How to generate the documentation for the or-tools library

Nikolaj van Omme[*]

July 5, 2012

Abstract

This little document explains how to generate and upload all the documentation for the Google or-tools library. This document doesn't explain how to write the documentation i.e. how to use of *Sphinx*, *Jinja2*, *html*, *css*, etc., only how to generate the output files once the documentation is written and how to upload the output files on the Google servers. All the scripts are written in Python and must be called in the correct sequence.

Only tested under Linux.

THIS DOCUMENT IS NOT UP TO DATE AND CONTAINS ERRORS!

Contents

1 Introduction (How it works) **2**
- 1.1 Vocabulary . . . 2
- 1.2 Directories . . . 2
- 1.3 Contents of the `SOURCES` directory . . . 3
- 1.4 Work flow . . . 4
- 1.5 Preferred use . . . 4

2 Install the documentation and the needed tools **5**
- 2.1 External libraries and tools . . . 5
- 2.2 Scripts . . . 5

3 or-tools Sphinx extension **5**
- 3.1 Links and references . . . 5

4 Add to the manual . . . **6**
- 4.1 a part . . . 6
- 4.2 a chapter . . . 6
- 4.3 a section . . . 6
- 4.4 Front material . . . 7

5 Generate the documentation **7**
- 5.1 The manual . . . 7
- 5.2 The documentation hub . . . 7
- 5.3 The tutorial code . . . 7
- 5.4 The slides . . . 7
- 5.5 The version . . . 7
- 5.6 Documentation Hub . . . 7

6 Global cripts **7**

7 Gotchas **7**

8 Resources **8**
- 8.1 Images . . . 8

[*]You can reach me at ortools.doc@gmail.com if you need some help. ;-)

1 Introduction (How it works)

This documentation and the process to generate the or-tools library's documentation was designed with the or-tools people in mind. As such, the whole generation process isn't bullet-proof nor idiot-proof.

You can get your local copy of the documentation and generate it if you want but there is no reason to do so as all the generated documentation is already available on the Google servers. If you find better ways to generate some parts of the documentation, please share with the whole community (but keep in mind that our main purpose is not to spend too much time on this).

This document itself is written in LaTeX and generated with PDFLaTeX.

1.1 Vocabulary

To avoid misunderstanding, let's agree on some wordings.

Source/Code file a file in wich you write the documentation. It can be restructuredText (.rst), LaTeX (.tex) or text (.txt, .css, html, ...)[1]. Some files are at the same time source and output files.

Output file a file that will be published on the server side, including html files, images, etc. Some files are at the same time source and output files.

Local home the computer you use to write the documentation.

Google servers the servers where the documentation is publicly accessible from the Internet.

Documentation hub a central html page from which all the documentation is accessible except the downloadable files. The url is: http://or-tools.googlecode.com/svn/trunk/documentation/documentation_hub.html.

1.2 Directories

There are seven main directories: four on the Local home side (`SOURCES`, `BUILD`, `DEPLOY` and `DOCUMENTATION`) and three on the Google servers side (`svn/doc_sources`, `files` and `svn/trunk/documentation`). Figure 1 illustrates those directories. The four local directories can be named to your liking (see section 2).

`SOURCES` This local directory contains the sources files, the scripts files and some configuration files. This directory is a local copy of the the directory `doc_sources`.

`BUILD` This local directory contains all the automatically generated output files.

`DEPLOY` This local directory contains all the generated output files (automatically or by hand). It is not a local copy of the `trunk/documentation` directory as the directory structure is slightly different. We use it essentially for testing purpose.

`DOCUMENTATION` The real thing, i.e. the documentation you can access from the documentation hub. This local directory is a local copy of the `trunk/documentation` directory. Actually, it is more the opposite: `svn/trunk/documentation` is a copy of your local directory `DOCUMENTATION` as we push (commit) the generated documentation on the Google servers without pulling (updating) it.

`svn/doc_sources` A server directory containing the sources files of the documentation. All the files you need to generate the documentation are stored in this directory. The full url is http://or-tools.googlecode.com/svn/doc_sources.

`files` A server directory containing the downloadable files. The full url is http://or-tools.googlecode.com/files. Note that you don't have public access to the directory, only to the files stored in it.

[1] Basically, *all* source files are text files.

