Sections: grouping content for permissions, navigation and designs

Source file: doc/eng-us/guides/sections.md Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/guides/sections.md

This guide teaches everything about sections in Exponential: what a section is and what it is not, the section pages of the administration interface (the list, one section, the edit form, the removal confirmation, assigning content), how a section is chosen for new content and how it changes when content moves, how roles and policies use sections, what the navigation part of a section does, how templates are chosen by section, what can be done from a shell or another program, three complete workflows, and what to do when something does not behave as expected.

It is written for administrators who set up who may see and edit what, and for site builders who give part of a site its own look. Read sections 1 and 2 first; after that every section stands on its own. Every page, field, rule and command below was checked against the code of this repository on 5 October 2026; file names are given in the References so that each claim can be checked again.

Guides · Related: Content jobs · Remote services (expsection) · Audit trail · Security and audit

In short

- A **section** is a named group of content objects. Every object belongs to exactly one section. A section has a name, an identifier, an ID and a navigation part, and nothing else.
- Sections are used in three places: **permissions** (a policy or a role assignment can be limited to sections), **the administration interface** (the navigation part decides which top menu tab is active while the content is viewed) and **designs** (templates can be overridden for one section, `Match[section]` or `Match[section_identifier]`).
- **Setup > Sections** (`/section/list`) shows every section with its objects, the roles that use it and whether it can be removed. From there you create, view, edit, remove and assign.
- **New content takes the section of its parent.** Moving a subtree carries the section of the new parent along to the objects that still had the old parent's section. To put existing content into a section, use **Assign content** or the Details tab of an object, and choose which objects below it get the section too: all of them (the default), only those in the section the chosen item had, or none (section 8.5).
- **A section can only be removed when nothing uses it**: no object (drafts and archived objects included), no policy with a Section limitation naming it and no role assignment limited to it. Removal asks for confirmation and cannot be undone.
- The default installation has five sections: Standard (1), Users (2), Media (3), Setup (4) and Design (5). The Anonymous role may read Standard only, which is why content put into a new section is hidden from visitors until a policy allows it.

Contents

- 1. What a section is, and what it is not
- 2. The sections of a new installation
- 3. The section list
- 4. One section's page
- 5. Creating a section
- 6. Editing a section, and what a change affects
- 7. Removing a section
- 8. Which section content is in: creating, moving and assigning (8.5 Which objects below get the section)
- 9. Sections in roles and policies
- 10. Navigation parts
- 11. Templates and designs by section
- 12. From a shell and from other programs
- 13. Workflows, step by step
- 14. Troubleshooting
- 15. Who may use the section pages
- References

1. What a section is, and what it is not

A section is a row in the table `ezsection` with four fields that matter:

Field	Example	Used for
ID	1	Everything that stores a reference: the `section_id` of every object, policy limitations, role assignments, `Match[section]` in override settings. It never changes.
Name	`Standard`	What people read: the section lists, role pages, the section choice of an object.
Identifier	`standard`	What templates and code name: `fetch( 'section', 'object', hash( 'identifier', 'standard' ) )`, `Match[section_identifier]=standard`. Letters, digits and _, starting with a letter, unique.
Navigation part	`ezcontentnavigationpart`	Which top menu tab of the administration interface is active while an object of the section is viewed or edited.

A fifth field, `locale`, exists in the table and is kept when a form sends it, but no page of the administration interface sets it and nothing in the kernel reads it to make a decision.

What a section is **not**:

- It is not a place in the content tree. A section has no root node and no URL. The objects of one section can be anywhere in the tree, and one folder can hold objects of several sections. Assigning content to a section starts at a node, but what is stored is the section of each

object.

- It is not a siteaccess. Which site a visitor sees is decided by the siteaccess; what the visitor may read in it is decided by roles, which can be limited by section.
- It is not a state. Object states (Setup > States) are a second, independent way to group objects for permissions; a policy can be limited by both.
- It does not belong to a location. An object with two locations has one section, the same at both.

