

Título do Documento		Projeto
<b>Tutorial do Doxygen</b>		
Escrito/Atualizado por		Data
<b>Eric Sonagli Abbade</b>		
Revisado por	Assinatura	Data
Aprovado por	Assinatura	Data

## 1. Objetivo

Este documento tem como objetivo ser um **tutorial de uso e instalação do Doxygen** para documentação de códigos. Foco será em documentações no formato HTML, para códigos C e Python. Esse tutorial é consideravelmente detalhado, mas não cobre todas as funcionalidades do Doxygen, por sua complexidade. Para isso, se dividiu o documento nos seguintes tópicos:

2. Índice;
3. Introdução;
4. Download de Softwares;
5. Configurando Doxywizard;
6. Geração de Documentação pelo terminal;
7. Abas da Documentação;
8. Observações para documentação de funções (Python e C);
9. Conclusão.

Primeiramente, serão mostrados exemplos de outros tutoriais do Doxygen e de uma documentação gerada, além de uma breve introdução (seção 3). Será também explicado como fazer os downloads necessários (seção 4). Após isso, será ensinado como gerar a documentação, pelo terminal e Doxywizard (seções 5 e 6). Algumas das possíveis diferentes abas da documentação, serão representadas abaixo:

[Página Principal](#)   [Páginas relacionadas](#)   [Tópicos](#)   [Namespaces ▼](#)   [Estruturas de dados ▼](#)   [Arquivos ▼](#)

Essas abas serão explicadas na seção 7.I, como: Página Principal, Páginas Relacionadas, Aba de Arquivos, Tópicos, Namespaces e Estrutura de Dados. Também, serão mostradas outras configurações na seção 7.II: Observações para documentação (Python e C). No final será feita a conclusão (seção 9). **Tudo isso será exemplificado por ilustrações.**

Esse tutorial busca ensinar diferentes formas de documentações, para que o leitor se adeque a seu propósito. Seções 1 a 5 já permitem uma documentação razoavelmente complexa (de forma rápida). Seções 7 e 8 possibilitam uma documentação mais complexa, mas demandam um maior tempo de uso. Outros tutoriais completam ainda mais a documentação.

## 2. Índice

3. Introdução.....	3
4. Download de Softwares.....	12
5. Geração de Documentação pelo terminal.....	13
6. Configurando Doxywizard.....	15
I.    Wizard-Project.....	16
II.   Wizard-Mode.....	17
III.  Wizard-Output.....	18
IV.   Wizard-Diagrams.....	21
V.    Expert-Diagrams.....	22
VI.   Run (executar).....	30
VII.  Salvando configurações .....	31
VIII. Gerando Doxyfile.....	32
7. Abas da Documentação.....	33
I.    Página Principal.....	33
i.  Página principal por comentários (C).....	34
ii. Exemplo de Página principal por comentários (C).....	39
iii. Página principal por comentários (Python).....	40
iv. Exemplo de Página principal por comentários (Python).....	40
v.  Página Principal por arquivos separados.....	41
II.   Páginas Relacionadas.....	43
i.  Páginas relacionadas por comentários nos códigos.....	43
ii. Exemplo em C de páginas relacionadas.....	44
iii. Exemplo em Python de páginas relacionadas.....	45
iv. Páginas relacionadas por arquivos separados.....	46
III.  Aba de arquivos.....	48
i.  Início da aba de arquivo.....	48
ii. Diagramas de funções.....	51
iii. Comentários de funções.....	53
iv. Páginas de arquivos.....	57
IV.   Tópicos.....	59
V.    Namespaces.....	62
VI.   Estrutura de Dados.....	64
8. Observações para documentação de funções (Python e C).....	65
I.    Doxygen e C na documentação de funções.....	65
II.   Doxygen e Python no acionamento de funções.....	66
III.  Doxygen e Python para programação orientada a objeto.....	68
9. Conclusão.....	72

### 3. Introdução

Doxygen é uma ferramenta para a geração de documentação de códigos. **Esse tutorial irá focar em documentações geradas no formato HTML, para códigos em diversas linguagens (especialmente C e Python).** Diversas discussões a respeito, da razão de seu uso, vantagens e desvantagens, além de tópicos correlatos foram feitas no seguinte documento: [https://cnpemcamp.sharepoint.com/:w:/r/sites/sei/\\_layouts/15/Doc.aspx?sourcedoc=%7B076f257b-491f-4f27-bf99-b7989d3cf79b%7D&action=edit&wdPreviousSession=fd54f04c-66a6-310c-45a0-45cea3300d03](https://cnpemcamp.sharepoint.com/:w:/r/sites/sei/_layouts/15/Doc.aspx?sourcedoc=%7B076f257b-491f-4f27-bf99-b7989d3cf79b%7D&action=edit&wdPreviousSession=fd54f04c-66a6-310c-45a0-45cea3300d03).

Esse último documento, porém, não cobre os detalhes técnicos da instalação e do uso do Doxygen. Por isso, esse tutorial tem como objetivo, justamente, atuar como um guia sobre os detalhes técnicos da instalação e do uso desse software. Dessa forma, ele complementa o documento mencionado no último link.

É importante ressaltar, como mencionado anteriormente, que devido à complexidade do Doxygen, esse documento não é capaz de cobrir todas as suas funcionalidades técnicas. Como o Doxygen é uma ferramenta muito usada atualmente, é possível encontrar diversas funcionalidades suas, pesquisando em sites de busca como o Google. Por isso, serão aqui mencionados alguns outros tutoriais para que o leitor possa buscar outras fontes:

- Resumo (**próprio site do Doxygen**): <https://www.doxygen.nl/manual/starting.html>
- Vídeo tutorial da documentação: <https://www.youtube.com/watch?v=RI50qI6e7HU>
- Documentação feita anteriormente: [SIMAR: SIMAR Software \(cnpem-emi.github.io\)](https://github.com/cnpem-emi/SIMAR)

Visando atingir o objetivo mencionado, primeiramente, é necessário fazer os downloads dos softwares. Por isso, na seção 4 será explicado como fazer os downloads do Doxygen e do GraphViz. Apesar desse tutorial ser focado no Doxygen, o GraphViz é um importante auxiliar ao próprio Doxygen. Ele permite que o Doxygen seja capaz de ter funcionalidades como a geração dos diagramas de funções (comentado na seção 7.III.ii).

Após isso, será comentado sobre como fazer as configurações padrões do Doxygen (seções 5 e 6). É possível fazer as configurações do Doxygen com o terminal de comandos, pela edição de parâmetros de um arquivo (seção 5). Na seção 6 será descrito como fazer as configurações pelo Doxywizard, que é um aplicativo baixado junto ao Doxygen. Ele possui uma interface gráfica de uso mais intuitivo. Por isso, esse tutorial focou mais em mostrar as configurações feitas por ele.

**Somente com essa edição de configurações (seções 5 e 6), feita após os downloads da seção 4, já é possível ter uma documentação significativamente detalhada.** A vantagem disso, é que esses procedimentos só precisam ser feitos 1 vez para todas as documentações desejadas. Portanto, após essa configuração inicial, é possível gerar essa documentação significativamente detalhada, de forma quase instantânea.

Será agora mostrado um exemplo de uma documentação gerada somente com as configurações feitas até a seção 5. Configurações da seção 5 podem ser substituídas pelas configurações da seção 6, já que essas são equivalentes. Agora será mostrado um exemplo de documentação. Esse exemplo é constituído pelo código fonte (arquivo de entrada) e imagens da documentação gerada. Um exemplo de código (Python) usado, é dado por:

```
import time
import random

var_state=False
var_reset=False

def F1():
    #Avisa que o status da operação é verdadeiro.
    #Gera um valor aleatório para contador.
    global var_state
    var_state=True
    F2(random.randrange(1,11)) #Números aleatórios de 1 a 10.

def F2(randon_counter):
    #Atualiza o contador para um valor aleatório (se for maior que o atual).
    #Depois, verifica se o contador já está em seu valor máximo (10).
    #Caso não esetja, ela o incrementa (desde que o status seja verdadeiro).
    #Além disso, gera o delay de 2 s e acionar a função F3.
    global var_counter
    global var_reset
    global var_state
    print("Variável Aleatória:", randon_counter)
    if randon_counter>var_counter:
        var_counter=randon_counter
    print("Variável Counter:", var_counter)
    if var_counter==10:
        var_reset=True
        F4()
    else:
        time.sleep(2)
        if var_state==True:
            var_counter=var_counter+1
            print("Variável Counter (pós acréssimo):", var_counter)
        F3()

def F3():
    global var_counter
    global var_reset
    #Função que identifica se o contador deve ser resetado (se for igual a
    10).
```

```

#Também, aciona a função F4.
if var_counter==10:
    var_reset=True
else:
    var_reset=False
print("Variável Reset:", var_reset)
F4()

def F4():
    global var_reset
    global var_counter
    #Reseta o contador caso isso seja demandado pela variável reset.
    if var_reset==True:
        var_counter=0
        var_reset=False

var_counter=0

while True:
    print("Novo ciclo")
    F1()
    time.sleep(5)

```

É importante ressaltar que o código acima é meramente ilustrativo, simulando processos comuns em programação, mas não exerce nenhuma aplicação prática específica. Por isso, ele não foi escrito da forma mais enxuta possível, já que sua função era somente facilitar a compreensão da documentação do leitor. Esse código acima terá suas funcionalidades brevemente descritas para facilitar a compreensão do leitor.

Existe um laço `while True` que aciona a função `F1()` a cada 5 segundos. Essa função `F1()` atualiza a variável `var_state` para verdadeiro a fim de sinalizar que a operação está começando, além de enviar um valor aleatório entre 1 e 10 para a função `F2()`.

Já essa outra função `F2()`, primeiramente, identifica se o valor aleatório é maior que o contador (`var_counter`) atual. Então, a variável `var_counter` irá assumir o maior valor entre os dois. Caso, essa variável seja igual a 10, será acionado a função `F4()`, após se ativar o reset (`var_reset` passa a ser verdadeiro). Se não, após o tempo de 1 segundo, o contador `var_counter` será atualizado em 1 e a função `F3()` será acionada. Isso é feito para simular um processo em que não se pode alterar a variável `var_counter` duas vezes em seguida, muito rápido.

Quando a função `F3()` é acionada, ela identifica se o contador `var_counter` é igual a 10, e caso seja, ativa o reset (`var_reset` passa a ser verdadeiro). Depois, a função `F4()` é acionada. Isso é feito para que a função `F4()` identifique se deve, ou não, zerar o controlador.

Por fim, quando a função `F4()` é acionada ela identifica se a variável (`var_reset`) é verdadeira, e caso seja reseta o contador (zerando-o). A variável `var_reset` é ativada sempre que o contador chega a 10. Dessa forma, se em qualquer momento do código o contador for igual a

10, a função `F4()` irá zerá-lo. Assim, o código simula um procedimento, no qual o controlador não pode ter um valor maior que 10.

