Integrating With Exponential 6.0

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

This guide connects other systems to an Exponential 6.0 installation: apps, shops, portals, search services, identity providers, log platforms, AI services and deployment pipelines. It shows which ways in and out the installation offers, how a machine signs in, how content and events leave the installation, and how to keep an integration safe and fast. It assumes a working, configured installation from Installing Exponential 6.0 and Configuring Exponential 6.0. Each section names the settings, shows a working example and links to the page with the full detail, so this page stays the map and the deeper pages stay the reference.

Every command is run from the installation root: the directory that holds `index.php`, `autoload.php`, `bin/` and `settings/`. The command console is `bin/php/console`; many installations add the shortcut `./console` (`ln -s bin/php/console console`). When you run a script as `root`, add `--allow-root-user`; the script says so when it is needed. The examples use `https://example.com` for the installation and `site` for its public siteaccess.

> **An integration is a user.** Every way in runs as a user of the installation, and that user's roles and policies decide what it may read and change, exactly as in the administration. Give each integration an account of its own with the smallest role that works, never the administrator's. See Service accounts, roles and policies.

Contents

1. The quick start: your first call
2. Choosing a way in
3. The REST content API and OAuth 2.0
4. Personal API keys and tokens
5. GraphQL
6. Remote services and ezjscore calls
7. SOAP, WebDAV and syndication
8. Content in and out: packages, migrations, imports, exports and feeds
9. Events out: webhooks, the audit trail, mail and the job queue
10. Users and sign-in: LDAP, single sign-on and the user console
11. Front ends: headless use, CORS, caching and Velocity
12. Search
13. AI services
14. Automation and CI: the console
15. Security for integrations
16. Troubleshooting

1. The quick start: your first call

Three steps take a script from nothing to a signed-in read of the content tree.

1. Check which ways in are switched on:

Shell

```
./console exp:ini get site.ini/ExtensionSettings/ActiveExtensions   # ezprestapi, expservices ...
./console exp:ini get rest.ini/Authentication/RequireAuthentication
./console exp:ini get rest.ini/ApiKeySettings/ApiKeys
./console exp:graphql:status --siteaccess=site
```

2. Make a personal API key for the integration's own account: sign in as that account, open **My account > API access**, choose the scopes and an end date, and copy the key. It is shown once. The account's role needs the policy `apikey/create`, which no role has until you give it (Personal API keys).

3. Call the REST content API with it:

Shell

```
curl -s -H 'Authorization: Bearer expk_...' https://example.com/api/ezp/v2/content/node/2
```

Without the key the same request answers `401`, because `rest.ini [Authentication] RequireAuthentication=enabled` is the default. With it, the answer is the content root as JSON, limited to what the key's owner may read.

From here, choose the surface that fits the job (section 2), and read Integrators: start here for the project side of an integration.

2. Choosing a way in

You want to	Use	Section
Read and write content from an app, a shop or a portal	The REST content API, or GraphQL	3, 5
Let users of another application act on the site as themselves	OAuth 2.0	3
Run a script of your own as one account	A personal API key (REST, GraphQL) or an `expservices` token	4
Drive administration functions (caches, users, shop, feeds) remotely	Remote services (`expservices`)	6
Mount the content tree as a drive, or call an old middleware	WebDAV, SOAP	7
Move content, classes and settings between installations	Packages, migrations, syndication	7, 8
Import from or export to files, databases and feeds	The import and export tools, RSS, sitemaps	8
Tell another system when something changes	GraphQL webhooks, audit trail sinks	9
Sign users in from a directory or another identity provider	Login and single sign-on handlers	10
Serve a front end on another host	CORS, the HTTP cache, static publishing	8, 11

You want to	Use	Section
Automate an installation from a pipeline	The console and its JSON output	14

The full inventory, with the direction, protocol, authentication and default of every surface, is Integration surfaces at a glance; the decision in more depth is Choosing an integration surface. Code that runs inside the installation (workflow events, event listeners, cronjob parts, your own module) is extending rather than integrating: see Extending Exponential 6.0.

3. The REST content API and OAuth 2.0

Full guide: REST API v1 and v2

The extension `ezprestapi` serves the content API through `index_rest.php`, below `/api/`:

- **Reads** answer at version 1 and version 2: `/api/ezp/v1/...` and `/api/ezp/v2/...`.
- **Writes** answer at version 2 only: create and delete work; the update route checks `content/edit` and answers 501 for now, so update content through GraphQL's `updateContent` (section 5).
- **Every read and write checks the user's policies** (`content/read`, `content/create`, `content/edit`, `content/remove`), whatever the request was signed in with.

