sphinx-theme-builder

Pradyun Gedam

Sep 19, 2023

CONTENTS

Streamline the Sphinx theme development workflow, by building upon existing standardised tools.

- simplified packaging experience
- simplified JavaScript tooling setup
- development server, with rebuild-on-save and automagical browser reloading
- consistent repository structure across themes

CHAPTER

ONE

GETTING STARTED

Sphinx Theme Builder provides a streamlined workflow for developing Sphinx themes. This tutorial walks through the process of setting up a new Sphinx theme, making changes to it and generating PyPI distribution files for it.

This tutorial expects that the reader has working knowledge of:

- Terminal / Command Prompt
- Python virtual environments
- Web technologies (HTML/JS/CSS)
- Git / GitHub

1.1 Installation

As a first step, let's install this tool with the `cli` extra, in a clean virtual environment:

```
$ pip install "sphinx-theme-builder[cli]"
```

1.2 Create a new theme

To create a new theme, you can use the `stb new` command.

```
$ stb new my-awesome-sphinx-theme
```

Todo: Actually write the template that this uses.

You will be prompted to answer a few questions. Once they've been answered, this command will generate a scaffold for your theme in a folder named `my-awesome-sphinx-theme`.

For the rest of this tutorial, we're going to exclusively work in this directory, so it's sensible to `cd` into it.

```
$ cd my-awesome-sphinx-theme
```

1.3 Install the theme

To work with your theme, it is necessary to install it in the virtual environment. Let's do an editable install, since that's usually what you would want to do for development.

```
$ pip install -e .
```

Note: an editable install with sphinx-theme-builder as backend requires a modern version of pip (>= 21.3)

1.4 Start the development server

To start a development server, you use the `stb serve` command. It needs a path to a directory containing Sphinx documentation, so we'll use the demo documentation that comes as part of the default scaffold:

```
$ stb serve docs/
```

This command will do a few things (we'll get to details of this later) and, after a short delay, opens your default browser to view the built documentation. Keep this terminal open/running.

The development server simplifies the workflow for seeing how a change affects the generated documentation fairly straightforward – save changes to a file, switch to the browser and the browser will update to reflect those changes.

1.5 Making changes

Todo: This requires that the template in `stb new` works, and uses `sphinx-basic-ng`.

To try out how the development server handles changes, create a new `sections/article.html` file in the `src/{your_package_name}/theme/{your_theme_name}` with the following content:

```
{{ content }}
```

The server should do a few things and the page will automagically reload with the new page contents.

1.5.1 How it works

The development server listens for changes in your theme or the documentation (i.e. when a file is saved/renamed/moved). When it detects that a change has been made, the server will:

1. Recompile your theme's assets
2. Rebuild the Sphinx documentation it is serving
3. Automagically reload open browser tabs of HTML pages served by it

If the theme's asset compilation or the documentation build (with Sphinx) fails, the server will print something in the terminal window about the failure.

1.6 Stopping the server

To stop the server, focus on the terminal where the server is running and press `Ctrl+C`.

1.7 Packaging the theme

When you wish to publish your theme on PyPI, you will need to package your theme into a few distribution files and upload these files to PyPI. This makes it possible to install the package for your theme, which can be downloaded and installed with `pip`.

To generate the distribution files, run:

```
$ stb package
```

This will generate files in `dist/`, that contain the relevant distribution files for your project. These can be uploaded to PyPI using twine.

1.8 Next Steps

Go write a Sphinx theme!

CHAPTER

TWO

BUILD PROCESS

This document describes the build process of a theme, built using Sphinx Theme Builder (`stb`). In other words, this document serves as an elaborate answer to the question: “What happens when I run `stb compile` or `stb package`?”

2.1 Asset Generation

2.1.1 nodeenv creation

`stb` invokes nodeenv, in a subprocess and creates an isolated installation of NodeJS in a `.nodeenv` directory. If a `.nodeenv` directory already exists, this step is skipped.

2.1.2 nodeenv validation

Once `stb` has a nodeenv available, it will run `node --version` using the nodeenv’s `node` and validate that it matches the requirements of the theme.

This serves as a validation check to ensure that the user does not incorrectly use a broken nodeenv (in which case, `node` will fail to run) or a NodeJS version that is incompatible with the theme (which can happen in some situations). Even when the nodeenv was just created, this serves as a sanity-check that the nodeenv that has been created is indeed functional and valid.