Esse código acima, gerou a seguinte documentação representada nas figuras abaixo. Essas figuras são referentes a página obtida após escolhida a aba “Namespaces” e selecionado a opção do arquivo usado. Elas podem ser visualizadas abaixo:

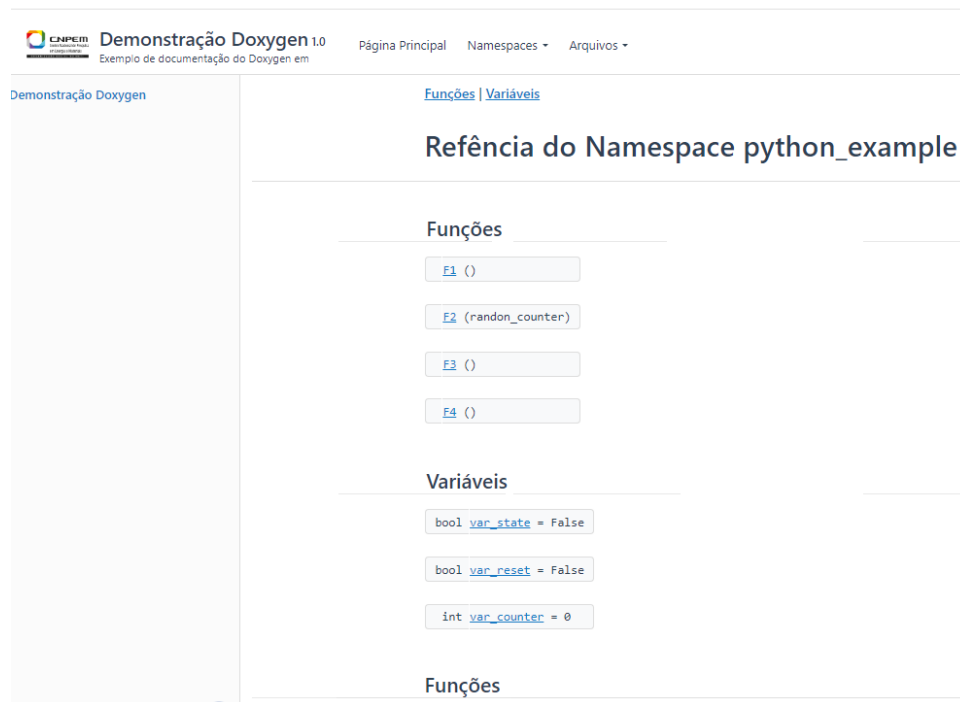


Figura 1: Primeira Imagem do Exemplo de Documentação.

## Funções

### ◆ F1()

F1 ( )

Definição na linha [7](#) do arquivo [python\\_example.py](#).



```
def F1():
    8 #Avisa que o status da operação é verdadeiro.
    9 #Gera um valor aleatório para contador.
    10 global var_state
    11 var_state=True
    12 F2(random.randrange(1,11)) #Números aleatórios de 1 a 10.
    13
    14
```

Este é o diagrama das funções utilizadas por essa função:



Figura 2: Segunda Imagem do Exemplo de Documentação.

### ◆ F2()

F2 ( *random\_counter* )

Definição na linha [15](#) do arquivo [python\\_example.py](#).



```
def F2(random_counter):
    16 #Atualiza o contador para um valor aleatório (se for maior que o atual).
    17 #Depois, verifica se o contador já está em seu valor máximo (10).
    18 #Caso não esetja, ela o incrementa (desde que o status seja verdadeiro).
    19 #Além disso, gera o delay de 2 s e acionar a função F3.
    20 global var_counter
    21 global var_reset
    22 global var_state
    23 print("Variável Aleatória:", random_counter)
    24 if random_counter>var_counter:
    25     var_counter=random_counter
    26 print("Variável Counter:", var_counter)
    27 if var_counter==10:
    28     var_reset=True
    29     F4()
    30 else:
    31     time.sleep(2)
    32     if var_state==True:
    33         var_counter=var_counter+1
    34         print("Variável Counter (pós acréssimo):", var_counter)
    35     F3()
    36
```

Este é o diagrama das funções utilizadas por essa função:

Figura 3: Terceira Imagem do Exemplo de Documentação.

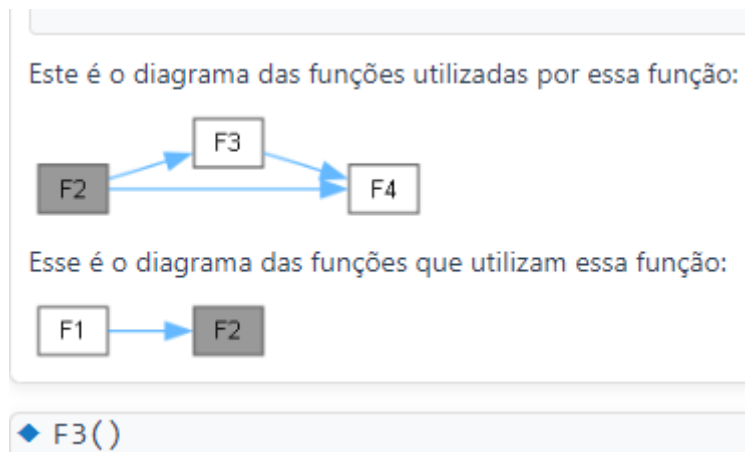


Figura 4: Quarta Imagem do Exemplo de Documentação.

◆ F3()

F3 ( )

Definição na linha [37](#) do arquivo [python\\_example.py](#).

```

37 def F3():
38     global var_counter
39     global var_reset
40     #Função que identifica se o contador deve ser resetado (se for igual a 10).
41     #Também, aciona a função F4.
42     if var_counter==10:
43         var_reset=True
44     else:
45         var_reset=False
46     print("Variável Reset:", var_reset)
47     F4()
48 
```

Este é o diagrama das funções utilizadas por essa função:

```

graph LR
    F3[F3] --> F4[F4]
  
```

Esse é o diagrama das funções que utilizam essa função:

```

graph LR
    F1[F1] --> F2[F2]
    F2 --> F3[F3]
  
```

◆ F4()

Figura 5: Quinta Imagem do Exemplo de Documentação.



◆ F4()

F4 ( )

Definição na linha [49](#) do arquivo [python\\_example.py](#).

```

49 def F4():
50     global var_reset
51     global var_counter
52     #Reseta o contador caso isso seja demandado pela variável reset.
53     if var_reset==True:
54         var_counter=0
55     var_reset=False
56

```

Esse é o diagrama das funções que utilizam essa função:

```

graph LR
    F1 --> F2
    F2 --> F3
    F2 --> F4
    F3 --> F4

```

## Variáveis

Figura 6: Sexta Imagem do Exemplo de Documentação.

## Variáveis

### ◆ var\_counter

```
int var_counter = 0
```

Definição na linha [57](#) do arquivo [python\\_example.py](#).

### ◆ var\_reset

```
bool var_reset = False
```

Definição na linha [5](#) do arquivo [python\\_example.py](#).

### ◆ var\_state

```
bool var_state = False
```

Definição na linha [4](#) do arquivo [python\\_example.py](#).

Figura 7: Sétima e Última Imagem do Exemplo de Documentação.

Essas setes imagens servem como referência, para que os leitores saibam o que esperar de uma documentação básica do Doxygen. Uma explicação detalhada de como essa geração é feita será na seção 5 desse tutorial. Na primeira das 7 imagens acima é possível ver outras abas opções diferente dos “Namespaces”, como a página principal e arquivos. Essas abas são existem em qualquer documentação. Outras documentações, podem ter mais abas, como:

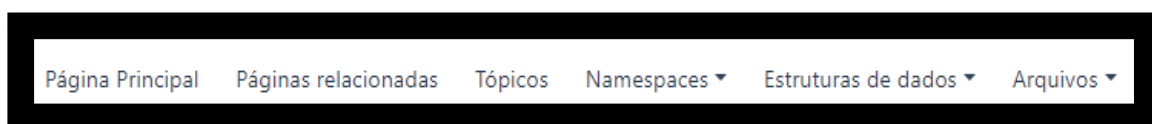


Figura 8: Abas de outras documentações.

Na figura acima, pode-se observar diversas abas geradas na documentação do Doxygen. Essas abas serão comentadas na seção 7. Quando a documentação é aberta, a tela inicial é a Página Principal. Somente com as configurações das seções 5 e 6, ela vem praticamente vazia, porém será descrito como preencher ela na seção 7.I.

Apesar da utilidade das representações acima, existem casos em que será necessário o desenvolvimento de uma documentação mais detalhada. Essa necessidade de se gerar documentações mais (ou menos) detalhadas varia conforme cada projeto. Portanto, cabe ao leitor identificar qual a real necessidade de detalhamento em seu próprio projeto.

É possível utilizar diversos recursos do Doxygen, para gerar uma documentação mais detalhada. Alguns desses recursos podem ser encontrados nos outros tutoriais mencionados anteriormente, ou até mesmo procurando em sites de busca como o Google. Outros desses recursos serão mencionados nesse documento, tais como as seções 7 a 12.

O formato de arquivo HTML, usado como padrão para a documentação nesse tutorial, permite o acesso de diversas páginas. Uma importante funcionalidade do Doxygen é a criação da página HTML principal (descrita na seção 7.I). Enquanto isso, também é possível gerar outras páginas HTML, da forma como desejado. Essas outras páginas serão denominadas: Páginas Relacionadas (descritas na seção 7.II). Na “Aba de arquivos” (seção 7.III) será comentado como gerenciar as páginas de cada um dos arquivos de código fonte.

Conforme as configurações definidas anteriormente, a documentação HTML pode ter outras abas, além das abas da página principal, páginas relacionadas e aba dos arquivos. Algumas dessas abas são as abas de Tópicos, Namespaces e Estrutura de Dados (serão comentadas nas seções 7.IV, 7.V e 7.VI). **Normalmente, essas abas (como as Páginas Relacionadas) não aparecem na documentação. Elas só aparecem, quando se colocam os comandos necessários** (que serão mencionados nas seções 7.II, 7.III, 7.IV e 7.V) nos arquivos fontes.

Existem ainda outros tipos de abas, mas que não serão comentados nesse tutorial, por fugir de seu escopo. Elas podem ser acessadas na parte superior da tela, como nas figuras que serão

mostradas abaixo (as vezes também na lateral esquerda, conforme as configurações definidas na seção 6.III).

Doxygen gera documentações de diversas linguagens, sendo que cada uma tem diversos comandos e bibliotecas. Por isso, é possível que parte da documentação não apareça como previsto, ao se usar comandos diferentes desse tutorial. Sendo assim, **é necessário conferir a documentação**, para evitar imprecisões. Caso, a documentação não esteja como planejada, é preciso um **estudo para que se encontre uma solução específica** a esse problema. É muito difícil um único tutorial (a exemplo desse próprio) cobrir todos os casos possíveis.

Apesar da dificuldade de se tratar com cada um desses casos em específico, foi feito um estudo prévio de algumas particularidades da documentação de códigos Python e C. Esse estudo não cobre todas as particularidades da documentação de códigos Python e C (já que existirem várias). No entanto, ele serve como um auxílio para leitores que estejam interessados em tal linguagem e foi descrito na seção 8). Finaliza-se o tutorial com a conclusão na seção 9.