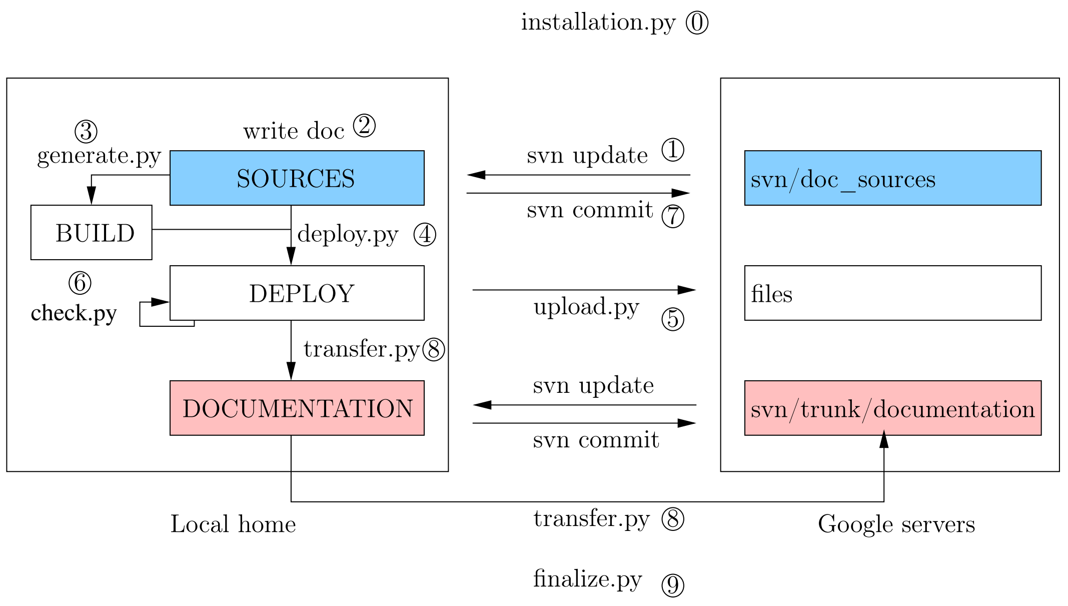

Figure 1: figure
The six main directories and the scripts/commands to move the files between them. The base url for the Google Servers is http://or-tools.googlecode.com/. The directories colored in the same color are svn copies of each other[2].

`svn/trunk/documentation` A server directory with the or-tools library's documentation (except for the downloadable files). The full url is http://or-tools.googlecode.com/svn/trunk/documentation.

1.3 Contents of the `SOURCES` directory

The documentation is explained on the documentation hub.

The local directory `SOURCES` contains the following files:

`current_version.txt` Text file with the actual release number (xx.yy.zz), i.e. the release number of the next documentation you will upload.

`global.rst` Rst file with global variables. You can update it manually if you need to.

`README.txt` You know...

`TODO.txt` What we plan for the future.

The local directory `SOURCES` contains the following directories:

`doc` The doc of the doc, i.e. the LaTeX files to generate this document.

`FAQ` Source files for the Frequently Asked Questions. The FAQ is generated by Sphinx.

`HUB` Sources files for the documentation hub. Written by hand but with the help of the Python `generate.py` script.

`internals` Internal files for the scripts. You shouldn't change anything here.

`LABS` Exercises.

[2]We don't consider white as a color, at least not in this document.

1.4 Work flow

The whole process is divided in 9 steps[3]. Most of the steps are automated by Python scripts and to simplify your life (and mine), several scripts are bundled in the global scripts (see section 6). These scripts need to be called in sequence. In each of the subdirectories of the directory `SOURCES` (see section 1.3), you will find the corresponding Python scripts if needed. If you don't find a specific script in a subdirectory, it means this script is not needed for that type of documentation.

Step ① Update the source code: (script: none)
Update your local copy of the documentation via svn: `svn update`.

Step ② Write the documentation: (script: none)
Let your talent speak!

Step ③ Generate the documentation: (script: `generate.py`)
Generate automatically the documentation. You will find the generated documentation in the `BUILD` directory. For the moment (as of May 27 2012), only the manual (subdirectory `MANUAL`) and the FAQ (subdirectory `FAQ`) are automatically generated.
The documentation hub is parlty generated automatically, partly generated by hand. See section 5.2 for more details.

