Common Lisp UltraSpec - A Project For Modern Common Lisp Documentation

Michal “phoe” Herda 2017

January 30, 2017

Abstract

TODO

1 Common Lisp UltraSpec - A Project For Modern Common Lisp Documentation

by Michal “phoe” Herda 2017

1.1 TODO Abstract

1.2 TODO Introduction

1.3 Previous work

1.3.1 ANSI CL Standard

1. Published standard

   The ANSI Common Lisp standard is the specification of the Common Lisp language, published in 1994.

   The specification itself is a written document of over one thousand pages of formatted text. Such a large amount of technical data was a natural candidate to be turned into a digital database browsable by humans.

2. Derived work

   The most famous work derived from the ANSI CL standard is the Common Lisp HyperSpec (henceforth abbreviated as CLHS) (http://www.lispworks.com/documentation/HyperSpec/Front/index.htm). It is a hyperlinked Web version of the original standard, which allows for easy navigation of the standard. Its HTML form also allows for search and quick access using external search engines, such as Google or IRC (Internet Relay Chat) bots.

   The CLHS is released under an essentially non-free license which allows verbatim copying of CLHS as a whole, but prohibits any changes to it and creating derivative works. It is therefore not possible to create a unified piece of Common Lisp documentation based on the CLHS.

1.3.2 Lisp content aggregators

The Common Lisp documentation spans far and wide beyond the Common Lisp standard. Even during the time of Common Lisp standarization, many extensions to the language existed, with their respective pieces of documentation.

From this arose the obvious need of aggregating Lisp content, both with respect to code and documentation. Below, I will outline three contemporary services which provide Lisp users with content.

1. l1sp.org

   The l1sp.org (http://l1sp.org) service is a content aggregation tool created by Zach Beane. Its main purpose is to enable lookup of symbols inside various pieces of documentation scattered around the web. It currently contains links to above 20 pieces of documentation, including the CLHS, documentation for various CL implementations and many commonly used language extensions and libraries, such as the Metaobject Protocol, ASDF and Alexandria.

This redirection service is very useful and allows for easy lookup, but such an approach depends on the presence of all the pieces of documentation in their respective places all around the Web. Also, the pieces of documentation are not linked to each other; for example, it is impossible to reach the Common Lisp reference from within the Metaobject Protocol reference (http://metamodular.com/CLOS-MOP/) and vice versa. The individual pieces of reference also greatly vary in style, which means both the typesetting and graphical layout and the textual form in which the information is presented to the reader.

2. Quicklisp

   Quicklisp (https://www.quicklisp.org/), created by Zach Beane, provides a centralized repository of Lisp libraries through a piece of Lisp code, which in turn allows the programmer to automatically resolve dependencies, download and compile a particular library on their Lisp system.

   While Quicklisp is invaluable as a library repository, it does not provide any sort of documentation service as such is out of scope of the Quicklisp project. Therefore it cannot be a direct aid in creating a Lisp documentation project.

3. Quickdocs

   The Quickdocs service (http://quickdocs.org/) is a content aggregation tool created by Eitaro Fukamachi expressly for automated collection and generation of documentation for Common Lisp libraries; therefore, it aids the issue outlined in the paragraph above by expanding Quicklisp with documentation capabilities. The documentation itself is generated automatically from the source code of libraries found in the Quicklisp repositories. It consists of a system's Quicklisp description and a list of exported symbols along with the type of objects they refer to and any documentation strings they may contain.

   Such an automation provides a very good and aesthetically pleasing means of reading about a given system's protocol. The issue with such automatic generation is, it forces the library authors to follow a convention of documenting their libraries in a particular way, which must be recognizable by the tool parsing the Quicklisp systems - otherwise, the documentation will not be visible in Quickdocs. Additionally, Quicklisp descriptions often contain little more than links to external websites documenting the code, which deprives Quickdocs of the ability to automatically generate documentation for it.

