Editing several items at once

Source file: doc/eng-us/bc/6.0/multi-node-edit.md Last changed 2026-11-04
https://doc.exponential.earth/exphelp/viewer/bc/6.0/multi-node-edit.md

Read this page if your editors change many items of the same kind, or if you maintain datatypes, edit handlers or admin overrides of the sub items list or the search results. The new view `content/multiedit` edits the full attribute set of up to 50 objects in one form, with the ordinary editor's own machinery. It is reached from the sub items list (**More actions → Edit selected**, and **Create multiple new**) and from admin search results.

In short

What changed	New view `content/multiedit` (policy `content/edit`, each object checked with `can_edit`). Search results in the admin get a checkbox column and **Edit selected**; the sub items list gets **Edit selected** and **Create multiple new**. Drafts autosave on the `ezautosave` settings.
Who is affected	Overrides of `content/searchresult.tpl`, `children_detailed.tpl`, `ezajaxsubitems_expdatatable.js` and `admin_datatable.css` (admin and admin4) do not show the new controls. Extensions with edit handlers: their handlers now also run here, per object. Datatypes need no change.
How to check	`grep -n -A8 "multiedit" kernel/content/module.php`
How to fix	Merge your overrides with the shipped files listed under Files. Nothing else.

Why it works at all

Every datatype edit template names its inputs after the **content object attribute id**:

Template

```
name="{$attribute_base}_ezstring_data_text_{$attribute.id}"
```

That id is unique per object, per version, per language. So one form can carry the attributes of any number of objects with no name collisions, and `validateInput()` / `fetchInput()` / `storeInput()` are called per object exactly as `content/edit` calls them for one.

Nothing in the datatype layer was changed, and nothing needs to be. A datatype that works in the normal editor works here, including datatypes that do not exist yet - the third party `sckenhancedselection`, `eztags`, `xrowmetadata` and `ezobjectrelationlist` all render and post correctly without knowing this view exists.

Why a separate view

`content/edit` is a single object state machine: language and translation choice, version conflicts, placement, browse-for-nodes, "somebody else is editing this". Threading a list through every one of its branches would put the thing everybody uses all day at risk for the sake of a thing they use occasionally. `multiedit` is a new assembly around the same parts, in the same module - so it inherits the module's permission context, its `redirectTo`, and its draft plumbing.

Using it

From a sub items list

Tick some items, then **More actions → Edit selected**.

The checkboxes there are named `DeleteIDArray[]` - a name that list reuses for every bulk action - and carry **node ids**. The view resolves them.

From a search result

`/content/search?SearchText=...` now has a checkbox column, a select-all box in the header, and an **Edit selected** button. This is how you edit items that do not share a parent: search the whole site, tick what you want, edit it in one form.

The search results template deliberately contains **no** `<form>`. It is included inside the search form, and a form inside a form is dropped by every browser - the inner one does not exist and its button posts to the outer action. The button builds a form in javascript instead, carrying the node ids and the request token.

Directly

POST to `content/multiedit` with any of:

field	carries
`ContentObjectIDArray[]`	object ids
`MultiEditObjectIDArray[]`	object ids
`ContentNodeIDArray[]`	node ids
`MultiEditNodeIDArray[]`	node ids
`DeleteIDArray[]`	node ids (what the sub items list sends)
`MultiEditReturnURI`	where Discard and a clean publish return to

All of them are accepted together and reduced to a unique list of object ids. A request token (`ezxform_token`) is required, as on any admin POST.

What the form does

- **Groups by content class.** One section per class, in the order the classes were first met. A mixed selection is not refused; it is drawn as sections.
- **One panel per object**, collapsible, open by default when three or fewer are in the group or when the object failed validation.
- **Opens an internal draft per object** on first load and carries the version numbers back in hidden `MultiEditDraft[objectID]` fields, so pressing a button twice does not stack a new version each time.
- **Says what it left out.** An object you may not edit, one that has gone, or one whose draft could not be opened is listed with the reason rather than silently dropped.