## 4. Download de Softwares

### Instruções de Instalação e uso Doxygen (e GraphViz):

- Download do Doxygen no link: <https://www.doxygen.nl/download.html>
- Após o download: aceitar tudo e instalar.
- Download do GraphViz no link: <https://graphviz.org/download/>
- Após o download: aceitar tudo e instalar.
- Observação: download do GraphViz é necessário para que os documentos gerados pelo Doxygen tenham mais funcionalidades, como por exemplo: diagramas do fluxo de conexão entre as funções.
- Abrir aplicativo Doxywizard:

Após o download do Doxygen, o Doxywizard pode ser encontrado dentro dos arquivos baixados, conforme pode ser observado na figura abaixo:

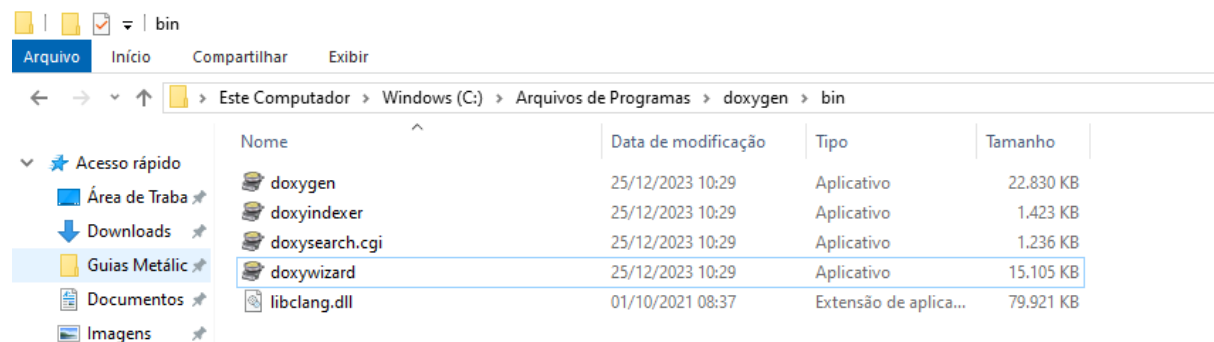


Figura 9: Local do Doxygen.

## 5. Gerando a documentação pelo terminal:

Uma maneira de se criar a documentação é pelo terminal de comandos, usando o comando: `doxygen \file\` (\file\ se refere ao caminho completo do arquivo com as instruções para a tarefa). Para isso, deve-se abrir o terminal que contenha o aplicativo Doxygen (que provavelmente é o mesmo terminal do aplicativo Doxywizard).

Isso pode ser feito, ao se digitar o comando: `cd /caminho/`, no terminal de comandos, sendo que: `/caminho/` se refere ao caminho do diretório. No caso deste computador, primeiro se apertou o comando: `cd ..`, duas vezes para que se entre no diretório `C:\` e depois o comando usado foi:

```
cd Program Files\doxygen\bin
```

Cada usuário deve substituir os comandos, com seus respectivos diretórios. Após entrar no diretório, basta usar o comando: `doxygen \file\`, substituindo `\file\` pelo caminho do arquivo com as configurações. Esse autor usou o seguinte comando:

```
doxygen C:\Users\eric.abbade\Downloads\doxyfile.txt
```

O arquivo tem várias opções de configurações. Esse tutorial focou no processo de geração da documentação pelo Doxywizard, por ter se julgado mais intuitivo, além de que o arquivo `doxyfile.txt` é muito grande. Logo, não irá se comentar sobre cada uma de suas configurações.

**É importante ressaltar que os comandos do Doxywizard podem gerar um arquivo `doxyfile.txt`, de forma relativamente intuitiva.** Por isso, para a criação de um desses arquivos totalmente “do zero” sugere-se a leitura da próxima seção que será relativa ao Doxywizard. **Após isso, é possível usar as mesmas configurações do Doxywizard para criar o arquivo `doxyfile.txt` a ser usado no terminal.**

Será referenciado um exemplo de arquivo usado (com configurações similares as que serão mostradas no Doxywizard) para utilização dos usuários interessados: [Doxygen/doxyfile.txt at master · cnpem-emi/Doxygen \(github.com\)](https://github.com/cnpem-emi/Doxygen/blob/master/cnpem-emi/Doxygen/doxyfile.txt). Outro exemplo é dado por: <https://github.com/cnpem-emi/simar-sw/blob/master/docs/Doxyfile>. Ambos os arquivos são do formato txt. Usuários podem substituir neles os parâmetros desejados. **Como várias configurações são similares ao Doxywizard, o tutorial da seção 6 serve como base de alguns dos principais parâmetros que podem ser editados em tal arquivo.**

Visando uma breve introdução ao usuário sobre o arquivo: duas de suas configurações mais importantes são as dos locais de entrada e saída. É importante que se edite a seção: `OUTPUT_DIRECTORY` com o local do diretório no qual se deseje que a documentação seja gerada. Por exemplo:

```
OUTPUT_DIRECTORY = C:\Users\eric.abbade\Downloads\DoxyFile_Output
```

Também, é importante que se edite a seção: INPUT, na qual se determina os arquivos de entrada. É necessário colocar o diretório local completo de cada uma das entradas. Vale ressaltar que caso não se preencha o diretório de um dos arquivos, será assumido que eles estão no mesmo diretório aberto no terminal. Um exemplo da definição de entradas é dado por:

```
INPUT                      = C:\Users\eric.abbade\Downloads\simar-sw-master\INPUT1  
C:\Users\eric.abbade\Downloads\simar-sw-master\INPUT2
```

No caso de mais de uma entrada pode-se colocar elas espaçadas, como no exemplo acima, dos diretórios: INPUT1 e INPUT2. Por fim, basta usar o comando mencionado para gerar a documentação no diretório de saída mencionado.

## 6. Configurar Doxywizard:

Doxywizard é um aplicativo capaz de gerar a documentação dos códigos. Uma outra maneira de se criar a documentação é pelo terminal de comandos, conforme comentado na seção anterior.

Serão comentadas algumas **vantagens e desvantagens** de cada um dos métodos mencionados. Doxywizard (que será comentado nessa seção) apresenta vantagens em relação a geração da documentação pelo terminal (seção anterior), por ser mais intuitivo. Apesar disso, uma desvantagem do Doxygen com relação ao outro método é que a geração da documentação por terminal pode ser mais rápida para ser feita (caso exista um arquivo configurado), além de que seu arquivo doxyfile.txt pode ser facilmente enviado para outros computadores.

Portanto, como ambos os métodos são válidos para gerar a documentação e possuem vantagens, em relação ao outro, esse tutorial irá explicar os dois métodos. Apesar disso, as configurações mais específicas deles serão mostradas no método do Doxywizard, por ser julgado mais intuitivo. Esse método é considerado mais simples por ele ter uma interface para definir as configurações.

Enquanto isso, o método da geração pelo terminal depende da definição dos parâmetros em um arquivo grande, o que pode ser menos intuitivo. Porém, como ambas as configurações são similares, é possível editar parâmetros parecidos no doxyfile.txt para o uso da geração de arquivos pelo terminal.

Doxywizard tem 3 abas principais: “Wizard” (configurações gerais), “Expert”(configurações específicas) e “Run” (execução). Será comentado primeiramente sobre a aba “Wizard”. Essa aba Wizard possui 4 “seções”: “Projeto”, “Modo”, “Saída” e “Diagramas”. Para melhor descrição da configuração serão mencionadas as configurações dessas 4 seções da aba “Wizard”, começando pela seção de projetos.

## I. Wizard-Project

Defina o diretório de trabalho (local onde Doxygen está armazenado), nome do projeto, diretório do código fonte (código do qual se deseja gerar a documentação) e diretório de destinação (local no qual a documentação será gerada). É importante ressaltar que **na seção de código fonte, deve-se escolher um diretório (pasta) contendo os arquivos**, sobre os quais se buscam gerar a documentação. É possível acrescentar outros diretórios fontes (de entrada), conforme será explicado na seção 6.V.

Esses parâmetros são fundamentais para o funcionamento do Doxygen, porém para uma documentação mais detalhada, existem outros parâmetros que também podem ser preenchidos. Exemplos desses outros parâmetros são: versão, sinopse, logo (imagem que será colocada em destaque, no caso se escolheu o logo do CNPEM). Todos esses parâmetros podem ser vistos na imagem abaixo:

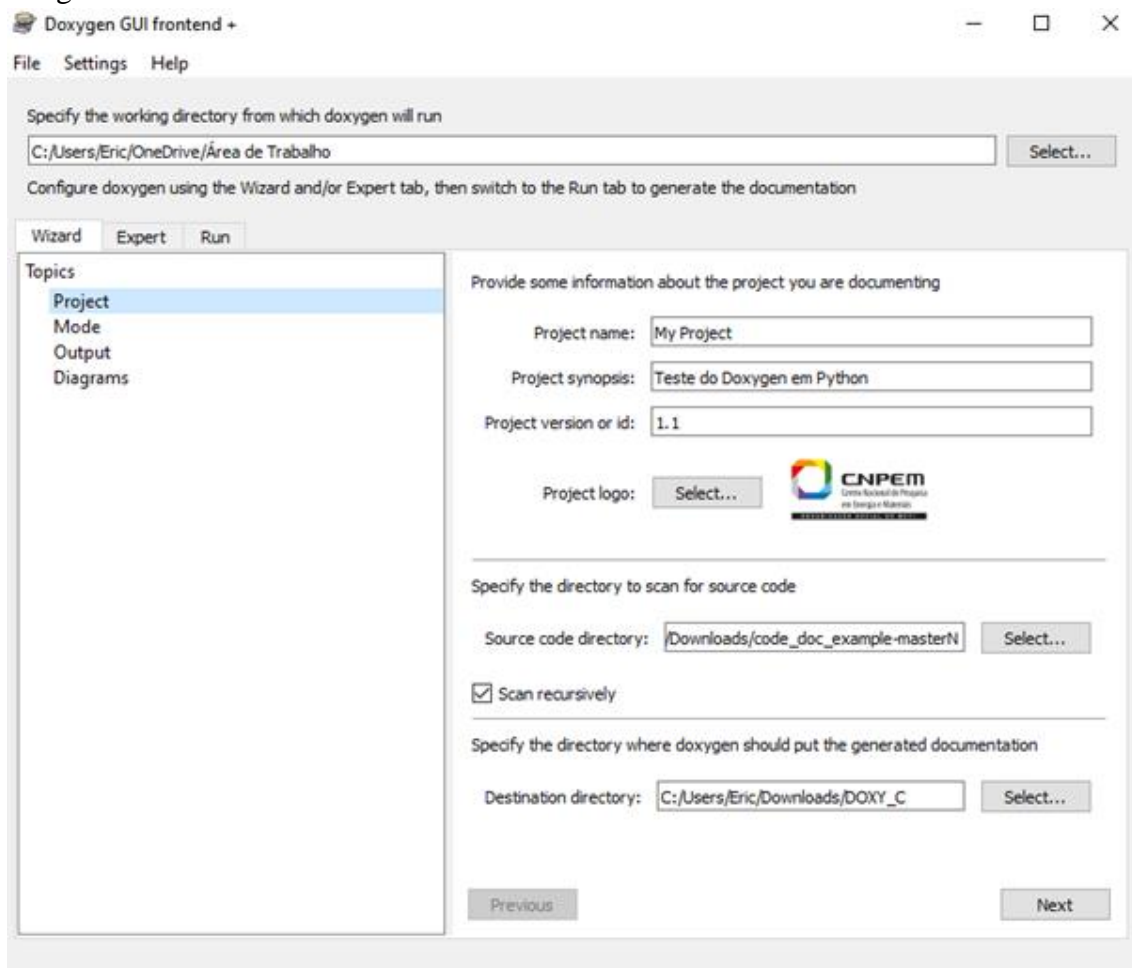


Figura 10: Instruções de configurações da seção: “Wizard-Projeto”.



