Using Experiments

- Enabling Experiments
  - Via WordPress Admin
  - Via wp-config.php (Constant)
  - Via Code (Filter)
- Testing Experiments
  - Step 1: Enable in Staging
  - Step 2: Verify Schema Changes
  - Step 3: Test Your Queries
  - Step 4: Monitor Performance
  - Step 5: Review Documentation
- Managing Experiments
  - Checking Experiment Status
  - Disabling Experiments
  - Updating Experiments
- Common Use Cases
  - Use Case 1: Testing New Scalar Types
  - Use Case 2: Enhanced Input Types
  - Use Case 3: Performance Improvements
- Environment-Specific Strategies
  - Development
  - Staging
  - Production
- Troubleshooting
  - Experiment Not Showing in Schema
  - Unexpected Behavior
  - Breaking Changes After Update
- Best Practices
  - Do:

- Don't:
- Providing Feedback
  - What to Share
  - Where to Share
- What Happens When Experiments Graduate
- What Happens When Experiments Are Deprecated
- Next Steps

This guide shows you how to enable, test, and manage experiments in WPGraphQL.

Enabling Experiments

Via WordPress Admin

The easiest way to enable experiments:

1. Navigate to **GraphQL > Settings** in your WordPress admin
2. Click the **Experiments** tab
3. Find the experiment you want to enable
4. Check the checkbox next to the experiment
5. Click **Save Changes**
6. Clear any schema caches if applicable

Experiments Settings Screenshot

Via wp-config.php (Constant)

For environment-specific control, use the `GRAPHQL_EXPERIMENTAL_FEATURES` constant:

```
// In wp-config.php

// Enable specific experiments
define( 'GRAPHQL_EXPERIMENTAL_FEATURES', [
    'email_address_scalar' => true,
    'one_of_inputs'        => true,
] );
```

Disable all experiments:

```
// Useful for production environments
define( 'GRAPHQL_EXPERIMENTAL_FEATURES', false );
```

Important: The constant takes precedence over admin settings. If you set the constant, the admin settings will be ignored.

Via Code (Filter)

For advanced programmatic control when the constant is not defined:

```
add_filter( 'graphql_experimental_features_override', function( $experiments ) {
    return [
        'email_address_scalar' => true,
    ];
} );
```

Important: This filter only works when the `GRAPHQL_EXPERIMENTAL_FEATURES` constant is not defined. If the constant is defined, it takes precedence and cannot be overridden by filters. This follows WordPress best practices where constants have the "final say" for configuration.

Testing Experiments

Step 1: Enable in Staging

Always test experiments in a staging environment first:

1. Create a staging site with a copy of your production data
2. Enable the experiment in **GraphQL > Settings > Experiments**
3. Clear schema cache (if your setup uses caching)
4. Test your GraphQL queries

Step 2: Verify Schema Changes

Use the GraphiQL IDE to inspect schema changes:

```
# Check if a new type was added
query IntrospectNewType {
  __type(name: "EmailAddress") {
    name
    kind
    description
  }
}
```

Step 3: Test Your Queries

Run your actual application queries against the staging environment:

- Test all query variations your app uses
- Check mutations if applicable
- Verify error handling
- Test edge cases

Step 4: Monitor Performance

Watch for any performance impacts:

- Query execution time
- Server resource usage
- Client-side rendering time
- Database query count

Step 5: Review Documentation

Each experiment should have documentation explaining:

- What it does
- What changes to expect
- Known limitations
- How to provide feedback

Managing Experiments

Checking Experiment Status

See which experiments are currently active:

```
// In your code
$registry = \WPGraphQL\Experimental\ExperimentRegistry::get_instance();
$experiments = $registry ? $registry->get_active_experiments() : [];

foreach ( $experiments as $slug => $experiment ) {
    echo "Active: " . $experiment->get_config()['title'];
}
```