2.1.3 `npm install`

Once the nodeenv is created, the JS dependencies of the theme are installed by running `npm install` using from the nodeenv. This will create a `node_modules` directory containing these dependencies.

If a `node_modules` directory already exists and is newer than the `package.json` file, this step is skipped.

2.1.4 `npm run-script build`

With all the NodeJS dependencies of the theme installed, the NodeJS-based build is performed by `npm` from the nodeenv. This build is done by using `npm run-script build` which looks at the `"build"` key in the `"scripts"` section of `package.json`.

For a package that uses Webpack for performing their JS build, this would mean that the `package.json` would look something like:

```
{
  "devDependencies": {
    "webpack": "...",
    "webpack-cli": "..."
  },
  "scripts": {
    "build": "webpack"
  }
}
```

This command is expected to generate the compiled assets for the theme. If the theme's `build` command is being executed, everything that `stb` does is working correctly.

2.2 Python Packaging stuff

`stb package` dispatches the task of performing the build to the build project, by invoking it in a subprocess. The build project orchestrates the details of the build process and ends up invoking `stb`'s build logic, in the appropriate manner.

The build project generates a source distribution, unpacks it and builds a wheel from the unpacked source distribution.

2.3 Source Distribution

For generating the source distribution, `stb` will generate a tarball that contains files from the `src/` directory of the project. Certain files are not included in this tarball, based on the following rules:

- Exclude hidden files.
- Exclude `.pyc` files.
- Exclude compiled assets.
- Exclude files that are excluded from version control (only git is supported).

2.4 Wheel Distribution

For generating the wheel distribution, `stb` will generate the theme's assets in production mode (using the *Asset Generation* process described above). Once this is generated, `stb` will generate a wheel file containing the package metadata and all the files in the `src/` directory.

Caution: It is expected that wheels would only be generated from unpacked source distributions. Attempting to generate a wheel from the source tree directly may result in incorrect contents in the wheel.

2.5 Appendix: Controlling NodeJS installation

nodeenv will prefer to use pre-built binaries, if they're available for the platform that the build is taking place on. If a pre-built binary is not available, it tries to build NodeJS from source on the machine.

By default, the pre-built binaries are fetched from:

- https://nodejs.org/download/release/ for supported platforms
- https://unofficial-builds.nodejs.org/download/release/ for musl-based platforms

It is possible to configure the behaviour of `nodeenv` using a `~/.nodeenvrc` file to change its behaviour, such as whether it uses a pre-built binary or what mirror it downloads from.

2.5.1 `STB_USE_SYSTEM_NODE`

When set to `true` or `1`, `stb` will ask nodeenv to use the `node` executable available on `PATH`, for creating the nodeenv.

This functionality is primarily for software redistributors who have policies against using prebuilt binaries from the NodeJS team, such as the ones that `nodeenv` tries to use by default.

2.5.2 `STB_RELAX_NODE_VERSION_CHECK`

When set to `true` or `1`, `stb` will *not* enforce that the NodeJS version in the `.nodeenv` directory exactly match the declared version in the theme. Instead, the check changes to ensuring that it has the same major version and an equal-or-higher minor version. The patch version is ignored.

This functionality is primarily for software redistributors who wish to use newer-but-still-compatible versions of the NodeJS.

CHAPTER

THREE

FILESYSTEM LAYOUT

Sphinx Theme Builder requires the themes to follow a fairly specific project layout and structure. This standard structure is what allows this tool to provide a sensible build pipeline, which in-turn enables the nice quality-of-life things like `stb serve`.

3.1 How it looks

```
- .gitignore # The theme should be under version control with git
- README.md
- LICENSE
- package.json # For Javascript-based build tooling.
- pyproject.toml # For Python package metadata and tooling.
- src:
    - my_amazing_theme: # The importable Python package (notice underscores)
        - __init__.py
        - [other Python files]
        - theme: # HTML templates
            - my-amazing-theme:
                - [various .html pages]
                - static - [any static assets that don't need to be compiled,
                  like images]
        - assets: # Static assets, SASS and JS.
            - [static assets that need to be compiled, possibly within folders]
            - styles:
                - index.sass # Compiled into the final CSS file.
                - [other Sass / SCSS files]
            - scripts:
                - my-amazing-theme.js # Compiled into the final JS file.
                - [other JS files]
```

3.2 Need for version control

Sphinx Theme Builder does not enforce the use of Git but, as part of the build process, it will exclude any files that are excluded from Git's tracking using any of the supported mechanisms (typically, a `.gitignore` file in the repository root). This information is queried as part of the source distribution generation process.

