes for builders.

Name patterns

The object name pattern builds the name from attribute values. Each `<identifier>` is replaced by that attribute's text; `<a|b>` uses `a`, or `b` when `a` is empty (`kernel/classes/eznamepatternresolver.php`):

Pattern	Result
`<title>`	the title
`<short_title|title>`	the short title, else the title
`<first_name> <last_name>`	a person's full name

The name is what URL aliases are made from unless the class has its own URL alias pattern, so a good pattern also gives good addresses. When a name has to be computed by code, a pattern can name a PHP resolver class (`content.ini` `[NamePatternSettings]` `ResolverClasses[]`, interface `expNamePatternResolverInterface`); see Names made by a PHP class.

Class groups

Classes are filed in **class groups** (Content, Users, Media, Setup in a new installation). Groups only organize the class list in the administration and the classes offered in each part of it; they do not change permissions. Class groups explains the pages and how to remove a group safely.

Changing a class that has content

Change	Effect on existing objects
Add an attribute	every object gets the attribute with an empty or default value; a required one is only enforced at the next edit
Remove an attribute	its values are removed from every object and every version; there is no undo
Change the name, description or category	nothing but the labels
Change an identifier	stored values stay, but every template, setting and import that used the old identifier breaks
Change the datatype	not offered: add a new attribute, copy the values with a script, remove the old one
Change the name pattern	names are rebuilt when objects are published again
Change objects to another class	not in the kernel; the `expchangeclass` extension does it with an attribute mapping (Case study: expchangeclass)

Before changing a class on a live site, take a database backup, try the change on a copy, and check the templates that use the class with `grep -rn "data_map.<identifier>" extension/` .

Classes in version control

A class made in the administration lives only in the database. To bring it to another installation:

- export it in a **package** (class export) and install the package there (Packages);
- or write a script that creates it: the RAD wizard **Content extension** (`/setup/contentextension`) generates one (RAD tools).

Either way, keep the result in your project's extension so the next installation is built from files, not by hand. Site packages and installers shows how whole sites are built that way.

Related pages

- Choosing datatypes
- Designing a content model
- eZContentClass reference
- Lesson 4.1: Classes, attributes and identifiers
- 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.