2. The sections of a new installation

A new installation (`share/db_data.dba`, `kernel/sql/common/cleandata.sql`) creates five sections:

ID	Name	Identifier	Navigation part	Holds
1	Standard	`standard`	Content structure (`ezcontentnavigationpart`)	the content tree from the root folder down
2	Users	`users`	User accounts (`ezusernavigationpart`)	the user accounts and user groups
3	Media	`media`	Media library (`ezmedianavigationpart`)	the media library: images, files, their folders
4	Setup	`setup`	Setup (`ezsetupnavigationpart`)	setup objects
5	Design	`design`	Design (`ezvisualnavigationpart`)	design objects

The default Anonymous role has `content/read` and `content/pdf` limited to Section Standard. That one limitation is what keeps user accounts, media folders and everything in other sections away from visitors who are not logged in, and it is the reason most of the workflows in section 13 begin with a new section.

Sites installed from packages or extensions often have more. On the demonstration server, for example, the newsletter extension created "CJW Newsletter" (34) and a package added "Restricted" (6) and "Sample: collaboration" (35). Two of them have no identifier, which the section list marks as needing attention (section 3.4).

3. The section list

Setup > Sections, address `/section/list`. The page answers four questions at a glance: which sections are there, what is in each, which ones does the permission system depend on, and which could go.

3.1 The overview

Six figures across the top, counted over all sections, not only the ones on the current page:

Figure	Counts
Sections	all sections
Published objects	published objects in all sections together
Used by roles	sections that a policy limitation or a role assignment names

Figure	Counts
Without published objects	sections with no published object (they may still hold drafts or archived objects)
Can be removed	sections that nothing uses (section 7)
Need attention	sections without an identifier, or with a navigation part that `menu.ini` does not list; shown in red when not 0

The numbers come from three grouped database queries, however many sections there are (one query per section and status on MongoDB), so the page stays fast with hundreds of sections and millions of objects.

3.2 Finding a section

Find a section searches the name, the identifier, the ID, the navigation part (its name and its identifier) and the names of the roles that use the section. Several words must all match. **Show** narrows the list to one kind: All, With published objects, Without published objects, Used by roles, Can be removed, Need attention. A line under the controls says how many sections of the page are shown; when none matches, the page says so.

Search and filter work on the sections of the current page. With the default of 25 per page this is every section on almost every site. Both need JavaScript; without it the controls are not shown and the list is complete.

3.3 A section's card

Each section is a card:

- **The tick box** selects the section for **Remove selected** (only for users who may edit sections).
- **The name** opens the section's page; next to it the identifier, in monospace, and the ID.
- **Badges**: the number of published objects (or "No published objects"), "Used by roles", "Can be removed", and the warnings "No identifier" and "Unknown navigation part".
- **View**, **Edit** and **Assign content**. Assign content is greyed out, with the reason in its tooltip, when your role does not allow you to assign this section (section 9.3).
- **Facts**: the navigation part, the objects (published, drafts, archived), the roles with a policy limited to the section (each a link to the role), and the number of role assignments limited to it.

Cards that need attention have an amber edge on the left.

3.4 What "Need attention" means

- **No identifier**: older installations and some packages created sections before identifiers existed. Such a section works for permissions and content, but templates, `fetch( 'section', 'object', hash( 'identifier', ... ) )` and `Match[section_identifier]` cannot name it, and the edit form will not save it until it has one (section 5).
- **Unknown navigation part**: the section names a part that no `menu.ini` lists, usually because the extension that added it is no longer active. Its content then shows no active top menu tab. Choose another part (section 10).

3.5 Page size, pages and the buttons

Per page under the list chooses how many sections a page shows. The sizes come from `admininterface.ini [PaginationSettings]` (`ItemsPerPage[section/list]=25` by default) and the choice is kept as the preference `admin_section_list_limit`. The page links (`/section/list/(offset)/25`) follow.

Below them: **Remove selected** (section 7) and **New section** (section 5); New section is also at the top. A message at the top of the list says what the last action did: "The section Members area was created.", "The section ... was saved.", "Removed: ...", or that Remove selected was pressed with nothing ticked.