1.4 My work

1.4.1 Idea

The idea of creating a unified, hyperlinked piece of Common Lisp documentation which additionally spanned over multiple language extensions and libraries had been growing in me since I began my journey with Common Lisp back in 2015. I had been irritated by the separation of particular bodies of Lisp knowledge and lack of connection between them. In the beginning of 2016, I started looking for means to improve this situation.

During my research, it became obvious to me that - no matter which particular way would be chosen in this case - the project of creating and maintaining a modern, unified repository of Common Lisp documentation would require substantial work. It would be necessary to choose the appropriate pieces of work the repository would consist of, find most recent versions of their documentation, solve any legal issues of creating derivative works of them, parse the existing documents and keep the repository maintained in the face of the changing versions of Common Lisp libraries.

1.4.2 Requirements

The idea for building such a piece of documentation was presented at the European Lisp Symposium 2016 during a lightning talk that I gave. I would like to expand on a particular slide of that presentation, which outlines the qualities I expect of a Common Lisp documentation project.

1. Editable

   It needs to be modifiable and extensible by anyone willing to expand it.

2. Complete

   It should aim for completeness and maximizing its coverage of the Common Lisp universe.

3. Downloadable

   It should be usable locally, without an Internet connection.

4. Mirrorable/Clonable

   It should be easy to create mirrors and copies of it on the Internet and on hard drives.

5. Versioned

   It should use version control.

6. Modular

   It should be splittable into separate modules with cross-module hyperlinks breaking as the only side effect.

7. Updatable

   It should be easy to update it to its newest version.

8. Portable

   It should be exportable as a static HTML website.

9. Unified

   It should be consistent in style.

10. Community-based

    It should belong to the Lisp community and be further developed and extended there.

The implementation of this idea is a project created by me that I have named the Common Lisp UltraSpec, henceforth abbreviated CLUS.

The dpANS source makes it **editable**.

Git (https://git-scm.com/) as version control makes it **downloadable**, **mirrorable**/**clonable**, **versioned** and **updatable**.

Hosting it on GitHub (https://github.com) allows it to be **community-based**.

DokuWiki (https://www.dokuwiki.org/) allows it to be **modular** and **portable**.

The goals are - to make it **complete** and **unified**.

1.4.3 Source - dpANS CL (see below)

The whole process was made possible by the availability of the LaTeX source code for "draft preview Americal National Standard", abbreviated as dpANS, for Common Lisp. These sources were put into public domain by Kent M. Pitman and other members of the X3J13 committee.

While not being the actual standard itself, the dpANS is close enough to it to be usable as a proper reference of Common Lisp while also being in the public domain, which allows me to create derivative works of it. It turned out to be a feasible source upon which I could begin implementing the first part of the UltraSpec.

1.4.4 Work done so far

At the moment of writing these words, I have translated six dictionaries from the dpANS sources into pages in DokuWiki markup syntax, corrected the pages and hyperlinked the code examples found inside.

Additionally, I have created a customized version of DokuWiki meant for displaying the CLUS content. While I have not yet published the source code of this modified DokuWiki instance, it was successfully deployed (http://phoe.tymoon.eu/clus/) with the specification data translated so far.

I expect to have the whole sources parsed and translated before the European Lisp Symposium 2017.

1.4.5 Demonstration of used methods and tools

The presence of feasible source for creating a unified and modernized piece of Common Lisp documentation allowed me to download the sources and start looking for means of parsing and processing it. The following subchapters describe the tools I have been using and explain the reasons for them being chosen.

