

# My Project

version

Author Name

September 21, 2022



# Contents

<b>SEaB projname's demo documentation</b>	<b>1</b>
Overview	1
Objective	1
Tools	1
Features	1
Other Content	3
More Detail	3
An Introduction to reStructuredText	4
Goals	4
History	5
Technical File	6
Technical File Information	6
Access to the Technical File	7
When should the file be created?	7
Delivery of the file	8
<b>Search</b>	<b>8</b>



# SEaB projname's demo documentation

## Overview

This is an overview document that follows on from the Documentation report issued last week.

This content is produced in [RST](#) (reStructuredText), but could also have equally been produced in [Markdown](#) although RST is more versatile and produces better looking web content.

RST has many advanced features, too many to list in this introductory document, but I will detail some within this content. One useful feature is the ability to include content from other sources by using the *include* directive.

The include directive looks like this:

```
..include:: demo.txt
```

The include will only display in a compiled HTML file, so in Github there will be a blank space below (where the include statement sits):

This is a plain text file that can be used to include repeatable text, or information that may need to be changed on an infrequent basis (such as address details).

## Objective

This content is here to show some of the features of RST and Sphinx, but its main purpose is to give you something to review and comment on.

### Note

RST is very sensitive about white space. It may be better to add comments/changes to the pdf version of the file rather than the RST itself

## Tools

Several software tools are being used in the creation of content:

- Sublime Text - the editor used for RST content
- MarkdownPad2 - the editor used for Markdown content
- Sphinx - the static site generator (HTML)
- Sphinx-intl -the localisation add in for Sphinx (to produce translation files)
- RST2PDF - the generator used to produce PDF files
- Pandoc - for conversion of files (e.g., from RST to .docx)
- Vale - a linter and style checker


With the exception of Sublime Text and MarkdownPad2, all the other tools are Linux CLI (command line) tools.

## Features

A few features of RST:

Tables:

Header row, column 1	Header 2	Header 3
body row 1, column 1	column 2	column 3
body row 2	Cells may span columns	


Image	Description
	Dangerous voltage

Admonitions:

**Warning**  
This is some warning text

**Attention!**  
This is some text to draw attention

You can also create custom admonitions:

	Warning! Dangerous voltage present
-----------------------------------------------------------------------------------	------------------------------------

**Custom**  
This is a custom admonition

Code:

Code can be displayed in a number of ways. In a table:

Code 1	Code 2
<pre>foo.c  extern int bar(int y); int foo(int x) {     return x &gt; 0 ? bar(x-1)+1                 : 0; }</pre>	<pre>bar.c  extern int foo(int x); int bar(int y) {     return y &gt; 0 ? foo(x-1)*2                 : 0; }</pre>

or in a code block:

*EXT:site\_package/Configuration/TCA/Overrides/sys\_template.php*

```
/**
 * Add default TypoScript (constants and setup)
 */
\TYPO3\CMS\Core\Utility\ExtensionManagementUtility::addStaticFile(
    'site_package',
    'Configuration/TypoScript',
    'Site Package'
);
```

Emphasis:

**Bold**

*Italic*

*Interpreted text*

Footnotes or endnotes <sup>1</sup>

Mathematics:

RST, Sphinx and RST2PDF all have support for mathematics and complex formulae.

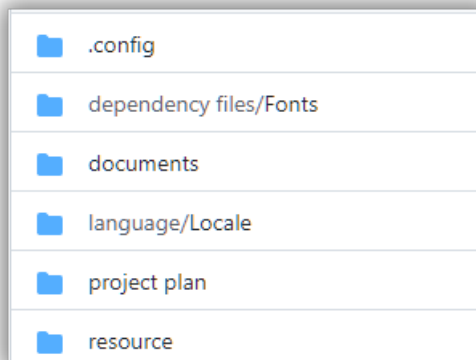
A couple of examples -

$$y - y_0 = m(x - x_0)$$

$$J(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

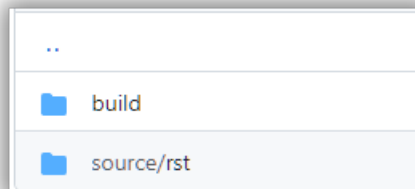
## Other Content

If you browse through the repository, you will find several other directories.



Browse through then to see the content. The **‘.config’** directory only contains a single file `conf.py`, which is the configuration file for Sphinx. The **‘language’** directories will contain the gettext strings for translation.

Within the **‘documents’** directory are some other sub-directories of interest:



You are already in the **‘source’** directory (and there is a Markdown file included in the **‘md’** subdirectory for comparison), but the **‘build’** directory contains the html and pdf versions of the RST source files. Take a look at both sets of files to get a feel for how content will be produced.