3.3 Auto-generated folders

The following folders will be auto-generated when the theme's assets are compiled. Add them to the project's `.gitignore`:

- `.nodeenv` - The NodeJS (+ `npm`) installation that is used to compile the theme's assets.
- `node_modules` - The NodeJS packages that are installed for use, to compile the theme's assets.
- `src/theme/<my-awesome-theme>/static/styles` - The compiled CSS assets for the theme
- `src/theme/<my-awesome-theme>/static/scripts` - The compiled JS assets for the theme

3.4 How to get this right

Nearly everything about the filesystem layout and the contents of the various configuration files are validated, as part of the build process. This means that you can get the layout and contents of the files correct by ensuring that the build of the theme succeeds, when using Sphinx Theme Builder.

This typical workflow for doing this looks something like:

1. Update the `build-backend` for the theme to Sphinx Theme Builder.

   Listing 1: pyproject.toml

   ```
   [build-system]
   requires = ["sphinx-theme-builder >= 0.2.0a14"]
   build-backend = "sphinx_theme_builder"
   ```

2. Run `stb package` in the same directory as the `pyproject.toml` file.
3. Fix the error presented.
4. Repeat the above two steps, that until the `stb package` command succeeds.
5. And.. done!

CHAPTER

FOUR

MANAGING THEME ASSETS

The `sphinx-theme-builder` intentionally separates asset source files from the compiled assets that are bundled with your theme. These generally fall into two folders:

- "Asset sources" in `src/my_awesome_theme/assets`

  The source files used for generating the theme's actual stylesheet/JavaScript files (e.g. `.scss` files, `.ts` files).

- "Compiled static assets" in `src/my_awesome_theme/theme/my-awesome-theme/static`

  These are the files that will be used by Sphinx (e.g. `.css` files generated from `.scss`, `.js` files after transpiling and bundling).

4.1 Compiling assets

As described in the *Build Process* document, themes are expected to compile assets through a NodeJS-based build system with `npm run-script build` being the entrypoint for it.

During theme development, it is expected that theme authors will not need to invoke npm manually. Instead, when you want to compile the theme's assets, you'll run:

```
$ stb compile
```

This will handle all the details of fetching NodeJS, the npm dependencies and other running the build via npm.

When previewing documentation using `stb serve`, this command will run prior to rebuilding the documentation. This means that your theme development workflow will not require manually invoking the rebuild. The live-reloading server ignores assets in the static folder, so generating files there does not trigger a re-build loop.

4.2 Example: Webpack-based asset generation

For an example, we'll use Furo's 2022.06.21 release. That version of the Furo theme uses a Webpack based asset generation pipeline, for compiling Sass and JS source assets into generated CSS and JS assets.

The relevant files for this setup are:

- https://github.com/pradyunsg/furo/blob/2022.06.21/pyproject.toml
  - Configures Sphinx Theme Builder
  - Declares Python metadata + dependencies
  - Notice `additional-compiled-static-assets` in there
- https://github.com/pradyunsg/furo/blob/2022.06.21/package.json

  - Declares JS build dependencies
  - Declares entrypoint for `npm run-script build` as `webpack`
- https://github.com/pradyunsg/furo/blob/2022.06.21/webpack.config.js
  - Configures Webpack to compile the assets into the relevant location
    * transpile Sass files into CSS and run PostCSS on it
    * bundle multiple JS files into a single JS file
- https://github.com/pradyunsg/furo/blob/2022.06.21/postcss.config.js
  - Configure PostCSS to run autoprefixer

CHAPTER

FIVE

CLI

Detailed information to the various CLI commands.

5.1 `stb compile`

This command compiles the current project's assets.

5.1.1 Options

`--production`

Runs the build with `NODE_ENV=production` (`development` by default).

5.2 `stb new`

This command uses cookiecutter under the hood, with a dedicated template, to help set up a new project.

This is based on `cookiecutter` which means it is also possible for theme authors to use tooling built for it – like cruft, to keep their theme up to date with the underlying template, as the template evolves.

5.2.1 Options

None.

5.3 `stb npm`

Interact with the npm, available in the environment.

This command requires the nodeenv to exist. Typically, you can create it by running the *stb compile* command successfully once.