1. Notepad++ (https://notepad-plus-plus.org/) - the text editor

   When it came to the main editor for doing most of the parsing work, I could choose between Emacs (https://www.gnu.org/software/emacs/) and Notepad++, a pair of GPL-licensed (https://www.gnu.org/licenses/gpl-3.0.en.html) programmer's editors. (Emacs is a keyboard-oriented editor, available for all major operating systems; Notepad++ is a WYSIWYG, keyboard-and-mouse-oriented editor written for Windows that I was able to run on my Linux setup using the Wine (https://www.winehq.org/) toolkit.) I chose the latter mostly because I have been using Notepad++ for the past few years and also due to the entry threshold associated with Emacs; I am still learning this editor despite having used it for more than a year now, and I have been using it mostly as a Lisp programming environment.

2. DokuWiki - the engine for displaying HTML

   DokuWiki is a GPL-licensed wiki software written in PHP (http://php.net/). In my experience, it was able to fulfill all the requirements I had for a displaying engine: it does not need database access and instead relies on flat files, which allows me for easy versioning the data with Git; it has a simple markup syntax that I consider sane; it is extensible and hackable, which so far proves very useful; I have had some previous experience in using and configuring; and last but not least, it simply works and allows me to deliver the contents in a readable and aesthetically pleasing way, which is the most important reason.

3. Regular expressions, Unix coreutils - the tool for parsing the sources

   The most important choice that I have had to make in the beginning was, how to parse the source files of the dpANS. The source code is a large body of LaTeX code, created by multiple people over a large span of time. It contains highly customized TeX macros, used irregularly among the source code.

   The initial research led me towards TeX parsers written in various languages, such as Parsec (https://wiki.haskell.org/Parsec) written in Haskell (https://wiki.haskell.org/). My initial attempts of feeding the dpANS sources to the parsers I found were failures though; the individual bodies of code were too complex and my knowledge about these parsers was too little for me to succeed. I realized that, in order to properly parse the TeX source code of the draft, I would need to create a substantially large set of parsing rules; even afterwards, I would need to spend a lot of time doing manual polishing and fixing of the corner cases, such as TeX macros used only in a few places within the source files or actual mistakes within formatting, such as utilizing function markup for macros and vice versa.

   Because of this, I decided to abandon the approach of parsing the standard with a parser capable of processing TeX directly and instead go for a simpler choice: utilizing a set of regular expressions to parse a subset of utilized TeX macros and formatting. It would mean later polishing the preprocessed data by hand, though I would like to note that this last step would be necessary anyway regardless of the technique used.

   My editor of choice, Notepad++, contained a powerful enough RegEx engine that was capable of guiding me through the process. Various bulk edits were also made through the assorted unix utilities: grep, sed, awk, rename.

4. Git - versioning system, GitHub - project hosting

   The data for the whole project is kept in a Git repository, stored at GitHub (https://github.com/phoe/clus-data) and publicly available. Because DokuWiki keeps all data as flat text files, I can easily modify and deploy new versions of data to upstream websites.

1.4.6 Problems encountered

Most of the problems I have encountered are connected with the dpANS sources being a big and complicated piece of documentation and usage of regular expressions to parse the TeX sources.

As I have mentioned before, the source code had been created over a lengthy period of time with multiple people contributing to it. Because of that, many parts of the specification are formatted differently: they utilize different TeX macros, specific to the people creating the source and the part of the language that was worked upon. Despite the irregularities, I was able to employ the regular expressions and capabilities of my editor to fix most of the cases globally and fix the corner cases manually.

A significant part of the required work was hyperlinking. Although I was able to parse the code for TeX glossary entries, I also needed to take the English grammar into account, such as plural and past forms of glossary entries.

I have had some minor problems with DokuWiki's rendering and markup capabilities, though none of them have been significant enough to be mentioned in detail here.

1.5 Conclusions and future work

1.5.1 Benefits/Disadvantages

The benefits of my approach come as logical continuations of the slogans used in section **Requirements** TODO FIX REFERENCE.

The most obvious one, which is also the goal of the project, is the construction of a contemporary source of Common Lisp documentation and a singular resource capable of containing most of the knowledge a Common Lisp programmer might need.