4. One section's page

Click a section's name or **View**: `/section/view/<ID>`.

1. **The title**: name, identifier and ID.
2. **Edit**, **Assign content** and **All sections**. Edit and Assign content appear only for users who may use them.
3. **The figures**: published objects, drafts of new objects (objects that were never published), archived objects, roles with policies for the section, role assignments limited to it.
4. **Details**: name, identifier (or a warning that it has none), ID, navigation part, and **Removal**: either "Can be removed" or "In use" with the rule.
5. **Roles with policies limited to this section**: each role with its ID, as a link, and the policies of that role whose Section limitation names this section, as `module/function` (for example `content/read`). Two roles with the same name are told apart by their IDs.
6. **Users and user groups with role limitations associated with this section**: every role assignment made with "Assign with limitation: Section" for this section (section 9.2), the user or group and the role.
7. **Objects within this section**: the published objects, newest first, with their class and the time they were last modified, 10, 25 or 50 at a time (the general list-length preference of the administration interface). Each name opens the object.

5. Creating a section

1. **Setup > Sections**, **New section**. The form opens at `/section/edit/0`.
2. **Name**: what people will read, for example `Members area`.
3. **Identifier**: filled in from the name while you type the name (`members_area`), as long as you have not typed an identifier yourself. It must start with a letter and contain only letters, digits and `_`, and no other section may have it. The page checks this before sending; the server checks it again.
4. **Navigation part**: the top menu tab that will be active for this content. For ordinary pages leave **Content structure**.
5. **Create section**. The list opens with "The section Members area was created."

The new section is empty. Nothing can see it yet except users whose policies are not limited by section, such as administrators. Section 13 shows the usual next steps.

The server refuses three identifiers, with the message shown above the form and next to the field:

Message	Cause
Identifier can not be empty	the field was empty, or only spaces
Identifier should consist of letters, numbers or '_' with letter prefix.	it starts with something other than a letter, or contains a space, a hyphen, an accented letter or another sign
The identifier has been used in another section.	another section has it (compared exactly)

The name is not checked; an empty name is stored as it is, which is why the form asks for one.

Every save is recorded in the audit trail as `content.section.change`, with the values before and after (Audit trail).

6. Editing a section, and what a change affects

Edit on the card or on the section's page opens `/section/edit/<ID>`. It is the same form, and below the fields it says what a change affects for this section: how many objects it holds and how many policies, roles and role assignments are limited to it.

You change	Effect
Name	Only what people read. Policies and role assignments store the ID, so no permission changes.
Identifier	Everything that names the section by its identifier stops matching: `Match[section_identifier]` in override settings, `fetch( 'section', 'object', hash( 'identifier', ... ) )`, code that calls `eZSection::fetchByIdentifier()`. Change those together with it.
Navigation part	The active top menu tab for every object of the section, from the next page view.

Saving clears the view cache of every object in the section (`eZContentCacheManager::clearContentCacheIfNeededBySectionID()`) and sends the event `content/section/cache`, so that listeners such as an HTTP cache can purge too. On a section with very many objects this takes a moment.

Cancel returns to the list without saving.

7. Removing a section

7.1 The rule

A section can be removed only when nothing uses it (`eZSection::canBeRemoved()`):

1. no object is in it, of any status: published, never published (a draft of a new object) or archived;
2. no policy has a Section limitation that names it;
3. no role assignment is limited to it.

The list shows which sections pass with the badge **Can be removed**, and the figure of the same name counts them.

7.2 Step by step

1. Tick the sections on the list and press **Remove selected**.
2. The confirmation page lists, in red, the sections that **will be removed**, with the warning, and separately the sections that **cannot be removed**, each with its reasons: "Holds 173 objects (173 published, 0 drafts, 0 archived)", "18 policies are limited to it" with links to the roles, "2 role assignments are limited to it".
3. **Remove N sections** removes the ones that may go. **Cancel** keeps every section. If none of the ticked sections may go, there is only **Back to the sections**.
4. The list opens with "Removed: ..." naming them.