Step ④ Deploy the documentation: (script: `deploy.py`)
Copy locally the output files in the directory `DEPLOY`. This directory is used to test the documentation and try new things without messing with the svn directory.

Step ⑤ Upload the documentation: (script: `upload.py`)
Before you can check the documentation in the `DEPLOY` directory, and if you want to test the download links, you have to upload the downloadable files. The `upload.py` script will be stuck if you try to upload a file that is already uploaded (based on the filename). If you want to update a downloadable file, you first have to delete the file from the Google servers. For the moment, this action has to be done manually through web user interface. If you know how to do it with a script, we are interested!

Step ⑥ Check the documentation: (script: `check.py`)
Check the documentation. As of May 27 2012, the `check.py` script only checks extern links. The rest is a manual (and important!) process.

Step ⑦ Update the documentation source: (script: none)
If you're happy with the new documentation, commit your changes on the Google solvers.

Step ⑧ Upload the documentation: (script: `transfer.py`)
Push (commit) the documentation on the Google servers. Prefer to use the `transfer.py` script over the svn commands.

Step ⑨ Prepare the next release of the documentation: (script: `finalize.py`)
The documentation has a release number (xx.yy.zz). It is used internally by the scripts to check some operations. The `finalize.py` script will copy the old number and update the new number (for instance, from 4.5.11 to 4.5.12 by default). You MUST use this script after step ⑧. If you want to update the release number to a higher release number, edit manually the file `current_version.txt` in the `SOURCES` directory.

Section 5 provides a more detailed account on how to generate the documentation.

1.5 Preferred use

These scripts are not bullet-proof nor idiot-proof. I strongly suggest that the source files always match the documentation files in http://or-tools.googlecode.com/svn/trunk/documentation . Once you have finished steps ① - ⑥, commit your changes on the Google solvers (step ⑦). If there is a conflict, start the whole process again later when the generated documentation is up to

[3]Once you get used to the documentation generation, you will be able to skip some of the steps (when you don't need to update the whole documentation tree) and to only use the specific manual commands you need.

date and matches the sources files on the Google servers. One easy way to ensure the consistency between the source and the documentation is to delegate to one single person the upload and commitment of the documentation (steps ⑧ - ⑨) (or to redesign the whole process ;-)). This design ask for some discipline but I think it is manageable among Googlers.

2 Install the documentation and the needed tools

To do.

2.1 External libraries and tools

The following list details all the libraries and tools that I use to generate the documentation. Most of them are written in Python.

Sphinx This is the main library. It transforms `restucturedText` into plenty of other formats.

Jinja2

LaTeX To generate the pdf version of the manual, we use LaTeX and `pdflatex` (and of course, the classical `makeindex/mkindex`, `bibtex`, . . .).

We use also Python (2.7), make, html, css, ...

2.2 Scripts

The Python scripts are not bullet-proof.

3 or-tools Sphinx extension

To do.

3.1 Links and references

The right way to define references is to subclass `XRefRole` from `sphinx.roles` and tho redefine its methods `process_link()` and `result_nodes()`. In the standard domain (`domain = env.domains['std']`), two dictionnaries are defined with anonymous labels and non anonymous labels. You can retrieve their values with:

```
docname, labelid = domain.data['anonlabels'].get(target, ('',''))
```

and

```
docname, labelid, sectname = domain.data['labels'].get(target, ('','','')).
```

However, some labels are not stored in those dictionnaries (I really have no idea why...). After some thoughts, I decided to construct my own references my own way.

3.1.1 Labels

I use the same labels:

```
.. _my_beautiful_label:
```

Note that you can only use one label per line. The labels have to be placed right above a title, a table, a topic, etc. as mentioned in the Sphinx documentation.

3.1.2 References

You still can use the classical references:

`:ref:‘My beautiful title <my_beautiful_label>‘` or `:ref:‘my_beautiful_label‘`

but their use is quite limited. Instead use the ones defined in the extension:

`:yref:‘My beautiful title <my_beautiful_label>‘` or `:yref:‘my_beautiful_label‘`.

Not only are they resolved differently (and correctly) in LaTeX and HTML but they are ”clever” and know what type of references they are (reference to a Figure, a title, a table, etc.).

If you want to start a sentence with a reference and thus ask for a first capitalized letter, use ”^” (the circumflex character) in front of the title or the label in the `yref`. Table 1 shows several examples.