Activate it with `ActiveExtensions[]=ezprestapi` (Configuring Exponential 6.0, section 8). The older `ezprestapiprovider` is deprecated: its last release is a compatibility layer on top of `ezprestapi`, so remove it from `ActiveExtensions[]` once `ezprestapi` is active.

Signing in

`rest.ini [Authentication] RequireAuthentication=enabled` (the default) makes every content route need a user. Two credentials are accepted:

- **A personal API key**, for a script of your own (section 4).
- **An OAuth 2.0 token**, for an application that lets *other* users sign in and act for them:

  1. Register the application at **Setup > oAuth admin** (`/oauthadmin/list`): it gets a client id, a secret and its redirect address.
  2. The application sends the user to `https://example.com/oauth/authorize?client_id=<client id>&redirect_uri=<redirect uri>&response_type=code&state=<random>`, with PKCE also `&code_challenge=<challenge>&code_challenge_method=S256`. The user needs the policy `oauth/authorize`.
  3. It exchanges the code at `POST /api/auth/v1/oauth/token` (`grant_type=authorization_code`) and sends the token as `Authorization: Bearer <token>`.
  4. It renews the token with `grant_type=refresh_token` before `rest.ini [OAuthSettings] TokenTTL` (one hour) ends.

  The token endpoint supports the grants `authorization_code` and `refresh_token`. A public client, such as a mobile app with no secret, must use PKCE. A token stops working as soon as its user is disabled or locked. Details: OAuth applications for integrations, PKCE and state and OAuth token hardening.

Shell

```
# the children of the content root, as the key's owner
curl -s -H 'Authorization: Bearer expk_...' https://example.com/api/ezp/v2/content/node/2/list/offset/0/limit/20
```

More calls, the route table and the settings: REST API v1 and v2 and The REST API for integrators.

4. Personal API keys and tokens

Full guide: Personal API keys

A personal API key lets a script call the REST API and GraphQL as one user, without that user's password, limited to the **scopes** the key was made for.

- **Made by** the user at **My account > API access**, with the policy `apikey/create`. The key starts with `expk_` and is shown once; only its hash is stored.
- **Sent as** `Authorization: Bearer expk_...`.
- **Limited by** `rest.ini [ApiKeySettings]`: `ApiKeys=enabled`, at most `MaxKeysPerUser=10` keys a user, `DefaultExpiryDays=90`, `MaxExpiryDays=365` and `RateLimitPerMinute=120` requests a minute per key.
- **Watched by** the audit trail: `access.apikey.create`, `.revoke`, `.use` and `.use.failed` (with the reason, such as `expired`, `scope` or `rate_limited`). The administration lists every key and can revoke any of them.

`expservices` has tokens of its own (`expt_` and 48 hex characters), made for its remote services (section 6). Which credential suits which integration: Choosing credentials for an integration and Personal API keys for integrations.

5. GraphQL

Full reference: The GraphQL API

The kernel answers GraphQL at `/api/graphql` in every siteaccess where it is switched on. The schema is generated from the installation's content classes: every class is a type, every attribute a field. A query runs as a user (signed in with an OAuth token, a personal API key, the administration's session, or anonymous) and that user's policies decide what it reads. Mutations change content with the same policies, workflows and audit as the administration.

Switch it on, for one siteaccess and one front end, in one command. Without `--apply` it shows what it would change:

Shell

```
./console exp:graphql:enable --siteaccess=site --origin=https://shop.example.com --key-for=frontend
./console exp:graphql:enable --siteaccess=site --origin=https://shop.example.com --key-for=frontend --apply
./console exp:graphql:status --siteaccess=site
```

It sets `graphql.ini [GraphQLSettings] Enabled=enabled` in `settings/override/`, allows the front end's origin (`rest.ini [CorsSettings]`), makes a key with the scope `graphql:read` for the user `frontend`, and prints a `curl` line to try. **Setup > GraphQL** has the same switches. Queries are limited (depth, complexity, list size; see `exp:graphql:status`), and persisted

queries are answered from a response cache.

Walk-throughs: GraphQL walkthrough and GraphQL recipes.