GitHub will give a preview of the pdf file (but none of the links will work), and will display the code for the HTML files, not the content. A zip file **‘html-docs.zip’** can be downloaded to allow viewing the HTML content as intended.

## Important

The HTML and PDF content are not yet styled and are using default settings

## More Detail

The other pages in this demo content are **‘guide.rst’** (a fairly long document that details many of the features of RST), and **‘example.rst’** (an example of a very boring Markdown file converted to RST using pandoc).

<sup>1</sup> A footnote or endnote

# An Introduction to reStructuredText

**Author:** David Goodger

**Contact:** [docutils-develop@lists.sourceforge.net](mailto:docutils-develop@lists.sourceforge.net)

**Revision:** \$Revision: 9051 \$

**Date:** \$Date: 2022-04-02 23:59:06 +0200 (Sa, 02. Apr 2022) \$

**Copyright:** This document has been placed in the public domain.

[reStructuredText](#) is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax and parser system. It is useful for inline program documentation (such as Python docstrings), for quickly creating simple web pages, and for standalone documents. [reStructuredText](#) is a proposed revision and reinterpretation of the [StructuredText](#) and [Setext](#) lightweight markup systems.

reStructuredText is designed for extensibility for specific application domains. Its parser is a component of [Docutils](#).

This document defines the [goals](#) of reStructuredText and provides a [history](#) of the project. It is written using the reStructuredText markup, and therefore serves as an example of its use. For a gentle introduction to using reStructuredText, please read [A ReStructuredText Primer](#). The [Quick reStructuredText](#) user reference is also useful. The [reStructuredText Markup Specification](#) is the definitive reference. There is also an analysis of the [Problems With StructuredText](#).

ReStructuredText's web page is <https://docutils.sourceforge.io/rst.html>.

## Goals

The primary goal of [reStructuredText](#) is to define a markup syntax for use in Python docstrings and other documentation domains, that is readable and simple, yet powerful enough for non-trivial use. The intended purpose of the reStructuredText markup is twofold:

- the establishment of a set of standard conventions allowing the expression of structure within plaintext, and
- the conversion of such documents into useful structured data formats.

The secondary goal of reStructuredText is to be accepted by the Python community (by way of being blessed by PythonLabs and the BDFL <sup>2</sup>) as a standard for Python inline documentation (possibly one of several standards, to account for taste).

<sup>2</sup> Python's creator and "Benevolent Dictator For Life", Guido van Rossum.

To clarify the primary goal, here are specific design goals, in order, beginning with the most important:

1. Readable. The marked-up text must be easy to read without any prior knowledge of the markup language. It should be as easily read in raw form as in processed form.
2. Unobtrusive. The markup that is used should be as simple and unobtrusive as possible. The simplicity of markup constructs should be roughly proportional to their frequency of use. The most common constructs, with natural and obvious markup, should be the simplest and most unobtrusive. Less common constructs, for which there is no natural or obvious markup, should be distinctive.
3. Unambiguous. The rules for markup must not be open for interpretation. For any given input, there should be one and only one possible output (including error output).
4. Unsurprising. Markup constructs should not cause unexpected output upon processing. As a fallback, there must be a way to prevent unwanted markup processing when a markup construct is used in a non-markup context (for example, when documenting the markup syntax itself).
5. Intuitive. Markup should be as obvious and easily remembered as possible, for the author as well as for the reader. Constructs should take their cues from such naturally occurring sources as plaintext email messages, newsgroup postings, and text documentation such as README.txt files.
6. Easy. It should be easy to mark up text using any ordinary text editor.
7. Scalable. The markup should be applicable regardless of the length of the text.
8. Powerful. The markup should provide enough constructs to produce a reasonably rich structured document.



9. Language-neutral. The markup should apply to multiple natural (as well as artificial) languages, not only English.
- 10 Extensible. The markup should provide a simple syntax and interface for adding more complex general markup, and custom markup.
- 11 Output-format-neutral. The markup will be appropriate for processing to multiple output formats, and will not be biased toward any particular format.

The design goals above were used as criteria for accepting or rejecting syntax, or selecting between alternatives.

It is emphatically *not* the goal of reStructuredText to define docstring semantics, such as docstring contents or docstring length. These issues are orthogonal to the markup syntax and beyond the scope of this specification.

Also, it is not the goal of reStructuredText to maintain compatibility with [StructuredText](#) or [Setext](#). reStructuredText shamelessly steals their great ideas and ignores the not-so-great.

Author's note:

Due to the nature of the problem we're trying to solve (or, perhaps, due to the nature of the proposed solution), the above goals unavoidably conflict. I have tried to extract and distill the wisdom accumulated over the years in the Python [Doc-SIG](#) mailing list and elsewhere, to come up with a coherent and consistent set of syntax rules, and the above goals by which to measure them.

