DocVar: Manage and use document variables

Zunbeltz Izaola and Paulo Ney de Souza

1 Introduction

In book production, we are frequently faced with the problem of using and reusing the same information in various locations of the products. Text strings like the title, author, ISBN, ... may appear in the book cover (in the front cover or the spine) and also in the colophon, as well in the text itself.

It is also desirable to be able to have "place holders" for this kind of information in templates that are used to produce each book of a collection. In the production of each book, ideally, the document will read these data from a database (or from an intermediate file derived from a database).

Figure 1 shows two books, members of the "Coleção Professor de Matemática" collection. You can see how they share the same design, while the first one shows one more piece of information: a subtitle. These two covers are generated from the same LaTeX file, but loading different metadata files.

Figure 2 shows a full cover of a book in the "Coleção Projecto Euclides" collection. It shows other kinds of metadata handled in a similar way (ISBN, title in the spine, ...).

2 Usage

The aim of our package is to facilitate the use of such "place holders" in a document, by making it easier to create, validate and use a document variable that is loaded from an external file.

At the beginning, we planned to call this package `metadata` because we are dealing with information that varies from book to book in a collection (author(s), title, date, subtitles, ...) and formatting variations (font size of title, subtitle, author(s), ...). But the name `metadata` has a rather specific meaning in the world of documents and there is already a package called `metadata` on CTAN. Therefore, the name `Document Variable` or `DocVar` for short, was selected.

The usage of document variables is done in three steps:

Define document variable Typically a class will define several `docvar`s that individual documents, applying the class, will set and use.

Set value of document variable The value of all `docvar`s used by a document is set; typically in a separate file that is included in the document.

Use value of document variable The `docvar` is replaced by the value to which it has been set.

2.1 Define document variables

Each `docvar` is defined by a unique ⟨*key*⟩. This ⟨*key*⟩ is the mandatory argument of `\definedocvar`. There are optional arguments that control the behaviour of the `docvar`.

The syntax for the `\definedocvar` macro is: `\definedocvar[`⟨*option*⟩`]{`⟨*key*⟩`}`. Table 1 lists all the options planned for the `\definedocvar` macro. At the time of writing, not all the options are yet implemented.

The `docvar` can be of different types: integer, float, string, length. Some variables can have multiple values and they will be treated as list-type variables (for example, a book may have multiple authors). It is possible to transform the value by applying a macro; e.g., in some book designs the title is uppercase. The value of a `docvar` may be defined by "inheritance" from another variable: In most cases the name of authors printed on the spine of the book will be the same as on the cover, but sometimes the names should be modified (space limitations, design). It may be useful to define different "error levels" if the variable is empty. The error levels are the same as described for the variable validation. The `DocVar` package defines a mechanism to validate the values give to the each key. See section 2.4 for more details.

2.2 Set document variables

The macro sets the value of a previously defined `docvar`. The intent is to set the value of the `docvar`s in a file loaded by a document, or alternatively to use the macro in the document `.tex` file itself.

The `\setdocvar` macro has two mandatory arguments, the `docvar` key and the value: `\setdocvar{`⟨*key*⟩`}{`⟨*value*⟩`}`.

2.3 Use document variables

The macro `\getdocvar[`⟨*option*⟩`]{`⟨*key*⟩`}` retrieves the value of the `docvar`. In general, it will mean to *print* the value of the `docvar`, but `docvar`s can also be used to set arguments of other macros.

`\getdocvar` accepts one option, `transform`; its value is a macro to be applied to the value of the ⟨*key*⟩. This transformation is applied after any other transformation defined by `\setdocvar`.

2.4 Data validation

The process of validating data may be complex. On one hand we can validate the value of isolated ⟨*key*⟩s, while on the other hand, we can validate the correctness of a value related to the value of other ⟨*key*⟩s. For example, if a `docvar` represents a zip code, we can validate its format (as a single key value), but

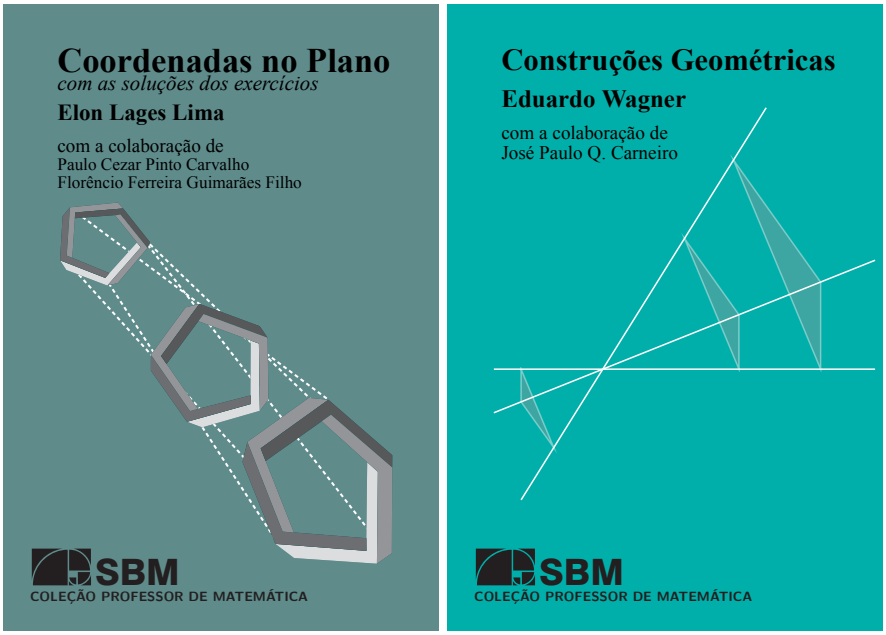

Figure 1: Two examples of covers from the CPM collection.