6. Remote services and ezjscore calls

Full guide: Remote services and apps

The extension `expservices` turns the administration's functions into more than a thousand named remote services (content, users, roles, caches, cronjobs, shop, feeds, media, tags, layouts, the audit), served as JSON by the `ezjscore` call interface:

Shell

```
curl -s 'https://example.com/ezjscore/call/expsession::whoami?ContentType=json'
```

A call runs as the signed-in session (with its form token), or as the owner of a personal `expservices` token (`expt_...`). The settings are in `extension/expservices/settings/expservices.ini` (`[Services] Enabled=enabled`); **Setup > RAD** (`/setup/rad`) lists the catalog of the installation. Your own `ezjscore` server functions answer the same way at `/ezjscore/call/<server>::<function>` and need the policy `ezjscore/call`.

More: Remote services: the whole admin as an HTTP API and Remote services with expservices.

7. SOAP, WebDAV and syndication

All three are **off by default** and ask for a password only, never for the second sign-in step (section 15).

Surface	Switch	Entry script	Page
SOAP server (SOAP 1.1, no WSDL)	`soap.ini [GeneralSettings] EnableSOAP=true`	`index_soap.php` (`soap.php` before 6.0.15)	The SOAP entry script
WebDAV (class 1), the content tree as a drive	`webdav.ini [GeneralSettings] EnableWebDAV=true`	`index_webdav.php` (`webdav.php` before 6.0.15)	The WebDAV entry script
`syndication`: content shared between installations as feeds, kept in step by cron	the extension, plus SOAP	`/syndication/menu`	syndication

When a site calls *other* SOAP servers, the client checks their certificates: SOAP client TLS. Both protocols for integrators: WebDAV and SOAP and Syndication between installations.

8. Content in and out: packages, migrations, imports, exports and feeds

Job	Tool	Page
Move content, classes, templates and files between installations	Packages (`.ezpkg`): Setup > Packages, or `ezpm` on the command line	Packages, ezpm

Job	Tool	Page
Apply the same changes to every installation of a site, in order, with a record and a rollback	Migrations: `exp:migration:migrate`, `exp:migration:status`, `--format=json`	Migrations, Migrations in the kernel
Import from files, databases, feeds and other platforms, on a schedule	The import tools, with one run history (**Imports**)	Which import or export tool do I need?, Transfer runs
Export content as CSV, JSON, XML or packages, to SFTP, S3, WebDAV or HTTP	`xrowextract`, the **Export** tab	xrowextract
Users in and out, as CSV or JSON, without passwords	`exp:user:export`, `exp:user:import`	User management from the console
Publish RSS 1.0 or 2.0, Atom, OPML or podcast feeds; import RSS	Setup > RSS; `/rss/feed/<address>`; the cronjob `rssimport.php`	RSS feeds
Tell search engines about every public page	`bcgooglesitemaps`, one `sitemap_<siteaccess>.xml` a siteaccess, by cronjob	bcgooglesitemaps
A static copy of a site on GitHub Pages, any git host or a plain web server	Static publishing: Setup > Static publishing, `exp:static:build`, `exp:static:publish`	Static publishing, Host static sites on GitHub Pages

An import should be keyed by the remote id of each object, so running it twice changes nothing the second time: Idempotent imports keyed by remote id. After a large import, clear the caches it affects (Caching after imports).

9. Events out: webhooks, the audit trail, mail and the job queue

GraphQL webhooks and subscriptions

Wherever content changes (the administration, a mutation, a command, a cronjob, an import), the kernel notes the event (`publish`, `update`, `trash`, `delete`, `move`, `hide`, `reveal`, `mutation`) in an event journal. From it:

- **Webhooks** POST a signed JSON body to another server, at least once, with retries for hours, logged: `./console exp:graphql:webhooks --add=... --apply`, or **Setup > GraphQL**.
- **Subscriptions** stream the events to a client that may read the content: Server-Sent Events at `/api/graphql/subscribe`, or a WebSocket on Velocity.

See Real time: webhooks and subscriptions.