Each removal clears the view cache of the section's objects (there are none, by the rule), sends `content/section/cache` and is recorded in the audit trail as `content.section.remove`. The confirmation is used up by the removal: posting the same form again removes nothing.

Removing needs `section/edit` without limitation (section 15).

7.3 What removal does not check

- `NewSection` **limitations** of `section/assign` policies. A policy that offered the section to assign keeps its ID, which then points nowhere. Edit such policies before or after.
- **Templates and settings.** `Match[section]=<ID>`, `Match[section_identifier]=...` and template code that names the section stop matching silently.
- **Section IDs used by settings**, such as `site.ini [UserSettings] DefaultSectionID` (section 8.1). It is not consulted.

There is no undo. A new section with the same name and identifier gets a new ID, so policies and settings that named the old ID do not apply to it.

8. Which section content is in: creating, moving and assigning

8.1 New content

- An object created below a node takes **the section of its parent's object** (`eZContentObject::createWithNodeAssignment()`), and on its first publish the publish operation step `update-section-id` fills in the parent's section for any object that has none yet.
- A section an object already has is kept when it is published. A copy starts without one and therefore takes the section of the place it is copied to.
- A user who registers on the site is created with `site.ini [UserSettings] DefaultSectionID`; the default `0` means "the section of the user group in `DefaultUserPlacement`", normally Users.

8.2 Moving and locations

- **Moving a subtree** (the main location of its top object) gives the new parent's section to every object in the subtree **that still had the old parent's section**. Objects that had a section of their own keep it. A secondary location being moved changes nothing.

- **Changing the main location** of an object does the same for its subtree.
- **Adding a secondary location** changes nothing: an object has one section, wherever it appears.

8.3 Assign content to a subtree: section/assign

This is the usual way to put existing content into a section.

1. On the section's card or page, **Assign content**. The content browser opens with a note at the top: "Choose start location for the <Members area> section".
2. Above the browser, choose **which objects get the section** (section 8.5): the chosen item and every object below it (preselected), the chosen item and the objects below it that have its section, or the chosen item only.
3. Navigate with the tabs, the tree and the list, choose the item with its radio button, and press **Select**.
4. For a single item with a large subtree (from `content.ini [ContentJobSettings] SynchronousLimit` items, 50 by default), a confirmation offers to do it **now** or **in the background** as a content job, with a progress page (Content jobs). Smaller subtrees, and "the chosen item only", are assigned at once.
5. Done at once, the section list opens again and every content view cache is cleared (`eZContentCacheManager::clearAllContentCache()`). Done as a job, its progress page opens; the job clears the view caches of the objects it changes, batch by batch.

Which objects change, with the first choice (with the second, the branches in another section are left out):

- Done at once (`eZContentObjectTreeNode::assignSectionToSubTree()`), the chosen item and every object below it **whose main location is in that subtree**. An object that only has a secondary location there keeps its section.
- Done as a content job, every object with a location in the subtree, and each one is checked against your permissions; objects you may not assign are skipped with a warning in the job's log.

Permissions are checked for the chosen items. If some may not be assigned, the others are, and a page lists the ones that were not. If your role allows the section but no class of object, or does not allow the section at all, the page says that instead (section 9.3).

8.4 One object: the Details tab and the edit form

On an object's page in the administration interface, the **Details** tab has a **Section** selector with the sections your role lets you assign to this object, for a node with children a second selector with the three choices of section 8.5 (**Whole subtree** preselected), and **Set**. The same choices are in the edit form of a draft. Both call `eZSection::applyTo()`, which changes the object, and with **Whole subtree every object below each of its locations** whose main location is there, updates the search index and clears the view caches, the static cache pages included when static caching is on. With **This object only** the object alone changes, whatever its locations.

8.5 Which objects below get the section

Every place that assigns a section offers the same three choices (`eZSection::SCOPE_*`):

Choice	Value	What changes
Whole subtree / The chosen item and every object below it	`subtree`	The object and every object below it, as before there was a choice. The default wherever nothing is chosen.
Subtree, except branches in another section / ... the objects below it that have its section	`inherited`	The object and the objects below it that are in the section the object had. An object below in another section keeps its section, and so does everything below it, even objects there in the old section.
This object only / The chosen item only	`object`	The object alone, at all its locations.

