Hawkmoth Documentation

Release 0.5

Jani Nikula

Jan 25, 2020

Contents

Hawkmoth is a minimalistic Sphinx `C Domain` autodoc directive extension to incorporate formatted C source code comments written in reStructuredText into Sphinx based documentation. It uses Clang Python Bindings for parsing, and generates C Domain directives for C API documentation, and more. In short, Hawkmoth is Sphinx Autodoc for C.

Hawkmoth aims to be a compelling alternative for documenting C projects using Sphinx, mainly through its simplicity of design, implementation and use.

Please see the Hawkmoth project GitHub page (or README.rst in the source repository) for information on how to obtain, install, and contribute to Hawkmoth, as well as how to contact the developers.

Read on for information about Hawkmoth usage; how to configure and use the extension and how to write documentation comments, with examples.

Contents:

CHAPTER 1

C Autodoc Extension

Hawkmoth provides a Sphinx extension that adds a new directive to the Sphinx `C domain` to incorporate formatted C source code comments into a document. Hawkmoth is Sphinx `sphinx.ext.autodoc` for C.

For this to work, the documentation comments must of course be written in correct reStructuredText. See *documentation comment syntax* for details.

1.1 Installation

Add `hawkmoth` to `extensions` in `conf.py`. Note that depending on the packaging and installation directory, this may require adjusting the `PYTHONPATH`.

1.2 Configuration

The extension has a few configuration options that can be set in `conf.py`:

cautodoc_root
: Path to the root of the source files. Defaults to the configuration directory, i.e. the directory containing `conf.py`.

  To use paths relative to the configuration directory, use `os.path.abspath()`, for example:

```
import os
cautodoc_root = os.path.abspath('my/sources/dir')
```

cautodoc_compat
: Compatibility option. One of `none` (default), `javadoc-basic`, `javadoc-liberal`, and `kernel-doc`. This can be used to perform a limited conversion of Javadoc-style tags to reStructuredText.

> **Warning:** The compatibility options and the subset of supported syntax elements are likely to change.

cautodoc_clang
A comma separated list of arguments to pass to `clang` while parsing the source, typically to define macros for conditional compilation, for example `-DHAWKMOTH`. No arguments are passed by default.

1.3 Directive

This module provides the following new directive:

`.. c:autodoc:: filename-pattern [...]`
Incorporate documentation comments from the files specified by the space separated list of filename patterns given as arguments. The patterns are interpreted relative to the *cautodoc_root* configuration option.

The `compat` option overrides the *cautodoc_compat* configuration option.

The `clang` option overrides the *cautodoc_clang* configuration option.

1.4 Examples

The basic usage is:

```
.. c:autodoc:: interface.h
```

Several files with compatibility and compiler options:

```
.. c:autodoc:: api/*.[ch] interface.h
   :compat: javadoc-basic
   :clang: -DHAWKMOTH
```

CHAPTER 2

Syntax

For the `c:autodoc` directive to work, the C source code must be documented using specific documentation comment style, and the comments must follow reStructuredText markup.

Optionally, there's limited support for some Javadoc and Doxygen style constructs for compatibility.

See *the examples section* for a quick tour of what's possible, and read on for documentation comment formatting details.

2.1 Documentation Comments

Documentation comments are C language block comments that begin with `/**`.

Because reStructuredText is sensitive about indentation, it's strongly recommended, even if not strictly required, to follow a uniform style for multi-line comments. Place the opening delimiter `/**` and closing delimiter `␣*/` on lines of their own, and prefix the lines in between with `␣*␣`. Indent the actual documentation at the third column, to let Hawkmoth consistently remove the enclosing comment markers:

```
/**
 * The quick brown fox jumps
 * over the lazy dog.
 */
```

One-line comments are fine too:

```
/** The quick brown fox jumps over the lazy dog. */
```

All documentation comments preceding C constructs are attached to them, and result in C Domain directives being added for them. This includes macros, functions, struct and union members, enumerations, etc.

Documentation comments followed by comments (documentation or not) are included as generic documentation.

2.2 Tags

Use reStructuredText field lists for documenting function parameters, return values, and arbitrary other data:

```
/**
 * The baznicator.
 *
 * :param foo: The Foo parameter.
 * :param bar: The Bar parameter.
 * :return: 0 on success, non-zero error code on error.
 * :since: v0.1
 */
int baz(int foo, int bar);
```

2.3 Compatibility

Hawkmoth supports limited Javadoc and Doxygen style constructs for compatibility. See the *extension documentation* on how to enable the support.

```
/**
 * The baznicator.
 *
 * @param foo The Foo parameter.
 * @param bar The Bar parameter.
 * @return 0 on success, non-zero error code on error.
 * @since v0.1
 */
int baz(int foo, int bar);
```

Warning: The compatibility support and its documentation are a work-in-progress.

2.4 Cross-Referencing C Constructs

Use `The C Domain` roles for cross-referencing as follows:

- `:c:data:`name`` for variables.
- `:c:func:`name`` for functions and function-like macros.
- `:c:macro:`name`` for simple macros and enumeration constants.
- `:c:type:`name`` for structs, unions, enums, and typedefs.
- `:c:member:`name.membername`` for struct and union members.

See the Sphinx `Basic Markup` and generic `Cross-referencing syntax` for further details on cross-referencing, and how to specify the default domain for brevity.

CHAPTER 3

Examples

This page showcases Hawkmoth in action.

Note: The documentation you are viewing was built without Hawkmoth and/or its dependencies (perhaps on https://readthedocs.org/). The output seen below was pre-generated statically using Hawkmoth, and should closely reflect actual results.

