Upgrading

- Semantic Versioning
- Categorizing Breaking Changes
  - Type of Change:
  - Impact and Effort:
- Communication of Breaking Changes
- Changelog and Upgrade Guide
- Release Cadence
- Deprecations
- Support Policy
  - What Does "Support" Mean?
  - Unsupported Versions
- Beta Period and Testing
- Process for Releasing Breaking Changes
- Frequently Asked Questions (FAQ)
  - How do I know if a release contains breaking changes?
  - What should I do if I experience issues after upgrading?
  - How can I test new versions before upgrading my production environment?
  - What are the supported versions of WPGraphQL?
  - How are deprecations handled?
  - How do I stay informed about new releases and breaking changes?
  - What should I do if I am using a deprecated feature?
  - Does WPGraphQL provide long-term support (LTS) for specific versions?
  - Are security updates backported to older versions?

Upgrading WPGraphQL is an important part of keeping your API secure, performant, and up to date with new features. However, as with any software, some updates may introduce breaking changes that require you to take action to maintain compatibility with your project.

This guide outlines the key steps and policies for upgrading WPGraphQL, including how we handle breaking changes, what to expect from versioning, and how you can plan your upgrades.

Semantic Versioning

WPGraphQL follows **Semantic Versioning** (SemVer) to communicate the meaning of each release and the nature of changes introduced. This ensures that users can easily understand the potential impact of any given release. Semantic Versioning uses the format **MAJOR.MINOR.PATCH**:

MAJOR (e.g., `2.0.0`): - A major version increment indicates a breaking change. Something in this release requires users to take action to remain compatible. Examples include changes to the API schema, function signatures, or other critical changes that could disrupt compatibility. - Major versions will contain clear documentation and upgrade guides to help users transition smoothly.

MINOR (e.g., `1.1.0`): - A minor version increment adds new functionality in a backward-compatible manner. Users should be able to upgrade without breaking their existing code or integrations. - Minor releases may include enhancements, new features, and optimizations, but they will not require users to make any changes to existing functionality.

PATCH (e.g., `1.0.1`): - A patch version increment is for backward-compatible bug fixes. These fixes address issues without affecting functionality or introducing breaking changes. - Patch releases are safe for users to apply without concern for changes to their existing workflows.

By adhering to Semantic Versioning, WPGraphQL clearly signals when users should expect breaking changes, when new features are available, and when critical bugs are fixed, helping the community make informed upgrade decisions.

Categorizing Breaking Changes

Breaking changes will be categorized based on two main factors: **Type of Change** and **Impact & Effort**.

Type of Change:

PHP API Changes: Changes to the underlying PHP codebase, which will affect plugins and custom code interacting with WPGraphQL internal APIs. - ex: Changes to action hooks, filters, or internal WPGraphQL PHP functions. **GraphQL API Changes**: Changes that impact how the API is consumed by clients and front-end applications. - ex: Modifications to the GraphQL schema, such as field removals or type changes, changes in GraphQL response structures, return codes, headers, or other external behavior, or new required arguments.

Impact and Effort:

Breaking changes are further classified by **Impact Scope** and **Effort to Upgrade**.

Impact Scope: - Low: Affects few users (niche use cases or specific extensions). - Medium: Affects a moderate number of users or common use cases. - High: Affects most or all users interacting with the API.

Effort to Upgrade: - Low: Likely Requires little to no action to remain compatible. - Medium: Likely Requires small adjustments (e.g., changing function signatures or making minor changes to a schema query). - High: Likely Requires refactoring large sections of code, schema, or plugin integrations.

Each breaking change will be classified using these dimensions (e.g., "Type: PHP API, Impact: High, Effort: Low").

> [!NOTE] Every project is unique, and we cannot know for sure how your project(s) may be impacted by a breaking change, but we will use our contextual and history knowledge to help provide an estimate for how the breaking changes in a release might impact projects looking to update.

Communication of Breaking Changes

Breaking changes will be documented centrally and then shared before and after release through multiple channels to ensure broad community awareness.

These channels include:

- GitHub Discussions: For community discussions and feedback around releases and breaking changes.
- WPGraphQL Discord
- Headless WordPress Discord
- X (formerly Twitter)
- Email

Changelog and Upgrade Guide

For any version containing breaking changes, the changelog will include:

- A summary of the breaking changes with categorization (Impact Scope + Effort to Upgrade).
- A highlighted link to a detailed upgrade guide.

The **Upgrade Guide** will provide:

- Step-by-step instructions to adapt to the breaking change.
- Code examples, if applicable.
- Recommendations for common issues during upgrades.

Release Cadence

WPGraphQL takes an approach of **smaller, more frequent releases**. This means:

- **Incremental Changes**: Breaking changes will be released in smaller increments, allowing users to upgrade more easily and manage the impact with minimal disruption.
- **Clear Communication**: Each breaking change will be categorized by **Impact Scope** and **Effort to Upgrade**, enabling users to make informed decisions about when to upgrade.
- **Targeted Version Support**: WPGraphQL will support only the latest `minor.patch` within the last 2 major versions to reduce fragmentation, ensuring that users have clear expectations regarding version support.

Deprecations

WPGraphQL may deprecate certain functionality before fully removing or changing it in a future release. Deprecated features will continue to function for a specified period but will trigger warnings, allowing users to transition before breaking changes are introduced.

> [!TIP] Deprecations are intended to give you time to adjust your code before a feature is fully removed or changed. Keep an eye on deprecation warnings to ensure smooth upgrades.

