Writing migrations in PHP

A hands-on page for PHP developers - the migration class and its namespace, the context API, references between migrations, exceptions and transactions per engine, the kernel APIs a migration uses, one complete example...

Source file: doc/eng-us/help/php-developers/writing-migrations-in-php.md Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/php-developers/writing-migrations-in-php.md

This page is for PHP developers who write the migrations of a site or an extension. In Exponential 6.0.15 a migration is a PHP class: you get the run's database, user and environment from a context object, use the kernel's own APIs to make the change, and return what you did. Everything below runs on PHP 7.4 and on every newer PHP.

The class

Shell

```
./console exp:migration:generate --skeleton press_releases --description="The folder Press releases" --apply
```

writes `migrations/20261101100000_press_releases.php` with a header, the namespace `Site\Migrations` and a class `PressReleases20261101100000` that extends `\expMigrationPhp`. An extension's skeleton (`--source=extension:<name>`) gets the namespace `Extension\<Name>\Migrations`. The rules:

- one non-abstract class per file that extends `\expMigrationPhp`, with a name no other file declares;
- no code outside the class: the runner loads the file with `require`;
- the leading backslash on kernel classes (`\eZRole`, `\expMigration`), because the file has a namespace;
- PHP 7.4 syntax: no `match`, no constructor property promotion, no named arguments, no `readonly`.

Method	Called by	Returns
`plan( $context )`	the dry run, the status page	what `up()` would do, one string per line; must not write
`up( $context )`	a run with `--apply`, the simulation	what it did; throw on failure
`down( $context )`	`exp:migration:rollback <id>`	what it undid; leave it out when the change cannot be undone by itself
`detect( $context )`	every state lookup	`\expMigration::DONE`, `PENDING`, `NOT_NEEDED` or `null` (the record decides)
`transactional()`	the runner	`false` for a migration that commits by itself

The context

Call	Gives
`$context->db()`	the run's connection (`eZDBInterface`), inside the run's transaction
`$context->engine()`	`mysql`, `postgresql`, `sqlite`, `oracle` or `mongodb`
`$context->isApply()`	`true` in a run that writes, `false` in a dry run
`$context->environment()`	`development`, `test`, `staging`, `production`, or what `--environment` says
`$context->runId()`, `$context->user()`, `$context->userLogin()`	the run's id and user
`$context->setReference( $name, $value )`, `reference( $name, $default )`	values passed to later migrations of the same run
`$context->log( $level, $text )`	a line in the run log and `var/log/migration.log`
`$context->option( $name )`	an option of the run (`force`, `reason`, `only` ...)

References live for one run. A migration that runs in a later run finds what it needs again by remote id or identifier, never through a reference.

Exceptions and transactions

Throw any exception from `up()` to fail the migration. The runner rolls back its transaction, records it as `failed`, stops the run, keeps the failed state in a caption and restores the caption taken before the run.

Engine	What the transaction undoes
PostgreSQL, SQLite	everything: schema and data
MySQL/MariaDB, Oracle	data only; `CREATE`, `ALTER` and `DROP` commit at once
MongoDB	nothing

So on MySQL/MariaDB, put a schema change and the data that depends on it in separate migrations, and write `up()` so it can meet a half-done state.

The kernel APIs you use

Change	API
a table of your own	`\expKernelTableSchema::install( array( $table ), $context->db(), $schema, $messages )`, `::exists()`
a column or an index	`\expUpgradeCatalog::hasColumn()`, `::hasIndex()`, `::execute()` with SQL from the engine's schema handler
content	`\eZContentFunctions::createAndPublishObject()`, `::updateAndPublishObject()`, `\eZContentObject::fetchByRemoteID()`
a content class	`\eZContentClass::create()`, `\eZContentClassAttribute::create()`, `\eZContentClassClassGroup::create()`

Change	API
roles and policies	`\eZRole::fetchByName()`, `::create()`, `appendPolicy()`, `assignToUser()`, `\eZRole::expireCache()`
settings	`\eZINI::instance( '<file>.ini.append.php', 'settings/override', null, false, null, true )`, `setVariable()`, `save()`

The recipes show each of them in a complete migration.

