Seventh LACCEI Latin American and Caribbean Conference for Engineering and Technology (LACCEI'2009)
"Energy and Technology for the Americas: Education, Innovation, Technology and Practice"
June 2-5, 2009, San Cristóbal, Venezuela.

Using Doxygen to generate Spanish and English documentation

Adolfo Di Mare

Escuela de Ciencias de la Computación e Informática – Universidad de Costa Rica
adolfo.dimare@ecci.ucr.ac.cr

ABSTRACT

We describe how to use Doxygen to generate specifications for C++ modules both in English and Spanish. The method is simple and also facilitates maintaining documentation in several languages, which helps module and program internationalization.

Keywords: documentation, data hidding, software engineering, implementation, abstraccion.

Globalization has brought programmers together into a large family that produces every day millions of lines of code. As it has happened many times over, if the program under construction will be used only in rare instances there is a strong possibility that its documentation will not be written. Although at first glance this looks like savings, the developer will soon discover that module specification facilitates his work and shortens the time needed to build programs. The reason for this reality is simple: whoever does not know where he goes, winds up elsewhere (Di Mare, 2007). It is also important to incorporate test data definition in each module specification, as discussed in (Di Mare, 2008).

When in here we speak about "languages" it should be understood as "spoken languages" as it is always assumed that programs are written in C++, but the documentation method for multiple languages described in here can be generalized to other computer languages in which the profesional documentation tool Doxygen is used (van Heesch, 2005). Doxygen uses embedded tags in the program source, which are similar to JavaDoc's: `/** Sorts the array */`. It generates HTML pages and Latex or RTF documents. The generated documentation highlights clases, methods and functions, and its options help tailoring it to different needs.

1. DOCUMENTATION BLOCKS FOR EACH LANGUAGE

When building programs it is natural for the programmer to write his specifications in his native language or English, which is the standard language for internationalized projects. Here English is used as the base language and Spanish as the other work language.

When the Doxygen configuration file includes the `PREDEFINED` option to define a macro, Doxygen uses it to generate the documentation. Using this Doxygen capability it is possible to define a block of documentation for each language, putting the documentation text surrounded by a `#ifdef` - `#endif` pair corresponding to that language.

```
#ifdef English_dox
/// Defined by the C++ standard library.
#endif
#ifdef Spanish_dox
/// Definido por la biblioteca estándar C++.
#endif
namespace std { } // trick to include it into the Doxygen documentation
```

Figure 1

Figure 1 contains the documentation in English and Spanish for namespace "`std`". In this example, macro "`English_dox`" marks the English documentation block, while macro "`Spanish_dox`" marks the Spanish documentation block. Here we see that documenting style may become very heavy, because to document the simple line "`namespace std { }`" 6 lines of documentation were necessary. It is therefore natural to try to prevent that so much documentation make program source code unreadable: for most programmers is easier to document the main sources in English, and leave in a separate file the Spanish documentation.

```
// CSV.h    (C) 2008 adolfo@di-mare.com
// ...

#ifdef English_dox
/** Deletes \c ch when it is the trailing character in \c str.
    - The deleted character always is \c ch.

    \dontinclude test_CSV.cpp
    \skipline    test::chop()
    \until       }}
    \see         test_CSV::test_chop()
*/
#endif
void chop( std::string & str , char ch=0 );

// ...
// EOF: CSV.h
```

Figure 2

Figure 2 is a piece of code taken from file "`CSV.h`" that contains the specification for function "`chop()`". To avoid putting the documentation in Spanish within the same file "`CSV.h`" it is necessary to include it into another file.

File "`CSV.es.h`" shown in Figure 3 does not contain C++ statements and therefore it is never used to compile the program: it only contains statements about whatever Doxygen is documenting. Because everything being documented belongs to other files, as "`CSV.h`", each documentation segment must declare what it documents. In this case, where what is being documented is function "`chop()`", it is necessary to use the "`\fn`" Doxygen command with the exact signature of the documented function. The short description in Spanish for this function appears after the "`\brief`" Doxygen command.

```
// CSV.es.h  (C) 2008 adolfo@di-mare.com

#ifdef Spanish_dox
    /// Documentación en español.
    #define Spanish_dox "Documentación en español"
   ///  \def  Spanish_dox    ///<  Marca  bloques  de  documentación  en
español.
#endif
// ...

/** \fn    void chop( std::string & str , char ch=0 );
    \brief Elimina \c ch si es el último caracter de \c str.
    - El caracter eliminado siempre es \c ch.

    \dontinclude test_CSV.cpp
    \skipline    test::chop()
    \until       }}
    \see         test_CSV::test_chop()
*/

// ...
// EOF: CSV.es.h
```