The audit trail and its sinks

Full reference: Audit and The audit trail

The audit trail records what happens in the installation and delivers its records to **sinks**: syslog (RFC 5424, or journald), e-mail, and a **signed webhook**. The webhook is off until it has a URL:

INI

```
# settings/override/audit.ini.append.php
[AuditSink_webhook]
URL=https://siem.example.com/exponential
SigningSecret=<a long random value>
Events[]
Events[]=content.object.publish
```

Each batch carries `X-Exponential-Signature` (HMAC-SHA-256 of the body), `X-Exponential-Timestamp` and `X-Exponential-Batch`; the receiver checks the signature and drops a batch id it has seen. The console's filter can also be exported as CSV, JSON lines or JSON (policy `audit/export`). How to receive and check it: Outgoing webhooks from the audit trail.

Notifications and mail

Notifications tell users by e-mail when content they follow changes. Mail leaves through `site.ini [MailSettings] Transport` (`sendmail`, `smtp` or `file`); see Configuring Exponential 6.0, section 5, Notifications for administrators and Mail and notifications integration.

The job queue

`expqueue` is a database-backed job queue for background work: producers queue a job with one call, and workers run it on the command line, from cron or as a systemd service, never inside a web request. It retries with backoff, has a dead letter, idempotency keys, priorities and concurrency limits that follow rate-limit hints. It is installed with every installation and stays inactive until activated; **Setup > Job queue** shows the jobs. See expqueue and AI providers, the job queue and MCP.

10. Users and sign-in: LDAP, single sign-on and the user console

Need	Setting	Page
Sign in against a directory (LDAP, Active Directory)	`site.ini [UserSettings] LoginHandler[]=LDAP`, with `settings/ldap.ini`	Login handlers: LDAP and textfile
Single sign-on from another system	`site.ini [UserSettings] SingleSignOnHandlerArray[]`, a handler class of your own	Single sign-on and social login
Social login, two-step sign-in, passkeys	`sevenx_authentication_2fa`	Two-step sign-in, Passkeys
Create, change, disable, approve and audit accounts from scripts	`exp:user:*` (`create`, `edit`, `disable`, `enable`, `reset-password`, `approve`, `import`, `export`, `bulk`, `audit` and more)	User management from the console

The login handlers are tried in the order of `LoginHandler[]`; keep `standard` in the list so local accounts, such as the administrator's, still sign in when the directory is down. SAML and OpenID Connect are not shipped: write a login or single sign-on handler, or put an identity-aware proxy in front.

11. Front ends: headless use, CORS, caching and Velocity

- **Headless.** A front end on another host reads content through GraphQL or the REST API and renders it itself. Content keeps its permissions, workflows and audit in Exponential.
- **CORS.** A browser page on another host may call the APIs only when its origin is allowed: `rest.ini [CorsSettings] Cors=enabled` with `AllowedOrigins[]`. It is off by default. Native apps, servers and `curl` do not need it. See CORS for the REST API.
- **The HTTP cache.** `settings/httpcache.ini` (`Enabled=disabled` by default) keeps whole pages per permission context and serves them before the kernel starts, to anonymous and signed-in visitors alike; the `X-Exp-Cache` header tells a hit from a miss. See The role-aware HTTP cache.
- **Velocity.** Exponential Velocity, the recommended web server, adds a response cache, WebSockets, Server-Sent Events and rooms for real-time front ends. See Velocity response cache and Velocity WebSockets, Server-Sent Events and rooms.
- **A static front end.** When the public site needs no PHP at all, static publishing writes it as files (section 8).

Performance advice for integrations: Velocity and performance for integrations.

12. Search

Exponential searches with its built-in index (`site.ini [SearchSettings] SearchEngine=eZSearchEngine`), filled as content is published and rebuilt with `php bin/php/updatesearchindex.php`. For larger sites, facets and other languages' stemming, a Solr server takes over through the `ezfind` extension; `exp:solr` starts, stops and checks it (Server control commands). Feeding an external search service, or reading search results into another system: Search engine integration.

13. AI services

`expai` (Exponential Intelligence) connects the administration to AI providers: editors ask in plain language, and the assistant writes drafts as the signed-in editor, with that editor's permissions; it never publishes on its own. Providers, keys, budgets and the local model are set under Setup > EI: Models and Setup > EI: Budgets; long work runs through `expqueue`. `expai` can also use read-only tools of MCP servers in its panel (policy `expai/mcp`). The kernel itself ships no MCP server. See expai and AI providers, the job queue and MCP.

14. Automation and CI: the console

Full reference: Exponential Console

Every script of Exponential runs through one entry point, `bin/php/console`, which returns the script's exit code unchanged, so it is safe in shell pipelines and CI jobs.