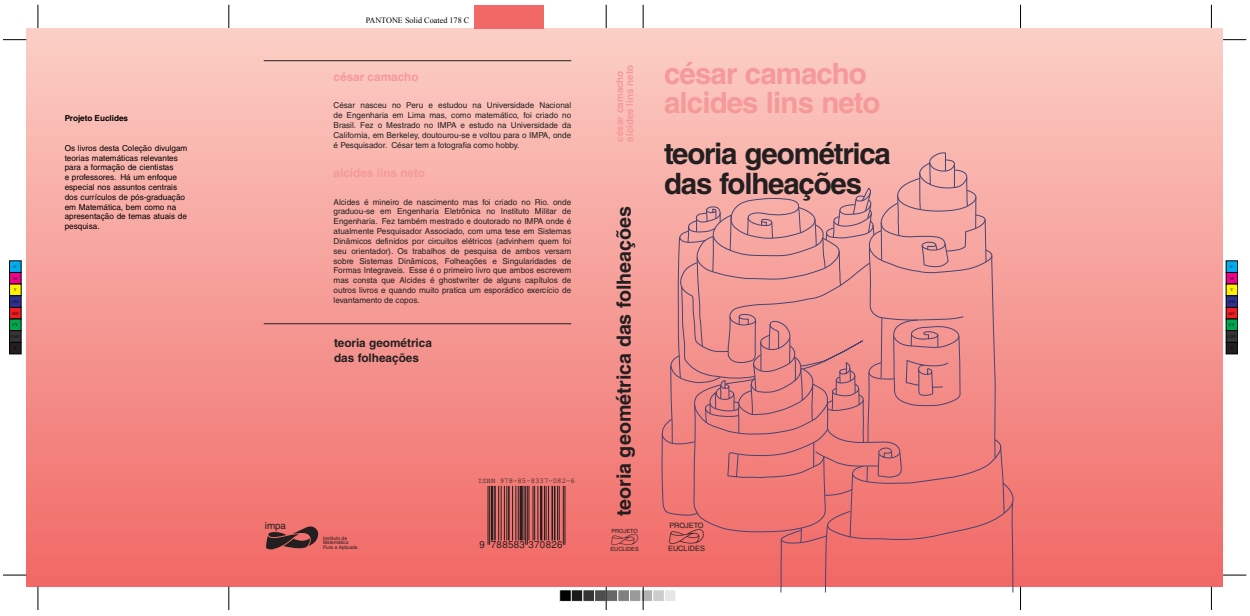

Figure 2: A full cover produced using DocVar

Option	value	Implemented?	Definition
`type`	`integer`, `float`, `string`, `length`	No	Type of the variable
`multiple`	`true`, `false`	No	Set to true if the `docvar` has multiple values
`empty`		No	Behaviour of `\usedocvar` when value is empty
`inherit`	⟨*key*⟩	Yes	`docvar` from which value may be inherited
`transform`	macro	Yes	Always transform value before using it

Table 1: Options of the `\definedocvar` macro.

may also be validated in relation to a "state" `docvar`, to which it is related.

Each validation has an "error level". There are five levels:[1]

none Nothing happens.

info Information is logged in the log file.

warning A warning message is logged to the terminal and the log file.

error An error message is logged to the terminal and the log file, and an error mark is shown in the document.

critical After issuing a critical error, TEX will stop reading the current input file. This may halt the TEX run (if the current file is the main file) or may abort reading a sub-file.

fatal After issuing a fatal error the TEX run halts.

The validation is defined with the macro:
`\setvalidation[`⟨*error*⟩`]{`⟨*keys*⟩`}{`⟨*validation*⟩`}`

The macro's first required argument ⟨*keys*⟩ is a comma-separated list; the list can have just one element. This is the list of related ⟨*keys*⟩ which will be validated together. The second mandatory argument is the macro to do the actual test. The optional argument is the *error level* associated to the test. The default value is `error`.

The package will provide several basic validation tests. The user-defined ⟨*validation*⟩ macro should accept as many arguments as ⟨*keys*⟩ are listed and it should return a boolean; true if the validation is passed and false if it fails. The ⟨*validation*⟩ macro will receive its arguments in the order given in ⟨*keys*⟩.

Some validations may be too complex to be programmed efficiently in LATEX. Examples of using external scripting languages (Lua, Bash, Perl, Python) will be provided.

When a `docvar` is used with the `\getdocvar` command, the package will execute all the validations containing the corresponding ⟨*key*⟩.[2]

3 Availability

The `DocVar` package is licensed under the LPPL, and copyrighted by Books in Bytes. The first public version should appear soon on CTAN; for development, see the repository at `https://gitlab.com/booksinbytes/docvar`.

◇ Zunbeltz Izaola
Durango
Spain
`zunbeltz (at) gmail dot com`

◇ Paulo Ney de Souza
Berkeley, CA
USA
`pauloney (at) gmail dot com,`
`paulo (at) berkeley.edu`
`http://booksinbytes.com/`

[1] These error levels are modelled after the `l3msg` package error levels.

[2] This may not be efficient because the same validation will be run several times. But it seems to be the only way to show error messages close to the point where the `docvar` is used.