Gatsby provides a rich set of lifecycle APIs to hook into its bootstrap, build, and client runtime operations.

Gatsby's design principles include:

- Conventions > code, but use low-level primitives to build conventions with code.
- Extracting logic and configuration into plugins should be trivial and encouraged.
- Plugins are easy to open source and reuse. They're just npm packages.

High level Overview

The following model gives a conceptual overview of how data is sourced and transformed in the process of building a Gatsby site:

Bootstrap sequence

During the main bootstrap sequence, Gatsby (in this order):

- reads and validates `gatsby-config.js` to load in your list of plugins (it doesn't run them yet).
- deletes HTML and CSS files from previous builds (public folder)
- initializes its cache (stored in `/.cache`) and checks if any plugins have been updated since the last run, if so it deletes the cache
- sets up `gatsby-browser` and `gatsby-ssr` for plugins that have them
- starts main bootstrap process
    - runs onPreBootstrap. e.g. implemented by `gatsby-plugin-typography`
- runs sourceNodes e.g. implemented by `gatsby-source-wikipedia`
    - within this, `createNode` can be called multiple times, which then triggers onCreateNode
- creates initial GraphQL schema
- runs resolvableExtensions which lets plugins register file types or extensions e.g. `gatsby-plugin-typescript`
- runs createPages from the gatsby-node.js in the root directory of the project e.g. implemented by `page-hot-reloader`
    - within this, `createPage` can be called any number of times, which then triggers onCreatePage
- runs createPagesStatefully
- runs source nodes again and updates the GraphQL schema to include pages this time
- runs onPreExtractQueries e.g. implemented by `gatsby-transformer-sharp` and `gatsby-source-contentful` , and extracts queries from pages and components (`StaticQuery`)
- compiles GraphQL queries and creates the Abstract Syntax Tree (AST)
- runs query validation based on schema
- executes queries and stores their respective results
- writes page redirects (if any) to `.cache/redirects.json`
- the onPostBootstrap lifecycle is executed

In development this is a running process powered by webpack and `react-refresh`), so changes to any files get re-run through the sequence again, with smart cache invalidation. For example, `gatsby-source-filesystem` watches files for changes, and each change triggers re-running queries. Other plugins may also perform this service. Queries are also watched, so if you modify a query, your development app is hot reloaded.

The core of the bootstrap process is the "api-runner", which helps to execute APIs in sequence, with state managed in Redux. Gatsby exposes a number of lifecycle APIs which can either be implemented by you (or any of your configured plugins) in `gatsby-node.js` , `gatsby-browser.js` or `gatsby-ssr.js` .

Build sequence

(to be written)

Client sequence

(to be written)

Please see the links along the left under "REFERENCE" for the full API documentation.