An example: a folder in Standard holds `a` (Standard), `b` (Members area) with `b1` (Standard) below it, and `c` (Standard). Assigning Intranet with `inherited` changes the folder, `a` and `c`; `b` stays in Members area and `b1` in Standard. With `subtree` all five are in Intranet, with `object` the folder only.

What it does not change: new content below still takes the section of its parent (section 8.1), and moving still follows section 8.2. A section stays a property of the object; `object` does not give one location a section of its own.

Whoever assigns, with every choice:

- **Only objects you may assign the section to change.** The chosen item needs it, and so does every object below it: an object your section/assign policy excludes (its Class, Section, Owner or Subtree limitation) keeps its section, whether it is done at once, as a content job or from the Details tab. An administrator without limitations changes them all. The kernel's own changes (a new main location takes its parent's section) are not checked.
- **A very large subtree goes to the background.** From the Details tab and the edit form, a subtree with more nodes than `content.ini [ContentJobSettings] NowLimit` (1000 by default) is assigned by the content job `section`, in batches: from the Details tab its progress page opens, from the edit form the draft stays open while it runs. **This object only** is always done at once.
- Each assignment writes the audit event `content.node.section` with the choice (`after.scope`) and the number of objects it changes; the search index and the view caches of the changed objects are updated.

9. Sections in roles and policies

Sections are first of all a permission tool. There are three ways a role can use one.

9.1 A policy with a Section limitation

A policy of the `content` module can be limited to sections. The policy then applies only to objects in those sections. These functions accept the Section limitation: `read`, `diff`, `view_embed`, `create`, `edit`, `publish`, `manage_locations`, `hide`, `translate`, `remove`, `versionread`, `versionremove` and `pdf`.

Example, the default Anonymous role:

Module	Function	Limitation
content	read	Section(Standard)
content	pdf	Section(Standard)

To let visitors also read a new section "Public documents": **Roles and policies** > Anonymous > **Edit** > the `content/read` policy > **Edit** > tick "Public documents" in the Section list next to Standard > **OK** > **Save**.

A Section limitation combines with the other limitations of the same policy by AND (Class, Owner, Node, Subtree, State...), and policies of the same function combine by OR. "Editors may edit articles in Members area" is one policy: `content/edit` with Class(Article) and Section(Members area).

The section's page (section 4) lists every role with such a policy, and the list counts the section as "Used by roles".

9.2 A role assigned with the limitation to a section

A whole role can be given to a user or group **for one section only**. On the role's page, below the list of assignments, choose **Section** in the limitation select, **Assign with limitation**, pick the section, then the users or groups. Every policy of the role then applies only to objects in that section (internally the policies get a `User_Section` limitation).

Use it when the same role is needed several times for different parts of the site: one "Editor" role, assigned to the Marketing group limited to Section Marketing and to the Sales group limited to Section Sales.

The section's page lists these assignments under "Users and user groups with role limitations associated with this section". They keep the section from being removed.

9.3 Who may assign sections: section/assign

`section/assign` decides who may put content into sections, with four limitations:

Limitation	Means
NewSection	the sections that may be assigned (the target)
Section	the sections the objects may be in now (the source)
Class	the classes of the objects that may be assigned
Owner: Self	only objects the user owns

Example: "Editors may move articles and folders from Standard into Members area, nothing else": `section/assign` with Class(Article, Folder), Section(Standard), NewSection(Members area).

Without a NewSection limitation every section may be assigned. **Assign content** on the list is greyed out for the sections a user may not assign.

9.4 The section module's own functions

Function	Opens
`section/view`	the list and each section's page

Function	Opens
`section/edit`	the edit form, New section, Remove selected
`section/assign`	Assign content, and the list and section pages

10. Navigation parts

The administration interface has a top menu: Content structure, Media library, Users, Setup and more. A navigation part is what links a page to one of those tabs. While an object is viewed or edited, the navigation part of **its section** decides which tab is active and which left menu is shown. That is why the Users section uses User accounts and the Media section Media library.

