Notifications: architecture, extending and testing

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

This guide is for developers. It explains how the notification system is built, how to add your own event type and handler (with a complete example that was run), how the mails are made and overridden, which settings and hooks you can use, and how to test without sending mail. About 45 minutes; you need a running installation, a shell and PHP knowledge. The exact data model and settings are in the specification and the INI reference. Every command and output below was run on a test installation with the mail transport forced to files.

1. Architecture

```
 publish operation --> eZNotificationEvent ('ezpublish')  \
 collaboration item -> eZNotificationEvent ('ezcollaboration') >-- eznotificationevent (status 0)
 a run ------------> eZNotificationEvent ('ezcurrenttime') /                |
                                                                             v
                        eZNotificationEventFilter::process( $ids )  --  for each pending event, on
e transaction:
                                                                             |
                     +-------------------------+-----------------------+-----+-----------------+
                     | ezsubtree               | ezcollaboration-      | ezgeneraldigest       |
 your handler
                     | (publish -> followers)  | notification          | (time event -> digests)|
                     +-----------+-------------+-----------+-----------+-----------+------------+
                                 v                         v                       v
                   eZNotificationCollection (subject, text)  +  eZNotificationCollectionItem (addr
ess, send_date)
                                 |  send_date 0: send now            | send_date > 0: kept for th
e digest
                                 v                                   v
                   eZMailNotificationTransport ('ezmail')  -->  eZMailTransport (site.ini Transpor
t: sendmail, smtp or file)
```

Part	Where	Role
Events	`kernel/classes/notification/eznotificationevent.php`, `eznotificationeventtype.php`, `event/<type>/<type>type.php`	A row plus a type class that fills it (`initializeEvent`) and gives its content (`eventContent`)
The filter	`eznotificationeventfilter.php`	Loads the handlers, hands each pending event to each handler, removes or keeps the event, cleans up
Handlers	`handler/<id>/<id>handler.php`	`handle( $event )`, `fetchHttpInput()`, `storeSettings()`, `cleanup()` and the attributes the settings templates read
Rules	`ezsubtreenotificationrule.php`, `ezcollaborationnotificationrule.php`, `ezgeneraldigestusersettings.php`	Who wants what
Collections and items	`eznotificationcollection.php`, `eznotificationcollectionitem.php`	The message and its recipients; `send_date` decides now or digest

Part	Where	Role
Transports	`eznotificationtransport.php`, `ezmailnotificationtransport.php`	`eZNotificationTransport::instance( 'ezmail' )->send( $addresses, $subject, $body, null, $parameters )`
The service	`kernel/classes/expnotificationservice.php`, `expnotificationjob.php`	Run, plan (dry run), status, lock, run record, audit, background job
Entry points	`cronjobs/notification.php`, `bin/php/notification*.php`, `kernel/notification/*.php`	Thin stubs; the code is in `kernel/private/classes/{cronjobs,commands,views}/`

The run lock, the run record and the audit event are described in the specification.

2. An event type and a handler: a complete example

The example adds the event type `examplealert`: code makes an event with a message and an address, and a handler mails the message to that address. Two files, one setting.

The event type

File `examplealert/examplealerttype.php`, in a directory listed under `[NotificationEventTypeSettings]`:

PHP

```php
<?php
/**
 * Event type "examplealert": an event made by code, with a message and the address it is for.
 */
class ExampleAlertType extends eZNotificationEventType
{
    const NOTIFICATION_TYPE_STRING = 'examplealert';

    public function __construct()
    {
        parent::__construct( self::NOTIFICATION_TYPE_STRING );
    }

    function initializeEvent( $event, $params )
    {
        $event->setAttribute( 'data_text1', (string)$params['message'] );
        $event->setAttribute( 'data_text2', (string)$params['address'] );
    }

    function eventContent( $event )
    {
        return $event->attribute( 'data_text1' );
    }
}

eZNotificationEventType::register( ExampleAlertType::NOTIFICATION_TYPE_STRING, 'ExampleAlertType'
 );
```

`initializeEvent()` receives the parameters given to `eZNotificationEvent::create()` and stores them in the event's `data_int1..4` and `data_text1..4` columns. `eventContent()` is what `$event->attribute( 'content' )` returns. The last line registers the type name with its class.