Or via GraphQL response extensions (when debug is enabled):

```
query CheckExperiments {
  # Active experiments appear in response extensions when GRAPHQL_DEBUG is enabled
  __typename
}
```

Response will include (when `GRAPHQL_DEBUG` is enabled):

```
{
  "data": { "__typename": "RootQuery" },
  "extensions": {
    "experiments": ["test_experiment", "email_address_scalar"]
  }
}
```

Benefits of GraphQL Extensions Response:

- **Client-side Detection**: Your GraphQL clients can automatically detect which experimental features are available
- **Debugging**: Easily identify which experiments are active without checking WordPress admin
- **Team Coordination**: Frontend developers can see experimental features without backend coordination
- **Development Only**: Only appears when `GRAPHQL_DEBUG` is enabled, keeping production responses clean

Example Client Usage:

```
// JavaScript example - only works when GRAPHQL_DEBUG is enabled
const response = await fetch('/graphql', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ query: '{ __typename }' })
});

const data = await response.json();

if (data.extensions?.experiments?.includes('email_address_scalar')) {
  // Use the EmailAddress scalar type
  console.log('EmailAddress scalar is available!');
}
```

Disabling Extensions Response:

If you need to disable the experiments extensions response:

```
add_filter( 'graphql_should_show_experiments_in_extensions', '__return_false' );
```

Disabling Experiments

If you encounter issues:

1. Go to **GraphQL > Settings > Experiments**
2. Uncheck the experiment
3. Click **Save Changes**
4. Clear schema cache
5. Refresh your application

Your site will immediately revert to stable behavior.

Updating Experiments

When a new version of WPGraphQL is released:

1. Review the changelog for experiment changes
2. Check if any active experiments have updates
3. Read the migration notes if breaking changes occurred
4. Test in staging with the new version

5. Update production after successful testing

Common Use Cases

Use Case 1: Testing New Scalar Types

```
# Before experiment (using String)
mutation UpdateUser {
  updateUser(
    input: {
      id: "user123"
      email: "notanemail" # No validation
    }
  ) {
    user {
      email
    }
  }
}

# After enabling EmailAddress scalar experiment
mutation UpdateUser {
  updateUser(
    input: {
      id: "user123"
      emailAddress: "notanemail" # Now validated!
    }
  ) {
    user {
      email
    }
  }
}
# Returns error: "notanemail is not a valid email address"
```

Use Case 2: Enhanced Input Types

```
# Experiment: oneOf inputs for better mutations
mutation CreatePost {
  createPost(
    input: {
      # Only one of these is required/allowed
      title: "My Post"
      # OR
      titleI18n: { en: "My Post", es: "Mi Publicación" }
    }
  )
}
```

Use Case 3: Performance Improvements

Some experiments might add query optimization features:

```
# Experiment: Query complexity analysis
query HugeQuery {
  posts(first: 100) {
    nodes {
      # Warns if query is too complex
    }
  }
}
```

Environment-Specific Strategies

Development

Enable experiments freely to stay current:

```
// wp-config.php (local development)
if ( defined( 'WP_ENVIRONMENT_TYPE' ) && 'local' === WP_ENVIRONMENT_TYPE ) {
    define( 'GRAPHQL_EXPERIMENTAL_FEATURES', true ); // Enable all
}
```

Staging

Enable specific experiments for testing:

```
// wp-config.php (staging)
if ( defined( 'WP_ENVIRONMENT_TYPE' ) && 'staging' === WP_ENVIRONMENT_TYPE ) {
    define( 'GRAPHQL_EXPERIMENTAL_FEATURES', [
        'email_address_scalar' => true,
    ] );
}
```

Production

Be conservative:

```
// wp-config.php (production)
if ( defined( 'WP_ENVIRONMENT_TYPE' ) && 'production' === WP_ENVIRONMENT_TYPE ) {
    define( 'GRAPHQL_EXPERIMENTAL_FEATURES', false ); // Disable all by default
}
```

Then explicitly enable only well-tested experiments via admin UI if needed.