5.3.1 Options

None.

5.4 `stb package`

Generate Python distribution files. (sdist and wheel)

This is done by running build in a subprocess.

5.4.1 Options

None.

5.5 `stb serve`

Serve the provided documentation path, with livereload on changes.

5.5.1 Usage

This start a long-running server with live-reload that watches for changes in the theme or documentation sources (using sphinx-autobuild).

When a change is made, it will rebuild the assets of the theme, rebuild the documentation using Sphinx and reload any open browser tabs that are viewing an HTML page served by the server.

5.5.2 Options

`--builder`

The Sphinx builder to build the documentation with.

Allowed values: `html` (default), `dirhtml`

`--host`

hostname to serve documentation on (default: 127.0.0.1)

`--port`

The port to start the server on. Uses a random free port by default.

Allowed values: INTEGER

`--pdb`

Run pdb if the Sphinx build fails with an exception.

`--open-browser / --no-open-browser`

Open the browser after starting live-reload server. This is done by default.

`--override-theme / --no-override-theme`

Override the `html_theme` value set in `conf.py`. This is not done by default.

CHAPTER

SIX

ERROR INDEX

Inspired by the Rust Compiler Error Index, this page describes the various errors that may be presented by `sphinx-theme-builder`, indicating known causes as well as potential solutions.

6.1 crash

There is an unexpected exception raised within `sphinx-theme-builder`. This is usually a symptom of an incorrect assumption/expectation or a bug in the implementation of `sphinx-theme-builder`.

What you can do: It is recommended to report this issue as a crash report, on the issue tracker. This error will print a detailed traceback above it, which should be included.

6.2 nodeenv-creation-failed

A NodeJS environment (nodeenv) could not be created, for some reason. Typically, the reason for the failure is indicated by the output above the error. This is an error from the underlying tooling that `sphinx-theme-builder` uses.

A `urllib.error.HTTPError` indicates that the issue is related to the network or the availability of NodeJS release files. It may mean the node version that this tool is trying to fetch is no longer available, for example if there is no compatible NodeJS binary for the operating system.

What you can do: When this error is encountered, a good place to look at is the nodeenv project's documentation and issue tracker.

6.3 non-boolean-env-variable-value

This error is raised when the user sets the a boolean environment variable to an invalid value. The valid values for boolean environment variables are `true`, `false`, `1` and `0`. They are case-insensitive. Providing any other value will cause this error to be raised.

What you can do: Provide a valid value, for the environment variable involved.

6.4 can-not-use-system-node-on-windows

This error is raised when the user tries to use the system NodeJS installation on Windows.

The underlying tooling that Sphinx Theme Builder uses (nodeenv) does not support using an existing system NodeJS installation for creating the nodeenv, so this is not permitted.

What you can do: Unset the `STB_USE_SYSTEM_NODE` environment variable, since it is not possible to use the system NodeJS installation on Windows.

6.5 js-build-failed

This error indicates that the build step using the Javascript tooling failed, which is a theme-specific issue. Typically, there is a JS error reported by the theme-specific tooling immediately above this error.

Sphinx Theme Builder executes `npm run-script build` for running the Javascript tooling. When that fails, this error is raised which means that something went wrong in the Javascript build pipeline of the theme that is being built/packaged.

What you can do: Look into the JS error reported, since that error is what needs to be looked at (and possibly, put into a search engine).

6.6 js-install-failed

This error indicates that the install step for the Javascript tooling failed, which is a theme-specific issue. Typically, there is a JS error reported by the theme-specific tooling immediately above this error.

Sphinx Theme Builder executes `npm install` for installing the Javascript tooling that the theme declares a dependency on. When that fails, this error is raised which means that something went wrong in the installation of the JS dependencies of the theme that is being built/packaged.

What you can do: Look into the JS error reported, since that error is what needs to be looked at (and possibly, put into a search engine).

6.7 nodeenv-unhealthy-npm-not-found

This error indicates that the nodeenv created for building this theme is broken, and does not have an `npm` executable. Typically, this is a symptom of an incomplete cleanup.

What you can do: Deleting the `.nodeenv` directory and trying again will usually resolve this issue.

If this happens while performing multiple builds of the same theme in parallel (which is not supported at this time), this is likely caused by a race condition due to the lack of support for this mode of usage. You'll need to ensure that no parallel builds are occurring in the same directory.