- **Machine output.** Newer commands print a versioned JSON schema with `--format=json`: `exp-user/1` for `exp:user:*`, `exp-migration/1` for `exp:migration:*`. The exit codes are documented per command: 0 done, 1 failed or refused, and so on.
- **Settings from a pipeline.** `exp:ini get|set|where` reads and writes the same files as the settings page (Change settings from the command line).

- **Deploys.** `exp:velocity deploy` runs the steps of a PHP deploy in order and stops at the first failure; migrations apply the content and setting changes (section 8).
- **Your own PHP with the kernel loaded.** `php bin/php/ezexec.php <script.php>`; see CLI scripts and ezexec.
- **Scheduled work.** Cronjob parts run imports and exports on a schedule (Cronjobs for integrations).

15. Security for integrations

Checklist: Integration security checklist

- **One account per integration**, with the smallest role that works, and keys with the narrowest scopes and an end date. Revoke a key the day the integration ends.
- **Two-step sign-in is for people at the sign-in form.** REST basic authentication, WebDAV and SOAP ask for the password only, so keep them off, or limited to accounts that do not need the second step. Machines use API keys or OAuth tokens, which are refused as soon as their user is disabled or locked (Two-step sign-in).
- **Secrets belong in the secret store**, never in an INI file or a repository: **Setup > Secrets**, or a name `env:<VARIABLE>` for containers (The secret store).
- **Rate limits.** API keys are limited per minute (`RateLimitPerMinute`); GraphQL limits depth and complexity; request rules can allow, deny or redirect any address before its view runs (Request rules).
- **Behind a proxy**, list it in `site.ini [HTTPHeaderSettings] TrustedProxies[]`, or the installation sees the proxy's address instead of the client's (Trusted proxies).
- **The Content Security Policy** decides what a page may load; an embedded widget or a script from another host has to be allowed in `site.ini [ContentSecurityPolicy] Directives[]` (Content Security Policy).
- **Watch it.** The audit trail records key use and refusals; `var/log/security.log` collects what looks like an attack (Request logs), and Monitoring integrations lists what to watch.

16. Troubleshooting

Symptom	Cause and fix
`401` from `/api/ezp/...`	No credential, or an expired or revoked one. Send `Authorization: Bearer <key or token>`; check the key's end date at My account > API access.
`403` on a REST or GraphQL call that works in the administration	The integration's account lacks the policy, or the key lacks the scope. Compare its role with the call; the audit's `access.apikey.use.failed` names the reason.
`404` for `/api/...`	`ezprestapi` is not active, or the web server does not route `/api/` to `index_rest.php`. Check `ActiveExtensions[]` and the rewrite rules of Installing Exponential 6.0, section 8.
A browser front end gets a CORS error	`rest.ini [CorsSettings] Cors=enabled` and its origin in `AllowedOrigins[]` (CORS for the REST API). Native apps and servers never need it.
`/api/graphql` answers that GraphQL is off	It is not switched on for that siteaccess: `./console exp:graphql:status --siteaccess=<sa>`, then `exp:graphql:enable ... --apply`.

Symptom	Cause and fix
A key is refused with `rate_limited`	More than `RateLimitPerMinute` requests a minute. Slow the client down, or queue the work through expqueue.
The webhook receiver rejects the audit's batches	The signature is checked over the raw body with the same `SigningSecret`; check the clock too, because the timestamp is signed.
WebDAV or SOAP answers nothing	Both are off by default (`EnableWebDAV`, `EnableSOAP`), and the entry scripts are `index_webdav.php` and `index_soap.php` since 6.0.15.
An LDAP user cannot sign in, local accounts can	Check `settings/ldap.ini` (server, base DN, the attribute that holds the login) and that LDAP is in `LoginHandler[]` (Login handlers: LDAP and textfile).
Changes made by an import do not show	The caches still hold the old pages; clear them after the import (Caching after imports).
A changed setting has no effect	The INI cache: `php bin/php/ezcache.php --clear-tag=ini`, and on Velocity a restart (Configuring Exponential 6.0, section 10).

More: Integrators FAQ, Error handling and logging and Testing integrations.

Related pages

- Installing Exponential 6.0, Configuring Exponential 6.0, Updating Exponential 6.0 and Extending Exponential 6.0.
- Integrators: start here and The integrator's reading path.
- API and protocol standards.
- Upgrades without breaking integrations.

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.