The three buttons

button	does
Publish all	validates everything, then publishes each object separately
Save drafts	stores what was typed without publishing; required fields are not insisted on
Discard	removes the drafts this form opened and returns to `MultiEditReturnURI`

Collapsing and expanding

The toolbar above the sections carries **Expand all** and **Collapse all**.

A closed panel still posts its fields - `<details>` hides its children, it does not remove them - so collapsing everything and pressing Publish saves and publishes exactly as much as leaving them open. Nothing is left out.

Panels open by default when their class group has three or fewer items, and an item that failed validation is **forced open when the form comes back**, whatever was collapsed before: being told there is a problem and then having to hunt for it is the worst of both.

Creating several at once

Create multiple new sits next to **Create new** above a sub items list. It asks what and how many, makes that many drafts under this parent, and opens them in the same form.

A new object and an existing one are the same thing once a draft is open, so nothing about the form changes: the same sections, the same collapse controls, the same per-object publish report.

- The type list is `canCreateClassList()` for this parent - the same list the **Create here** menu is built from - so it already answers "may this reader put this class in this place". The button is disabled when nothing may be created here.
- At most `eZMultiEdit::MAX_OBJECTS` (50) at a time.
- Nothing is published until you say so. Abandoning the page leaves internal drafts, the same as editing does, and the draft list of the ordinary editor lets you discard them.

The trap in this, for anyone changing it

A newly instantiated object gets its **node assignment on version 1**. If the form then opens a *second* version for editing - which `openDrafts()` will do if it is not told the object already has a draft - then version 2 is what publishes, version 2 has no node assignment, and the object is published **with no location at all**: status 1, no row in `ezcontentobject_tree`, invisible in every list and impossible to find.

So `createDrafts()` returns `objectID => version`, and the view seeds `MultiEditDraft` with it before the selection is opened. After creating, check that the new items really appear under their parent: it is the only check that catches this.

Autosave

Drafts are saved automatically, on the settings the ordinary editor uses - `autosave.ini [AutosaveSettings] Interval` (default 180 seconds) and `TrackUserInput` (default `enabled`),

shipped by the `ezautosave` extension in `extension/ezautosave/settings/autosave.ini.append.php`. Turn autosave off there, or deactivate the `ezautosave` extension, and it is off here too. A small line beside the collapse controls says what is happening.

ezautosave itself is not reused, and could not be. Its `Y.eZ.AutoSubmit` binds one form to an endpoint naming *one* object and *one* version - `ezautosave::savedraft::<objectID>::<version>::<language>` - which cannot express a form holding many. So the form is posted back to its own **Save drafts** action instead: one request, every draft stored, by exactly the code the button uses.

Details worth knowing:

- **Nothing is published.** Autosave is a store, and storing skips the required-field check, so it never blocks and never publishes.
- **File fields are left out of the payload.** A file cannot meaningfully be re-sent every few minutes, and a datatype that receives no upload keeps what it already has.
- **A save in flight does not swallow later typing.** The dirty flag is only cleared when the response arrives, so anything typed meanwhile is saved by the next pass.
- Pressing a real button clears the flag so autosave does not race the submit.
- It needs `fetch` and `FormData`; without them it simply does not start and the buttons behave as before.

Extension edit handlers

`eZContentObjectEditHandler::validateInputHandlers()` and `executeInputHandlers()` are called per object, as `content/attribute_edit` calls them for one, after `initialize()`.

This matters more than it looks: an extension that hooks editing - to validate, to fill something in, to refuse a save - is otherwise **bypassed in silence**. A warning an extension returns has no attribute to attach to, so it is shown as its own line in that object's "needs attention" list. A warning given as a plain string is read as readily as the usual name/text pair.

Custom actions

A datatype that needs a round trip of its own - **Find object** on a relation, an upload on a file field - posts `CustomActionButton[<attributeID>_<action>]`. Those are parsed and handed to every object's `fetchInput()`.

The attribute id in that name is what makes it safe here: ids are unique, so each object takes only the entries that belong to it, and one parse serves the whole form.

Important details

Publishing several objects is not one act

Each object is its own `eZOperationHandler::execute( 'content', 'publish', ... )`. Any one of them can fail its own validation or be **suspended by a workflow** - an approve event, for instance. The result is therefore a list, not a yes:

- **published** - through

- **waiting for approval** - a workflow took it; not published, not failed
- **could not be published** - the operation refused it

If everything published and nothing is pending, the form redirects to `MultiEditReturnURI`. Anything else keeps you on the page with the report.

Validation is all-or-nothing before publishing

If any object fails validation, **nothing is published** and everything typed is kept as a draft. The failing objects are marked, and each names its own failing fields with the datatype's own message, e.g.

> Needs attention before this item can be published: Number of columns — No POST variable. Please check your configuration.

That message usually means a **required attribute with no value whose widget posts nothing when empty** - a multi-select with nothing chosen, for example. Such an object cannot be published in the ordinary editor either; multiedit simply surfaces it for several objects at once.

Drafts are real, and visible

Opening the form creates an internal draft per object. Until you publish or discard, the ordinary editor will show "you have a draft" for those objects. Abandoning the page leaves them behind.

The ordinary editor's "you have a draft" notice lets you discard each one.

Limits and permissions

- **50 objects** per form (`eZMultiEdit::MAX_OBJECTS`). Beyond that the selection is truncated: every object is a draft, a form section and a publish.
- The view requires the `content/edit` policy, and **each object is checked individually** with `can_edit`. Objects you may not edit are reported, not hidden.
- `MultiEditReturnURI` is only honoured when it is a site relative path, so a posted field cannot redirect elsewhere.

Language

One language per form. It comes from `/content/multiedit/(language)/eng-US`, otherwise each object's own initial language. An object that does not carry the requested language falls back to what it has rather than drawing nothing.

Security notes

Every object id, version number, class id, parent id and return address this view works from arrives in the request. None of them is taken on trust, and each check below closes something that was actually exploitable during review, not a hypothetical.

A draft is only yours to touch

`eZMultiEdit::mayUseDraft()` gates both discarding and adopting a draft. Three conditions, all required:

1. the version is a draft (never a published one),
2. the reader may edit the object (`can_edit` - the module's `content/edit` policy only says they may edit *something*),
3. the version's `creator_id` is the current user.

Without the third, naming any object and version in `MultiEditDraft[...]` with **Discard** deleted that draft - any editor's work in progress, anywhere on the site, on objects the attacker had no rights to and had never opened. It is now skipped silently; an adopted draft is refused with a reason instead.

Somebody else's draft is deliberately **not** offered here. The single object editor has a version chooser for that conversation; this form refuses and says so.

The return address cannot leave the site

`MultiEditReturnURI` is stripped of control characters, refused if it contains a backslash, and must begin with exactly one slash not followed by another separator. A leading slash and no second one is not enough on its own: browsers read `/\example.com` as protocol relative, and a newline in the value can carry past a naive check into a header.

Types are forced before use

Request fields are not assumed to have the shape the form gives them. Sending `MultiEditObjectIDArray` as a plain string used to reach `$_POST['MultiEditObjectIDArray'][] = ...` and fatal with "[] operator not supported for strings" - a crash from one crafted field. Both that and `MultiEditDraft` are forced to arrays first.

The language is only believed if it exists

The language comes off the url. It is matched against `eZContentLanguage::fetchList()` and otherwise treated as not asked for, rather than being carried into `createNewVersionIn()` or printed as typed. (It was not reflected into the page even before this - the templates escape what they print - but an unchecked locale had no business reaching the content API.)

Creating is checked against the parent, not the view

The view's declared policy is `content/edit`. Creating needs more, so `createDrafts()` asks `$parent->checkAccess( 'create', $classID )` - the same question the **Create here** menu asks - and the class list offered is `canCreateClassList()` for that parent, so a class that may not go there is never even shown.

