Pulse Preview – Basic Interface Technical Guide

This document dives into the front-end that powers `preview/basic/index.html`. It explains layout structure, key modules, state orchestration, and how the preview drives the live Pulse survey tag. The goal is to give engineers enough depth to extend or troubleshoot the experience quickly.

---

1. Entry Point Overview

`preview/basic/index.html` loads a single module script (`preview/basic/preview.js`). All runtime behaviour branches out from this file, which:

- wires the control rail UI (survey controls, theme generator, examples, triggers, tools),
- manages asynchronous data sources (Google Sheet demo surveys, generated themes, example themes),
- bootstraps the live Pulse tag,
- tracks "applied CSS" so operators can inspect/remove theme injections, and
- exposes a console-style log panel for diagnostics.

Supporting modules include:

Module	Responsibility
`preview/basic/theme-generator-client.js`	Generates theme variants from an arbitrary URL (colour analysis + Sass compilation).
`preview/basic/theme-css.js`	Small CSS token helpers, transformation utilities used during generation.
`preview/basic/preview.js`	Main controller handling UI wiring, state, and tag integration.

The page stylesheet (`preview/styles/generator/basic.css`) defines the dark UI shown in the screenshot.

---

2. Layout & DOM Contract

Control Rail (`<aside id="control-rail">`)

Sections are organised as accordion items. `preview.js` treats them as static DOM handles; no framework is used.

1. **Survey**
   - `#survey-select`: populated from the Google Sheet feed. Holds the survey ID.
   - `#present-btn`: triggers a forced presentation.
   - `#applied-css-list`: renders currently-applied stylesheets (Pulse default, example themes, generated themes, manual CSS).
2. **Theme Generator**
   - `#theme-url-input`: entry field for the site URL to scan.
   - `#generate-theme-btn`: kicks off analysis via `generateThemesFromUrl`.
   - `#generated-theme-section` + `#generated-theme-select`: hidden until themes are generated.
3. **Examples**
   - `#industry-select`: filters down example themes.
   - `#theme-select`: holds the (manifest-driven) curated themes and the "no override" placeholder.
4. **Triggers**
   - `#trigger-buttons`: container for buttons built from `TRIGGER_CONFIG`.

5. **Tools**
   - `#tag-status`, `#survey-status`: badge indicators driven by runtime events.
   - `#toggle-log-btn`: opens the side log panel (`#log-panel`).

Stage (`<section id="site-root">`)

The preview injects a live site (or fallback host) into `#site-root` via `loadHostPage`. Surveys render on top of that DOM as the Pulse tag injects widgets.

Log Panel (`<aside id="log-panel">`)

`#log-list` is populated with timestamped entries. Visibility toggles between `#toggle-log-btn` and `#close-log-btn`.

---

3. Application State & Boot Sequence

`preview/basic/preview.js` starts by declaring numerous DOM references and state holders. Key structures:

- **DOM handles** (`const surveySelect = document.getElementById('survey-select')`, etc.) for quick access.
- **Configuration constants**: Google Sheet URL (`SHEET_CSV_URL`), Lipsum default host, etc.
- **Runtime state flags**:
  - `loadedAssets` (`Set`) to avoid double-injecting styles/scripts,
  - `tagReady`, `pendingSurveyId`, `lastPresentedId` for tag lifecycle,
  - `surveyRecords`, `generatedThemes`, `generatedThemeAssets` for data caches,
  - `stylesheetRegistry` (`Map`) storing metadata about every linked stylesheet (source, label, removable flag).

`init()` Sequence

1. Log “Initializing basic preview…”.
2. Call `populateSurveySelect()`:
   - fetches the Google Sheet CSV, parses it, filters for demos that match `?demo=` query params.
   - populates the select and returns initial survey metadata (ID, background URL, identifier).
3. Apply the identifier via `applyIdentifier`.
4. Load the host page (`loadHostPage`) – either the CSV-provided background URL or a Lipsum default.
5. Initialise theme select UI (`initializeThemeSelect`), which:
   - populates base options (“No theme override” placeholder),
   - loads the manifest of curated themes (`preview/styles/examples/generated/themes.json`),
   - populates the industry filter and theme dropdown, but does **not** auto-apply a theme.
6. Boot the Pulse tag (`bootPulseTag`).
7. Wire UI listeners (`wireUi`).
8. Set initial control/log states (`setRailOpen(false)`, `setLogVisibility(false)`).
9. Automatically present the first survey if we have a valid ID and host page.

---

4. Survey Lifecycle

Loading Available Surveys

`populateSurveySelect` pulls down Google Sheet data, normalises each row into a survey record (`surveyId`, `identifier`, `inlineTargetSelector`, etc.), and stores the filtered list in `surveyRecords`. The dropdown is replaced entirely, ensuring stale options are removed.

If the dataset is empty, we fall back to the default hard-coded option bundled in the HTML.

Presenting a Survey

- `presentSurvey(id, { force })` is triggered by select change or “Present Survey” button.
- If the Pulse tag is not ready, it sets `pendingSurveyId` and waits for `tagReady`.
- When ready:
  - Deduplicates repeated `pi('present',…)` calls via `lastPresentedId`, unless `force` is supplied.
  - Writes the action to the log (`addLog`).
  - Invokes `window.pi('present', id)` to call the production tag.
- The `applyIdentifier` helper ensures `window.pi('identify', identifier)` runs whenever we switch accounts.

Tag Boot (Pulse Tag Integration)

`bootPulseTag` embeds scripts from `/preview/scripts/surveys-tag.js`, which in turn loads the official `surveys.js` tag. It monitors for tag readiness by polling `window.pi` and `window.PulseInsightsObject`. Relevant lifecycle steps:

1. Inject stub.
2. Wait for initialization in `waitForOfficialTag`.
3. Once ready:
   - apply any staged identifier,
   - replay pending survey presentations.
4. Update badges (`setTagStatus`, `setSurveyStatus`).