## II. Wizard-Mode

Nessa seção é importante escolher o modo desejado de extração selecionado como: todas entidades (“All Entites”). Pode-se escolher incluir referência cruzada no código fonte na saída. Sobre a linguagem de programação para se otimizar os resultados, deve-se escolher a qual se julga mais próxima à usada no código fonte. Tais configurações podem ser visualizadas na figura abaixo:

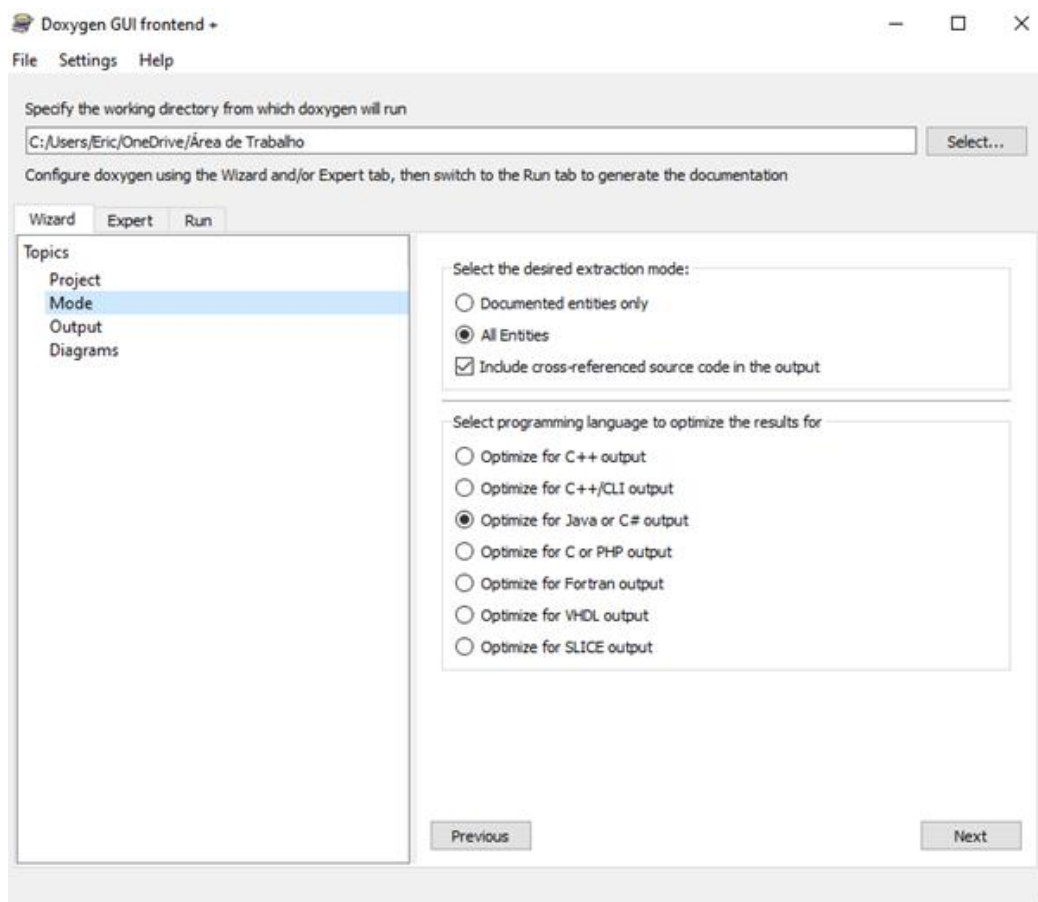


Figura 11: Instruções de configurações da seção: “Wizard-Mode”.

### III. Wizard-Output

Nessa seção se devem escolher os formatos de documentos de saída que se deseja gerar. Essa escolha pode ser feita conforme a preferência do usuário.

Destacam-se alguns formatos de saída que são mais comuns, tais como: HTML (mais usado com Doxygen, e, portanto, tem uma variedade de recursos maior do que se comparado a outros formatos), LaTeX e RTF. Pelas razões mencionadas, o formato de saída recomendado nesse tutorial é o: HTML e é sobre ele que essas instruções irão se basear.

É possível escolher a cor do documento, também, conforme as preferências do usuário. Por fim, é importante ressaltar que é possível para o usuário escolher se deseja que o documento html tenha um painel de navegação, seja comprimido (por .chm) ou tenha uma função de busca, além de outras configurações nessa mesma seção.

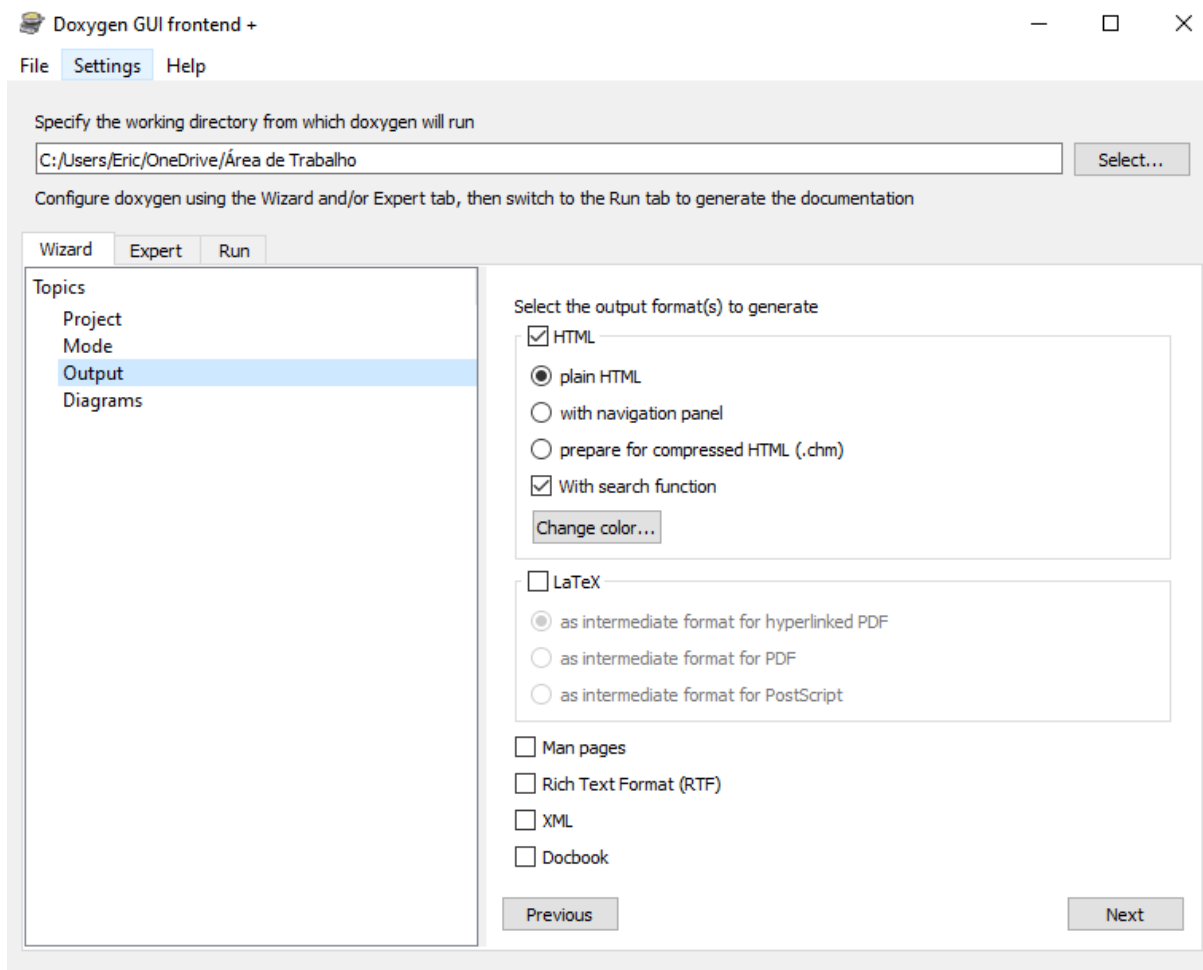


Figura 12: Instruções de configurações da seção: “Wizard-Output”.

Caso se escolha que o documento terá um painel de navegação, essa configuração irá gerar uma aba na lateral da página, de modo a facilitar a navegação do usuário. Essa aba permite o usuário navegar entre diversas “seções” do documento, bem como suas “subseções”. Cada uma dessas seções é uma outra página, com diferentes detalhes da documentação.

Apesar disso, a navegação entre essas “seções” já pode ser feita em uma aba que aparece por padrão na parte de cima do documento. Essa aba padrão superior permite acessar as diferentes “subseções” ao se clicar nas diferentes “seções”.

Uma desvantagem, porém, dessa configuração padrão, é que as subseções não ficam fixas na tela. Com o painel de navegação, as “subseções” ficam fixas na tela, além das “seções”, após clicar em uma seção específica. Desse modo, com o painel, o usuário pode navegar pela documentação, de forma mais intuitiva. Porém, o painel de navegação tem a desvantagem de ocupar um grande espaço, podendo causar desorganização na visualização da documentação. Portanto, cabe ao usuário determinar conforme suas preferências, a utilização, ou não do painel de navegação. Um exemplo do uso desse painel pode ser encontrado abaixo, no canto esquerdo da figura:



Figura 13: Documentação COM Painel de Navegação.

Na figura acima, é possível notar que quando se clica na seção “Arquivos” o painel mostra as respectivas “subseções” como “Lista de Arquivos”, “Globais”, etc. Apesar disso, essas mesmas seções podem ser acessadas na aba superior, mesmo que essa não fixe as “subseções”. Um exemplo de como a documentação fica sem o painel de navegação, referente ao mesmo código acima, pode ser visto abaixo:

## Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:  
[nível de detalhes 123456]

▾ <a href="#">Users</a>	
▾ <a href="#">eric.abbade</a>	
▾ <a href="#">Downloads</a>	
▾ <a href="#">simar-sw-master</a>	
▾ <a href="#">bme280</a>	
<a href="#">bme2.c</a>	
<a href="#">bme2.h</a>	
<a href="#">bme2_defs.h</a>	

Figura 14: Documentação SEM Painel de Navegação.

Como pode-se observar na figura acima, as “subseções” não ficam fixadas. Porém, pode-se acessar elas na aba de cima, além de se ter um visual mais organizado e menos “poluído”. Dessa forma, como já explicado, sugere-se a utilização, ou não, da aba conforme as preferências do usuário.

