The PracTEX Journal, 2010, No. 1
Article revision 2010/01/01

# DOXYGEN e LATEX2ε: As definitivas ferramentas para documentar seu código-fonte

Francisco Reinaldo[1], Maria Tereza de Castro Costa[2], Tiago Faria Bicalho[3], and Victor Vasconcelos Moreira[4]

Email [1]reinaldo.opus@gmail.com, [2]maryxb@gmail.com, [3]tiagofariabicalho@gmail.com, [4]victorvasconcelosfox@gmail.com

Resumo In this paper we present how programmers can document source-code and have updated reports during the elaboration/implemention phase. We focus mainly on two tools specifically aimed at this purpose: DOXYGEN and LATEX2ε.

## 1 Introdução

Profissionais do desenvolvimento de sistemas computacionais comumente documentam seus códigos-fonte ou parte deles somente após o término da implementação. Infelizmente, este tipo de decisão não acompanha a real preocupação de manutenibilidade no desenvolvimento de programas e faz com que o programa em desenvolvimento ou a documentação gerada não esteja sincronizada. Outro ponto a ressaltar é que a documentação acontece longe dos arquivos de código-fonte, o que inviabiliza sua manutenção e confunde a equipe de planejamento com informações antigas.

Este artigo apresenta duas ferramentas para solucionar o problema no sincronismo da documentação de programas e a compatibilidade da ferramenta de apoio. Primeiramente tem-se DOXYGEN como um gerador de documentação de software junto ao desenvolvimento e/ou atualizações de códigos-fonte e LATEX2ε como formatador tipográfico adequado. Com DOXYGEN e LATEX2ε, a documentação já formatada para apresentação como relatório final é automaticamente gerada pela extração dos comentários do programador/analista e em paralelo durante a implementação do código-fonte. Os membros do Laboratório de Inteligência Computacional (LIC), do Curso de Computação - Sistemas de Informação,

do Centro Universitário do Leste de Minas Gerais (UnilesteMG, Brasil), utilizam estas ferramentas durante a implementação de código para geração de relatórios atualizados.

## 2 Revisão de Literatura

TEX é um excelente sistema de tipografia desenvolvido por Donald E. Knuth para produção de livros, artigos e relatórios que exigem alta qualidade tipográfica. TEX é de fato um processador de macros que também possui uma poderosa flexibilidade para programação. LATEX2$\varepsilon$ é um conjunto de macros de TEX que implementam um sistema de preparação de documentos. O LATEX2$\varepsilon$ define uma linguagem de marcação de mais alto nível e que permite descrever o documento em termos de sua estrutura lógica e não apenas do seu aspecto visual. Usando diferentes classes de documentos e pacotes adicionais, o usuário de LATEX2$\varepsilon$ pode produzir uma grande variedade de leiautes. A primeira versão de LATEX2$\varepsilon$ largamente utilizada foi a 2.09, lançada em 1985[2] por Leslie Lamport a partir do TEX desenvolvido por Donald Knuth[1].

Noutra vertente, Doxygen é uma ferramenta geradora de documentação que oferece os recursos necessários para documentar os diferentes aspectos do código-fonte implementado através dos comentários dos programadores/analistas nas linguagens de programação `C/C++/C#`, `Java`, `IDL` (Corba, Microsoft, e KDE-DCOP flavors) e para algumas extensões como PHP. Doxygen funciona em plataformas UNIX, Microsoft e Macintosh OS X. Doxygen é o tipo de ferramenta que pode ajudar o programador a documentar seu código no formato *online* em HTML e manuais de referência *offline* no formato PDF conectado através do LATEX2$\varepsilon$. A documentação é extraída diretamente do código-fonte, deixando a documentação consistente dentro do código-fonte. Para maiores informações, a documentação original do sistema Doxygen bem como a aplicação estão disponíveis *online* no endereço `http://www.doxygen.org/`.

# 3 Desenvolvimento

## 3.1 Inserindo os comentários no código-fonte com Doxygen

A função primária do Doxygen é extrair comentários definidos no código-fonte e gerar relatórios ou manuais. Nesta mesma linha de raciocínio o LaTeX2$\varepsilon$ é utilizado para criar um manual *offline* de referência de boa qualidade e mulitplataforma. Quando unidas, as ferramentas Doxygen e LaTeX2$\varepsilon$ oferecem relacionamento entre os vários elementos que podem ser visualizados através de gráficos de dependências, gráficos hierárquicos e diagramas de colaboração que podem ser gerados automaticamente.