There will inevitably be people who disagree with my particular choices. Some desire finer control over their markup, others prefer less. Some are concerned with very short docstrings, others with full-length documents. This specification is an effort to provide a reasonably rich set of markup constructs in a reasonably simple form, that should satisfy a reasonably large group of reasonable people.

David Goodger ([goodger@python.org](mailto:goodger@python.org)), 2001-04-20

## History

reStructuredText, the specification, is based on [StructuredText](#) and [Setext](#). StructuredText was developed by Jim Fulton of [Zope Corporation](#) (formerly Digital Creations) and first released in 1996. It is now released as a part of the open-source "Z Object Publishing Environment" ([ZOPE](#)). Ian Feldman's and Tony Sanders' earlier [Setext](#) specification was either an influence on StructuredText or, by their similarities, at least evidence of the correctness of this approach.

I discovered [StructuredText](#) in late 1999 while searching for a way to document the Python modules in one of my projects. Version 1.1 of StructuredText was included in Daniel Larsson's [pythondoc](#). Although I was not able to get pythondoc to work for me, I found StructuredText to be almost ideal for my needs. I joined the Python [Doc-SIG](#) (Documentation Special Interest Group) mailing list and found an ongoing discussion of the shortcomings of the StructuredText "standard". This discussion has been going on since the inception of the mailing list in 1996, and possibly predates it.

I decided to modify the original module with my own extensions and some suggested by the Doc-SIG members. I soon realized that the module was not written with extension in mind, so I embarked upon a general reworking, including adapting it to the "re" regular expression module (the original inspiration for the name of this project). Soon after I completed the modifications, I discovered that StructuredText.py was up to version 1.23 in the ZOPE distribution. Implementing the new syntax extensions from version 1.23 proved to be an exercise in frustration, as the complexity of the module had become overwhelming.

In 2000, development on StructuredTextNG ("Next Generation") began at [Zope Corporation](#) (then Digital Creations). It seems to have many improvements, but still suffers from many of the problems of classic StructuredText.

I decided that a complete rewrite was in order, and even started a [reStructuredText SourceForge project](#) (now inactive). My motivations (the "itches" I aim to "scratch") are as follows:

- I need a standard format for inline documentation of the programs I write. This inline documentation has to be convertible to other useful formats, such as HTML. I believe many others have the same need.
- I believe in the Setext/StructuredText idea and want to help formalize the standard. However, I feel the current specifications and implementations have flaws that desperately need fixing.
- reStructuredText could form part of the foundation for a documentation extraction and processing system, greatly benefitting Python. But it is only a part, not the whole. reStructuredText is a markup language

specification and a reference parser implementation, but it does not aspire to be the entire system. I don't want reStructuredText or a hypothetical Python documentation processor to die stillborn because of over-ambition.

- Most of all, I want to help ease the documentation chore, the bane of many a programmer.

Unfortunately I was sidetracked and stopped working on this project. In November 2000 I made the time to enumerate the problems of StructuredText and possible solutions, and complete the first draft of a specification. This first draft was posted to the Doc-SIG in three parts:

- [A Plan for Structured Text](#)
- [Problems With StructuredText](#)
- [reStructuredText: Revised Structured Text Specification](#)

In March 2001 a flurry of activity on the Doc-SIG spurred me to further revise and refine my specification, the result of which you are now reading. An offshoot of the reStructuredText project has been the realization that a single markup scheme, no matter how well thought out, may not be enough. In order to tame the endless debates on Doc-SIG, a flexible [Docstring Processing System framework](#) needed to be constructed. This framework has become the more important of the two projects; [reStructuredText](#) has found its place as one possible choice for a single component of the larger framework.

The project web site and the first project release were rolled out in June 2001, including posting the second draft of the spec <sup>3</sup> and the first draft of PEPs 256, 257, and 258 <sup>4</sup> to the Doc-SIG. These documents and the project implementation proceeded to evolve at a rapid pace. Implementation history details can be found in the [project history file](#).

In November 2001, the reStructuredText parser was nearing completion. Development of the parser continued with the addition of small convenience features, improvements to the syntax, the filling in of gaps, and bug fixes. After a long holiday break, in early 2002 most development moved over to the other Docutils components, the “Readers”, “Writers”, and “Transforms”. A “standalone” reader (processes standalone text file documents) was completed in February, and a basic HTML writer (producing HTML 4.01, using CSS-1) was completed in early March.

[PEP 287](#), “reStructuredText Standard Docstring Format”, was created to formally propose reStructuredText as a standard format for Python docstrings, PEPs, and other files. It was first posted to [comp.lang.python](#) and the [Python-dev](#) mailing list on 2002-04-02.

Version 0.4 of the [reStructuredText](#) and [Docstring Processing System](#) projects were released in April 2002. The two projects were immediately merged, renamed to “Docutils”, and a 0.1 release soon followed.