The handler

File `examplealert/examplealerthandler.php`, in a directory listed under `[NotificationEventHandlerSettings]`. The file name is `<id>handler.php` and the class is `<id>handler`:

PHP

```php
<?php
/**
 * Handler "examplealert": mails the message of an examplealert event to the address of the event
.
 */
class examplealerthandler extends eZNotificationEventHandler
{
    const NOTIFICATION_HANDLER_ID = 'examplealert';

    public function __construct()
    {
        parent::__construct( self::NOTIFICATION_HANDLER_ID, 'Example alert handler' );
    }

    function handle( $event )
    {
        if ( $event->attribute( 'event_type_string' ) != 'examplealert' )
            return true; // not ours: the other handlers get it

        $collection = eZNotificationCollection::create( $event->attribute( 'id' ), self::NOTIFICA
TION_HANDLER_ID, 'ezmail' );
        $collection->setAttribute( 'data_subject', 'Example alert' );
        $collection->setAttribute( 'data_text', $event->attribute( 'data_text1' ) );
        $collection->store();
        $collection->addItem( $event->attribute( 'data_text2' ) ); // send_date 0: send now

        $addresses = array();
        foreach ( $collection->attribute( 'items_to_send' ) as $item )
        {
            $addresses[] = $item->attribute( 'address' );
            $item->remove();
        }
        eZNotificationTransport::instance( 'ezmail' )->send( $addresses, $collection->attribute(
'data_subject' ),
                                                             $collection->attribute( 'data_text'
 ) );
        $collection->remove();
        return true;
    }
}
```

`handle()` is called for every pending event; a handler ignores what is not its own and returns true. To send at once, add an item with no send date and send the items that have none (the example does). To keep a message for a digest, give the item a send date (`eZNotificationSchedule::setDateForItem( $item, array( 'frequency' => 'day', 'hour' => 8 ) )` then `$item->store()`) and do not send: the general digest handler sends it when a time event passes that date. Send first and remove the items only when `send()` returned true; on false call `eZNotificationEventFilter::noteDeliveryFailure()` and leave the items, so the next run retries them (see the upgrade notes). Do not remove an item that waits. The filter removes the event when no item is left for it, and keeps it (handled) otherwise.

The settings

In an extension, create `extension/<yours>/settings/notification.ini.append.php` and put the files in `extension/<yours>/notification/handler/examplealert/examplealerthandler.php` and `extension/<yours>/notificationtypes/examplealert/examplealerttype.php`:

INI

```
<?php /* #?ini charset="utf-8"?

[NotificationEventTypeSettings]
ExtensionDirectories[]=<yours>

[NotificationEventHandlerSettings]
ExtensionDirectories[]=<yours>
AvailableNotificationEventTypes[]=examplealert

*/ ?>
```

Activate the extension (`[ExtensionSettings] ActiveExtensions[]`), regenerate nothing (handlers are found by path, not by the autoload array) and clear the INI cache: `php bin/php/ezcache.php --clear-tag=ini --allow-root-user`.

The run below did not use an extension: it put the two example files in `var/tmp/notification-example/` and added that directory with `RepositoryDirectories[]` in the process's own settings (never written to a file). That is the same lookup with a different directory; `ExtensionDirectories[]` was not exercised here.

