PropGen Documentation

Release 0

Holger Karl

February 19, 2012

CONTENTS

Contents:

CHAPTER

ONE

WHY THIS TOOL?

Writing an application for a research project is a challenging task: good ideas are needed, background research checks, a research hypothesis and a research program have to be formulated. When applying for a larger project, this is typically done by a group of people, coming from different organizations. A lot of work goes into the creative process - the mere act of writing the proposal, collecting information about the program structure, putting it in Gantt charts and tables of deliverables etc. should get out of the way!

Anybody who has tried to write a proposal for one of the European Union's Framework Programs knows that it can be a cumbersome process. The EU provides a relatively strict template which information to provide: information about work packages, tasks, deliverables, milestones, Gantt charts, etc. Much of this information is repeated at several places in the document, in various forms of presentation (tables, charts, free text). Merely keeping this information in synch can be a formidable challenge, in particular, when several people work on a proposal. To make matters worse, the EU only provides an MS Word template (and not a particularly well done one, either). There is no support to get all the administrative work out of the way.

This was the very situation we were in when we developed a proposal for a reasonably large EU proposal (an integrated project with about 15 partners). Instead of going down the Word-road, we decided to put all the information on a Wiki and to generate the actual proposal from there, using LaTeX to typeset the actual document and generating all the administrative information automatically. This has three main advantages:

- Wikis are easy to use even for novel users who are not used to using version control systems for collaborative work (let alone trying to distribute these files via email). Wikis naturally split up text in separate sections, circumventing the often problematic features of word processors to split up a document in smaller files.
- All the administrative information only needs to be entered ONCE. All possible presentations are automatically generated. They are guaranteed to stay synchronized. There is no time wasted for such work. Even non-trivial operations can be done until late in the proposal preparation without any risk (e.g., we decided to move a deliverable around a few hours before proposal submission - that would have been impossible with conventional tools).
- Wikis allow us to concentrate on the content, on our research ideas. We do not have to waste time fighting with a word processor.

To give one example: a task description for a workpackage looks as described in the figure in a moinmoin wiki. This then gets translated automatically into a Gantt chart for the workpackage (and into a Gantt chart for the project as a whole, and in tables, and in ...).

Tasks

Label	Start	Duration	Name	Lead partner
architectureDesign	1	3	Architecture Design	UE
archImprovement	4	2	Improvement of the Architecture	ABC
archImprovement	8	2	Improvement of the Architecture	ABC
archImprovement	12	2	Improvement of the Architecture	ABC
archFalsify	10	10	Falsification of the proposed Architecture	ISC

Figure 1.1: Example of a Wiki page, specifying several tasks for a workpackage (the architecture improvement task even has three phases).

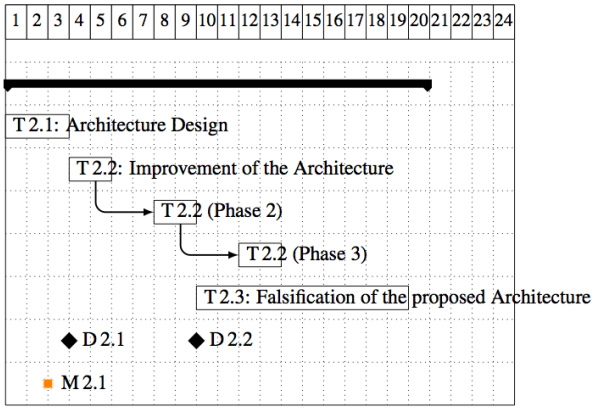

Figure 1.2: Resulting Gantt chart from the example task table (deliverable and milestones shown in this Gantt chart are defined in other Wiki tables)

Hence, the approach to go from a wiki to latex to PDF, and to submit this PDF file, has worked out nicely. It has produced a workflow that was reasonably easy for everybody, with full version control support without less IT-savy users needing to worry about it.

We felt that such a tool might be beneficial for a wider audience. So here it is - feel free to use it, to modify it, and to write interesting proposals using it. Our hope is that it will free up time from the mundane and boring tasks and enable all of us to concentrate more on the creative aspects of research.

Holger Karl

PS: When I write “we”, I refer to the team of colleagues engaged in the writing of said initial integrated research proposal. In particular, Bengt Ahlgren, Dirk Kutscher and Börje Ohlman deserve my thanks and gratitude for bearing through the rough-shot development of the inital version of this tool. I am indebted to them for constructive criticisms, ideas, and encouragement.

CHAPTER

TWO

INSTALLATION

How to install. Explain server vs. client installation. How to integrate with SVN.

CHAPTER

THREE

HOW TO USE PROPGEN

What to put on Wiki. How to trigger builds. What should go in which tables. What to watch out for when editing.

CHAPTER

FOUR

HOW TO CUSTOMIZE PROPGREN

How to customize the project.

4.1 Simple customization

Turn on, off certain parts, set colors.

4.2 Customize LaTeX templates

All the stuff that happens in template/latexTemplate.cfg

4.3 Complex customization

When you really have to work with the pyton code...

CHAPTER

FIVE

OPEN ISSUES

5.1 Known bugs

None, of course :-). If you find any, let me know!

5.2 Things still to do (TODO)

1. Put the bibtex file onto the Wiki as well. Probably better than to rely on version control to distribute it.

5.3 Debatable aspects

1. The settings files could be put on the Wiki as well. Two-edged sword: Might make it easier for everybody to configure things, but that is a serious downside as well. Technically, this would not be difficult to do. Not made up my mind yet.

