Write a workflow event type

The PHP side of a workflow event type - the class extending eZWorkflowEventType, registerEventType(), setTriggerTypes(), execute() and its statuses, the process parameters, settings per event, deferring to cron, and...

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

This page is for PHP developers who add a step that administrators can put into workflows: notify an outside system when content is published, check a field before publishing, wait for an approval. A workflow runs when a trigger fires (for example `content/publish/before` or `after`); each of its events is an instance of an event type, which is a class extending `eZWorkflowEventType`. The complete example with templates is on Writing a workflow event type.

Contents

1. Registration
2. The class
3. What execute() receives
4. The statuses
5. Settings per event
6. Deferring to cron
7. Rules
8. Event type or event listener?

Registration

```
extension/acmereport/eventtypes/event/acmenotify/acmenotifytype.php
extension/acmereport/settings/workflow.ini.append.php
```

INI

```ini
<?php /* #?ini charset="utf-8"?
[EventSettings]
ExtensionDirectories[]=acmereport
AvailableEventTypes[]=event_acmenotify
*/ ?>
```

`AvailableEventTypes[]` entries are the group (`event`) and the type string joined by `_`, like the kernel's `event_ezapprove` in `settings/workflow.ini`.

The class

PHP

```php
<?php
class acmeNotifyType extends eZWorkflowEventType
{
    const WORKFLOW_TYPE_STRING = 'acmenotify';

    public function __construct()
    {
        parent::__construct( self::WORKFLOW_TYPE_STRING, ezpI18n::tr( 'extension/acmereport', 'No
tify the archive' ) );
        $this->setTriggerTypes( array( 'content' => array( 'publish' => array( 'after' ) ) ) );
    }

    function execute( $process, $event )
    {
        $parameters = $process->attribute( 'parameter_list' );
        $object = eZContentObject::fetch( (int)$parameters['object_id'] );
        if ( !$object )
            return eZWorkflowType::STATUS_WORKFLOW_CANCELLED;

        acmeReportQueue::add( $object->attribute( 'id' ), (int)$parameters['version'] );
        return eZWorkflowType::STATUS_ACCEPTED;
    }
}

eZWorkflowEventType::registerEventType( acmeNotifyType::WORKFLOW_TYPE_STRING, 'acmeNotifyType' );
```

`setTriggerTypes()` limits the triggers the event can be used with, so administrators cannot attach it where it makes no sense. The kernel's `eZWaitUntilDateType` (`kernel/classes/workflowtypes/event/ezwaituntildate/`) is a complete example.

What execute() receives

Argument	Is
`$process`	the `eZWorkflowProcess`; `$process->attribute( 'parameter_list' )` holds the operation's parameters
`$event`	the `eZWorkflowEvent`, the configured instance: its settings are in its `data_int1`, `data_text1` ... attributes

For the content publish operation the parameters include `object_id` and `version`. Cast them: they are values, not objects.

The statuses

`eZWorkflowType` (`kernel/classes/ezworkflowtype.php`) defines the answers of `execute()`:

Status	Effect
`STATUS_ACCEPTED`	the step is done; the next event runs
`STATUS_REJECTED`	the operation is refused
`STATUS_DEFERRED_TO_CRON`, `STATUS_DEFERRED_TO_CRON_REPEAT`	wait; the workflow cronjob runs the step again
`STATUS_FETCH_TEMPLATE`, `STATUS_FETCH_TEMPLATE_REPEAT`	show the user a template (a choice, a form)

Status	Effect
`STATUS_REDIRECT`, `STATUS_REDIRECT_REPEAT`	send the user elsewhere, for example to a payment provider
`STATUS_WORKFLOW_CANCELLED`	stop the workflow
`STATUS_RUN_SUB_EVENT`, `STATUS_WORKFLOW_RESET`, `STATUS_WORKFLOW_DONE`	special flows of the multiplexer and resets

Settings per event

What an administrator sets for the event in the workflow editor is read by overriding `fetchHTTPInput()` and `validateHTTPInput()` and stored in the event's columns. Ship the templates `workflow/eventtype/edit/event_acmenotify.tpl` and `workflow/eventtype/view/event_acmenotify.tpl` in your design, like the kernel's `event_ezapprove.tpl`.

Deferring to cron

An event that waits (for an approval, a date, an outside answer) returns `STATUS_DEFERRED_TO_CRON_REPEAT`; the `workflow` cronjob script runs `execute()` again later, and **Setup > Workflow processes** shows the waiting processes. The editor sees the publish as pending until the event accepts.

Rules

- **Fast.** `execute()` runs inside the editor's publish request. Queue slow work (outside calls, mail, image processing) instead of doing it there (Background work).
- **No exceptions out of `execute()`.** Catch what can fail, log it, and return a status.
- **Idempotent.** A deferred event runs more than once; make the second run harmless.
- **Logic outside the class.** Put the decision in a service you can test without a workflow.

Event type or event listener?

Need	Use
administrators decide where it runs, in which workflow, in which order	a workflow event type
a step can refuse or pause an operation	a workflow event type
your code reacts to something the kernel does, always	an event listener (Events, listeners and filters)

Related pages

- Workflows, triggers and events
- eZWorkflowType API
- Workflow event 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.