Running it

Make an event from your code and run the filter for it:

PHP

```
$event = eZNotificationEvent::create( 'examplealert', array( 'message' => 'The example alert work
s.', 'address' => 'nottest-ui@nottest.invalid' ) );
$event->store();
eZNotificationEventFilter::process( array( (int)$event->attribute( 'id' ) ) );
```

The complete script that was run (it also forces the file transport and watches the mail):

PHP

```
<?php
/** Proves the custom event type and handler of the developer guide (files in var/tmp/notificatio
n-example), file transport only. */
$ini = eZINI::instance();
$ini->setVariable( 'MailSettings', 'Transport', 'file' );
$ini->setVariable( 'MailSettings', 'FileTransportDirectory', 'var/tmp/notification-mail' );
$n = eZINI::instance( 'notification.ini' );
$types = $n->variable( 'NotificationEventTypeSettings', 'RepositoryDirectories' );
$types[] = 'var/tmp/notification-example/eventtypes';
$n->setVariable( 'NotificationEventTypeSettings', 'RepositoryDirectories', $types );
$h = $n->variable( 'NotificationEventHandlerSettings', 'RepositoryDirectories' );
$h[] = 'var/tmp/notification-example/handlers';
$n->setVariable( 'NotificationEventHandlerSettings', 'RepositoryDirectories', $h );
$a = $n->variable( 'NotificationEventHandlerSettings', 'AvailableNotificationEventTypes' );
$a[] = 'examplealert';
$n->setVariable( 'NotificationEventHandlerSettings', 'AvailableNotificationEventTypes', $a );
echo "handlers: " . implode( ', ', array_keys( eZNotificationEventFilter::availableHandlers() ) )
 . "\n";
```

```
$event = eZNotificationEvent::create( 'examplealert', array( 'message' => 'The example alert work
s.', 'address' => 'nottest-ui@nottest.invalid' ) );
$event->store();
$seen = array();
eZMailNotificationTransport::observe( function ( $addresses, $subject, $body ) use ( &$seen ) { $
seen[] = $addresses[0] . ' | ' . $subject . ' | ' . trim( $body ); }, false );
$r = eZNotificationEventFilter::process( array( (int)$event->attribute( 'id' ) ) );
echo "result: " . json_encode( $r ) . "\n";
foreach ( $seen as $s )
    echo "mail: $s\n";
$left = eZDB::instance()->arrayQuery( 'SELECT COUNT(*) AS n FROM eznotificationevent WHERE id=' .
 (int)$event->attribute( 'id' ) );
echo "event left: " . $left[0]['n'] . "\n";
echo $r['failed'] == 0 && count( $seen ) == 1 ? "PASS\n" : "FAIL\n";
```

```
$ php bin/php/ezexec.php <the script> --allow-root-user
handlers: ezgeneraldigest, ezcollaborationnotification, ezsubtree, examplealert
result: {"events":1,"removed":1,"kept":0,"failed":0}
mail: nottest-ui@nottest.invalid | Example alert | The example alert works.
event left: 0
PASS
```

`process( $ids )` handles only those events; without ids it handles every pending event. In production the event is simply left pending and the next run of the notification cronjob handles it.

3. The mails and their templates

Every handler renders a template and reads variables it sets. For `ezsubtree`, `notification/handler/ezsubtree/view/plain.tpl` has the variable `$object` (the content object) and `$sender` and sets, with `{set-block scope=root variable=...}`:

Variable	Becomes
`subject`	the subject
`from`	the sender text (the display name; the address is `EmailSender`)
`message_id`, `reply_to`, `references`	`Message-ID`, `In-Reply-To`, `References`
`content_type`	the content type (for an HTML mail)

The text of the template is the body. The digest uses `notification/handler/ezgeneraldigest/view/plain.tpl`, which asks `fetch( notification, digest_handlers, hash( date, ..., address, ... ) )` for the handlers that have items for the address and includes `notification/handler/<handler>/view/digest_plain.tpl` for each; the subtree handler's is `ezsubtree/view/digest_plain.tpl` (with `digest_element_plain.tpl` per item). The collaboration mails are in `ezcollaboration/view/` (`plain.tpl` and one directory per type, `ezapprove/`).

