Comprehensive Guide to SwupJS Integration for WordPress/Bricks Builder

Technical Background

What is Swup?

Swup is a JavaScript library that transforms traditional multi-page websites into Single Page Application (SPA)-like experiences. Instead of performing full browser page reloads when clicking links, Swup:

1. **Intercepts link clicks** - Captures navigation events before the browser handles them
2. **Fetches pages via AJAX** - Loads new page content in the background using the Fetch API
3. **Replaces content dynamically** - Swaps only the designated content containers, preserving the rest of the DOM
4. **Animates transitions** - Applies CSS classes during the transition for smooth visual effects
5. **Manages browser history** - Uses the History API to maintain proper URL states and back/forward navigation

Why Use Swup with Bricks Builder?

Bricks Builder generates standard WordPress pages with full HTML documents. Without Swup, each page navigation:

- Triggers a complete browser reload
- Re-parses all HTML, CSS, and JavaScript
- Reinitializes all scripts and components
- Creates a visible "flash" between pages

With Swup, your Bricks-built site gains:

- **Perceived performance improvement** - Pages appear to load instantly
- **Smooth visual transitions** - Professional fade, slide, or scale effects
- **Reduced server load** - Cached pages load from memory
- **Better user experience** - App-like navigation feel

How the Script Works

The swup-bricks-integration.js file is a self-contained integration that:

1. **Loads Swup and plugins from CDN** - No npm installation required
2. **Injects transition CSS automatically** - Styles are added to the document head
3. **Configures all options in one place** - The SWUP_CONFIG object centralizes settings
4. **Handles WordPress/Bricks specifics** - Excludes admin links, handles common plugins
5. **Provides reinitalization hooks** - The reinitializeScripts function handles third-party scripts

---

Installation Instructions

Method 1: Bricks Settings (Recommended)

1. **Access Bricks Settings**
   - Navigate to WordPress Admin → Bricks → Settings
   - Go to the "Custom Code" tab
2. **Add the Script**
   - Locate "Body (footer) scripts" section
   - Paste the entire contents of swup-bricks-integration.js wrapped in `<script>` tags:

```
<script>
// Paste the entire script content here
</script>
```

3. **Save Settings**
   - Click "Save Settings" at the bottom of the page

Method 2: Code Element in Template

1. **Edit Your Header/Footer Template**
   - Open Bricks Builder on your header or footer template
   - Add a "Code" element at the end of the template
2. **Configure the Code Element**
   - Set the rendering to "Execute Code"
   - Paste the script with `<script>` tags
3. **Save and Publish**
   - Save the template and publish changes

Method 3: Child Theme (Advanced)

1. **Create or Edit functions.php**

```
function enqueue_swup_integration() {
    wp_enqueue_script(
        'swup-bricks',
        get_stylesheet_directory_uri() . '/js/swup-bricks-integration.js',
        array(),
        '1.0.0',
        true // Load in footer
    );
}
add_action('wp_enqueue_scripts', 'enqueue_swup_integration');
```

2. **Place the Script File**
   - Create a `js` folder in your child theme
   - Save the script as `swup-bricks-integration.js`

Configuration Guide

Essential Settings

Content Container

The most critical setting is containers. This defines which parts of the page Swup will replace:

```
containers: ['#brx-content'],
```

Technical explanation: Swup finds elements matching these selectors on both the current page and the fetched page, then replaces the current element's innerHTML with the new content. The default `#brx-content` is Bricks' main content wrapper.

Important requirements:

- The container must exist on **every page** of your site
- The selector must match **exactly one element** per page
- The container should be **inside** `<body>`

Common Bricks containers:

- `#brx-content` - Main content area (default)
- `#brx-header` - If your header changes between pages
- `.custom-sidebar` - If you have a dynamic sidebar

Animation Selector

```
animationSelector: '[class*="transition-"]',
```

Technical explanation: This CSS attribute selector matches any element with a class containing "transition-". Swup monitors these elements for CSS `transitionend` and `animationend` events to know when animations complete.

How to use:

1. Add `transition-fade`, `transition-slide`, or `transition-scale` class to your content wrapper in Bricks
2. The injected CSS handles the actual animations
3. Swup waits for these elements to finish animating before showing new content

Animation Settings

Native View Transitions

```
nativeViewTransitions: false,
```

Technical explanation: The View Transitions API is a new browser feature (Chrome 111+, Edge 111+) that provides hardware-accelerated transitions between page states. When enabled:

- Browser creates a snapshot of the current page
- New content is rendered off-screen
- Browser animates between snapshots using GPU acceleration

When to enable: Only if your target audience primarily uses Chrome/Edge and you want smoother animations.