3

The second draft of the spec:

- [An Introduction to reStructuredText](#)
- [Problems With StructuredText](#)
- [reStructuredText Markup Specification](#)
- [Python Extensions to the reStructuredText Markup Specification](#)

4

First drafts of the PEPs:

- [PEP 256: Docstring Processing System Framework](#)
- [PEP 258: DPS Generic Implementation Details](#)
- [PEP 257: Docstring Conventions](#)

Current working versions of the PEPs can be found in <https://docutils.sourceforge.io/docs/peps/>, and official versions can be found in the [master PEP repository](#).

## Technical File

### Technical File Information

All CE marking directives (and this applies also to UKCA marking) require the manufacturer of the product to create a technical file which should contain the information required to show that the product properly complies with the requirements of the directives which apply to it.

The directives contain clauses which give some general details of the kind of information which will be required in the technical file, but this is couched in the most general of terms. As a general guide, the following items should be included:

- Description of the apparatus, usually accompanied by a block diagram
- Wiring and circuit diagrams
- General Arrangement drawing
- List of standards applied
- Records of risk assessments and assessments to standards
- Description of control philosophy/logic
- Data sheets for critical sub-assemblies
- Parts list
- Copies of any markings and labels
- Copy of instructions (user, maintenance, installation)
- Test reports
- Quality control & commissioning procedures
- Declaration of Conformity

There is nothing to stop the file containing a great deal more information than is listed here - for instance, copies of the engineering drawings for any bespoke parts could also be included. However, these should not be used to pack the file at the expense of the more relevant information on how the equipment operates and how it meets the safety objectives of the Directives. Of course, for simple products (e.g. household domestic appliances) the control philosophy may be so simple as to be self-evident, in which case it is unnecessary for the file to include this information. Often, this sort of product will be the subject of a notified body report confirming compliance to a harmonised standard and a copy of the report should be included in the file. The importance of including a basic general description of the appliance cannot be overstated. Furthermore, the action of providing a properly documented description of the control system will often allow the designer to spot potential flaws in the logic of the operation.

### Access to the Technical File

It is important to understand that only the authorities given the power to enforce the directives have a right to see the technical file. It does not need to be published or given to customers. With the exception of the Medical Devices Directive, there is no clear requirement that the file should be kept on EU soil. However, for a manufacturer based outside of the EU, any approach made by the authorities will initially come to the authorised agent or the importer of the goods, so they must have access to the file to be able to fulfil their legal obligations. Access to technical information may also be a condition of contract laid down by some customers.

The Machinery Directive requires manufacturers based outside Europe to appoint someone within Europe to act as the contact point for their technical documents.

### When should the file be created?

Different directives treat the creation of the file with different priorities. The LVD strictly requires the manufacturer to have the file in place before the CE mark is applied, whereas the Machinery Directive says that the file does not have to be in constant existence and thereby provides for a period of grace in which it can be compiled. In practice, it usually makes sense to ensure the file is created early in the life of a product and is then kept up to date.

For some directives the creation of the file and its assessment by a Notified Body are a pre-requisite for proper compliance with the Directives. This is the case for the higher classes of medical device, personal protective equipment and for equipment for explosive atmospheres. Assessment of the technical file is also a requirement for Type Approval under the Machinery Directive.

There are also other reasons why it makes sense to compile the file at the time the product is originally designed rather than waiting until a request is received from an enforcement body. Firstly, there is the problem that much of the information required for the file (particularly component data and supplier information) may no longer be available several years after the product design has been completed. Secondly, the file can form a useful repository for

information on changes to the design or manufacturing processes used to make the product. Finally, if used correctly, the file can itself become a useful way to manage the process of properly completing the CE marking process: if a list of the contents required for the file is drawn up at an early stage then this list can be used to identify whether all of the necessary steps have been completed by identifying the relevant information in the file for each stage in the process.

### Delivery of the file

The purpose of the Technical File is to provide evidence for an enforcement authority that the product has correctly completed the assessment and attestation procedures of the relevant directive(s).

The enforcement authorities have the right to demand a copy of the file for up to 10 years after the last date on which the product(s) described in the file was made. This is by no means a trivial requirement - changes in personnel, company facilities, IT systems and file formats all represent major challenges to guaranteeing the availability of the required information and need to be considered carefully.

The file must be delivered 'in material form' which may be taken to mean as a paper copy. In practice, however, the enforcement bodies are likely to be satisfied with an electronic version, and in fact may only demand specific sections of the file if their concerns relate to specific aspects of the product's compliance with the directives.

The file has to be drawn up in a European language, and will normally be requested by the relevant enforcement body for the country and region in which the manufacturer of the product is located (the "Home Authority Principle"). The manufacturer is not obliged to translate the file before delivery to an enforcement authority from outside their home country.

## Search

- **search**