6.8 nodeenv-unhealthy-file-not-found

This error indicates that the nodeenv created for building this theme is broken, and does not contain a file that should have been there (the `node` executable). Typically, this is a symptom of an incomplete cleanup.

What you can do: Deleting the `.nodeenv` directory and trying again will usually resolve this issue.

If this happens while performing multiple builds of the same theme in parallel (which is not supported at this time), this is likely caused by a race condition due to the lack of support for this mode of usage. You'll need to ensure that no parallel builds are occurring in the same directory.

6.9 nodeenv-unhealthy-subprocess-failure

This error indicates that the nodeenv created for building this theme is broken, and can not be used.

Sphinx Theme Builder executes `node --version` using the NodeJS available in the nodeenv to determine what version is available in the nodeenv. This error indicates that this subprocess failed, which should never happen for a valid and functional nodeenv.

What you can do: This is not a typical failure and may be a symptom of a different underlying issue (incompatible architecture, broken OS libraries etc). It may be possible to work around this by deleting the `.nodeenv` directory and trying again.

6.10 nodeenv-version-mismatch

This error indicates that the nodeenv created for building this theme does not match the declared requirements of the theme, and can not be used. Typically, this is because the declared requirement of the theme has been changed.

What you can do: Deleting the `.nodeenv` directory and trying again will usually resolve this issue. If it does not, please see *Appendix: Controlling NodeJS installation*. If you're a redistributor of software (rather than a user or theme author), the aforementioned link should provide relevant information.

6.11 unable-to-cleanup-node-modules

This error indicates that the `node_modules` folder could not be deleted.

What you can do: This is not a typical failure and will need diagnosis of why the deletion might have failed (eg: permission issues, filesystem issues).

6.12 invalid-pyproject-toml

This error indicates that the theme does not have a valid `pyproject.toml` file in the root, due to the contents of the file. Typically, this is related to the the `project` table and mentions the problematic key.

What you can do: Resolve the issue as suggested by the error message. You can find more details by looking up documentation about the `[project]` table (eg: **PEP 621**).

6.13 pyproject-missing

This error indicates that theme does not have a `pyproject.toml` file in the root directory of the theme's package (typically, the repository root). It is required for using Sphinx Theme Builder.

What you can do: Create a `pyproject.toml` file and try again. You'll get a new error, which will help guide you forward.

6.14 pyproject-could-not-parse

This error indicates that the theme do not have a valid `pyproject.toml` file. It needs to be a valid TOML 1.0.0 file.

What you can do: Investigate why the file is not a valid TOML file and fix the issue.

6.15 pyproject-no-project-table

This error indicates that the `pyproject.toml` file in the theme does not contain the `[project]` table. That table provides required metadata and must be specified.

What you can do: Declare your project's metadata in the `[project]` table. See **PEP 621** for more details on the syntax.

6.16 pyproject-no-name-in-project-table

This error indicates that the `pyproject.toml` file in the theme does not contain the required `name` key in the `[project]` table.

What you can do: Declare your project's name in the `[project]` table.

6.17 pyproject-non-canonical-name

This error indicates that the `pyproject.toml` file in the theme has a `name` key that is not formatted in a canonical format. This needs to be a `dashed-name-with-only-lowercase-characters`.

What you can do: Fix your project's `name` in the `[project]` table.

6.18 pyproject-invalid-version

This error indicates that the `pyproject.toml` file in the theme has a `version` key that is not a string.

What you can do: Fix your project's `version` in the `[project]` table.

6.19 project-init-missing

This error indicates that the theme's `__init__.py` file could not be found.

What you can do: Move/create your theme's importable package in the location expected. See *Filesystem Layout* for more details.

6.20 project-init-invalid-syntax

This error indicates that the theme's `__init__.py` file is not a valid Python file. This file is parsed by Sphinx Theme Builder, to look up the Python version of that file.

What you can do: Fix the file to be valid Python code.

6.21 project-double-version-declaration

This error indicates that the theme has declared its version in both `__init__.py` file and the `pyproject.toml` file. This is not supported.

What you can do: Declare your theme's version in only one location. Typically, you can do this by removing the version declaration in the `pyproject.toml` file.

6.22 project-missing-dynamic-version

This error indicates that the theme's `__init__.py` file does not contain a version declaration in the form expected by Sphinx Theme Builder.

This needs to be a top-level assignment of the form `__version__ = <string>`.

What you can do: Declare your theme's version in the form expected.