## IV. Wizard-Diagrams

Nesta seção se deve informar sobre a existência ou não de diagramas. Foi definido o uso da ferramenta “dot” do pacote “GraphViz”

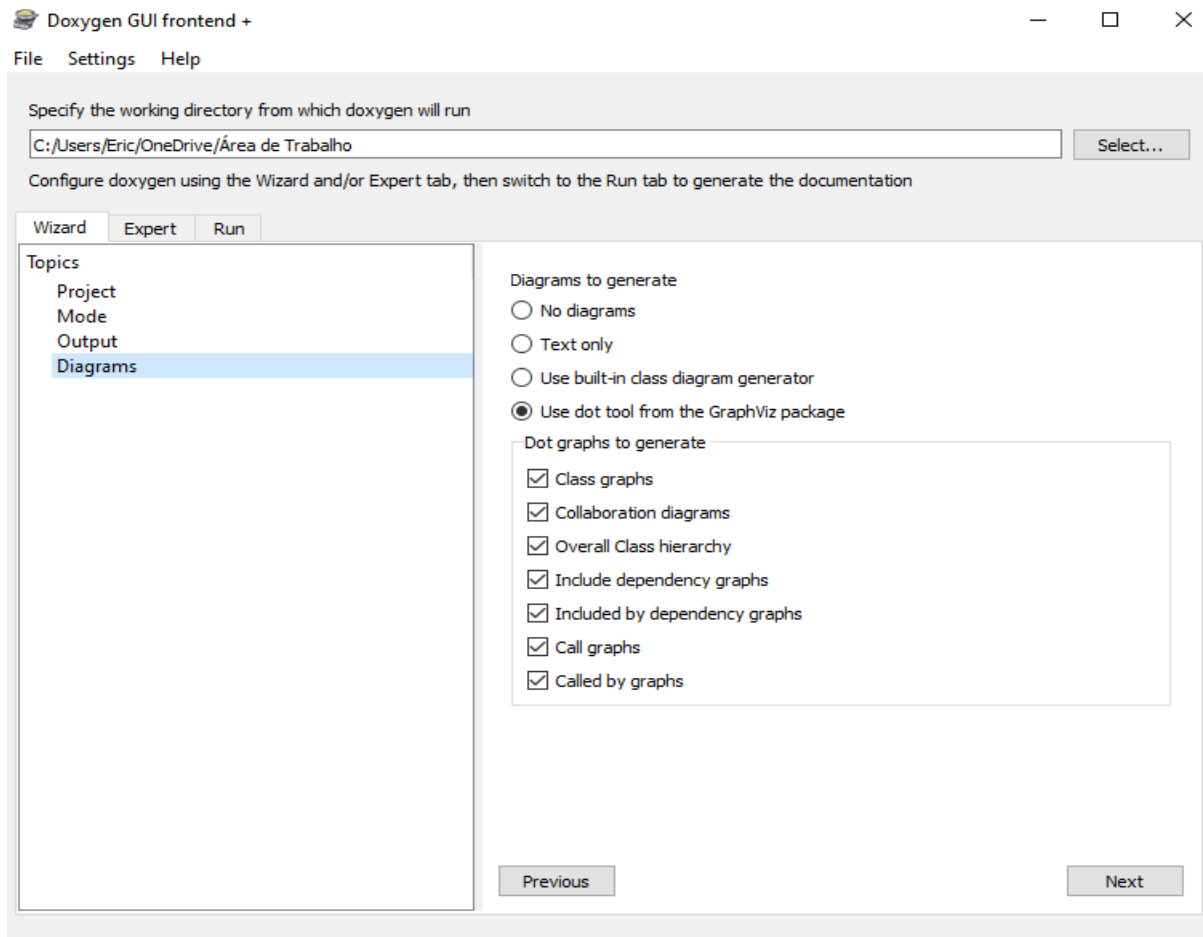


Figura 15: Instruções de configurações da seção: “Wizard-Diagrams”.

## V. Expert

Agora, irá se comentar sobre a seção: “Expert”. Nessa seção se definem as configurações mais específicas. Ela possui muitas abas, sendo que muitas vezes, não se alteram as configurações de várias. Por isso, somente será ilustrado as abas nas quais se considerou importante fazer alterações.

Existem algumas seções com muitas configurações, sendo que a maioria dessas é apenas preencher um “checkbox”. Portanto, para simplificar o documento serão mostradas somente as imagens dessas abas, de modo que os usuários possam reproduzi-las, de maneira simples. Apesar disso, serão comentadas nesse documento as funcionalidades que foram julgadas ser de maior importância, visando um melhor esclarecimento das configurações do Doxygen.

Parâmetros que foram alterados, com relação as configurações padrão, estão com a fonte do texto em **vermelho**, nas imagens abaixo. Parâmetros que não estão com a fonte em vermelho, são configurações que já vieram por padrão e NÃO precisam ser alterados. Alguns campos precisam dos locais de outros diretórios, mas essas definições já foram realizadas nas seções do “Wizard”, por isso, não precisam ser alterados nessa seção e não serão comentados. Então, as configurações são dadas por:

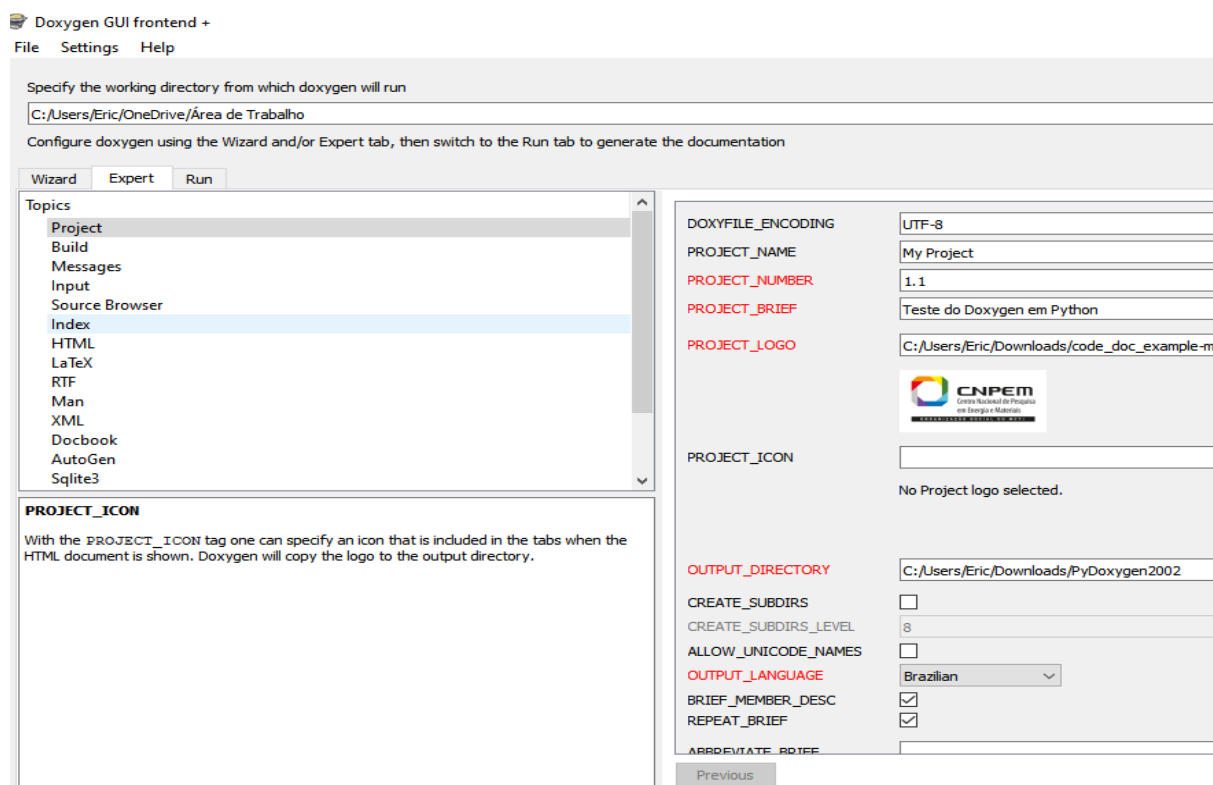


Figura 16: Instruções de configurações da seção: “Expert-Project”.

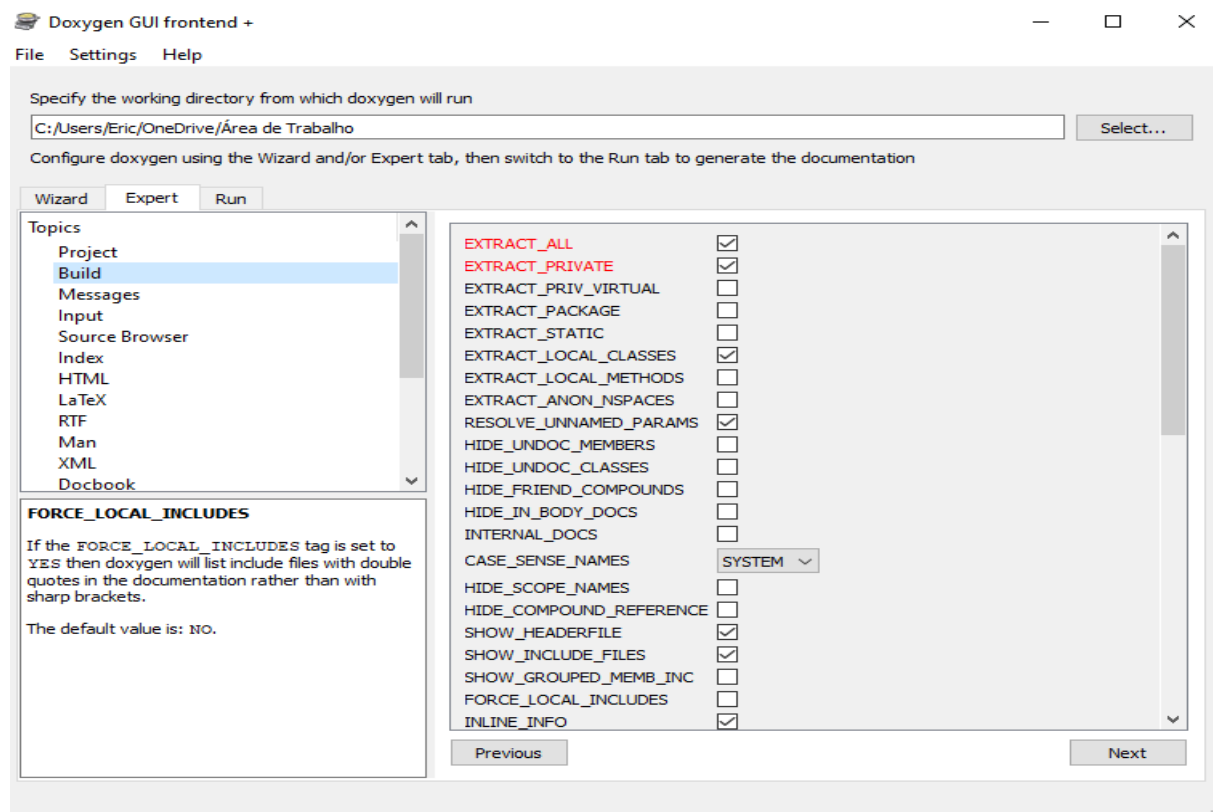


Figura 17: Instruções de configurações da seção: “Expert-Build”.

Existe uma seção denominada: “Expert-Input”. É necessário remover todos os arquivos do campo: FILE\_PATTERNS dessa seção, de modo que as configurações fiquem semelhantes com as figuras abaixo. Nessa seção, pode-se acrescentar mais arquivos no campo: INPUT, além do diretório acrescido em: “Wizard-Project”, caso se deseje gerar a documentação para mais de uma pasta. Deve-se lembrar que após colocar o caminho do arquivo no campo INPUT, deve-se clicar em: +. Português pode ser definido como a linguagem, na seção: “Expert-Project”, pela escolha de: Brazilian.