Troubleshooting

Experiment Not Showing in Schema

1. **Verify it's enabled**: Check GraphQL > Settings > Experiments
2. **Clear cache**: If using object caching, clear it
3. **Check for conflicts**: Disable other plugins temporarily
4. **Review error logs**: Look for PHP errors or warnings
5. **Check constant**: Ensure `GRAPHQL_EXPERIMENTAL_FEATURES` isn't set to `false`

Unexpected Behavior

1. **Disable the experiment**: Return to stable behavior
2. **Review documentation**: Check for known limitations
3. **Test in isolation**: Disable other experiments to isolate the issue

4. **Report the issue**: Create a GitHub issue with:
   - WPGraphQL version
   - WordPress version
   - Steps to reproduce
   - Expected vs actual behavior
   - Error messages (if any)

Breaking Changes After Update

1. **Check the changelog**: Look for breaking change notices
2. **Review migration guide**: Follow provided migration steps
3. **Test thoroughly**: Don't skip staging testing
4. **Provide feedback**: Let the team know if migration was difficult

Best Practices

Do:

- **Test in staging first**: Never enable experiments directly in production
- **Read documentation**: Understand what each experiment does
- **Provide feedback**: Help improve experiments with your real-world usage
- **Monitor updates**: Keep an eye on experiment status changes
- **Have a rollback plan**: Know how to disable if issues arise
- **Document your usage**: Note which experiments you're using and why

Don't:

- **Assume stability**: Experiments can have breaking changes
- **Skip testing**: Always test updates in staging
- **Ignore deprecation notices**: Start planning migration when you see them
- **Use in production without testing**: Risk of unexpected behavior
- **Depend on experiments indefinitely**: They're temporary by design
- **Mix experimental and stable without testing**: Ensure compatibility

Providing Feedback

Your feedback shapes which experiments graduate to core!

What to Share

Positive feedback:

- Specific use cases that work well
- Performance improvements you've noticed
- Developer experience wins
- Features that solve real problems

Constructive feedback:

- Bugs or unexpected behavior
- Performance issues
- Missing functionality
- Confusing documentation
- Difficult migration paths

Where to Share

1. **GitHub Issues**: For bugs and specific problems
   - https://github.com/wp-graphql/wp-graphql/issues
2. **Slack**: For quick questions and discussions
   - https://wpgraphql.com/community
3. **Social Media**: Share wins and experiences
   - Tag @wpgraphql on Twitter/X
4. **GitHub Issues**: Best for detailed feedback and use cases
   - https://github.com/wp-graphql/wp-graphql/issues

What Happens When Experiments Graduate

When an experiment graduates to stable:

1. **It becomes always-on**: You can't disable it anymore (it's part of core)
2. **The experiment setting is deprecated**: Shows a notice in admin
3. **Breaking changes stop**: Feature now follows semantic versioning
4. **Documentation moves**: From experiments to main documentation
5. **Long-term support begins**: Feature is maintained like any core feature

Example notice you might see:

> ⚠ This experiment has graduated to core and is now enabled by default. This setting will be removed in the next major release.

What Happens When Experiments Are Deprecated

When an experiment is marked for deprecation:

1. **Deprecation notice appears**: In admin and potentially in GraphQL responses
2. **Documentation updated**: Shows deprecation status and migration path
3. **Grace period begins**: At least one major release before removal
4. **Migration guide published**: Instructions for adapting your code
5. **Community notification**: Announced in release notes and issues

Example deprecation notice:

> ⚠ This experiment is deprecated and will be removed in v3.0.0. Please migrate to [alternative solution]. [Read migration guide →]

Next Steps

- Creating Experiments - Learn how to build experiments
- Contributing Experiments - Contribute to WPGraphQL core
- What are Experiments? - Back to overview

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