Animation Scope

```
animationScope: 'html',
```

Options:

- `'html'` - Animation classes (`is-changing`, `is-animating`, etc.) are applied to `<html>` element
- `'containers'` - Classes are applied directly to container elements

Recommendation: Keep as `'html'` for easier CSS targeting with selectors like `html.is-animating .transition-fade`.

Cache Configuration

Cache TTL (Time-To-Live)

```
cacheTTL: 5 * 60 * 1000, // 5 minutes in milliseconds
```

Technical explanation: Swup stores fetched page HTML in a JavaScript Map object in memory. The TTL determines how long cached pages remain valid. After expiration, Swup fetches a fresh copy.

Considerations:

- **Dynamic content sites**: Lower TTL (1-5 minutes) ensures content freshness
- **Static content sites**: Higher TTL (30-60 minutes) improves performance
- **Real-time content**: Set to `0` to disable caching entirely

Maximum Cache Entries

```
maxCacheEntries: 20,
```

Technical explanation: Limits memory usage by removing oldest cached pages when the limit is exceeded. This is implemented via the cache:set hook that prunes excess entries.

Preload Settings

```
preloadEnabled: true,
preloadOnHover: true,
preloadVisibleLinks: false,
preloadThrottle: 3,
```

How preloading works:

1. **Hover preload**: When user hovers over a link, Swup immediately fetches that page
2. **Visible links**: Intersection Observer API detects links entering the viewport
3. **Throttle**: Limits concurrent fetch requests to prevent server overload

Technical benefit: By the time the user clicks, the page is already cached, resulting in instant navigation.

WordPress-Specific Settings

Ignore Selectors

```
ignoreSelectors: [
  '[data-no-swup]',
  '#wpadminbar a',
  'a[href*="wp-admin"]',
  'a[href*="wp-login"]',
  'a[href*="/cart"]',
  // ... more
],
```

Technical explanation: The buildIgnoreVisit function creates a callback that checks every link click against these selectors. Matching links trigger normal browser navigation instead of Swup transitions.

Why ignore these:

- **Admin bar**: WordPress admin links should always perform full page loads
- **WooCommerce cart/checkout**: These pages often require full reloads for session management
- **Login pages**: Authentication typically requires traditional form submission

---

Adding Transitions to Your Bricks Site

Step 1: Add Transition Class to Content

1. **Open Bricks Builder** on your main template (usually "Single" or "Archive")
2. **Select your main content wrapper** (the element containing page content)
3. **Go to Style → CSS Classes**
4. **Add class**: `transition-fade`

Alternatively, target the default Bricks content wrapper in your CSS:

```
#brx-content {
  /* This becomes the animated element */
}
```

And modify the script's animationSelector:

```
animationSelector: '#brx-content',
```

Step 2: Customize Transition Styles

The script injects default styles via the TRANSITION_CSS constant. You can override these in Bricks' custom CSS:

```
/* Slower fade transition */
html.is-changing .transition-fade {
  transition: opacity 0.5s ease-in-out;
}

/* Custom slide direction */
html.is-leaving .transition-slide {
  transform: translateX(-100%);
}

html.is-rendering .transition-slide {
```

```
  transform: translateX(100%);
}
```

Step 3: Test the Transitions

1. **View your site** (not in Bricks editor)
2. **Open browser DevTools** → Console
3. **Enable debug mode** in the script:

   ```
   debug: true,
   ```

4. **Click links** and observe console logs showing transition lifecycle

---

Handling Third-Party Scripts

The Reinitialization Problem

Technical background: When Swup replaces content, it injects new HTML but doesn't re-execute JavaScript. Any scripts that:

- Query the DOM for elements
- Attach event listeners
- Initialize components (sliders, lightboxes, etc.)

...will not work on the new content because they only ran once on initial page load.

Solution: The reinitializeScripts Function

The reinitializeScripts function is called after every page transition:

```
function reinitializeScripts() {
  // Bricks frontend JS
  if (typeof bricksInit === 'function') {
    bricksInit();
  }

  // Custom event for your scripts
```

```
  document.dispatchEvent(new CustomEvent('swup:contentReplaced', {
    detail: {swup: window.swup}
  }));

  // Third-party reinitializations...
}
```

Adding Your Own Reinitializations

Method 1: Modify the Function

Add your code directly to reinitializeScripts:

```
function reinitializeScripts() {
  // ... existing code ...

  // Your custom initialization
  if (typeof MyCustomSlider !== 'undefined') {
    MyCustomSlider.init();
  }
}
```

Method 2: Listen for Custom Event

In a separate script:

```
document.addEventListener('swup:contentReplaced', function(event) {
  // Reinitialize your scripts here
  initializeMyComponents();
});
```

Method 3: Use the onPageView Callback

In the SWUP_CONFIG object:

```
onPageView: function(visit) {
  // Called after every page transition
  console.log('Navigated to:', visit.to.url);
```

```
  // Reinitialize components
  initializeLightboxes();
  initializeSliders();
}
```

Analytics Integration

Google Analytics 4

The script includes GA4 tracking in onPageView:

```
if (typeof gtag !== 'undefined') {
  gtag('config', 'GA_MEASUREMENT_ID', {
    page_path: visit.to.url
  });
}
```

Important: Replace `GA_MEASUREMENT_ID` with your actual measurement ID (e.g., `G-XXXXXXXXXX`).