Deprecations will be clearly communicated in release notes and upgrade guides, providing users with sufficient time to adapt their projects before the feature is fully removed or changed.

Support Policy

WPGraphQL will support only the latest `minor.patch` version of the **last 2 major versions**. This means that older versions within a major release (e.g., `1.1.x` if `1.2.x` exists) and versions beyond this window will no longer receive official support, and users will be encouraged to upgrade to the latest version.

> [!NOTE] Only the latest `minor.patch` of each of the last 2 major versions is supported. Versions within a major release (e.g., `1.1.x` if `1.2.x` exists) or beyond the last 2 major versions will not receive support.

What Does "Support" Mean?

"Support" includes:

- **Bug Fixes**: Active work will be done to resolve bugs and issues reported on supported versions. Bug fixes will be prioritized by severity and impact on WPGraphQL's core functionality. Known bugs, as well as newly reported ones, will be addressed based on development priorities.
- **Security Updates**: General and critical security fixes will be backported to the latest `minor.patch` within the last 2 major versions, ensuring that these versions remain secure.
- **Compatibility**: Supported versions will be tested against newer versions of WordPress core to ensure continued functionality. While WPGraphQL may be tested with certain versions of PHP and popular extensions, compatibility with these cannot be guaranteed.
- **Documentation**: Clear documentation and upgrade guides will be provided for supported versions, including guidance on adapting to new features or breaking changes.

> [!NOTE] Compatibility testing is primarily focused on WordPress core updates. WPGraphQL may also be tested with certain PHP versions and popular extensions, but compatibility with these is not guaranteed.

Unsupported Versions

Versions older than the last 2 major versions or outdated `minor.patch` versions within a major release will no longer receive:

- Bug fixes
- Security updates

- Compatibility testing or guarantees with new versions of WordPress, PHP, or third-party extensions

While older versions may still function, the quality and speed of support will be higher for those using the latest versions.

Beta Period and Testing

WPGraphQL will establish a **beta period** of approximately 30 days for major releases. During this time, users are encouraged to:

- Test the upcoming release on a specific **release branch**.
- Provide feedback and report bugs by participating in the Pull Request (PR) discussion opened for the specific beta release.

By gathering feedback in the beta PR, we ensure all discussions and issues are centralized and visible to the community.

Process for Releasing Breaking Changes

Pre-release Preparation: - Categorize the breaking change (Impact Scope + Effort to Upgrade). - Document the change in the upgrade guide. - Prepare a release branch for major releases with a beta period (30 days).

Pre-release Communication: - Share early release notes in community channels (Discord, Twitter, Facebook, etc.). - Encourage community testing and feedback during the beta period.

Post-release: - Announce the release and provide a link to the changelog and upgrade guide. - Monitor feedback and issue patches as necessary.

Frequently Asked Questions (FAQ)

How do I know if a release contains breaking changes?

Breaking changes will be clearly communicated in the release notes, changelog, and upgrade guide for each version. Each breaking change is categorized by Impact Scope and Effort to Upgrade so you can quickly understand the potential impact on your project.

What should I do if I experience issues after upgrading?

If you encounter issues after upgrading, we recommend:

- Checking the Upgrade Guide for the version you're upgrading to.
- Reviewing the release notes for any breaking changes that might apply to your setup.
- Searching the WPGraphQL Discord or GitHub Discussions to see if others have encountered similar issues.
- If the problem persists, open a GitHub Issue or ask for help in the WPGraphQL Discord.

> [!TIP] For best results, review the upgrade guide and release notes, and consider reaching out to the community on Discord or GitHub Discussions if issues persist.

How can I test new versions before upgrading my production environment?

We offer a **beta period** of approximately 30 days for major releases, where you can test the upcoming version on a specific **release branch**. During this period, you can help identify any issues and provide feedback. We recommend setting up a staging environment for testing major upgrades before deploying to production.

What are the supported versions of WPGraphQL?

WPGraphQL supports the latest `minor.patch` of the last 2 major versions. If you're on an older version, we encourage you to upgrade to one of the latest supported versions to ensure continued support, access to new features, and security patches.

How are deprecations handled?

Before we remove or change a feature that could break backward compatibility, we may deprecate the functionality first. Deprecations are communicated in release notes, and deprecated features will continue to function for a specified period but will trigger warnings. This gives you time to adjust your code before the feature is fully removed or changed.

How do I stay informed about new releases and breaking changes?

You can stay informed by following these communication channels:

- WPGraphQL Discord
- Headless WordPress Discord
- X (formerly Twitter)
- Email notifications

We also publish detailed release notes and upgrade guides with each new version.

What should I do if I am using a deprecated feature?

If you're using a deprecated feature, we recommend reviewing the release notes for the deprecation details. You'll usually have a specified period during which the deprecated feature will continue to function. Use this time to update your code to the newer methods before the feature is fully removed.

Does WPGraphQL provide long-term support (LTS) for specific versions?

At this time, WPGraphQL does not offer long-term support for specific versions. We recommend staying up to date with the latest version to ensure compatibility and support.

Are security updates backported to older versions?

General and critical security updates are only backported to the latest `minor.patch` of the last 2 major versions when possible in a non-breaking way. If you are using an older version of WPGraphQL, we recommend upgrading to a supported version to ensure your site remains secure.

□□□ #1

□ □□□ □□□ 2026-08-29 15:36:26 UTC

□ □□□ □□□ 2026-08-29 15:36:26 UTC