---

5. Styling & Theme Management

Tracking Applied CSS

`registerStylesheet` / `unregisterStylesheet` maintain `stylesheetRegistry`. Each entry records:

- source (`default`, `examples`, `generated`, `manual`),
- label (friendly name for the manifest),
- removable toggle (disables removal for base CSS like `/preview/styles/generator/basic.css`).

`updateCssManifest` rebuilds the `<ol id="applied-css-list">`, listing checkboxes that enable/disable each stylesheet by manipulating `<link>` `disabled` attribute.

Example Themes

`initializeThemeSelect` and `renderThemeOptions`:

- load curated themes via `fetchLatestThemes` (JSON manifest).
- Filter by industry.
- Provide a “No theme override” option plus generated options grouped by account.
- Themes are **not** auto-applied; the user must choose and press “Apply Theme”.

`applyThemeById` fetches the selected manifest entry, creates `<link id="preview-theme-css">`, and registers it as the active theme. Clearing the theme removes the link and reverts to widget defaults.

Generated Themes

- `generateThemesFromUrl` (in `theme-generator-client.js`) analyses the provided URL, determines colour palettes, compiles up to 4 variants via the theme generator runtime (`compileTheme`), and returns CSS strings.

- `handleGenerateTheme` drives the async workflow:
  i. Disable the button (`setGeneratingState(true)`).
  ii. Call the generator and store results in `generatedThemes` .
  iii. Populate the generated themes select (`populateGeneratedThemeSelect`).
  iv. Auto-apply the first variant (`applyGeneratedTheme`).
  v. Re-enable controls.
- Generated themes are turned into `Blob` URLs, cached within `generatedThemeAssets` so we can revoke them (`clearGeneratedThemeAssets`) when new themes are generated.

Manual CSS

Operators can paste any stylesheet URL into the “Applied CSS Files” section:

- `applyManualCss(url)` injects a `<link>` pointing to the provided URL.
- The manifest tracks the link so it can be toggled or removed later.

---

6. Trigger Simulation

`TRIGGER_CONFIG` defines synthetic events (`exit-intent` , `rage-click` , etc.). `renderTriggers` (module from `preview/app/ui/triggers.js`) maps these definitions to buttons. At runtime:

1. Clicking a trigger button calls `handleTrigger` .
2. `handleTrigger` logs the action and delegates to `sendTrigger` .
3. `sendTrigger` posts a `window.pi('command', ['eventName'])` depending on the mapping.

This allows sales/demo teams to showcase behaviour without editing console configuration.

---

7. Tools & Logging

- `addLog(message, level)` creates entries in `#log-list` with consistent prefixes (`[preview]`).
- `setRailOpen` and `setLogVisibility` toggle CSS classes to show/hide the control rail and log panel.
- Status badges (`#tag-status` , `#survey-status`) use simple `textContent` updates for real-time feedback.

---

8. External Data & Build Inputs

Source	Usage
Google Sheet CSV (`SHEET_CSV_URL`)	Production demo survey metadata, including IDs, inline selectors, and background hosts.
`preview/styles/examples/generated/themes.json`	Output of the nightly theme export pipeline – curated Example themes.
`theme-generator/output/...`	Default theme manifest and client theme CSS; exposed through `/preview/styles/...` paths.
`/preview/scripts/surveys-tag.js`	Loader shim that keeps the preview’s tagging sequence close to production.

No backend APIs are required; everything is fetched over HTTP from the static server.

---

9. Extensibility Notes

- **Adding Triggers**: Update `TRIGGER_CONFIG` (UI) and extend the `sendTrigger` mapping inside `preview.js`.
- **Supporting New Data Feeds**: `populateSurveySelect` and `fetchSurveySheet` assume CSV structure—extend index mapping or provide an alternate loader.
- **Theme Variants**: `handleGenerateTheme` always expects four variants (if available). Swap or add strategies in `SimpleThemeGenerator` within `theme-generator-client.js`.
- **Tag Diagnostics**: The stub script logs readiness; insert further listeners on `window.PulseInsightsObject` or `window.addEventListener('pulseinsights:ready', …)` as needed.

---

10. Known Constraints

- The UI assumes a Chromium browser when using advanced features like File System Access (used by the newer "Manage Themes" overlay in `preview/index.html`; the Basic preview does not surface it).
- If the Google Sheet is unreachable, the preview falls back to bundled options and logs warnings.
- Generated theme assets are stored in-memory; refreshing the page clears them.
- The "Applied CSS Files" manifest does not diff file content—it only tracks `href` and DOM presence.

---

11. Related Assets

- **Scripts & Services**
  - `preview/scripts/build-demo-data.js`: builds JSON fallback for demos.
  - `scripts/launch-preview.sh`: orchestrates the local server and rebuild pipeline.
- **Styling**
  - `preview/styles/generator/basic.css`: main UI styling.
  - `preview/dist/default.css`: default Pulse theme (used elsewhere, not auto-applied in basic preview).

---

12. Summary Checklist

When debugging or extending the Basic preview, verify:

1. Google Sheet feed reachable (`populateSurveySelect` success).
2. Background host loads (CORS can block some domains).
3. Pulse tag ready (`tagReady === true`) before presenting.
4. `stylesheetRegistry` shows the expected links.
5. Generated themes are revoked when regenerating (check console for revoked Blob URLs).
6. Trigger buttons send the intended `pi` commands (observe network or `window.pi.q`).

Armed with the above, you should be able to modify the previews, add new demos, or debug Pulse tag behaviour with confidence.