Figure 3

2. CONFIGURATION FILES FOR EACH LANGUAGE

"`CSV.cpp`" is a library to read and write files in CSV (Comma Separated Value) format (Shafranovich, 2005). The specifications for the 2 main functions in this library include examples that are test data taken form test program "`test_CSV.cpp`" using "`BUnit.h`" (Di Mare, 2008). To compile the test program it is necessary to use a project that includes these 3 files: "`CSV.cpp`" + "`CSV_line.cpp`" + "`test_CSV.cpp`". The compiler also processes the "`.h`" header files corresponding to those sources: "`CSV.h`" + "`CSV_line.h`".

However, to generate the Doxygen English documentation it is not necessary to use the Spanish documentation files. Therefore, in the Doxygen configuration file used to generate the English documentation the files listed are the same files that the compiler processes:

- "`CSV.h`" + "`CSV_line.h`"
- "`CSV.cpp`" + "`CSV_line.cpp`" + "`test_CSV.cpp`"

In addition, in the Doxygen configuration file macro "`English_dox`" must be declared for Doxygen to extract the English documentation form the blocks marked by this macro:

```
PREDEFINED = ... "English_dox" ...
```

To obtain the Spanish documentation it is necessary to use a different Doxygen configuration file. One way to do this is to create a file with extension "`.es.h`" for each English source file: "`CSV.h`" → "`CSV.es.h`" and "`CSV_line.h`" → "`CSV_line.es.h`" It's best to use the "`.es.h`" instead of "`.h.es`" so that text editors process these files with syntax highlighting. It is unfortunate that if these documentation files are mentioned, Doxygen will includes them as part of the documented files. That is why it seems a better solution to combine all these Spanish documentation files in one large file called "`Spanish.txt`":

```
X:\SubDir\CSV> copy CSV_line.es.h+CSV.es.h Spanish.txt
```

Using this trick, all the Spanish documentation will be in file "`Spanish.txt`" for Doxygen to extract if the configuration file "`Spanish.dxg`" contains these 2 sentences:

```
INPUT      =  ... Spanish.txt ...
PREDEFINED = ... "Spanish_dox" ...
```

In this situation, the English documentation gets built using this command:

```
X:\SubDir\CSV> doxygen test_CSV.dxg
```

To build the Spanish documentation the command is the following:

```
X:\SubDir\CSV> doxygen Spanish.dxg
```

3. COMPARISON OF ENGLISH VS. SPANISH CONFIGURATIONS

```
Comparing files test_CSV.dxg and Spanish.dxg
***** test_CSV.dxg
PROJECT_NAME           = "CSV:"
OUTPUT_LANGUAGE        = English
OUTPUT_DIRECTORY       = .
***** Spanish.dxg
PROJECT_NAME           = "CSV:"
OUTPUT_LANGUAGE        = Spanish
OUTPUT_DIRECTORY       = .
*****
```

```
***** test_CSV.dxg
INPUT                  = CSV.cpp      CSV_line.cpp test_CSV.cpp \
                         CSV.h        CSV_line.h
RECURSIVE              = NO
***** Spanish.dxg
INPUT                  = CSV.cpp      CSV_line.cpp test_CSV.cpp \
                         CSV.h        CSV_line.h                \
                         Spanish.txt
#                        copy CSV_line.es.h+CSV.es.h Spanish.txt
RECURSIVE              = NO
*****

***** test_CSV.dxg
PREDEFINED             = "DOXYGEN_COMMENT" \
                         "English_dox" \
                         "__cplusplus" \
***** Spanish.dxg
PREDEFINED             = "DOXYGEN_COMMENT" \
                         "Spanish_dox" \
                         "__cplusplus" \
*****
```

Figure 4

The differences between the 2 Doxygen configuration files shown in Figure 4 can be obtained using the "`FC.exe`" command :

```
X:\SubDir\CSV> fc test_CSV.dxg Spanish.dxg
```

4. FINAL DETAILS

At the beginning of header file "`CSV.h`" both macro "`English_dox`" and macro "`Spanish_dox`" are used. Even though this is not necessary for Doxygen to document correctly, it is nice to mention that documentation is available in these 2 languages (although it is possible to use more languages).

```
// CSV.h    (C) 2008 adolfo@di-mare.com

#ifdef English_dox
    /// Doxygen English documentation.
    #define English_dox   "Doxygen English documentation"
   /// \def English_dox  ///< Macro used to have Doxygen generate English doc.
#endif
#ifdef Spanish_dox
    /// Documentación en español.
    #define Spanish_dox "Documentación en español"
   /// \def Spanish_dox  ///< Macro usado para que Doxygen genere en español.
#endif

// ...
// EOF: CSV.h
```