The parts are listed in `settings/menu.ini` `[NavigationPart]`:

Identifier	Name
`ezcontentnavigationpart`	Content structure
`ezmedianavigationpart`	Media library
`ezusernavigationpart`	User accounts
`ezshopnavigationpart`	Webshop
`ezvisualnavigationpart`	Design
`ezsetupnavigationpart`	Setup
`ezmynavigationpart`	My account
`expauditnavigationpart`	Audit

Extensions add their own in their `settings/menu.ini.append.php` (`Part[eznewsletternavigationpart]=...` and so on), and the section form lists them. A section whose part is not listed any more (the extension was deactivated) shows "Unknown navigation part" and its content no active tab.

The navigation part changes nothing for visitors: the public siteaccesses have no such menu. The admin pagelayouts also put the identifier and `section_id_<ID>` into the class of the page, for styling.

11. Templates and designs by section

11.1 Override a template for one section

`override.ini` (of a siteaccess, or in an extension) can choose a template by section, by ID or by identifier:

INI

```ini
[members_article_full]
Source=node/view/full.tpl
MatchFile=full/members_article.tpl
Subdir=templates
Match[class_identifier]=article
Match[section_identifier]=members_area
```

`Match[section]=<ID>` does the same by ID. The node views, content/edit, search, history and

the template functions `node_view_gui`, `content_view_gui` and the others set the keys `section` (and `section_identifier` where the section has one). Clear the template override cache after changing `override.ini`:

Shell

```
php bin/php/ezcache.php --clear-id=template-override,template,content --allow-root-user
```

A section without an identifier cannot be matched with `section_identifier`; give it one first.

11.2 Templates that ask about sections

```
{* one section, by ID or by identifier: exactly one of the two *}
{def $members = fetch( 'section', 'object', hash( 'identifier', 'members_area' ) )}

{* every section *}
{def $sections = fetch( 'section', 'list' )}

{* published objects of a section, 10 from the start; archived ones with status 'archived' *}
{def $objects = fetch( 'section', 'object_list', hash( 'section_id', $members.id, 'limit', 10, 'o
ffset', 0 ) )
     $count = fetch( 'section', 'object_list_count', hash( 'section_id', $members.id ) )}

{* the roles whose policies are limited to it, and the role assignments limited to it *}
{def $roles = fetch( 'section', 'roles', hash( 'section_id', $members.id ) )
     $assigned = fetch( 'section', 'user_roles', hash( 'section_id', $members.id ) )}

{* the children of a node that are in one section *}
{def $children = fetch( 'content', 'list', hash( 'parent_node_id', $node.node_id,
                                                 'attribute_filter', array( array( 'section', '='
, $members.id ) ) ) )}
```

`object_list` sorts by object ID, newest first, unless `sort_order` says otherwise. In a node template the section is `$node.object.section_id`. The content list and tree fetches can also sort by section (`sort_by`, `array( 'section', true() )`).

12. From a shell and from other programs

There is no dedicated command for creating or removing sections; use the pages, or the remote services below.

Assign a subtree from a shell, as a content job:

Shell

```
php bin/php/expcontentjob.php section <node ID> <section ID>               # now, with progress
php bin/php/expcontentjob.php section <node ID> <section ID> --background  # queued for the worke
r
php bin/php/expcontentjob.php section <node ID> <section ID> --scope=inherited  # subtree (defaul
t), inherited or object
./console exp:expcontentjob section 1234 7 --allow-root-user               # the same through the
 console
php bin/php/expcontentjob.php list                                         # what is running, wha
t finished
```

The job runs as the user given with `--login`, else as `site.ini [UserSettings] UserCreatorID`, and that user's permissions apply to every object. The cronjob part `contentjobs` starts queued jobs and resumes interrupted ones (Content jobs).

