Getting Started

Once Sphinx is installed, you can proceed with setting up your first Sphinx project. To ease the process of getting started, Sphinx provides a tool, **sphinx-quickstart**, which will generate a documentation source directory and populate it with some defaults. We're going to use the **sphinx-quickstart** tool here, though it's use by no means necessary.

Setting up the documentation sources

The root directory of a Sphinx collection of reStructuredText document sources is called the source directory. This directory also contains the Sphinx configuration file `conf.py`, where you can configure all aspects of how Sphinx reads your sources and builds your documentation. [1]

Sphinx comes with a script called **sphinx-quickstart** that sets up a source directory and creates a default `conf.py` with the most useful configuration values from a few questions it asks you. To use this, run:

```
$ sphinx-quickstart
```

Answer each question asked. Be sure to say yes to the `autodoc` extension, as we will use this later.

There is also an automatic "API documentation" generator called **sphinx-apidoc**; see sphinx-apidoc for details.

Defining document structure

Let's assume you've run **sphinx-quickstart**. It created a source directory with `conf.py` and a master document, `index.rst` (if you accepted the defaults). The main function of the master document is to serve as a welcome page, and to contain the root of the "table of contents tree" (or *toctree*). This is one of the main things that Sphinx adds to reStructuredText, a way to connect multiple files to a single hierarchy of documents.

The `toctree` directive initially is empty, and looks like so:

```
.. toctree::
   :maxdepth: 2
```

You add documents listing them in the *content* of the directive:

```
.. toctree::
   :maxdepth: 2

   usage/installation
   usage/quickstart
   ...
```

This is exactly how the `toctree` for this documentation looks. The documents to include are given as document names, which in short means that you leave off the file name extension and use forward slashes (`/`) as directory separators.

Read more about the toctree directive.

You can now create the files you listed in the `toctree` and add content, and their section titles will be inserted (up to the `maxdepth` level) at the place where the `toctree` directive is placed. Also, Sphinx now knows about the order and hierarchy of your documents. (They may contain `toctree` directives themselves, which means you can create deeply nested hierarchies if necessary.)

reStructuredText directives

`toctree` is a reStructuredText *directive*, a very versatile piece of markup. Directives can have arguments, options and content.

Arguments are given directly after the double colon following the directive's name. Each directive decides whether it can have arguments, and how many.

Options are given after the arguments, in form of a "field list". The `maxdepth` is such an option for the `toctree` directive.

Content follows the options or arguments after a blank line. Each directive decides whether to allow content, and what to do with it.

A common gotcha with directives is that **the first line of the content must be indented to the same level as the options are.**

Adding content

In Sphinx source files, you can use most features of standard reStructuredText. There are also several features added by Sphinx. For example, you can add cross-file references in a portable way (which works for all output types) using the `ref` role.

For an example, if you are viewing the HTML version you can look at the source for this document – use the "Show Source" link in the sidebar.

Todo

Update the below link when we add new guides on these.

 See reStructuredText for a more in-depth introduction to reStructuredText, including markup added by Sphinx.

Running the build

Now that you have added some files and content, let's make a first build of the docs. A build is started with the **sphinx-build** program:

```
$ sphinx-build -b html sourcedir builddir
```

where *sourcedir* is the source directory, and *builddir* is the directory in which you want to place the built documentation. The `-b` option selects a builder; in this example Sphinx will build HTML files.

 Refer to the sphinx-build man page for all options that **sphinx-build** supports.

However, **sphinx-quickstart** script creates a `Makefile` and a `make.bat` which make life even easier for you. These can be executed by running **make** with the name of the builder. For example.

```
$ make html
```

This will build HTML docs in the build directory you chose. Execute **make** without an argument to see which targets are available.

How do I generate PDF documents?

`make latexpdf` runs the `LaTeX builder` and readily invokes the pdfTeX toolchain for you.

Todo

Move this whole section into a guide on rST or directives

Documenting objects

One of Sphinx's main objectives is easy documentation of *objects* (in a very general sense) in any *domain*. A domain is a collection of object types that belong together, complete with markup to create and reference descriptions of these objects.

The most prominent domain is the Python domain. For example, to document Python's built-in function `enumerate()`, you would add this to one of your source files.

```
.. py:function:: enumerate(sequence[, start=0])

   Return an iterator that yields tuples of an index and an item of the
   *sequence*. (And so on.)
```


This is rendered like this:

enumerate(*sequence*[, *start=0*])

Return an iterator that yields tuples of an index and an item of the *sequence*. (And so on.)