6.23 project-improper-dynamic-version

This error indicates that the `pyproject.toml` file in the theme has a `version` key *and* declares that the version will be picked up from the `__init__.py` file (by including version in the `dynamic` key).

What you can do: Declare your theme's version in the form expected.

6.24 project-no-version-declaration

This error indicates that Sphinx Theme Builder could not locate a version declaration for this theme. Typically, this is because the version has not been declared in either of the `pyproject.toml` or `__init__.py` files.

What you can do: Declare your theme's version in the `pyproject.toml` file or in the theme's "base" `__init__.py` file.

6.25 project-invalid-version

This error indicates that the theme's declared version is not a valid Python Package version (see **PEP 440** for the specification).

What you can do: Fix your theme's version.

6.26 no-license-declared

This error indicates that the `pyproject.toml` file does not declare a license, which is not permitted when using Sphinx Theme Builder.

What you can do: Declare a license.

6.27 unsupported-dynamic-keys

This error indicates that there are unsupported values the `dynamic` key in `pyproject.toml` file.

What you can do: Remove the unsupported values.

6.28 no-node-version

This error indicates that no NodeJS version was configured in the `pyproject.toml` file.

What you can do: Configure the NodeJS version in the theme's `pyproject.toml` file.

6.29 node-version-mismatch

This error indicates that the NodeJS version in the nodeenv does not match what was configured in the `pyproject.toml` file.

What you can do: Typically, this is only seen when *the default NodeJS handling is overridden* and is a symptom of misconfiguration (either of sphinx-theme-builder or within the environment). Ensure that a matching NodeJS version for the project being build is available and, if so, delete the `.nodeenv` directory and try again.

If this happens while performing multiple builds of the same theme in parallel (which is not supported at this time), this is likely caused by a race condition due to the lack of support for this mode of usage. You'll need to ensure that no parallel builds are occurring in the same directory.

6.30 missing-theme-conf

This error indicates that the `theme.conf` file for the theme could not be located.

This is typically a genuinely missing file, a symptom of a misconfiguration or incorrect filesystem layout of the package. The misconfiguration is typically forgetting to set `theme-name` in `[tool.sphinx-theme-builder]` in pyproject.toml, when it does not match the PyPI package name.

What you can do: Ensure that your theme name is correct and, if so, move/create the `theme.conf` file, in the expected location. If you're creating this file, you can find more details in Sphinx's theme creation documentation.

6.31 missing-javascript

This error indicates that the Javascript file for the theme could not be located.

This is typically a genuinely missing file, a symptom of a misconfiguration or incorrect filesystem layout of the package. The misconfiguration is typically forgetting to set `theme-name` in `[tool.sphinx-theme-builder]` in pyproject.toml, when it does not match the PyPI package name.

What you can do: Ensure that your theme name is correct and, if so, ensure that the Javascript file is available, in the expected location.

6.32 missing-stylesheet

This error indicates that the CSS file for the theme could not be located.

This is typically a genuinely missing file, a symptom of a misconfiguration or incorrect filesystem layout of the package. The misconfiguration is typically forgetting to set `theme-name` in `[tool.sphinx-theme-builder]` in pyproject.toml, when it does not match the PyPI package name.

What you can do: Ensure that your theme name is correct and, if so, ensure that the CSS file is available in the expected location.

6.33 could-not-read-theme-conf

This error indicates that `theme.conf` could not be read.

Sphinx Theme Builder tries to read the `theme.conf` file of the theme being built, parse it and read certain configuration variables declared in it. This error is presented when that operation fails. More details are typically available from the "context" information provided in the error message (the line after "Could not open/parse).

What you can do: Ensure that this file can be pasted as an INI file.

6.34 theme-conf-incorrect-stylesheet

This error indicates that the style file declared in `theme.conf` does not match the stylesheet that was expected based on the theme name.

Sphinx Theme Builder enforces this consistency to make it easier to avoid collisions within derived themes and ensure that CSS file for the theme could not be located.

What you can do: Update the filename of the CSS file in `theme.conf`, to match what is expected. You may also need to update your build system to generate the file in the corresponding location.

6.35 missing-command-line-dependencies

This error indicates that one or more of the dependencies that are needed to use the `stb` command are not installed.

This is typically a symptom of doing `pip install sphinx-theme-builder` instead of `pip install "sphinx-theme-builder[cli]"`.