Matomo/Piwik

Also included:

```
if (typeof _paq !== 'undefined') {
  _paq.push(['setCustomUrl', visit.to.url]);
  _paq.push(['trackPageView']);
}
```

Other Analytics Platforms

Add similar code to onPageView:

```
// Facebook Pixel
if (typeof fbq !== 'undefined') {
  fbq('track', 'PageView');
```

```
}

// Plausible
if (typeof plausible !== 'undefined') {
  plausible('pageview');
}
```

Troubleshooting

Issue: Hard page reload instead of transition

Causes:

1. **Container not found**: Ensure `#brx-content` exists on all pages
2. **Link is ignored**: Check if the link matches any ignoreSelectors
3. **External link**: Swup only handles same-origin links

Debug steps:

1. Enable `debug: true`
2. Check console for Swup logs
3. Verify container exists: `document.querySelector('#brx-content')`

Issue: Animations not working

Causes:

1. **Missing transition class**: Add `transition-fade` to content wrapper
2. **CSS not injected**: Check if `#swup-transitions` style element exists in `<head>`
3. **Animation selector mismatch**: Ensure your class matches `animationSelector` pattern

Debug steps:

1. Inspect `<html>` element during navigation for `is-changing`, `is-animating` classes
2. Check computed styles on transition elements

Issue: Scripts not running after navigation

Cause: JavaScript only executes on initial page load.

Solution: Add reinitialization code to reinitializeScripts or listen for `swup:contentReplaced` event.

Issue: Forms not submitting properly

Cause: Swup doesn't handle form submissions by default.

Solutions:

1. Add `data-no-swup` to the form's submit link/button
2. Install the Swup Forms Plugin (add to CDN URLs and enable)

Issue: Scroll position issues with fixed header

Solution: Adjust scrollOffset:

```
scrollOffset: 80, // Height of your fixed header in pixels
```

Performance Optimization

Recommended Production Settings

```
const SWUP_CONFIG = {
  // Disable debug logging
  debug: false,

  // Optimize caching
  cacheEnabled: true,
  maxCacheEntries: 30,
  cacheTTL: 10 * 60 * 1000, // 10 minutes
```

```
  // Enable preloading
  preloadEnabled: true,
  preloadOnHover: true,
  preloadThrottle: 5,

  // Disable unused plugins
  scriptsEnabled: false, // Only enable if absolutely needed
  progressBarEnabled: true,

  // Enable native transitions for supported browsers
  nativeViewTransitions: true,
};
```

Measuring Impact

Use browser DevTools to compare:

1. **Network waterfall**: Compare full page load vs. Swup navigation
2. **Performance timeline**: Measure time-to-interactive
3. **Core Web Vitals**: Monitor LCP, FID, CLS before/after implementation

---

Advanced Customization

Custom Animation per Link

Use `data-swup-animation` attribute:

```
<a href="/gallery" data-swup-animation="slide">View Gallery</a>
```

This adds `to-slide` class to `<html>` during transition. Create corresponding CSS:

```
html.is-changing.to-slide .transition-fade {
  transition: transform 0.4s ease;
}
```

```
html.is-leaving.to-slide .transition-fade {
  transform: translateX(-100%);
}

html.is-rendering.to-slide .transition-fade {
  transform: translateX(100%);
}
```

Programmatic Navigation

Access the Swup instance globally:

```
// Navigate to a page
window.swup.navigate('/about');

// Navigate without animation
window.swup.navigate('/contact', { animate: false });

// Clear cache
window.swup.cache.clear();
```

Excluding Specific Pages

Dynamically ignore pages based on URL:

```
ignoreVisit: function(url, { el } = {}) {
  // Ignore checkout pages
  if (url.includes('/checkout/')) return true;

  // Ignore specific page
  if (url === '/special-page/') return true;

  // Check element against ignore selectors (default behavior)
  const { ignoreSelectors } = SWUP_CONFIG;
  if (el) {
    for (const selector of ignoreSelectors) {
      if (el.closest(selector)) return true;
    }
```

```
    }

    return false;
}
```

This guide covers the complete setup and customization of the SwupJS integration for Bricks Builder. For further questions or issues, refer to the official Swup documentation or examine the detailed comments within the swup-bricks-integration.js file.

Revision #1
Created 2026-06-21 05:01:16 UTC by art10m
Updated 2026-06-21 05:01:53 UTC by art10m