Bounds

A selection is cut to `MAX_OBJECTS` (50) and a create count is clamped to it. Each object in a form is a draft, a form section and a publish, so this bounds both the page and the write amplification of a crafted request.

Checked against a real restricted account

The ownership checks above can be demonstrated with a draft whose creator differs from the session user, but that proves only half of it. So the checks are also run as an account that may edit **one subtree and nothing else**:

attempt	result
edit an item inside its subtree	allowed - the gate discriminates rather than denying everything
edit an item outside it	refused, and the item is named with the reason
adopt another user's draft on an object it may not touch	refused
discard that draft	refused; the version is still there afterwards

To repeat this, make a test account with a role limited to one subtree on a copy of the installation (the scripts used for the original check are private), and try the four attempts above.

Two things that account is worth keeping in mind for:

- a policy check that passes for an administrator says nothing about anyone else, and `content/edit` without a limitation is not the shape most editors have;
- `eZUser::setInformation()` only writes the password hash when the password and its confirmation match. Passing it once leaves an account that exists, is enabled, has its role, and cannot log in, with nothing to say why - so check that the test account can log in (`eZUser::loginUser()`) before trusting a refusal.

Covered by

Private regression scripts covered each case above (draft ownership, open redirect, type confusion); they are not shipped. To repeat the draft-ownership case by hand you need a draft owned by another user and its object and version numbers.

Files

file	part
`kernel/content/module.php`	the `multiedit` view declaration
`kernel/content/multiedit.php`	the view's entry point; the code is in `kernel/private/classes/views/content/multiedit.php`
`kernel/content/multiedit_functions.php`	`eZMultiEdit`, all the working parts
`design/admin/templates/content/multiedit.tpl`	the form
`design/admin/templates/content/searchresult.tpl`	checkbox column and the button
`design/admin/templates/children_detailed.tpl`	the `Edit selected` and `Create multiple new` labels, and the url
`design/admin/javascript/ezajaxsubitems_expdatatable.js` (and the `admin4` copy)	the More actions entry and the Create multiple new button
`design/admin/stylesheets/theme/admin_datatable.css` (and the `admin4` copy)	the styling for that menu entry
`design/admin4/templates/content/multiedit.tpl`	the admin4 copy of the form

The form includes `design:content/edit_attribute.tpl` - the same include the single object editor uses - and gives it the same variables `content/attribute_edit` gives it, including `content_attributes_grouped_data_map`, which is what the admin override of that template actually draws from.

Tests

Private browser scripts covered: the view renders, groups, carries drafts and emits unique attribute names; More actions and search both round trip and Discard returns; Create multiple new (the button, the chooser, the drafts, publishing and that new items get a location); autosave (the indicator, the settings, that a change stores every draft without publishing); security (draft ownership, open redirect, type confusion); and the same checks as a user restricted to one subtree. They are not shipped. Do the "By hand" checks in the sections above on a copy of the installation. Check the view is registered: `grep -n -A8 "multiedit" kernel/content/module.php`.

Known gaps

- **No bulk field operation.** Setting one field across many objects - a section, a template, a code - is a different UI and a different validation story. This view edits whole objects.
- **No per-object publish.** Publish is all or nothing per form; an object that fails keeps the rest from publishing.
- **Translation is not handled.** The form edits one language and does not offer the translate-from behaviour the single editor has.
- **Placement is not editable.** An existing object keeps the locations it has; a newly created one gets the parent it was created under, and no way to add a second location or choose a different one from this form.
- **Draft conflicts are refused, not resolved.** An object somebody else is editing is left out with a reason, where the single editor offers the version chooser.
- **No preview.** ezautosave's preview pane belongs to a single object and has no meaning for a form holding many.

Related pages

- Sub items table options and Sub items: copy selected
- Content jobs
- September 2026, first half: 14 September
- Changelog 6.0.15
- Content model and editing guide

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.