The argument of the directive is the *signature* of the object you describe, the content is the documentation for it. Multiple signatures can be given, each in its own line.

The Python domain also happens to be the default domain, so you don't need to prefix the markup with the domain name.

```
.. function:: enumerate(sequence[, start=0])

   ...
```

does the same job if you keep the default setting for the default domain.

There are several more directives for documenting other types of Python objects, for example `py:class` or `py:method`. There is also a cross-referencing *role* for each of these object types. This markup will create a link to the documentation of `enumerate()`.

```
The :py:func:`enumerate` function can be used for ...
```

And here is the proof: A link to `enumerate()`.

Again, the `py:` can be left out if the Python domain is the default one. It doesn't matter which file contains the actual documentation for `enumerate()`; Sphinx will find it and create a link to it.

Each domain will have special rules for how the signatures can look like, and make the formatted output look pretty, or add specific features like links to parameter types, e.g. in the C/C++ domains.


See Domains for all the available domains and their directives/roles.

Basic configuration

Earlier we mentioned that the `conf.py` file controls how Sphinx processes your documents. In that file, which is executed as a Python source file, you assign configuration values. For advanced users: since it is executed by Sphinx, you can do non-trivial tasks in it, like extending **`sys.path`** or importing a module to find out the version you are documenting.

The config values that you probably want to change are already put into the `conf.py` by **sphinx-quickstart** and initially commented out (with standard Python syntax: a `#` comments the rest of the line). To change the default value, remove the hash sign and modify the value. To customize a config value that is not automatically added by **sphinx-quickstart**, just add an additional assignment.

Keep in mind that the file uses Python syntax for strings, numbers, lists and so on. The file is saved in UTF-8 by default, as indicated by the encoding declaration in the first line. If you use non-ASCII characters in any string value, you need to use Python Unicode strings (like `project = u'Exposé'`).


See Configuration for documentation of all available config values.

Todo

Move this entire doc to a different section

Autodoc

When documenting Python code, it is common to put a lot of documentation in the source files, in documentation strings. Sphinx supports the inclusion of docstrings from your modules with an *extension* (an extension is a Python module that provides additional features for Sphinx projects) called *autodoc*.

In order to use *autodoc*, you need to activate it in `conf.py` by putting the string `'sphinx.ext.autodo` assigned to the `extensions` config value. Then, you have a few additional directives at your disposal.

For example, to document the function `io.open()`, reading its signature and docstring from the source file, you'd write this:

```
.. autofunction:: io.open
```

You can also document whole classes or even modules automatically, using member options for the auto directives, like

```
.. automodule:: io
   :members:
```

autodoc needs to import your modules in order to extract the docstrings. Therefore, you must add the appropriate path to **sys.path** in your `conf.py`.

Warning

autodoc imports the modules to be documented. If any modules have side effects on import, these will be executed by `autodoc` when `sphinx-build` is run.

If you document scripts (as opposed to library modules), make sure their main routine is protected by a `if __name__ == '__main__'` condition.

 See **sphinx.ext.autodoc** for the complete description of the features of autodoc.

Todo

Move this doc to another section

Intersphinx

Many Sphinx documents including the Python documentation are published on the internet. When you want to make links to such documents from your documentation, you can do it with **sphinx.ext.intersphinx**.

In order to use intersphinx, you need to activate it in `conf.py` by putting the string `'sphinx.ext.intersphinx'` into the **extensions** list and set up the **intersphinx_mapping** config value.

For example, to link to `io.open()` in the Python library manual, you need to setup your **intersphinx_mapping** like:

```
intersphinx_mapping = {'python': ('https://docs.python.org/3', None)}
```

And now, you can write a cross-reference like :py:func:`io.open`. Any cross-reference that has no matching target in the current documentation set, will be looked up in the documentation sets configured in **intersphinx_mapping** (this needs access to the URL in order to download the list of valid targets). Intersphinx also works for some other domain's roles including `:ref:`, however it doesn't work for `:doc:` as that is non-domain role.

 See **sphinx.ext.intersphinx** for the complete description of the features of intersphinx.

More topics to be covered

- Other extensions:
- Static files
- Selecting a theme
- Setuptools integration
- Templating
- Using extensions
- Writing extensions

v: master

Footnotes

[1] This is the usual layout. However, `conf.py` can also live in another directory, the configuration directory.

Refer to the sphinx-build man page for more information.