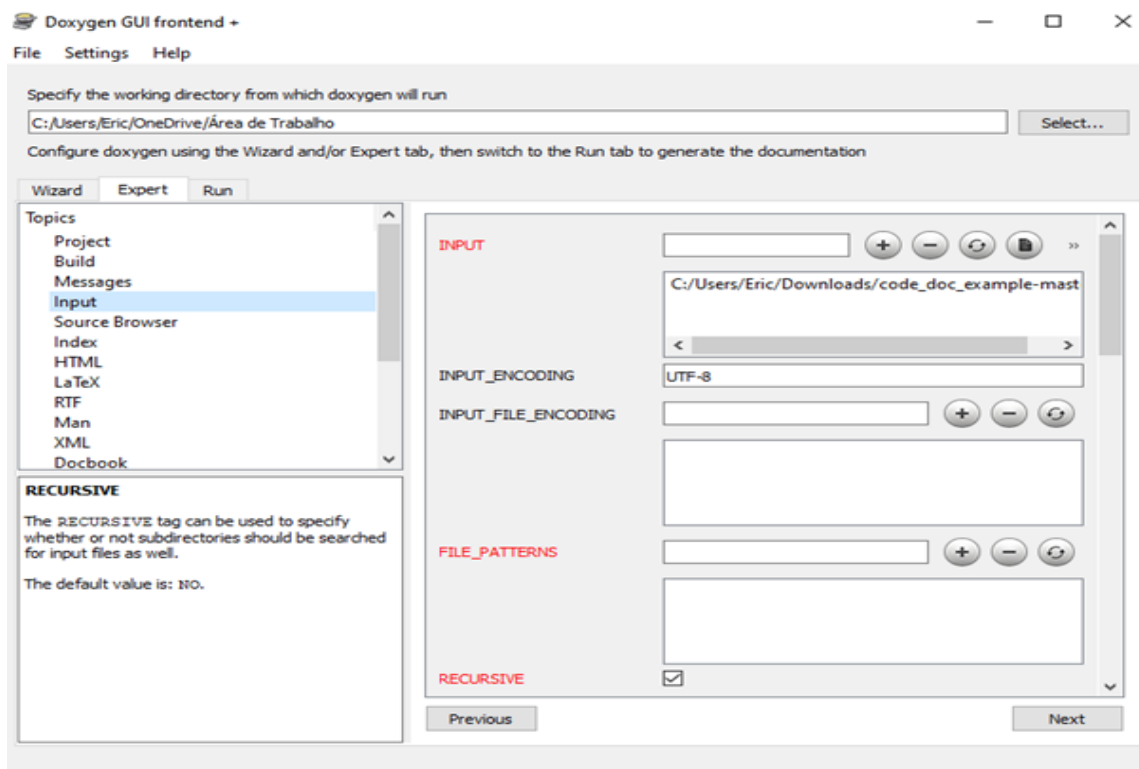


Figura 18: Instruções de configurações da seção: “Expert-Input”.

Uma **importante observação é que as próprias linhas de código de uma função podem ser adicionadas na documentação ao preencher o campo: INLINE\_SOURCES** na seção: “Expert-Source Browser”. Nas fotos abaixo, essa configuração foi preenchida. Caso, o usuário deseje retirar as linhas da documentação, basta retirar essa configuração.

O acréscimo da representação das linhas na documentação deixa o arquivo mais completo, por permitir que sejam vistas quais as linhas de comandos contidas na função. Isso evita com que o leitor tenha o trabalho de ter que procurar no arquivo original do código sobre quais linhas que determinada função se referia.

Por outro lado, essa funcionalidade pode deixar a documentação muito maior, ficando mais desorganizada (“poluída”). Logo, é importante que o usuário tenha em mente seus objetivos na geração do arquivo de documentação, para que opte, ou não, pelo uso dessa função.



Para melhor ilustração da questão mencionada acima, referente a demonstração, ou não, das linhas do código no arquivo de documentação, serão mostradas imagens de como fica a documentação em cada um desses casos, para que o usuário faça a melhor escolha:

◆ get\_open\_iter()

void get\_open\_iter ( struct bme\_sensor\_data \* sensor )

Updates door opening status.

Parameters

[in] sensor : Pointer to sensor

Considering the closed server racks work under negative pressure, a sustained "significant" rise in pressure may indicate a door opening.

However, a change in temperature may (albeit slowly) alter the pressure.

Therefore, the best solution is to maintain a moving open/closed pressure average and listen for sudden changes.

Returns

void

Definition at line 40 of file bme.c.

Here is the caller graph for this function:

main

→

update\_open

→

get\_open\_iter

Figura 19: Exemplo de Documentação SEM as Linhas de Código.

◆ get\_open\_iter()

void get\_open\_iter ( struct bme\_sensor\_data \* sensor )

Updates door opening status.

Parameters

[in] sensor : Pointer to sensor

Considering the closed server racks work under negative pressure, a sustained "significant" rise in pressure may indicate a door opening.

However, a change in temperature may (albeit slowly) alter the pressure.

Therefore, the best solution is to maintain a moving open/closed pressure average and listen for sudden changes.

Returns

void

Definition at line 40 of file bme.c.

Here is the caller graph for this function:

main

→

update\_open

→

get\_open\_iter

```

40 |
41 | // 0.23 refers to the standard deviation of the population over a period of 3 minutes
42 | if (sensor->average - sensor->data.pressure < -0.23 ||
43 |     (sensor->open_average != 0 && sensor->open_average - sensor->data.pressure < 0.23)) {
44 |     if (sensor->strikes_closed > (WINDOW_SIZE - 1))
45 |         sensor->is_open = 1;
46 |     else
47 |         sensor->strikes_closed++;
48 | } else {
49 |     if (sensor->strikes_closed < 1)
50 |         sensor->is_open = sensor->open_average = 0;
51 |     else
52 |         sensor->strikes_closed--;
53 | }
54 | }

```

Figura 20: Exemplo de Documentação COM as Linhas de Código.

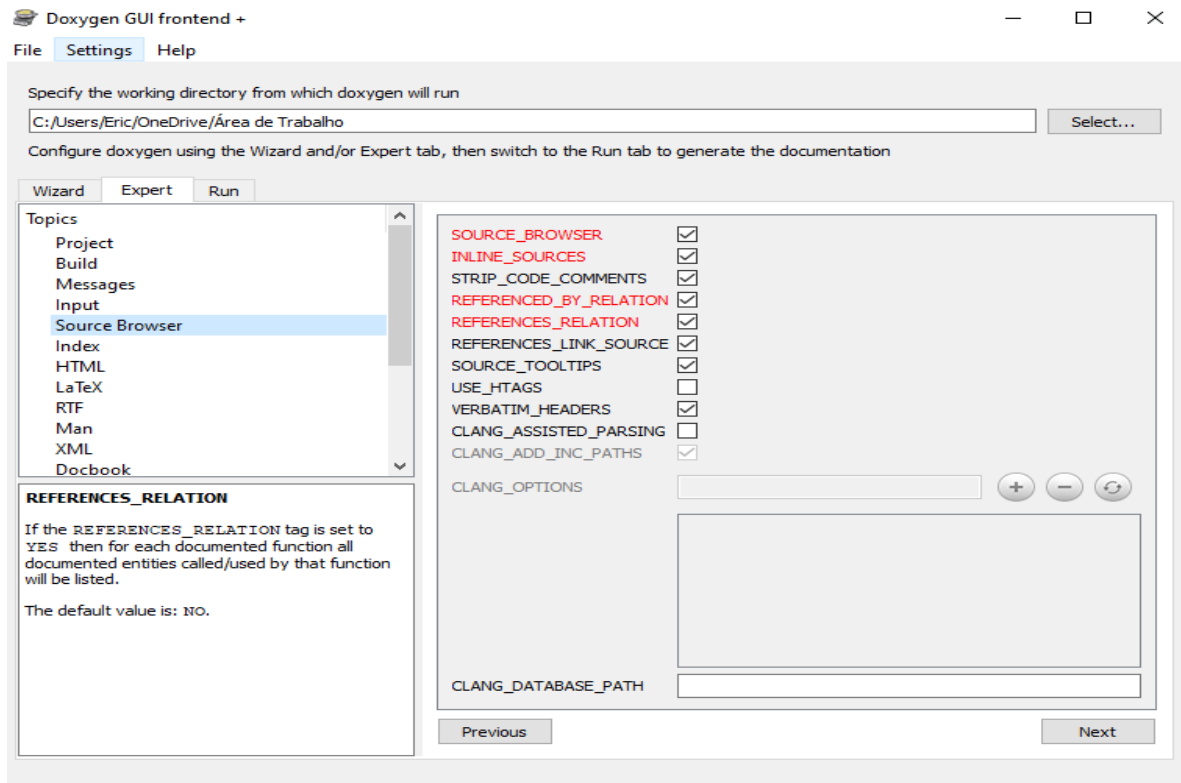


Figura 21: Instruções de configurações da seção: “Expert-Source Browser”.

Na figura abaixo irá se explicar sobre a seção do HTML. Como explicado, esse é um dos formatos de arquivos de saída mais usados. Para aprimoração de questões estéticas relativas ao design de uma página HTML é possível inserir um arquivo .css no campo: HTML\_STYLESHEET. A figura abaixo mostra esse procedimento, sendo que o arquivo usado pode ser encontrado em: <https://github.com/cnpem-emi/simar-sw/blob/master/docs/doxygen-awesome.css>

Esse arquivo .css pode ser customizado conforme as preferências do usuário para que se tenha uma página com o design e a estética desejados.

No campo HTML\_EXTRA\_FILES é possível se definir arquivos que serão representados nas: “Páginas relacionadas” (será explicado melhor sobre na seção 7.II). do arquivo final da documentação. Assim, basta colocar arquivos (podendo ser mais de 1) com formatos compatíveis, para que as páginas HTML sejam geradas.

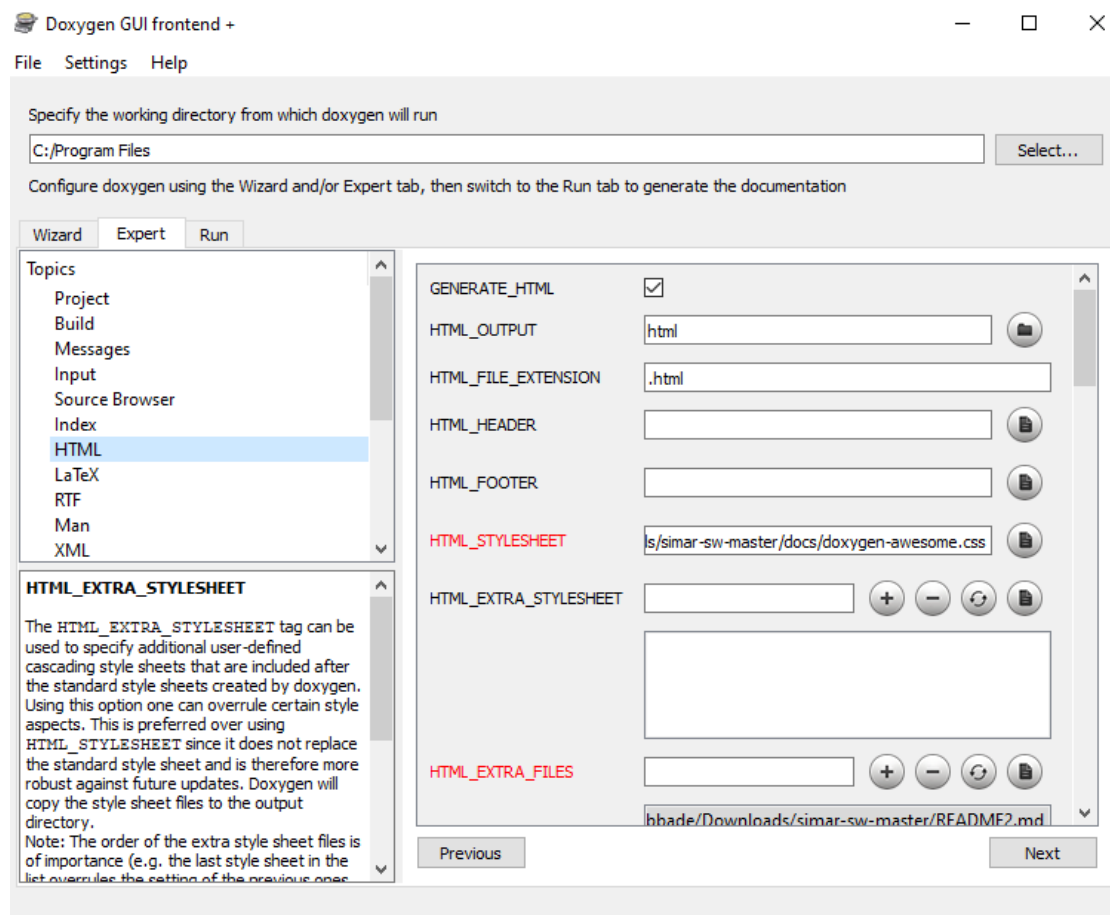


Figura 22: Instruções de configurações da seção: “Expert-HTML”.