The templates are ordinary templates. To change a mail, override it in your design with the usual override mechanism (see template override ordering): copy the file to `design/<yours>/override/templates/` with an override rule for `notification/handler/ezsubtree/view/plain.tpl`, or put a changed file with the same path in your design's `templates/` folder. Keep the `set-block` lines for the variables you need. This guide did not run an override; the standard templates were run, and their output is shown in

the user's guide.

The settings page uses `notification/handler/<id_string>/settings/edit.tpl`, which receives `handler`, and in admin4 also the variables the settings view passes (`subscriptions`, `subscription_total`, `confirm_remove`, ...). A handler that is not one of the three built-in ones gets a generic card with its `edit.tpl`; its `fetchHttpInput( $http, $module )` and `storeSettings( $http, $module )` are called when the page is posted (`Store`).

4. The file transport and where mail goes

`eZFileTransport` writes one file per message. `[MailSettings] FileTransportDirectory` in `site.ini` chooses the directory (default `var/log/mail`):

PHP

```php
$ini = eZINI::instance();
$ini->setVariable( 'MailSettings', 'Transport', 'file' );
$ini->setVariable( 'MailSettings', 'FileTransportDirectory', 'var/tmp/notification-mail' );
```

`setVariable()` changes the settings of the running process only; nothing is written to a file. This is how the console option `--mail-file-dir` and every test work. The files are named `<time>-<number>.mail` and hold the headers and the body.

5. Extension points

Point	How
A new event type	a class file and `notification.ini` as in section 2
A new handler	the same
Switch a handler off	remove it from `[NotificationEventHandlerSettings] AvailableNotificationEventTypes[]`
A new transport	a class `<name>notificationtransport` in `<path><name>notificationtransport.php`, `[TransportSettings] TransportPluginPath[]=<path>` (the path ends with a `/`), and call `eZNotificationTransport::instance( '<name>' )`
Another way to send mail	`[MailSettings] TransportAlias[<name>]=<class>` extending `eZMailTransport`, and `Transport=<name>`
Watch or suppress the mail	`eZMailNotificationTransport::observe( $callback, $suppress )`; the callback gets `( $addresses, $subject, $body, $parameters, $sent )`; `$sent` is whether the transport took the message (absent when suppressed)
Hook into the commands, the cronjob part and the views	`site.ini [RunnableSettings] Listeners[]` and `Implementation[<class>]`, see the runnable classes
Replace the service's behaviour	extend `expNotificationService` and the commands by `Implementation[]` for the command classes; the service itself is static
The audit trail	runs from the console and the web emit `system.command.run`; the cronjob run `system.cronjob.run` (audit event model)

6. Testing

The test class `tests/tests/kernel/classes/notification/NotificationSystemTest.php` is a pattern: live style (it runs on the installation it finds and is skipped where there is none), test content named `NOTTEST`, addresses on `nottest.invalid`, and at the start:

PHP

```php
$ini = eZINI::instance();
$ini->setVariable( 'MailSettings', 'Transport', 'file' );
$ini->setVariable( 'MailSettings', 'FileTransportDirectory', $mailDir );
// refuse to run unless trim( $ini->variable( 'MailSettings', 'Transport' ) ) === 'file'
```

It runs only its own events (`process( $ids )`), so the installation's pending events and digests are not touched, reads the mail from the files, and removes what it made. Run it:

Shell

```shell
php vendor/bin/phpunit tests/tests/kernel/classes/notification/NotificationSystemTest.php
```

```
OK (23 tests, 246 assertions)
```

A digest test needs no waiting: create the event with a time in the future, `eZNotificationEvent::create( 'ezcurrenttime', array( 'time' => $time ) )`, store it and process it by id.

Related pages

- The specification, the INI reference, the command reference
- The administrator's guide, the user's guide
- Extensions, commands, cronjob parts and module views as classes

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.