Para que os comentários possam ser localizados e extraídos do código-fonte pelo Doxygen e formatados para LaTeX2$\varepsilon$, as informações para o Doxygen precisam ser referenciadas por comandos especiais. As formas mais comuns para definir comentários em linha para o Doxygen são aqueles que usam blocos de comentário. Assim, não é necessário o caracter * antes dos comandos do Doxygen ao iniciar e terminar um bloco de comentários, apesar de autores utilizarem este sinalizador para uma clara legibilidade do código-fonte, como seguem os exemplos abaixo.

**Exemplo 1**: Para comentários de linha simples.

```
/** comentário. **/
/// comentário. ///
/*! comentário. !*/
//! comentário. !//
```

**Exemplo 2**: Para comentários multi-linha:

```
/**
 * comentário
 */
```

Dessa forma, o Exemplo 2, de estrutura de comentário acima apresentado, abre espaço para uma descrição mais detalhada, reconhecendo uma série de comandos do Doxygen. Os comandos a serem inseridos nesta estrutura tem o formato `@comando` ou `\comando`. Por exemplo, o comando `\brief` permite o programador definir uma descrição compact em sua linha de código.

Nesta primeira etapa criamos as seções da documentação, tal como apresenta abaixo:

```
/**
\mainpage (Nome do sistema)
\image (Insere imagem na documentação)
\section (Insere seção, para comentários)
\file (Insere nome do arquivo fonte principal)
\brief (Insere comentário referente ao último comando)
\author (Insere autor)
\version (Insere versão do sistema)
\date (Insere data de criação)
*/
```

A seguir, dois exemplos simples de código-fonte em linguagem C são apresentados. O primeiro exemplo, ver *Listing* 1, apresenta o excerto de código do arquivo `structcmd.h`. Já o segundo exemplo, ver *Listing* 2, apresenta a forma declarada da documentação do código nos padrões do DOXYGEN. Optamos por manter a originalidade do texto, para estes *Listings* que se seguem.

Listing 1: Código Base

```
1 #define MAX(a,b) (((a)>(b))?(a):(b))
2 typedef unsigned int UINT32;
3 int errno;
4 int open(const char *,int);
5 int close(int);
6 size_t write(int,const char *, size_t);
7 int read(int,char *,size_t);
```

Segue abaixo o mesmo código-fonte do *Listing* 1, mas obedecendo os padrões de comentários estabelecidos pelo DOXYGEN para se construir a documentação do código.

Listing 2: Código Base com comentários

```
1 /*! \mainpage Esta é uma aplicação teste.
2  \section Introdução simples e resumida de um programa.
3  \section Seção para explicar os procedimentos de instalação.
4  \subsection Passo 1: instalar.exe:  Apenas clique em instalar.
```

```
 5  \author Tiago Faria Bicalho
 6  \version 1.0.0.0
 7  \date 03/12/2009
 8  \bug Não existem Bugs
 9   \warning Aviso que esta aplicação está ok.
10 */
11
12 /*! \file structcmd.h
13  \brief A Documented file.
14 */
15
16 /*! \def MAX(a,b)
17  \brief A macro that returns the maximum of \a a and \a b.
18 */
19 #define MAX(a,b) (((a)>(b))?(a):(b))
20
21 /*! \var typedef unsigned int UINT32
22  \brief A type definition for a .
23 */
24 typedef unsigned int UINT32;
25
26 /*! \var int errno
27  \brief Contains the last error code.
28  \warning Not thread safe!
29 */
30 int errno;
31
32 /*! \fn int open(const char *pathname,int flags)
33  \brief Opens a file descriptor.
34  \param pathname The name of the descriptor.
35  \param flags Opening flags.
36 */
37 int open(const char *,int);
38
39 /*! \fn int close(int fd)
40  \brief Closes the file descriptor \a fd.
41  \param fd The descriptor to close.
42 */
43 int close(int);
44
45 /*! \fn size_t write(int fd,const char *buf, size_t count)
46  \brief Writes \a count bytes from \a buf to the filedescriptor \a fd.
```

```
47  \param fd The descriptor to write to.
48  \param buf The data buffer to write.
49  \param count The number of bytes to write.
50 */
51 size_t write(int,const char *, size_t);
52
53 /*! \fn int read(int fd,char *buf,size_t count)
54  \brief Read bytes from a file descriptor.
55  \param fd The descriptor to read from.
56  \param buf The buffer to read into.
57  \param count The number of bytes to read.
58 */
59 int read(int,char *,size_t);
```

Segue abaixo a explicação de cada comando acima utilizado:

- `\param` Define o parâmetro da função.
- `\brief` Define o comentário para o comando.
- `\fn` Define as funções.
- `\var` Define as variáveis.
- `\var typedef` Define as variáveis e qual módulo ela pertence.
- `\def` Define as variáveis ou funções do comando `#define`.
- `\file` Define o arquivo lido.

