Contributing Experiments

- Before You Start
  - Is an Experiment Right for Your Idea?
  - Read the Guidelines
- Contribution Process
  - Step 1: Propose the Experiment
  - Step 2: Discuss and Refine
  - Step 3: Implement the Experiment
    - Fork and Branch
    - Create the Experiment
    - Follow Code Standards
    - Write Great Commit Messages
  - Step 4: Submit Pull Request
  - Step 5: Code Review
  - Step 6: Merge and Release
- After Your Experiment Ships
  - Monitor Feedback
  - Be Available
  - Iterate if Needed
- Graduation Process
  - 1. Evaluate Readiness
  - 2. Create Graduation Plan
  - 3. Implement Graduation
  - 4. Support During Transition
- Deprecation Process
  - 1. Understand Why
  - 2. Add Deprecation Notice
  - 3. Create Migration Guide
  - 4. Communicate

    - 5. Support Users
- Tips for Success
    - Technical Tips
    - Communication Tips
    - Process Tips
- FAQ
    - How long does the process take?
    - Can I propose multiple experiments?
    - What if my experiment isn't approved?
    - Do I need to be a core committer?
    - Will I get credit?
- Getting Help
- Resources
- Pull Request Templates

This guide covers how to contribute experiments to WPGraphQL core, from proposal to implementation.

Before You Start

Is an Experiment Right for Your Idea?

Experiments are best suited for:

- **Proposed core features** that need real-world validation
- **Breaking changes** you want to test before committing
- **Controversial features** that need community feedback
- **Performance improvements** that need testing at scale

Experiments are **not** suitable for:

- **Plugin-specific features** (create a WPGraphQL extension instead)
- **Bug fixes** (submit a regular pull request)
- **Minor enhancements** (submit a regular pull request)
- **Features that are clearly out of scope** for WPGraphQL core

Read the Guidelines

Before proposing an experiment:

1. Review WPGraphQL's vision and scope
2. Search existing GitHub Issues to avoid duplicates
3. Read the Contributing Guide
4. Understand the experiment lifecycle

Contribution Process

Step 1: Propose the Experiment

Create a GitHub Issue using the "Experiment Proposal" template (a maintainer will add the "experiment: needs refinement" or "experiment: approved" label):

1. Go to Create New Issue
2. Select "Experiment Proposal" from the template dropdown
3. Fill out the template with your proposal details

The template includes all the necessary sections to help you structure your proposal effectively.

Step 2: Discuss and Refine

Engage with the community:

- **Respond to questions** promptly and thoughtfully
- **Consider feedback** and adjust the proposal
- **Address concerns** about scope, implementation, or impact
- **Build consensus** among maintainers and community

A WPGraphQL maintainer will label your issue:

- `experiment: approved` - Ready to implement
- `experiment: needs refinement` - More discussion needed
- `experiment: not suitable` - Better as extension or PR

Step 3: Implement the Experiment

Once approved, create your implementation:

Fork and Branch

```
# Fork the repository on GitHub first

# Clone your fork
git clone git@github.com:YOUR-USERNAME/wp-graphql.git
cd wp-graphql

# Add upstream remote
git remote add upstream git@github.com:wp-graphql/wp-graphql.git

# Create feature branch
git checkout -b experiment/email-address-scalar
```

Create the Experiment

Follow the Creating Experiments guide:

1. Create experiment class in `src/Experimental/Experiment/`
2. Register in `ExperimentRegistry.php`
3. Write comprehensive tests
4. Add inline documentation

Follow Code Standards

```
# Run code sniffer
composer phpcs

# Fix auto-fixable issues
composer phpcs:fix

# Run PHPStan
composer phpstan

# Run tests
composer test
```

Write Great Commit Messages

```
# Good commit message format
git commit -m "feat(experiments): Add EmailAddress scalar experiment

- Adds EmailAddress scalar type with validation
- Updates User.email field to use new scalar
- Includes comprehensive tests for valid/invalid emails
- Documents validation rules and limitations

Closes #1234"
```

Step 4: Submit Pull Request

Create a pull request with this template:

Title: `[Experiment] Email Address Scalar`

Description:

```
## Description

Implements the EmailAddress scalar experiment as proposed in #1234.

## Changes

- [ ] Created `EmailAddressScalarExperiment` class
- [ ] Registered EmailAddress scalar type
- [ ] Updated User.email field to use EmailAddress scalar
- [ ] Added 15+ tests covering validation, schema changes, and edge cases
- [ ] Added inline documentation
- [ ] Updated experiments roadmap

## Testing

Tested in:

- [x] Local development environment
- [x] WordPress 6.4
- [x] PHP 8.1, 8.2
- [x] All existing tests pass
```

```
- [x] New tests pass

## Screenshots

(If applicable, add screenshots of the Experiments settings page or GraphiQL IDE)

## Breaking Changes

When enabled, this experiment changes:

- User.email field from String → EmailAddress
- Validation now happens at GraphQL layer
- Invalid emails will throw GraphQL errors

## Documentation

- [x] Inline code documentation
- [ ] User-facing documentation (will add after approval)
- [x] Updated README in src/Experimental/

## Related Issues

Closes #1234
Relates to #5678 (EmailAddress validation discussion)

## Checklist

- [x] Code follows project style guidelines
- [x] All tests pass
- [x] PHPStan passes
- [x] PHPCS passes
- [x] Experiment can be enabled/disabled without errors
- [x] No breaking changes to core when experiment is disabled
- [x] Commit messages follow conventional commits
```

Step 5: Code Review

Be prepared for feedback:

- **Respond to review comments** within a few days

- **Make requested changes** promptly
- **Ask questions** if feedback is unclear
- **Be open** to different approaches

Common feedback topics:

- Code quality and standards
- Test coverage
- Performance implications
- Documentation clarity
- Naming conventions
- Backwards compatibility

Step 6: Merge and Release

Once approved:

1. Core team will merge your PR
2. Experiment will be included in next release
3. You'll be credited in release notes
4. Community testing begins

After Your Experiment Ships

Monitor Feedback

Pay attention to:

- GitHub Issues mentioning your experiment
- Discussions asking questions
- Performance reports
- Edge cases discovered in the wild

Be Available

As the experiment author, you're expected to:

- Answer questions from users
- Address bugs that arise
- Consider enhancement requests

- Help with documentation improvements
- Participate in graduation/deprecation discussions

Iterate if Needed

Experiments can receive breaking changes, so don't be afraid to:

- Refactor based on feedback
- Add missing functionality
- Improve performance
- Simplify the API

Submit additional PRs as needed with the `[Experiment Update]` prefix.

Graduation Process

When your experiment is ready to graduate:

1. Evaluate Readiness

Check the criteria:

- ☐ At least 2-3 releases have passed
- ☐ Positive feedback from production users
- ☐ No major unresolved issues
- ☐ Performance is acceptable
- ☐ Documentation is complete
- ☐ Tests have good coverage
- ☐ Core team agrees it's ready

2. Create Graduation Plan

Open a new GitHub Issue:

Title: `[Experiment Graduation] Email Address Scalar`

Content:

```
## Experiment Summary
```

The EmailAddress scalar experiment has been active for 3 releases and is ready to graduate to core.

Usage Stats

(If available)

- X downloads/installs with experiment enabled
- Y production sites using it
- Z positive feedback comments

Feedback Summary

Positive:

- Users love the built-in validation
- Reduced boilerplate code
- Clear error messages

Concerns Addressed:

- Initial performance concern resolved in v2.1
- Custom validation filter added in v2.2

Graduation Plan

- [ ] Move implementation from experiment to core
- [ ] Keep experiment class with deprecation notice
- [ ] Update documentation
- [ ] Add to changelog
- [ ] Provide migration guide
- [ ] Set removal date (2 releases out)

Breaking Changes

For users who enabled the experiment:

