Curious how Gatsby works under the hood? The pages in this section describe how a Gatsby build works from an internal code/architecture point of view. It should be useful for anyone who needs to work on the internals of Gatsby, or for those who are simply curious how it all works, or perhaps you're a plugin author and need to understand how core works to track down a bug? Come one, come all!

If you're looking for information on how to *use* Gatsby to write your own site, or create a plugin, check out the rest of the Gatsby docs. This section is quite low level.

These docs aren't supposed to be definitive, or tell you everything there is to know. But as you're exploring the Gatsby codebase, you might find yourself wondering what a concept means, or which part of the codebase implements a particular idea. These docs aim to answer those kinds of questions.

A few more things. These docs are mostly focused on the build lifecycle (`gatsby build`). Operations specific to the develop lifecycle (`gatsby develop`) are mostly ignored. Though this may change in the future. Also, they mostly focus on the happy path, rather than getting bogged down in details of error handling.

The graph below shows roughly how all the sub systems of Gatsby fit together and the input/output artifacts at different parts of the build. To find out how different parts work, click on the nodes in the graph, or use the menu on the left.

> *Note: This diagram isn't up to date with the latest version of Gatsby. You can help by making a PR to update this documentation*

```
digraph graphname {

  node [ style = filled, fillcolor = white ];

  ## Legend

  subgraph cluster_legend {
    label = "Legend";
    gatsby [ label = "Gatsby", width=1 ];
    redux [ label = "redux namespace", shape = box, fillcolor = skyblue, width=1 ];
    cache [ label = "site/.cache/", shape = cylinder, fillcolor = moccasin, width=1
];
    public [ label ="site/public/", shape = cylinder, fillcolor = palegreen, width=1
];
    siteData [ label = "site/external data", shape = cylinder, fillcolor = gray,
width=1 ];

    siteData -> gatsby [ style = invis ];
    gatsby -> redux [ style = invis ] ;
    redux -> cache [ style = invis ];
    cache -> public [ style = invis ];
  }

  ## Source Nodes

  dataSource [ label = "data sources. e.g. file, contentful", shape = cylinder,
fillcolor = gray ];
  sourceNodes [ label = "source nodes" URL = "/docs/node-creation/" ];
  nodes [ label = "nodes", shape = box, fillcolor = skyblue, URL = "/docs/node-
creation/" ];
```

```
  nodesTouched [ label = "touchedNodes", shape = box, fillcolor = skyblue, URL =
"/docs/node-creation/#freshstale-nodes" ];
  rootNodeMap [ label = "rootNodeMap", shape = box, fillcolor = skyblue, URL =
"/docs/node-tracking/" ];

  dataSource -> sourceNodes;
  sourceNodes -> nodes;
  sourceNodes -> nodesTouched;
  sourceNodes -> rootNodeMap;

  ## Schema

  pluginResolvers [ label = "plugin resolvers", shape = cylinder, fillcolor = gray,
URL = "/docs/schema-input-gql/#inferring-input-filters-from-plugin-fields" ];
  generateSchema [ label = "generate schema", URL = "/docs/schema-generation/" ];
  schema [ label = "schema\l (inc resolvers)", shape = box, fillcolor = skyblue ];

  nodes -> generateSchema;
  nodes -> schema;
  pluginResolvers -> generateSchema;
  rootNodeMap -> generateSchema;
  generateSchema -> schema;

  ## Pages

  componentFiles [ label = "React components\l (src/template.js)", shape = cylinder,
fillcolor = gray ];
  createPages [ label = "site.createPages", URL = "/docs/page-creation/" ];
  pages [ label = "pages", shape = box, fillcolor = skyblue ];
  components [ label = "components", shape = box, fillcolor = skyblue ];

  schema -> createPages;
  componentFiles -> createPages;
  createPages -> pages;
  createPages -> components;

  ## Query

  fragments [ label = "query fragments *.js", shape = cylinder, fillcolor = gray ];
  runQueries [ label = "extract and run queries", URL = "/docs/query-behind-the-
scenes/" ];
  componentsWithQueries [ label = "components\l (with queries)", shape = box,
fillcolor = skyblue ];
  queryResults [ label = "JSON result\l /public/static/d/dataPath", shape =
cylinder, fillcolor = palegreen, URL = "/docs/query-execution/#save-query-results-
to-redux-and-disk" ];
  dataPaths [ label = "jsonDataPaths", shape = box, fillcolor = skyblue ];

  fragments -> runQueries;
  schema -> runQueries;
  pages -> runQueries;
  components -> runQueries;
```

```
  runQueries -> componentsWithQueries;
  runQueries -> queryResults;
  runQueries -> dataPaths;

  ## Write Pages

  writePages [ label = "writePages", URL = "/docs/write-pages/" ];
  dataJson [ label = "data.json", shape = cylinder, fillcolor = moccasin ];
  asyncRequires [ label = "async-requires.js", shape = cylinder, fillcolor =
moccasin ];
  syncRequires [ label = "sync-requires.js", shape = cylinder, fillcolor = moccasin
];
  pagesJson [ label = "pages.json", shape = cylinder, fillcolor = moccasin ];

  dataPaths -> writePages;
  components -> writePages;
  pages -> writePages;
  writePages -> dataJson;
  writePages -> asyncRequires;
  writePages -> syncRequires;
  writePages -> pagesJson;

  ## App.js

  appWebpack [ label = "configure webpack\l (`build-javascript`)", URL =
"/docs/production-app/#webpack-config" ];
  productionApp [ label = "production-app.js", shape = cylinder, fillcolor =
moccasin, URL = "/docs/production-app/#production-appjs" ];
  buildJavascript [ label = "build-javascript.js", URL = "/docs/production-app/" ];
  componentChunks [ label = "component chunks\l component---src-blog-[hash].js",
shape = cylinder, fillcolor = palegreen, URL = "/docs/how-code-splitting-works/" ];
  appChunk [ label = "app-[hash].js", shape = cylinder, fillcolor = palegreen ];
  webpackStats [ label = "webpack.stats.json", shape = cylinder, fillcolor =
palegreen, URL = "/docs/how-code-splitting-works/#webpackstatsjson" ];
  chunkMap [ label = "chunk-map.json", shape = cylinder, fillcolor = palegreen, URL
= "/docs/how-code-splitting-works/#chunk-mapjson" ];

  appWebpack -> buildJavascript;
  asyncRequires -> productionApp;
  dataJson -> productionApp;
  productionApp -> buildJavascript;
  buildJavascript -> componentChunks;
  buildJavascript -> appChunk;
  buildJavascript -> webpackStats;
  buildJavascript -> chunkMap;

  queryResults -> componentChunks;

  ## Generate html

  htmlWebpack [ label = "configure webpack\l (`build-html`)", URL = "/docs/html-
generation/#webpack" ];
```

```
  staticEntry [ label = "static-entry.js", shape = cylinder, fillcolor = moccasin,
URL = "/docs/html-generation/#static-entryjs" ];
  buildHtml [ label = "build-html.js", URL = "/docs/html-generation/" ];
  pageRenderer [ label = "page-renderer.js", shape = cylinder, fillcolor = palegreen
];
  htmlFiles [ label = "html files\l (index.html)", shape = cylinder, fillcolor =
palegreen ];

  htmlWebpack -> buildHtml;
  syncRequires -> staticEntry;
  dataJson -> staticEntry;
  webpackStats -> staticEntry;
  chunkMap -> staticEntry;
  staticEntry -> buildHtml;
  buildHtml -> pageRenderer;
  pages -> buildHtml;
  pageRenderer -> buildHtml;
  buildHtml -> htmlFiles;
}
```