5.4 Ideas for future features

1. Integrate a version control system like SVN for the produced LaTeX files
2. Generate PDF files directly from the Wiki, make it possible to trigger that at least.
3. Integrate Etherpad into Wiki
4. Build a bridge to the financial planning of a project.

- Either by parsing from/ writing to an Excel (or similar) spreadsheet. Relatively easy, but hard to make this general
- Or by putting spreadsheet-functionality onto the wiki. Hard to do for different wiki types (a nightmare, probably).

5. Build support for latexdiff. Possibly triggered from wiki as well?
6. Better support for figures in both wiki in latex. One idea might be to upload a PDF to the wiki and have the Wiki convert it to a PNG file. And the pull the PDF file directly from the wiki, without need to manually put it in the LaTeX figure directory.

CHAPTER

SIX

SOURCE CODE DOCUMENTATION

The code described here lists in the bin directory. Some general remarks:

- The invocation sequence is pullproject -> generateXML -> latexFromWiki -> latexFromXML -> ensureSymbolicLinks
- Details are in the makefile in the main directory
- Many functions get passed a parameter "config". This is the content of settings.cfg, as parsed by the standard python configuration file parser ConfigParser.SafeConfigParser (see python library documentation for details).

6.1 pullproject

Pull the raw wiki files from wherever is specified in settings.cfg. Store the raw wiki syntax in the wikipath directory.

`pullProject.`**`ensureDirectories`**(*config*)
A small helper function that makes sure that all the directories that are mentioned in settings.cfg PathNames section actually exist. This can be useful after a make clean or in case directories have been manually and inadvertently removed.

`pullProject.`**`getPartners`**(*masterPage*, *pullInstance*, *config*, *parser*, *verbose=False*)
get all the partner description files

`pullProject.`**`getProposalStructure`**(*masterPage*, *pullInstance*, *config*, *parser*, *verbose=False*)
Extract all the relevant files for the actual proposal text from the wiki.

`pullProject.`**`getWorkpackages`**(*masterPage*, *pullInstance*, *config*, *parser*, *verbose=False*)
Identify all the workpackages and download them

6.2 wikiParser

6.2.1 The wikiParser module as such

We need to parse various wiki formats into useable latex. This module provides an abstract base class wikiParser that implements a lot of basic functions e.g., to extract tables, lists, etc.

This base class has to be subclassed to specialize for specific Wiki syntax variants. The subclasses can be fairly slim and mostly specify regular expressions to use (e.g., how to recognize headings).

A factory function is called to obtain an instance of such a parser.

`wikiParser.`**`wikiParserFactory`**(*config*)
Construct an instance of the correct parser class, choice depends on what is selected in settings.cfg.

6.2.2 The wikiParser base class

class wikiParser.**wikiParser**
Base class to get the interface for turning wiki syntax into useful stuff

constructLabel(*t*)
Given a heading, construct a suitable label out of it. Remove whitspaces and obvious strange characters.

getLaTeX(*t*)
turn all of the wiki into LaTeX

getList(*wiki*)
turn the first itemize in the wiki into a list

getListAsDict(*wiki*, *delimiter=':'*)
tmp is an aray of strings, assumed to be key/values delimited by delimited split them up, return a proper dictionary for that

getSection(*wiki*, *title*, *level*)
extract the section with title at level

getTable(*wiki*)
turn the first table into list of dictionaries, using the first row as keys for the dictionaries. Removing boldfacing from the first row entries.

moveCommissionHints(*t*)
Make sure that commission hints appear after the first heading!

6.2.3 The moinmoin parser

class wikiParser.**wikiParserMoinmoin**(*config*)
Specialized for Moinmoin

getSection(*wiki*, *title*, *level*)
extract the section with title at level

6.2.4 The twiki parser

class wikiParser.**wikiParserTwiki**(*config*)
Specialized for Twiki

getSection(*wiki*, *title*, *level*)
extract the section with title at level

localHeading(*title*, *level*)
How does a heading with the given title, at the given level, look in this wiki style? Describe it as a regular expression.

6.3 latexFromWiki

latexFromWiki.**handleFile**(*f*, *outdir*, *parser*, *config*, *verbose=False*)
Translate a wiki file with name f to the corresponding LaTeX file.

Information where and how to translate are giving in config. Parser is a parser object for the correct wiki style.

6.4 latexFromXML

general docu for latexFromXML

6.5 settings

settings.**getSettings**(*filename*)
Try to find the settings file, turn it into a configParser object, and do some first preprocessing on it.

6.6 ensureSymbolicLinks

Make sure that the symbolic links from the manual to the generated subtree exist, unless there is already a real file there.

ensureSymbolicLinks.**createLinks**(*config*)
Look into config, in the Paths section, for any directory with generated as prefix, and has a corresponding manual directory as peer. Put symbolic links there if necessary to all files in generated.

6.7 utils

utils.**roundPie**(*l*)
round the values to 100%, input: list of (name, value) tuples

utils.**searchListOfDicts**(*l*, *key*, *value*, *returnkey*)
Search a list l which contains dictionaries for an entry where key has value, and return the value of returnkey.

utils.**treeReduce**(*l*, *reducefct*)
recursively apply a reduce function to a nested list structure. Atomic elements must be boolean values.

utils.**writefile**(*t*, *f*)
Write text t into file f. If flag utf8conversion is set, try to run a conversion into UTF-8.

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX