Testing integrations

A test strategy for Exponential integrations - unit tests without a database for mappings and parsers, test doubles for the other system, script tests with the kernel loaded, dry runs on a copy, the idempotency test...

Source file: `doc/eng-us/help/integrators/testing-integrations.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/integrators/testing-integrations.md

This page is for integration developers and testers. An integration has two systems, a network and data that changes under it; most of its failures can be found before go-live with a few kinds of tests. The extension-side mechanics are in Tests for extensions.

Contents

1. The test pyramid of an integration
2. Unit tests without a database
3. The other system as a test double
4. Tests with the kernel loaded
5. Dry runs and the idempotency test
6. Mail, webhooks and sandboxes
7. Acceptance with the customer

The test pyramid of an integration

Level	What it proves	How often
Unit tests (no database)	Mapping, parsing, validation, payload building, signature checks	Every commit, in CI
Contract tests with a test double	Your client handles every answer of the other system, including errors	Every commit
Script tests with the kernel	Objects are created, found by remote id, updated only when changed	Before each release, on a development installation
Dry runs on a copy of production	Real data parses; the numbers are plausible	Before go-live and after source format changes
End-to-end in a staging environment with the other system's test instance	The whole flow works, with real credentials of the test systems	Before go-live and after upgrades
Acceptance with the customer	The business result is right	Once, signed

Unit tests without a database

Keep the logic that needs no kernel in plain classes: the CSV parser, the mapping from a source row to attribute values, the payload builder, the signature check. They run in milliseconds with the PHPUnit of the kernel's `vendor/`:

PHP

```php
<?php
use PHPUnit\Framework\TestCase;

final class ProductMappingTest extends TestCase
{
    public function testPriceIsWrittenInThePriceDatatypeFormat(): void
    {
        $mapping = new myIntegrationProductMapping();
        $attributes = $mapping->attributes( array( 'ARTNR' => '10442', 'NAME' => ' Trail shoe ',
'PRICE' => '119.00', 'STOCK' => '7' ) );
        $this->assertSame( '119.00|1|1', $attributes['price'] );
        $this->assertSame( 'Trail shoe', $attributes['name'] );
    }

    public function testRowsWithoutAnArticleNumberAreRefused(): void
    {
        $this->assertNull( ( new myIntegrationProductMapping() )->attributes( array( 'ARTNR' => '
', 'PRICE' => '1' ) ) );
    }
}
```

Run from the extension directory with `php ../../vendor/bin/phpunit` and a `phpunit.xml.dist` as in the chapter above (DB-less PHPUnit tests).

The other system as a test double

Put the other system behind an interface of your own (`myIntegrationHubClient` with `send()`), and give tests a fake that records calls and answers what the test needs: success, `429` with `Retry-After`, `500`, a timeout, an invalid body. Then test that your sender marks rows sent only on success, backs off on `429`, and moves a row to the dead letter after the last attempt. Keep a few recorded real answers of the other system as fixtures, and update them when its API changes.

Tests with the kernel loaded

A test script run with `php bin/php/ezexec.php <script> -s <siteaccess>` on a development installation can create objects with a test prefix, check them and remove them:

PHP

```php
<?php
// extension/myintegration/tests/scripts/upsert_check.php, run with bin/php/ezexec.php on a devel
opment copy only
$importer = new myIntegrationProductImport( $cli, false );
$remoteID = 'test-erp-product-' . time();
// ... call the upsert with a test record, then:
$object = eZContentObject::fetchByRemoteID( $remoteID );
$cli->output( $object instanceof eZContentObject ? 'PASS created' : 'FAIL not created' );
if ( $object instanceof eZContentObject )
{
    $object->removeThis();   // or remove its node to the trash; development copies only
}
```

`ezexec.php` enables the test classes of an extension's `tests/` directory for a script inside it. Never run such scripts against production.

Dry runs and the idempotency test

1. Copy production (database and `var/storage`) to a test installation; anonymize personal data if the copy leaves the production environment.
2. Run the import with `--dry-run`: the counts must be plausible (not "20 000 created" on a site that has them all).
3. Run it for real; check a sample of objects.
4. Run it again at once: it must report only `unchanged`. Any `updated` means the change detection compares formats that differ.
5. Remove a record from the source and run again: the removal policy must apply, and the safety stop must trigger when the source is truncated on purpose.

Mail, webhooks and sandboxes

- **Mail**: `site.ini [MailSettings] Transport=file` on every test installation, and check `settings/override` does not force another transport; test addresses on `example.invalid`.
- **Outgoing webhooks**: point them at a receiver of your own in the test network, and test the signature check with a wrong secret and an old timestamp.
- **Payment providers and identity providers**: use their sandboxes; never real cards or real accounts.
- **API keys**: separate keys for test installations, with short lifetimes.

Acceptance with the customer

Write acceptance cases in the customer's language, with the data they know: "Article 10442 changed in the ERP at 14:00 shows the new price on the product page by 14:10." Sign them off before go-live, and keep them for the next upgrade, when they become the regression test (Upgrades without breaking integrations).

Related pages

- Tests for extensions
- Testing applications
- Quality gates and acceptance
- Integration go-live checklist
- 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.