## 3.2 Ajustando a codificação do Arquivo Fonte

Por padrão, o DOXYGEN gera relatórios `.tex` na codificação UTF8, assim é importante atentar-se a codificação dos arquivos fonte. Caso estes não estejam em UTF8 estes devem ser convertidos utilizando algum editor específico.

## 3.3 Ajustando o Idioma

O DOXYGEN permite a documentação na língua portuguesa. Para isto basta selecionar na aba "Expert" em "Project" e encontrar a opção OUTPUT_LANGUAGE, selecione a língua portuguesa e então vá em "Run" para iniciar a documentação, o sistema irá gerar a documentação com êxito, mas irá advertí-lo de uma possível falha.

```
Warning: The selected output language "portuguese" has not been
updated since release 1.3.3. As a result some sentences may appear in
English.
```

Esta mensagem indica que a codificação para língua portuguesa no formato Brasil não é atualizada desde a versão 1.3.3 e que algumas palavras ainda podem aparecer em inglês. Contudo podemos contornar este problema. Após gerar a documentação em LaTeX2$\varepsilon$, abra o arquivo principal denominado `refman.tex`, localize:

```
\usepackage[utf8]{inputenc}
Portuguese
```

e substitua por

```
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[brazil]{babel}
```

# 4 Gerando Manuais de Referência em LaTeX2$\varepsilon$

A geração de um manual de referência em LaTeX2$\varepsilon$ é simples e pode acontecer em duas diferentes etapas, tais como a etapa *Wizard* e a etapa *Expert* - para ambas, ver Figura 1. A etapa *Wizard* é designada para usuários com pouca ou nenhuma experiência com Doxygen. Nesta etapa, o processo de extração de comentários já está previamente configurado. Na guia apresentada, Figura 1, únicos campos a serem preenchidos são o nome do sistema, versão, o caminho do diretório contendo os arquivos fonte e o caminho do diretório onde o Doxygen irá gerar a documentação.

Entretanto, a etapa *Expert* necessita de maiores conhecimentos para alterar algumas configurações, tais como codificação, tipo de papel e outras.

Para acompanhar a evolução do Doxygen, durante a leitura do código-fonte, você pode escolher a opção “Run” - ver Figura 2.

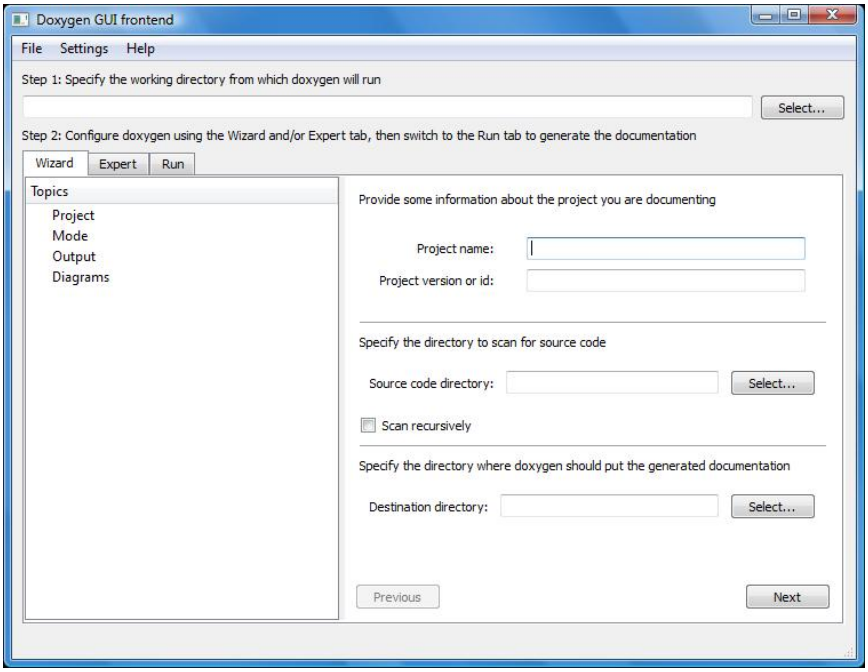


Figura 1: GUI para gerar relatórios.