- No changes needed, it becomes always-on

```
For users who didn't enable the experiment:

- User.email changes from String → EmailAddress
- Migration guide: Update queries expecting String type

## Migration Timeline

- v2.5: Experiment graduates, remains available with deprecation notice
- v2.6: Deprecation notices continue
- v3.0: Experiment removed, feature is always-on
```

3. Implement Graduation

Follow the graduation process:

1. Move core code to appropriate location
2. Add deprecation message to experiment config
3. Update documentation
4. Write migration guide
5. Update changelog

4. Support During Transition

During the deprecation period:

- Answer migration questions
- Help users update their code
- Document common migration patterns
- Create example code snippets

Deprecation Process

If your experiment needs to be deprecated:

1. Understand Why

Common reasons for deprecation:

- Better solution emerged
- Performance issues can't be resolved
- Breaking changes are too severe
- Core team decided against the feature
- Low adoption/interest

2. Add Deprecation Notice

Update the experiment config:

```
'deprecationMessage' => __(
    'This experiment is deprecated and will be removed in v3.0.0. Use [alternative] instead.
See migration guide: [link]',
    'wp-graphql'
)
```

3. Create Migration Guide

Document how users should migrate:

- What the alternative is
- How to update their code
- Timeline for removal
- Where to get help

4. Communicate

Announce via:

- Release notes
- GitHub Issue
- Social media
- Documentation updates

5. Support Users

During deprecation:

- Answer migration questions
- Help troubleshoot issues

- Provide code examples
- Be empathetic - change is hard!

Tips for Success

Technical Tips

1. **Keep it focused**: One experiment = one feature
2. **Write tests first**: TDD helps clarify the API
3. **Document as you go**: Don't leave docs for last
4. **Consider performance**: Profile your code early
5. **Think about migration**: How will this graduate or deprecate?

Communication Tips

1. **Be clear**: Explain the "why" not just the "what"
2. **Be responsive**: Reply to feedback promptly
3. **Be humble**: Your first design might not be the best
4. **Be patient**: Consensus takes time
5. **Be grateful**: Thank people for their feedback

Process Tips

1. **Start small**: Propose before implementing
2. **Build consensus**: Address concerns early
3. **Follow up**: Don't abandon your experiment after shipping
4. **Iterate**: Use feedback to improve
5. **Know when to pivot**: Not every experiment will graduate

FAQ

How long does the process take?

From proposal to merge:

- Discussion phase: 1-2 weeks minimum

- Implementation: 1-2 weeks (depends on complexity)
- Review: 1-2 weeks
- **Total: ~1-2 months for simple experiments**

Can I propose multiple experiments?

Yes, but focus on getting one merged successfully first. This builds trust and demonstrates your ability to follow through.

What if my experiment isn't approved?

That's okay! Consider:

- Building it as a separate extension
- Refining the proposal based on feedback
- Starting a smaller proof-of-concept
- Contributing to other experiments

Do I need to be a core committer?

No! Community contributions are welcome. Many successful experiments come from non-core contributors.

Will I get credit?

Yes! You'll be:

- Listed in the experiment file's `@author` tag
- Mentioned in release notes
- Credited in commit history
- Recognized in the community

Getting Help

Stuck or have questions?

- **GitHub Issues**: Create an issue (a maintainer will add the "question" label)
- **Slack**: Ask in #contributors channel

- **Office Hours**: Join WPGraphQL office hours (if scheduled)
- **Mentorship**: Ask if a core member can mentor you

Resources

- Creating Experiments Guide
- Using Experiments Guide
- Contributing Guide
- Testing Guide
- Code Standards

Ready to contribute? Start by creating an experiment proposal issue using the "Experiment Proposal" template!

Pull Request Templates

When implementing your approved experiment, use our specialized PR templates:

1. **Go to**: Create New Pull Request
2. **Select**: The default template will show a "chooser" with links to specialized templates
3. **Click**: The appropriate template link (e.g., "New Experiment Template")
4. **Fill out**: The template with your implementation details

This ensures your PR includes all the necessary information for efficient review.

□□□ #2
□ □□□ □□□ 2026-08-29 23:36:26 UTC
□ □□□ □□□ 2026-08-29 23:36:26 UTC