Table 1: Several examples using `yref`

code	HTML	LaTeX
`:yref:‘^Tabula rasa <my_ref>‘`	The Tabla rasa	Tabula rasa *x*
`:yref:‘<^ref_to_my_tabel>‘`	The table *tablename*	Table *x*
`:yref:‘Tabla <ref_to_my_tabel>‘`	the Tabla *tablename*	Tabla *x*
`:yref:‘<ref_to_my_tabel>‘`	The table *tablename*	Table *x*

4 Add to the manual . . .

To do.

This section covers only the manual. To correct or add some material, just open the corresponding `rst` file, follow the `rst` syntax and the more specialized `Sphinx` syntax and adapt the file to your needs. If you want to add a chapter or a section, read the next two sections. We discuss also how to add a label and a reference in section **??** and the front material as it requires a special treatment.

4.1 a part

4.2 a chapter

To add a chapter, follow the next steps:

1. In `SOURCES/MANUAL/source/manual`, create a file `mychapter.rst` and a folder `mychapter` where you will write the different sections of the chapter. See the other `rst` files.

2. Add an entry in `index.rst` in the table of contents at the end. This will add you chapter to the manual but will not make it visible yet.

3. Update manually the table of contents and possibly renumber the other chapters:

   - In `index.rst`, update the sidebar (`.. sidebar:: Content at a glance`);
   - In `MANUAL/source/doctemplates`, update `myglobaltoc.html`.This file is used to generate the toc in the sidebar for the first page of each chapter in the `html` version.

4. To make you chapter visible, open `config.py` and update `html_sidebars` accordingly.

There is nothing else to do for the LaTeX version.

4.3 a section

Just write your section in a `rst` file in the folder corresponding to the chapter and add an entry in the corresponding `rst` file of the chapter.

4.4 Front material

Again, we have duplicated the text as `Sphinx` treats the `pdf` and `html` versions differently.

4.4.1 Title page

4.4.2 Foreword

4.4.3 Table of contents

5 Generate the documentation

5.1 The manual

5.1.1 *Final* or *draft* release

5.2 The documentation hub

5.3 The tutorial code

5.4 The slides

5.5 The version

All the automatic generation of the doc is based on the current version number, so don't mess with it[4]! ;-) The file `current_version.txt` contains the current version (the one that you will upload on the server). You can update it by hand. By default, after a deploy and an update, the next version is incremented by 1. For instance, if the current version was `1.2.23` before deploying and updating, then the next version will be automatically set to `1.2.24`.

Each documentation of subdirectory `SOURCES` has an individual `deploy_xxx` and `upload_xxx` than can be called manually if desired. Pay attention to the order in which you call them. We detail each of them in the right sequence in the next subsections.

5.6 Documentation Hub

5.6.1 The change files/directory

changes.txt This is where you write what changed since the last upload of the documentation. This file is automatically inserted in `documentation_hub.html` so be carefull! ;-) Lines starting with `#` are comments that are not written in the html file. Don't add the version, this is done automatically. Note that this file is not automatically updated. You are responsible for its content.

changes_list.txt This file is automatically updated with the content of `changes.txt`.

changes This directory contains copies of all the `changes.txt` files.

6 Global cripts

7 Gotchas

- When in draft mode (see 5.1.1), the section numbers in the html pages are wrong.
- LaTeX slides files have to start with `\documentclass{}` on the first line (required by `generate_slides.py`).
- Manual: the preface is copied once. This is necessary as Sphinx doesn't allow to differentiate between titles in LaTeX and Sphinx.
- There is no automatic update between the Getting started page of the wiki and the one of the manual. If you change one, you have to change the other manually.

[4]An assert-like script is run by all the other scripts to verify that at least the current version is greater than the version before.

- When you want to use class names in titles and want to talk about them in the plural, you have to add `\s`, not just `s`:

```
``SearchMonitor``\s
-------------------
```

- Capitalized letters in a reference become lower letters and dashes ("-") become underscores ("_") in the LaTeX version of the references. So for instance, if the reference in your `rst` file `my_file.rst` in the directory `manual/my_chapter/` is `MyStrange_referenceIa`, the LaTeX reference will be `manual/my_chapter/my_file:mystrange-referenceia`.

8 Resources

8.1 Images

Xfig

fig2pdf