3.1 Macro

3.1.1 Source

```
/**
 * Failure status.
```

(continues on next page)

(continued from previous page)

```
 */
#define FAILURE 13

/**
 * Terminate immediately with failure status.
 *
 * See :c:macro:`FAILURE`.
 */
#define DIE() _exit(FAILURE)

/**
 * Get the number of elements in an array.
 *
 * :param array: An array
 * :return: Array size
 */
#define ARRAY_SIZE(array) (sizeof(array) / sizeof(array[0]))

/**
 * Variadic macros
 *
 * :param foo: regular argument
 * :param ...: variable argument
 */
#define VARIADIC_MACRO(foo, ...) (__VA_ARGS__)
```

3.1.2 Directive

```
.. c:autodoc:: examples/example-10-macro.c
```

3.1.3 Output

FAILURE
Failure status.

DIE()
Terminate immediately with failure status.

See *FAILURE*.

ARRAY_SIZE(array)
Get the number of elements in an array.

Parameters

- **array** – An array

Returns Array size

VARIADIC_MACRO(foo, ...)
Variadic macros

Parameters

- **foo** – regular argument
- **...** – variable argument

3.2 Variable

3.2.1 Source

```
/**
 * The name says it all.
 */
const int meaning_of_life = 42;

/**
 * The list of entries.
 *
 * Use :c:func:`frob` to frobnicate, always in :c:macro:`MODE_PRIMARY` mode.
 */
static struct list *entries;
```

3.2.2 Directive

```
.. c:autodoc:: examples/example-20-variable.c
```

3.2.3 Output

const int **meaning_of_life**
: The name says it all.

struct *list* * **entries**
: The list of entries.

  Use *frob()* to frobnicate, always in *MODE_PRIMARY* mode.

3.3 Typedef

3.3.1 Source

```
/**
 * Typedef documentation.
 */
typedef void * list_data_t;
```

3.3.2 Directive

```
.. c:autodoc:: examples/example-30-typedef.c
```

3.3.3 Output

list_data_t
: Typedef documentation.

3.4 Enum

3.4.1 Source

```
/**
 * Frobnication modes for :c:func:`frob`.
 */
enum mode {
        /**
         * The primary frobnication mode.
         */
        MODE_PRIMARY,
        /**
         * The secondary frobnication mode.
         */
        MODE_SECONDARY,
};
```

3.4.2 Directive

```
.. c:autodoc:: examples/example-40-enum.c
```

3.4.3 Output

enum **mode**

Frobnication modes for *frob()*.

MODE_PRIMARY

The primary frobnication mode.

MODE_SECONDARY

The secondary frobnication mode.

3.5 Struct

3.5.1 Source

```
/**
 * Linked list node.
 */
struct list {
        /** Next node. */
        struct list *next;

        /** Data. */
        list_data_t data;
};
```

3.5.2 Directive

```
.. c:autodoc:: examples/example-50-struct.c
```

3.5.3 Output

struct **list**
Linked list node.

struct *list* * **next**
Next node.

int **data**
Data.

3.6 Function

3.6.1 Source

```
/**
 * List frobnicator.
 *
 * :param list: The list to frob.
 * :param mode: The frobnication mode.
 * :return: 0 on success, non-zero error code on error.
 * :since: v0.1
 */
int frob(struct list *list, enum mode mode);

/**
 * variadic frobnicator
 *
 * :param fmt: the format
 * :param ...: variadic
 */
int frobo(const char *fmt, ...);
```

3.6.2 Directive

```
.. c:autodoc:: examples/example-70-function.c
```

3.6.3 Output

int **frob**(struct *list* * *list*, enum *mode* *mode*)
List frobnicator.

Parameters

- **list** – The list to frob.
- **mode** – The frobnication mode.

Returns 0 on success, non-zero error code on error.

Since v0.1

int **frobo**(const char * *fmt*, ...)
variadic frobnicator

Parameters

- **fmt** – the format
- ... – variadic

3.7 Preprocessor

3.7.1 Source

```
/**
 * Answer to the Ultimate Question of Life, The Universe, and Everything.
 */
#ifdef DEEP_THOUGHT
#define MEANING_OF_LIFE 42
#else
#error "Does not compute."
#endif
```

3.7.2 Directive

```
.. c:autodoc:: examples/example-70-preprocessor.c
   :clang: -DDEEP_THOUGHT
```

3.7.3 Output

MEANING_OF_LIFE
Answer to the Ultimate Question of Life, The Universe, and Everything.

3.8 Compat

3.8.1 Source

```
/**
 * List frobnicator.
 *
 * @param list The list to frob.
 * @param mode The frobnication mode.
 * @return 0 on success, non-zero error code on error.
 * @since v0.1
 */
int frob2(struct list *list, enum mode mode);
```

3.8.2 Directive

```
.. c:autodoc:: examples/example-80-compat.c
   :compat: javadoc-liberal
```

3.8.3 Output

int **frob2**(struct *list* * *list*, enum *mode* *mode*)
List frobnicator.

Parameters

- **list** – The list to frob.
- **mode** – The frobnication mode.

Returns 0 on success, non-zero error code on error.

Since v0.1

3.9 Generic

3.9.1 Source

```
/**
 * Source files may contain documentation comments not attached to any C
 * constructs. They will be included as generic documentation comments, for
 * example to describe the design of :c:type:`list` frobnication using
 * :c:func:`frob`.
 */
```

3.9.2 Directive

```
.. c:autodoc:: examples/example-90-generic.c
```

3.9.3 Output

Source files may contain documentation comments not attached to any C constructs. They will be included as generic documentation comments, for example to describe the design of *list* frobnication using *frob()*.

CHAPTER 4

Indices and tables

- genindex
- search

Index