A complete example

The folder Press releases below the content root, found by its remote id on every installation, and a role that may read it:

PHP

```php
<?php
// description: The folder Press releases and the role Press readers that may read it
// author: Site team
namespace Site\Migrations;

final class PressReleases20261101100000 extends \expMigrationPhp
{
    const REMOTE_ID = 'site-press-releases-folder';
    const ROLE = 'Press readers';

    public function plan( $context )
    {
        return array( 'publish the folder Press releases below node 2 (remote id ' . self::REMOTE
_ID . ')',
                      'create the role ' . self::ROLE . ' with content/read on that folder' );
    }

    public function detect( $context )
    {
        $done = \eZContentObject::fetchByRemoteID( self::REMOTE_ID ) instanceof \eZContentObject
             && \eZRole::fetchByName( self::ROLE ) instanceof \eZRole;
        return $done ? \expMigration::DONE : null;
    }

    public function up( $context )
    {
        $done = array();
        $object = \eZContentObject::fetchByRemoteID( self::REMOTE_ID );
        if ( !$object instanceof \eZContentObject )
        {
            $root = \eZContentObjectTreeNode::fetch( 2 );
            if ( !$root instanceof \eZContentObjectTreeNode )
                throw new \RuntimeException( 'there is no content root (node 2)' );
            $object = \eZContentFunctions::createAndPublishObject( array(
                'parent_node_id' => 2,
                'language' => $root->attribute( 'object' )->attribute( 'initial_language_code' ),
                'class_identifier' => 'folder',
                'creator_id' => \eZUser::currentUserID(),
                'remote_id' => self::REMOTE_ID,
                'attributes' => array( 'name' => 'Press releases' ),
            ), false );
            if ( !$object instanceof \eZContentObject )
                throw new \RuntimeException( 'the folder Press releases could not be published' )
;
            $done[] = 'published the folder Press releases (object ' . $object->attribute( 'id' )
 . ')';
```

```
        }
        $node = $object->mainNode();
        $context->setReference( 'press_releases_node', (int)$node->attribute( 'node_id' ) );

        if ( !\eZRole::fetchByName( self::ROLE ) instanceof \eZRole )
        {
            $role = \eZRole::create( self::ROLE );
            $role->store();
            $role->appendPolicy( 'content', 'read', array( 'Subtree' => array( $node->attribute(
'path_string' ) ) ) );
            \eZRole::expireCache();
            $done[] = 'created the role ' . self::ROLE . ' with content/read on ' . $node->attrib
ute( 'path_string' );
        }
        return $done;
    }

    public function down( $context )
    {
        $done = array();
        $role = \eZRole::fetchByName( self::ROLE );
        if ( $role instanceof \eZRole )
        {
            $role->removeThis();
            \eZRole::expireCache();
            $done[] = 'removed the role ' . self::ROLE;
        }
        $object = \eZContentObject::fetchByRemoteID( self::REMOTE_ID );
        if ( $object instanceof \eZContentObject )
        {
            $object->purge();
            $done[] = 'removed the folder Press releases';
        }
        return $done;
    }
}
```

What to notice: `up()` checks before it creates, so a second run after a partial failure completes the change; `detect()` lets an installation that already has both parts skip it; the folder is created as the run's user (`\eZUser::currentUserID()`); a later migration of the same run reads the node id with `$context->reference( 'press_releases_node' )`.

Shell

```
./console exp:migration:check
./console exp:migration:dry-run --simulate --diff     # PostgreSQL and SQLite
./console exp:migration:migrate --apply
./console exp:migration:rollback 20261101100000_press_releases --apply
```

Testing with PHPUnit on SQLite

Load the file with the kernel's own classes and run it on a throwaway SQLite database, as `tests/tests/kernel/classes/migration/expMigrationRunnerSqliteTest.php` does:

PHP

```php
$locator = new expMigrationLocator( array( array( 'source' => 'site', 'path' => __DIR__ . '/migra
tions' ) ) );
$errors = array();
$migrations = expMigrationRunner::load( ( new expMigrationGraph( $locator->find() ) )->migrations
(), $errors );
$context = new expMigrationContext( $db, true, array( 'environment' => 'testing', 'no-mirror' =>
true,
                                                      'user' => eZUser::fetchByName( 'admin' ) )
);
$report = ( new expMigrationRunner( $context ) )->run( $migrations );
$this->assertSame( 0, $report['exit'], implode( '; ', $report['errors'] ) );
```

Run `up()`, then `down()`, then `up()` again in one test: it proves the migration can be taken back and applied again. Test `plan()` without a database by building it from pure functions.

Where to go next

- PHP migrations: the full reference of the class and the context
- Writing a reversible down()
- Create or change a content class and add roles and policies
- Testing migrations
- Start now: the tutorial

Related pages

- Migrations: the hub
- Migrations: the guide
- Migrations: the tutorial
- PHP migrations
- Migrations in the administration
- The migration commands
- The migration settings
- The migration file format
- When a migration fails
- Troubleshooting migrations
- Migrations versus migrating a site
- PHP Developers: start here
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

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.