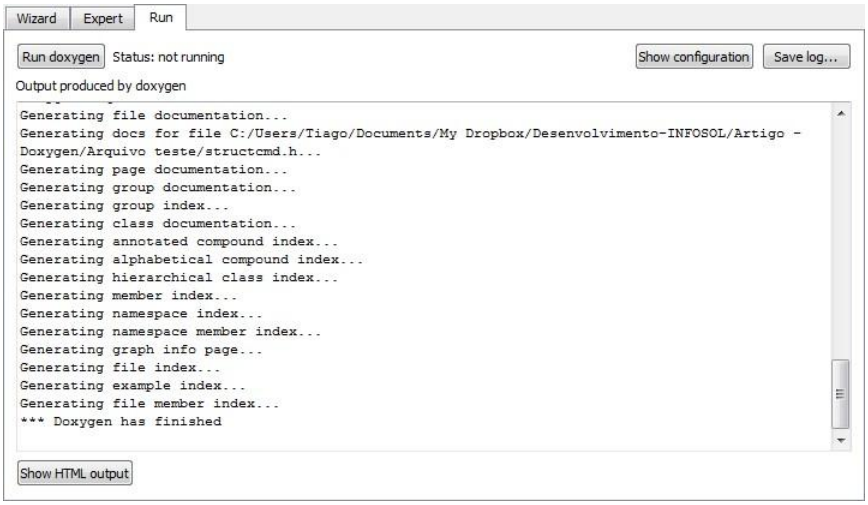


Figura 2: Leitura na geração de relatório.

# 5 Conclusão

O sucesso na documentação de um código-fonte proporciona uma melhoria no processo de desenvolvimento de software. Analistas de sistemas terão uma documentação precisa e fiel durante as atividades e/ou atualizações que foram realizadas no processo de implementação.

Ao documentar um código-fonte, utilizando as formatações do DOXYGEN, podemos perceber que o arquivo fonte tem a tendência de aumentar o número de linhas. Por outro lado, a documentação é embutida junto ao código e com apenas alguns cliques, esta documentação estará atualizada de acordo com as alterações no código além de não correr o risco de perdermos a documentação uma vez que esta faz parte do sistema, ou por incompatibilidade com editores de texto WYSIWYG (*What You See Is What You Get*).

# 6 Agradecimentos

Os autores agradecem aos professores Delaine Vasconcelos Rosa e José Geraldo Costa pela revisão gramatical final.

# Referências

[1] Reginaldo J. Santos. Introdução ao latex. *Departamento de Matemática-ICEx Universidade Federal de Minas Gerais*, page 68, 2002.

[2] Klaus Steding-Jessen. Latex: Uma alternativa mais eficiente comparada aos sistemas wysiwyg. 1998.

# APÊNDICE

# Programa Exemplo Doxygen e LaTeX

1.0

Gerado por Doxygen 1.6.1

Thu Dec 17 17:17:33 2009

# Sumário

# Capítulo 1

# Indice dos Arquivos

## 1.1 Lista de Arquivos

Lista de todos os ficheiros documentados com uma breve descrição:

# Capítulo 2

# Documentação do Arquivo

## 2.1 Referência ao Arquivo C:/Users/Tiago/Desktop/Teste/structcmd.h

A Documented file.

### Macros

- #define MAX(a, b) (((a)>(b))?(a):(b))

  *A macro that returns the maximum of a and b.*

### Definições de tipos

- typedef unsigned int UINT32

  *A type definition for a .*

### Funções

- int open (const char *, int)

  *Opens a file descriptor.*

- int close (int)

  *Closes the file descriptor fd.*

- size_t write (int, const char *, size_t)

  *Writes count bytes from buf to the filedescriptor fd.*

- int read (int, char *, size_t)

  *Read bytes from a file descriptor.*

## Variáveis

- int errno
  *Contains the last error code.*

### 2.1.1 Descrição detalhada

A Documented file. Details.

### 2.1.2 Documentação das macros

#### 2.1.2.1 #define MAX(a, b) (((a)>(b))?(a):(b))

A macro that returns the maximum of *a* and *b*. Details.

### 2.1.3 Documentação dos tipos

#### 2.1.3.1 typedef unsigned int UINT32

A type definition for a . Details.

### 2.1.4 Documentação das funções

#### 2.1.4.1 int close (int *fd*)

Closes the file descriptor *fd*.

**Parâmetros:**

***fd*** The descriptor to close.

#### 2.1.4.2 int open (const char * *pathname*, int *flags*)

Opens a file descriptor.

**Parâmetros:**

***pathname*** The name of the descriptor.
***flags*** Opening flags.

#### 2.1.4.3 int read (int *fd*, char * *buf*, size_t *count*)

Read bytes from a file descriptor.

**Parâmetros:**

***fd*** The descriptor to read from.
***buf*** The buffer to read into.
***count*** The number of bytes to read.

#### 2.1.4.4 size_t write (int *fd*, const char * *buf*, size_t *count*)

Writes *count* bytes from *buf* to the filedescriptor *fd*.

**Parâmetros:**

***fd*** The descriptor to write to.

***buf*** The data buffer to write.

***count*** The number of bytes to write.

### 2.1.5 Documentação das variáveis

#### 2.1.5.1 int errno

Contains the last error code.

**Aviso:**

Not thread safe!