Existe apenas uma seção na qual foi necessário o preenchimento de diretórios. Na seção: “Expert-Dot” necessitou-se informar o local da pasta: bin do GraphViz baixado, para dois campos distintos. No exemplo mostrado, o local era: C:/ Program Files/Graphviz/bin , mas cada usuário deve informar o local da pasta em que o próprio fez esse download), nos campos: DOT\_PATH e DIA\_PATH.

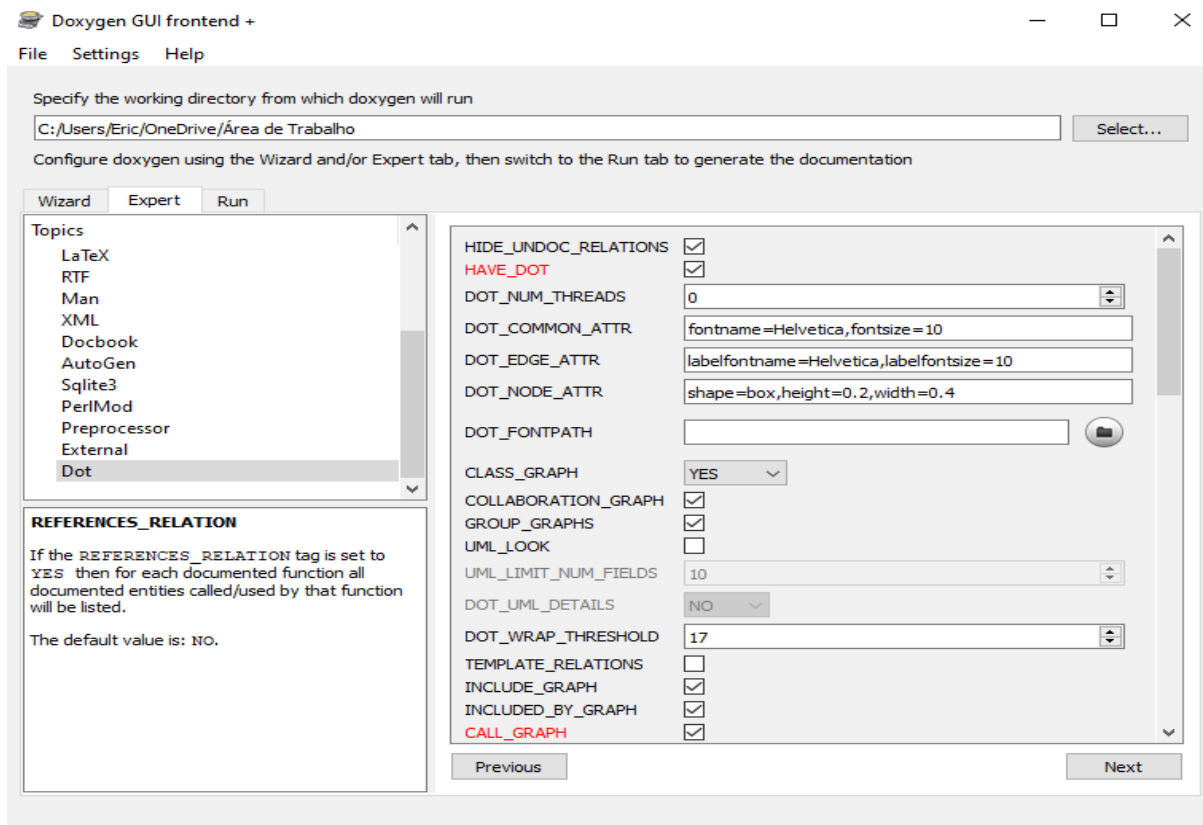


Figura 23 : Instruções de configurações da seção: “Expert-Dot” Parte 1.

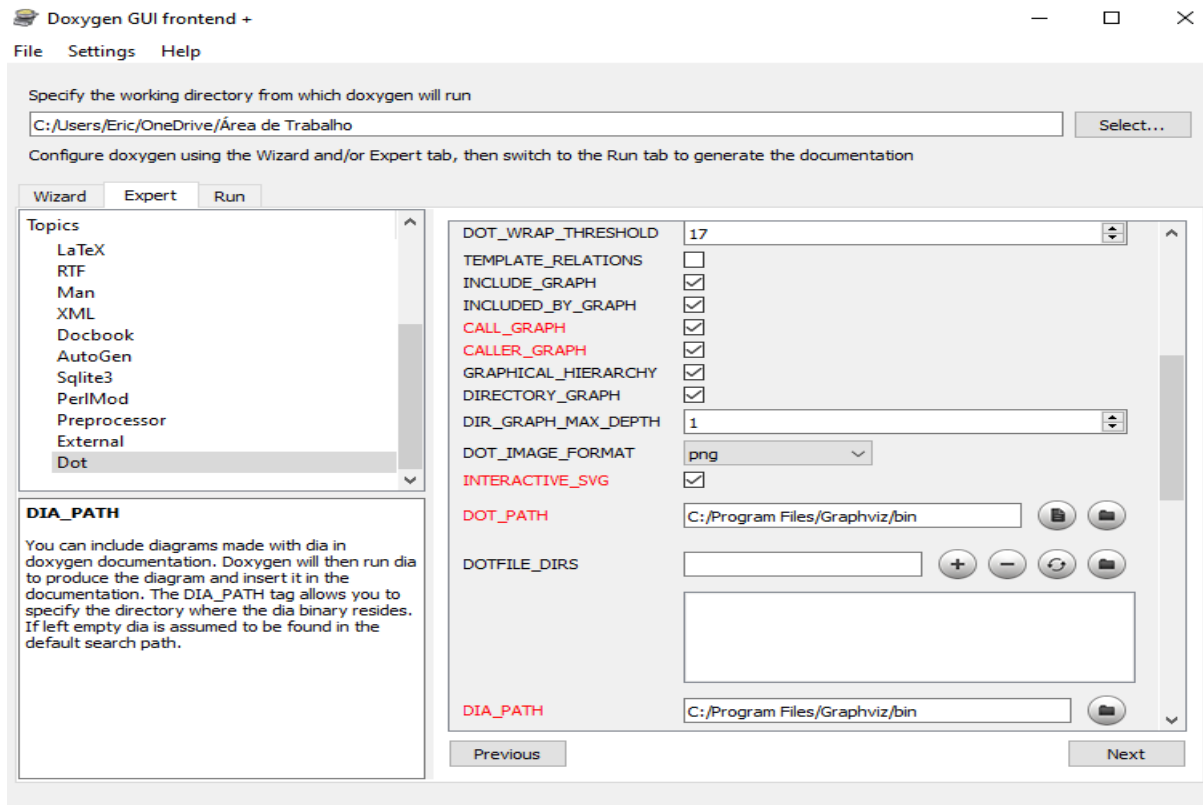


Figura 24: Instruções de configurações da seção: “Expert- Dot” Parte 2.

## VI. Run (executar)

Após, feita as configurações, basta clicar no botão: Run doxygen , conforme a imagem abaixo, para gerar a documentação. A documentação será gerada no diretório de saída definido na seção: “Wizard-Project”. Os arquivos para a visualização da documentação estão nas pastas conforme o tipo de saída escolhido.

Caso, tenha se escolhido o tipo de saída: html, a documentação estará na pasta: html (que está dentro da pasta definida) e o arquivo inicial é: index.html (apesar, de que pode ser renomeado, conforme preferências do usuário). Apesar disso, para a visualização de toda documentação são necessários todos os documentos da pasta: html (index.html é somente o arquivo da página principal).

Para que se visualize a documentação completa, é possível clicar no botão: Show HTML output . Isso fará com que o arquivo index.html seja aberto, de forma automática, em um navegador de internet.

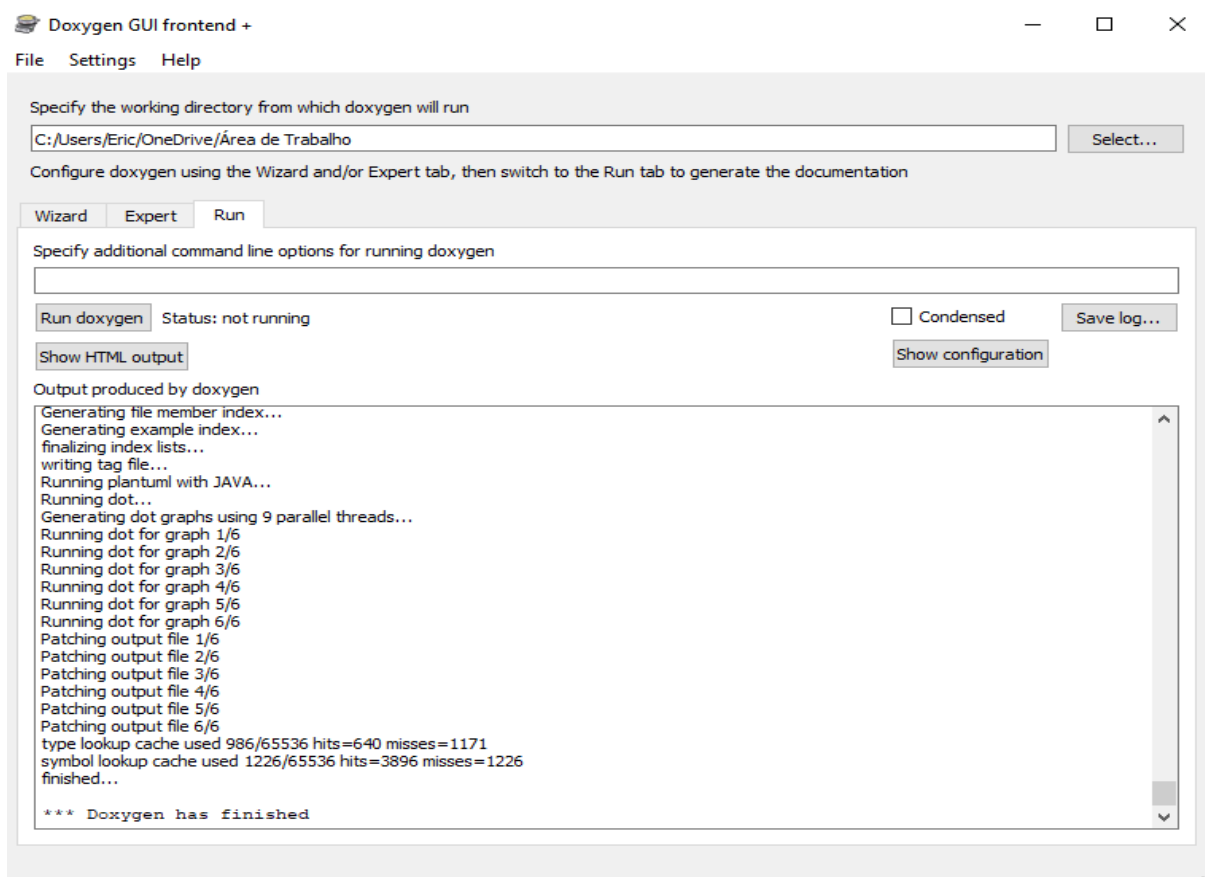


Figura 25: Instruções de configurações da aba: “Run”.