Another upside is modernization of the specification by fixing its issues and bugs, expanding its examples sections, clarifying any inconsistencies and questions that have emerged since the creation of the standard and giving it a more aesthetically pleasing look.

A beneficial side effect of my approach is generation of a version of the Common Lisp specification in a markup format. Such a format can then be easily parsed by automated tools to produce a document of any required typesetting qualities.

The disadvantages of my current approach occur on different layers.

First of all, it is easy to keep a single static website on the Web for years without any changes, but CLUS is far from static because of its design. The body of code that CLUS will turn into, as the time progresses, will require maintenance in order to stay clear and readable; it will require reviewers to check the input from anyone wanting to contribute to the CLUS repositories.

Second, although it does apply specifically to the dpANS sources, parsing and hyperlinking the chapters of the specification takes significant time. Additionally, because of the variety of forms other bodies of Lisp documentation have, it will be non-trivial to import them into CLUS - it will require separate effort to have them parsed and prepared for inclusion.

Third, the legal status and licensing issues of the various pieces of documentation will require separate thought. Creating a compilation work of all these elements will be essentially creating a derivative of them all and legal caution will need to be taken in case of documents with unknown or confusing legal status. It might be required to negotiate the terms of inclusion of particular pieces of work into CLUS with the respective holders of rights to them.

1.5.2 Thoughts

Among all the literature available for studying Common Lisp, I would like to mention the the dpANS source files as a valuable read from a non-technical point of view.

The standard was created before the era of ubiquitous versioning systems. Because of this, the draft source contains many comments, some of them timestamped. They show the technical problems and decisions the langauge specifiers faced and solved in the process of creating a formal standard for a programming language. They alsooutline the features which were deprecated and removed - or, on the contrary, created and added along the way, some of which I personally find quite enlightening. What I want to emphasize here, though, is that they show X3J13 as a group of human beings working on a common goal. The comments there show various aspects of their work: from communicating messages between particular people, through decision-making and commented-out pieces of specification itself, to the in-jokes and humor of the people.

In my opinion, studying the original sources for all three draft previews (all of which are available online) might be valuable for any person who wants to research specification development or software development in general from a more humane point of view as well as Lisp programmers who are interested in extending their background and the process through which Common Lisp came to life.

Another thought that I would like to mention here is the fact that, in the beginning, I had imagined my work as simple translation of the sources from their TeX format into wiki markup in order to let the DokuWiki engine format them into HTML. Reality has verified these ideas - I quickly realized that the standard itself has its share of inconsistencies, bugs and other issues. It is of course expected for such a huge body of documentation to have issues and these issues do not undermine the value of the specification as a whole, but I have unexpectedly found myself to be able to fix them as I progress through the sources.

Suddenly, from a simple translator, I had become an editor of the Common Lisp standard itself. What I am creating right now is not the draft sources being translated into DokuWiki markup - it is an edited version which contains many improvements and fixes to many issues that were impossible to fix in the previous CL specifications based on the work of X3J13.

It is a very responsible role that has emerged - but also one that I consider very satisfying.

1.5.3 Plans

It is impossible to speak of future plans without mentioning the Lisp community here.

The Common Lisp UltraSpec was meant from the start to be a community-based project, meaning that it belongs to the Lisp community and is meant to be utilized and expanded within it. I hope that other people will aid me in my process by suggesting changes, submitting patches, possibly integrating the documentation for respective Common Lisp libraries into the code and maintaining them later on.

Once the specification is completely integrated, I intend on extending its scope to include common facilities and extensions included and/or used in most contemporary Common Lisp implementations, such as the Metaobject Protocol, ASDF, Quicklisp and the compatibility libraries which provide cross-platform functionalities not included in the standard such as parallelism or networking.

I want to create quality standards for the respective types of pages and enforce them in order to keep the quality of the documentation high and its style consistent across pages and modules.

1.6 TODO Bibliography