What you can do: Run `pip install "sphinx-theme-builder[cli]"` to install the missing dependencies.

6.36 can-not-overwrite-existing-python-project

This error is raised to prevent users from overwriting existing Python projects with `stb new`, since that command does not attempt to preserve any file and is not designed to be used on an existing Python project.

What you can do: Use a clean directory for running `stb new`.

6.37 cookiecutter-failed

This error indicates that `stb new` could not create the repository. This is currently expected as the repository that `stb new` tries to use is not set up.

What you can do: Manually create your theme's codebase, using existing themes like Furo or pydata-sphinx-theme as the base for that.

6.38 no-nodeenv

This error is raised by `stb npm` when the user tries to use it without creating the nodeenv for the project.

What you can do: Run `stb compile` once, to create the nodeenv.

6.39 autobuild-failed

This error is raised when `sphinx-autobuild` exits with an error, which is typically caused by a fatal error or interrupt.

What you can do: Investigate why the Sphinx build failed. Typically, the output for failure above this error will provide relevant information.

CHAPTER

SEVEN

DEVELOPMENT

Todo: Flesh this out.

CHAPTER

EIGHT

CHANGELOG

8.1 0.2.0b2

- Adopt the newer copy of `copyfileobj_with_hashing`
- Correctly encode `RECORD` hashes
- Document a previously undocumented error case
- Document the `--pdb` flag
- Improve documentation to pass nit-picks
- Improve the `autobuild-failed` documentation
- Use tomllib on Python 3.11+

8.2 0.2.0b1

- Add `--host` to `stb serve`.
- Document a theme asset management approach.
- Fix the generator value.
- Generate a `package-lock.json` file, if it does not exist.
- Switch to `pyproject-metadata` (from `pep621`).

8.3 0.2.0a15

- Add `--pdb` flag to `stb serve`.
- Accept more values for `STB_USE_SYSTEM_NODE`, error out on invalid ones.
- Add `STB_RELAX_NODE_VERSION_CHECK`.
- Fix typing-related import for Python 3.7 compatibility.
- Document all errors in the error index, describing what the user can do.
- Fix project source URL in metadata.
- Improve the getting started tutorial.
- Tweak how links are presented in errors.

8.4 0.2.0a14

- Don't pin the upper Python version.
- Present the traceback on Sphinx failures.
- Update error message for `nodeenv-creation-failed`
- Quote the `sys.executable`.
- Fix mis-formatted README opening.
- Back to development.

8.5 0.2.0a13

- Simplify system node usage logic.
- Use the correct binary directory on Windows.
- Reducing the size of the generated nodeenv.
- Add TODOs to the tutorial, to reflect it is incomplete.

8.6 0.2.0a12

- Fix Windows compatibility.

8.7 0.2.0a11

- Fix Python 3.7 compatibility.
- Fix handling of missing `node` executable on system.
- Explicitly declare the LICENSE.

8.8 0.2.0a10

- Fix improper RECORD file generation.

8.9 0.2.0a9

- Try to fix improper RECORD file generation.

8.10 0.2.0a8

- Add `stb compile --production`
- Improve documentation on what the project layout is.

8.11 0.2.0a7

- Allow setting alternative theme name.
- Enable users to specify custom “additional compiled static assets”.
- Present error when npm is not found.
- Present more context when deciding on using `system` nodeenv.
- Run `nodeenv` with rich traceback installed.
- Search `PATH` for executables to run in nodenv.
- Suppress exception stack from click.

8.12 0.2.0a6

- Include parent paths of compiled files, when computing files for the wheel archive.
- Fix release version management.

8.13 0.2.0a5

- Include setuptools as a dependency.

8.14 0.2.0a4

- Add `stb npm` command, to make it easier to run npm within the nodeenv.
- Properly handle `nodeenv` and CLI colours.
- Get `node-version` from project configuration.
- Use the `node` from PATH, if it matches the required version
- Handle aborts coming out of click.
- Handle unclean exits in `build`.

8.15 0.2.0a3

- Improve `stb serve`.
- Improve handling and presentation of errors from `main`.
- Run project structure validation in more situations.
- Consolidate compiled asset calculation.
- Add a direct dependency on `nodeenv`.

8.16 0.2.0a2

- Update the paths that source assets are stored in.
- Correctly handle `[project]` in the error output.

8.17 0.2.0a1

Initial release.

CHAPTER NINE

LICENSE

INDEX