Figure 5

Sometimes the documentation has only one line, as it happens when documenting the class fields. In these cases, it is necessary to put the documentation outside the class declaration, because leaving it inside makes source code harder to read.

```
#ifdef English_dox
/// Uses \c getNextCSV() to scan and store complete CSV lines.
/// - Oftentimes it helps a lot to \c chop() out trailing CR+LF's.
/// - This class is derived from std::vector and has
```

```
///   all vector operations and interfaces.
#endif
class CSV_line : public std::vector {
public:
    CSV_line();
    CSV_line( const std::string & str );
    CSV_line( const char *pz_str , size_t n=0 );
    CSV_line( const CSV_line & );
    void setData( const std::string & str );
    void setData( const char *pz_str , size_t n=0 );
    friend class test_CSV;
};

#ifdef English_dox
/// \class   test_CSV;
/// \brief   Tests cases for \c CSV_line.
#endif

#ifdef English_dox
/// Constructor.
#endif
inline CSV_line::CSV_line() { }
```

Figure 6

In the example in Figure 6 the documentation for "`test_CSV`" is outside the class, just like the constructor's, as leaving it inside stains the code a lot.

```
class CSV_line : public std::vector {
public:
    #ifdef English_dox
    /// Constructor.
    #endif
    CSV_line();
    CSV_line( const std::string & str );
    CSV_line( const char *pz_str , size_t n=0 );
    CSV_line( const CSV_line & );
    void setData( const std::string & str );
    void setData( const char *pz_str , size_t n=0 );
    #ifdef English_dox
    /// Tests cases for \c CSV_line.
    #endif
    friend class test_CSV;
};
```

Figure 7

In Figure 7 its easy to see how charged the source code becomes when multi-language Doxygen documentation blocks get included. Therefore, in general, it's a good idea to put the documentation outside the class.

5. CONCLUSIONS

It is well known that Doxygen is an ideal tool for documenting because it is used in several and valuable projects. With a little effort it is possible to maintain documentation for several languages without staining much the code. The way to do it includes:

- Marking code blocks with a macro for each language such as "`English_dox`" for English and "`Spanish_dox`" for Spanish.
- In the main file goes is the documentation in the mother language, which usually is English.
- In other files goes the documentation for other languages.
- In many cases it is better to document fields, methods and functions outside the class, to avoid overloading the source.

- Despite the fact that each documentation block must be surrounded by the `#ifdef - #endif` pair corresponding to the language, the resulting code is readable.

REFERENCES

Di Mare, Adolfo (2007): Uso de Doxygen para especificar módulos y programas, Reporte Técnico ECCI-2007-02, Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica, 2007.

http://www.di-mare.com/adolfo/p/Doxygen.htm

Di Mare, Adolfo (2008): BUnit.h: Un módulo simple para aprender prueba unitaria de programas en C++, Reporte Técnico ECCI-2008-02, Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica, 2008.

http://www.di-mare.com/adolfo/p/BUnit.htm

Shafranovich, Y. (2005): Common Format and MIME Type for Comma-Separated Values (CSV) Files, Request for Comments: 4180, IETF The Internet Engineering Task Force, Octubre 2005.

http://tools.ietf.org/html/rfc4180

van Heesch, Dimitri (2005): Doxygen, 2005.

http://www.doxygen.org/index.html

SOURCE CODE

CSV.zip:

- http://www.di-mare.com/adolfo/p/CSV/CSV.zip
- http://www.di-mare.com/adolfo/p/CSV/es/index.html
- http://www.di-mare.com/adolfo/p/CSV/en/index.html

getNextCSV() & setQuotedCSV():

- http://www.di-mare.com/adolfo/p/CSV/es/CSV_8cpp.html
- http://www.di-mare.com/adolfo/p/CSV/en/CSV_8cpp.html

CSV.h:

- http://www.di-mare.com/adolfo/p/CSV/es/CSV_8h-source.html
- http://www.di-mare.com/adolfo/p/CSV/en/CSV_8h-source.html

test_CSV.cpp:

- http://www.di-mare.com/adolfo/p/CSV/es/classtest__CSV.html
- http://www.di-mare.com/adolfo/p/CSV/en/classtest__CSV.html

Authorization and Disclaimer

Authors authorize LACCEI to publish the paper in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.