Remote services of the expservices extension, under `/ezjscore/call/expsection::<method>`: `list`, `count`, `get`, `getByIdentifier`, `objects`, `objectCount`, `usage`, `canRemove`, `assignable`, `navigationParts`, `ofObject`, `ofNode`, and the writes `create`, `update`, `remove`, `assign` and `assignSubtree` (both take the POST field `scope`, section 8.5), each with the policy it needs (the service list).

PHP, in an extension or a script started with `php bin/php/ezexec.php <script> --allow-root-user`:

PHP

```php
$section = eZSection::fetchByIdentifier( 'members_area' );   // or eZSection::fetch( 7 )
$object  = eZContentObject::fetch( 1234 );
$section->applyTo( $object );                                // checks section/assign for the cur
rent user
$section->applyTo( $object, eZSection::SCOPE_INHERITED );    // or SCOPE_OBJECT; SCOPE_SUBTREE is
 the default
eZContentObjectTreeNode::assignSectionToSubTree( 1234, 7, false, true, eZSection::SCOPE_INHERITED
 ); // what section/assign does
$free    = $section->canBeRemoved();                         // the rule of section 7.1
```

13. Workflows, step by step

13.1 A members area

Goal: pages that only logged-in members may read.

1. **New section**: Name `Members area`, identifier `members_area`, navigation part Content structure. **Create section**.
2. Create a folder `Members` where it belongs in the tree, with the pages below it.
3. On the Members area card, **Assign content**, choose the `Members` folder, **Select**. The list now shows "N published" on the card.
4. Check as a visitor (logged out): the folder and its pages are gone from menus and give "access denied", because Anonymous may read Standard only.
5. **Roles and policies** > the role your members have (or a new role "Member" assigned to the Members user group) > **Edit** > **New policy** > `content` > `read` > Section(Members area) > **OK** > **Save**.
6. Log in as a member: the pages are there. The section's page now lists the role.

13.2 An intranet with departments

Goal: each department edits its own pages, with one shared Editor role.

1. One section per department: `Sales` (`sales`), `Marketing` (`marketing`). Assign each department's folder to it.
2. Give everyone read access: add `content/read` Section(Sales, Marketing) to the role all staff have.
3. On the Editor role's page, choose **Section** in the limitation select, **Assign with limitation**, choose Sales, then the Sales group. Repeat for Marketing.
4. Each section's page now shows its group under "Users and user groups with role limitations associated with this section"; a sales editor can edit Sales pages and not Marketing pages.

13.3 Staging content before it goes public

Goal: prepare pages in the real tree without visitors seeing them, then release them in one step.

1. **New section** `Staging` (`staging`). Do not add it to the Anonymous role.
2. Give editors `content/read` and `content/edit` for Section(Staging).
3. Create the new pages in their final place, then **Assign content** their top item to Staging. Visitors do not see them; editors do.
4. To release: on the **Standard** card, **Assign content**, choose the same top item. The pages become readable at once, because every content view cache is cleared.

To see staging pages in their own look while they wait, give the section its own template with `Match[section_identifier]=staging` (section 11.1).

14. Troubleshooting

Symptom	Cause	What to do
"The sections cannot be removed"	Objects, policies or role assignments still use it; the page names which	Assign its content to another section; edit the roles listed; remove the assignments. Drafts count: see the next line.
A section with "No published objects" still cannot be removed	It holds drafts of objects that were never published, or archived objects; the section's page counts them	Let their owners publish or remove the drafts. Drafts older than `content.ini [VersionManagement] DraftsDuration` (90 days) are removed by the cronjob script `old_drafts_cleanup.php` when it is scheduled.
Content disappeared for visitors after Assign content	The new section is not in the Anonymous role's `content/read`	Add the section to that policy (section 9.1), or assign the content back.
Visitors still see content of a section they may not read	Cached pages: static cache, an HTTP cache, a proxy	Assign content clears the content view cache; clear the static cache and any HTTP cache too.
Some objects below the chosen item kept their old section	Their main location is elsewhere; they only have a secondary location in the subtree	Assign content at their main location, or run the assignment as a content job (section 8.3).
Assign content is greyed out	Your role has no `section/assign` policy whose NewSection includes this section	An administrator adds the section to NewSection, or a policy without it (section 9.3).
"There are no objects in the system that you could assign the section ... to"	Your `section/assign` policy allows the section but no class	Widen the Class limitation of that policy.
"The identifier has been used in another section."	Identifiers are unique	Choose another, or rename the other section's identifier first.
The edit form will not save an old section	It has no identifier; the form requires one	Give it one; check that no template expected it to have none.
"Unknown navigation part"	The extension that defined the part is inactive	Choose a listed part, or activate the extension.

