The content model

For CMS developers new to Exponential, the whole content model on one page. It covers classes, attributes and datatypes, objects, versions and translations, nodes in the content tree, sections, object states and how…

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

This page is for developers who build sites on Exponential and need the content model in their head before they design anything. It names every building block, shows how they connect and links to the page that goes deep on each.

Contents

1. The model in one picture
2. Classes, attributes and datatypes
3. Objects, versions and translations
4. Nodes and the content tree
5. Sections and object states
6. Users are content too
7. Identifiers you will use every day
8. What lives outside the content model
9. Where to go next

The model in one picture

```
Content class (article)            the type: a name, an identifier and a list of attributes
  Class attribute (title, body)    each one of a datatype: ezstring, ezxmltext, ezimage ...
Content object (one article)       an instance of a class
  Version 1, 2, 3 ...              draft, published, archived; each version holds every translati
on
    Object attribute               the value of one class attribute in one version and language
  Node (location) 61, 74           the object's places in the content tree; one is the main node
Section                            a label on the object for access rules and designs
Object state                       labels of your own (in review, approved ...), grouped
```

Everything an editor creates, from a page or a news article to an image, a file, a user account or a user group, is a **content object** of some **content class**. There is no separate "page" table, "media" table or "user profile" table that you design around: you design classes.

Classes, attributes and datatypes

A **content class** is the type definition. It has a name (translatable), an identifier such as `article`, a list of attributes, an object name pattern (for example `<title>`), an optional URL alias pattern, a flag that says whether objects of the class may have children (`is_container`), the default sort order of those children, and an *always available* flag. These are the columns of

`ezcontentclass` (`kernel/classes/ezcontentclass.php`).

Each **class attribute** has an identifier, a datatype and flags: required, searchable, translatable and information collector (`kernel/classes/ezcontentclassattribute.php`). The **datatype** decides how the value is stored, validated, edited, searched and shown. The kernel ships 35 datatypes in `settings/content.ini [DataTypeSettings] AvailableDataTypes[]`, and extensions add more.

Details: Content classes and attributes and Choosing datatypes.

Objects, versions and translations

A **content object** never changes in place. Every edit creates a new **version** (a draft); publishing it makes it the current version and archives the previous one. A version holds all the translations of the object, and each translation is a set of object attributes in one language. The number of archived versions kept is limited per class (`settings/content.ini [VersionManagement]`).

Because every change is a version, you get history, comparison and restore for free, and you must never write the content tables directly: the kernel's publish operation creates the version, runs workflows, updates the search index and clears caches (Content from PHP).

Details: Versions, drafts and publishing and Translations and languages.

Nodes and the content tree

An object appears on the site through its **nodes** (also called **locations**). The nodes form one tree. An object can have several nodes, so the same article can appear under *News* and under *Sport*; one of them is the **main node**, and the others point to the same object. The tree decides:

- the navigation and the URL of each page (the URL alias of a node is built from its path);
- the scope of permissions (a policy can be limited to a subtree);
- which template is used (overrides can match a parent node or a depth);
- what a fetch returns (children, a subtree, a depth).

The shipped tree has fixed tops: the content root is node 2, the users are below node 5, the media library below node 43 (`settings/content.ini [NodeSettings]`). Details: Objects, nodes and locations.

Sections and object states

A **section** is one label per object, used by roles (a policy can be limited to a section), by template overrides (`Match[section]`) and by navigation parts in the administration. A new installation has five: Standard, Users, Media, Setup and Design (Sections).

Object states are labels of your own, in groups, for example a review state with *draft*, *in review* and *approved*. Roles can be limited to states, so they are how you model editorial stages beyond draft and published (Object states).

Details: Sections, roles and policies.

Users are content too

A user account is an object of the `user` class (class ID 4, `site.ini [UserSettings] UserClassID`) with an `ezuser` attribute that holds the login, e-mail address and password hash. User groups are objects of the `user_group` class placed in the user tree, and a user belongs to the groups where its nodes are. Roles are assigned to users or groups. That means you can add attributes to the user class like to any other class, and fetch users like content (Users and registration).

Identifiers you will use every day

Identifier	Example	Use it for
Class identifier	`article`	templates, overrides, fetch filters, imports; stable across installations
Attribute identifier	`title`, `body`	`$node.data_map.title` in templates, `attributes` in PHP
Node ID	2	links and fetches on one installation only
Object ID	57	relations; one installation only
Remote ID	`myimport-article-42`	a stable ID of your own for objects you import or reference from code

Node and object IDs differ between installations. Code, settings and packages that must work on every installation refer to class identifiers and remote IDs, not numbers.

What lives outside the content model

Some things are not content objects: settings (INI files), templates and designs, roles and policies, URL wildcards, layouts of Exponential Layouts (their own tables, linked to content by rules), workflows and triggers, orders of the shop and information collected from forms. Each has its own page in this section.

Where to go next

1. Designing a content model: the task, from requirements to classes.
2. Creating a content class: the task in the administration.
3. Mapping content to Exponential: the same model seen from a migration, with a mapping table for other systems.

Related pages

- The content model and editing content: the administrator's guide
- Content classes for builders: the builder's chapter of the book
- Nodes, objects and attributes in templates: the designer's view
- Glossary for CMS developers
- 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.