Symptom	Cause	What to do
A template override by section does not apply	The override cache is stale, the identifier changed, or the section has no identifier	Clear `template-override`, `template` and `content`; check `Match[...]` against the section's page.
A role assignment limited to a section did not keep it from being removed (SQLite, PostgreSQL, MongoDB, before October 2026)	The kernel compared the assignment's limitation case-sensitively	Fixed: `eZRole::fetchRolesByLimitation()` now compares without case.

15. Who may use the section pages

Page	Address	Needs
The list	`/section/list`	`section/view`, `section/edit` or `section/assign`
One section	`/section/view/<ID>`	`section/view` or `section/assign`
New and Edit	`/section/edit/<ID>`, `/section/edit/0`	`section/edit`
Remove selected	the list	`section/edit` without limitation
Assign content	`/section/assign/<ID>`	`section/assign` with the section in NewSection (or none)

The list and the section's page show Edit, New section and the tick boxes only to users who may edit sections, and Assign content only where it is allowed.

References

- Module and views: `kernel/section/module.php`, `kernel/private/classes/views/section/list.php` (the overview: `ListView::overview()`, `overviewFromDatabase()`), `view.php`, `edit.php`, `assign.php`.
- Templates: `design/admin4/templates/section/` and the same files in `design/admin/templates/section/` (`list.tpl`, `view.tpl`, `edit.tpl`, `confirmremove.tpl`, `assign_notification.tpl`, `browse_assign.tpl`, `exp_style.tpl`).
- The section class: `kernel/classes/ezsection.php` (`canBeRemoved()`, `applyTo()`, `fetchByIdentifier()`).
- Template fetch functions: `kernel/section/function_definition.php`, `kernel/section/ezsectionfunctioncollection.php`.
- Assigning a subtree: `eZContentObjectTreeNode::assignSectionToSubTree()` in `kernel/classes/ezcontentobjecttreenode.php`; the content job in `kernel/classes/contentjob/expcontentjobsectionsubtree.php`.
- Section on create and publish: `eZContentObject::createWithNodeAssignment()`, `eZContentOperationCollection::updateSectionID()`; on move: `kernel/classes/ezcontentobjecttreenodeoperations.php`.
- Permissions: the Section limitation of the content functions in `kernel/content/module.php`;

`eZUser::canAssignSection()`, `canAssignSectionToObject()`, `canAssignSectionList()` in `kernel/classes/datatypes/ezuser/ezuser.php`; role assignments with a section in `eZRole::assignToUser()` and `eZRole::fetchRolesByLimitation()` (`kernel/classes/ezrole.php`).

- Navigation parts: `settings/menu.ini [NavigationPart]`, `kernel/classes/eznavigationpart.php`, `kernel/classes/eznodeviewfunctions.php`.
- Override keys: `kernel/common/eztemplatedesignresource.php`, `kernel/classes/eznodeviewfunctions.php`, `settings/override.ini`.
- Caches: `eZContentCacheManager::clearContentCacheIfNeededBySectionID()` and `clearAllContentCache()` in `kernel/classes/ezcontentcachemanager.php`; the event `content/section/cache`.
- Default data: `share/db_data.dba`, `kernel/sql/common/cleandata.sql`.
- Paging: `settings/admininterface.ini [PaginationSettings]`, Admin list paging.
- Tests: `tests/tests/kernel/classes/expSectionOverviewTest.php` (the overview without a database, and against the kernel's own lookups on a live installation).

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.