## VII. Salvando configurações

Por fim, como é possível de se observar nesse relatório, existem muitas configurações que devem ser feitas no Doxygen, para que ele atue, de forma adequada. A fim de evitar ter que refazê-las toda vez em que se reinicia o Doxywizard, pode-se salvar as configurações, ao se clicar em: Settings e depois: Use current settings at startup , como é possível ver, na figura abaixo. Dessa forma, é possível se armazenar as configurações, para o próximo uso.

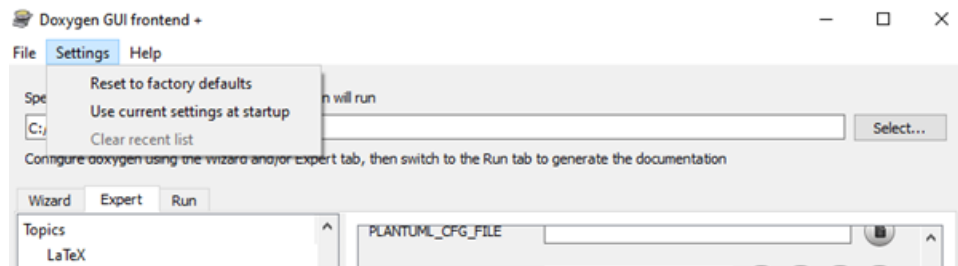


Figura 26: Instruções para salvar as configurações.

## VIII. Gerando Doxyfile

As configurações acima podem ser salvas em um arquivo. Isso pode ser feito por clicar no seguinte botão na mesma janela mencionada acima:

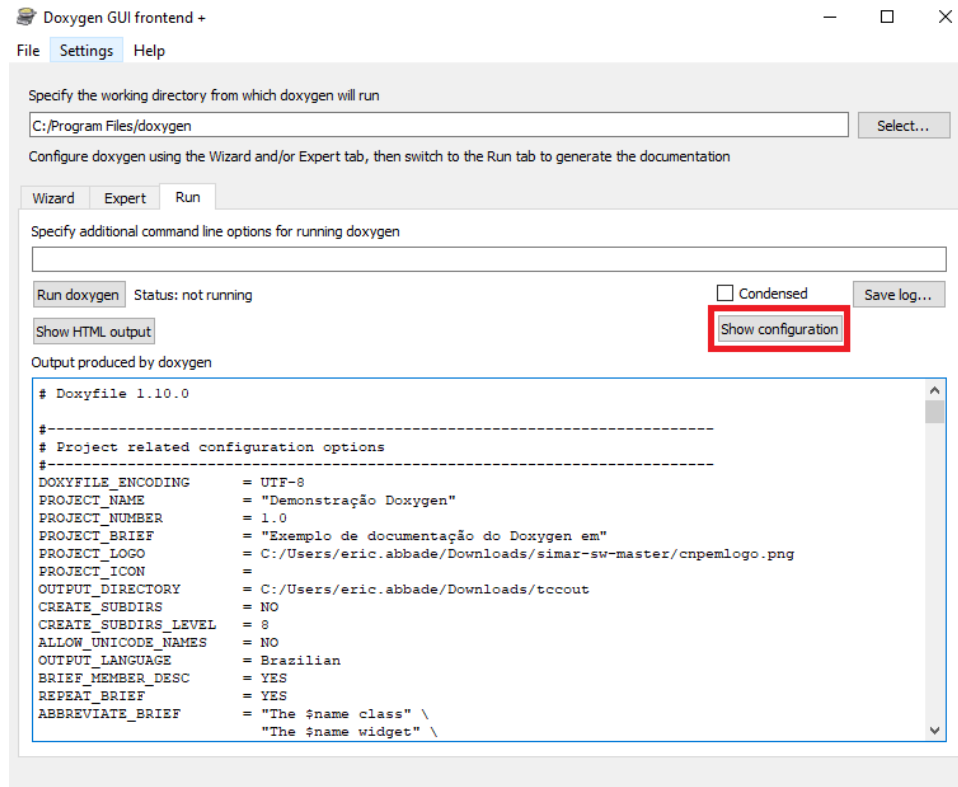


Figura 27: Salvando as configurações no Doxyfile.

Conforme é possível ver na figura acima, basta clicar no botão circulado em **vermelho**: Show configuration para que se visualize as configurações geradas no retângulo denotado em azul. Essas configurações podem ser salvas em um arquivo (por exemplo, com o nome: doxyfile.txt).

Esse arquivo pode ser usado da forma como mencionado na seção anterior para gerar a documentação, de forma independente dessa aba do Doxywizard. Além disso, pode ser importado para outros computadores, ou mesclado com outros arquivos (até possivelmente mais completos).



## 7. Abas da Documentação

### I. Página Principal

Como um dos usos mais comuns do Doxygen é gerar documentações no formato HTML, que é baseado em páginas web, é bastante útil usar o Doxygen para criar páginas específicas de documentação. É importante ressaltar que com as configurações mencionadas nas seções anteriores, diversas páginas já são automaticamente geradas. Porém para que se criem outras páginas, totalmente customizáveis, são necessários comandos adicionais.

No Doxygen, existe uma seção para a página principal (que tem somente 1 página), sendo que essa é a primeira página que aparece quando se entra na documentação. Além disso, existe outra seção para as “Páginas relacionadas” (que podem conter diversas outras páginas) e páginas específicas para arquivos. Caso nenhuma página relacionada seja criada, essa seção não irá existir no código.

Tais páginas podem ser criadas tanto por comentários nos códigos, como por arquivos separados dos códigos. Será comentado sobre como fazer para criá-las por ambos os métodos. Existem diversas linguagens que o Doxygen é capaz de mostrar a documentação, como mostrado anteriormente. Para fins demonstrativos, os comentários serão mostrados nas linguagens C e Python, já que são linguagens distintas, porém muito usadas.

Outras linguagens mencionadas podem ser usadas para gerar páginas pelo Doxygen. Detalhes relativos ao uso de comentários delas para a geração de páginas, devem ser estudados por outros tutoriais. Porém, vale a ressalva de que como será possível perceber adiante, a estrutura dos comentários em C é extremamente similar à do Python. Por isso, outras linguagens, possivelmente, terão uma sintaxe para os comentários no Doxygen similares a essas do C e do Python.

Um resumo de como documentar outros formatos de arquivo, elaborado no próprio site do Doxygen, é dado por: <https://www.doxygen.nl/manual/docblocks.html#specialblock>

## i. Página principal por comentários (C):

Agora, será explicado como gerar uma página principal por comentários em um código C. Primeiramente, é importante ressaltar que os comandos para gerar a página devem estar entre os seguintes caracteres:

```
/**
 *
 *
 *
 *
 */
```

Todos os comandos devem estar dentro as linhas com os caracteres de asterisco: `*`. O número de linhas com o asterisco: `*` é determinado pelo número de linhas com comandos, já que toda a linha com comando deve iniciar com esse asterisco. Pular uma linha (desde com asterisco no começo), não altera o funcionamento do processo de geração da página. Um exemplo do código completo para a geração da página será mostrado, adiante.

Para criar a página principal e definir seu título, deve-se usar o seguinte comando:

```
* @mainpage "Título Inicial"
```

Esse comando acima deve ser colocado no início dos comandos da página principal, logo após do: `/**`. Como resultado o título da página principal pode ser visualizado a seguir:



Figura 28: Título da Página Principal

É importante ressaltar que no comando acima, no local da representação do título, se colocou ele em aspas. **Todos os locais de textos que aparecem diretamente nas páginas foram colocados entre aspas nesse documento. Isso foi feito, para ilustrar ao usuário, os locais quais ele deve substituir para alterar os textos das páginas.** Portanto, no exemplo acima, para alterar o título para: Novo Título, basta substituir o trecho: `"Título Inicial"`, de modo que:

```
* @mainpage Novo Título
```

Dessa forma, o novo título da página principal fica:

## Novo Título

Figura 29: Novo Título da Página Principal.

Essa mesma lógica da escrita dos textos que aparecem diretamente no documento em aspas, será usada nas próximas representações dos comandos nessa seção.

Outro comando é o de “brief” que adiciona um breve texto, embaixo do título. Um exemplo de se uso, é dado por:

```
* @brief "Exemplo de briefing"
```

Seu resultado pode ser visto, na imagem abaixo:

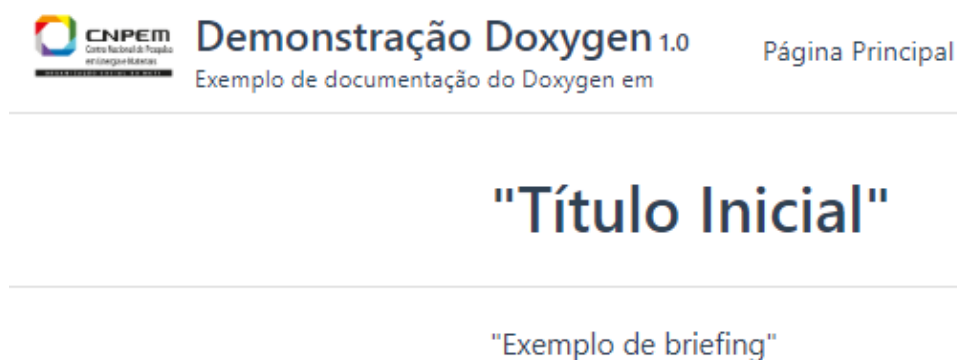


Figura 30: Exemplo do Comando: brief.

Lembrando que para alterar o texto “Exemplo de briefing”, basta substituir o texto entre aspas, como já explicado.

Vale lembrar que a ordem dos comandos, corresponde a ordem vertical (sentido de cima para baixo) da representação dos comandos na página de documentação. Ou seja, as linhas de comandos que forem colocadas em cima de outras linhas no código, resultarão em respectivas representações na documentação HTML que também serão colocadas acima das representações dos outros comandos.

Outro comando é o de texto. Um texto simples pode ser adicionado a página, pelo simples comando:

```
* "Exemplo de Texto."
```

Para que o texto seja representado na forma de tópicos, basta ser adicionado o caractere: - antes do texto, conforme visto na sequência de comandos abaixo:

```
* - "Tópico 1"  
* - "Tópico 2"
```

Textos podem ser adicionados em qualquer local da página, quantas vezes necessárias. Os resultados desses comandos podem ser vistos, na imagem abaixo:



Figura 31: Exemplo de Comando de Texto.

Outro comando é o de seção. Uma seção pode ser adicionada à página, pelo comando:

```
* @section main_page_section "Título Seção"  
* "Texto Seção"
```

Nesse comando representado acima, a seção é criada e o seu título pode ser alterado no comando da linha superior ("Título da Seção"), enquanto o seu texto pode ser alterado na linha inferior ("Texto da Seção"). Podem existir mais de uma linha de texto na seção e mais de uma seção na página, conforme o desejo do usuário. Uma representação visual do uso desse comando é dada por:

Seu resultado pode ser visto, na imagem abaixo:



Figura 32: Exemplo de Comando de Seção.

Existe também o comando do autor. Autores podem ser adicionados às páginas, pelos comandos:

```
* @author "Eric Sonagli Abbade"  
* @author "Outro Autor"
```

Pode ser adicionados qualquer número de autores (para os casos em que existe mais de um autor). Diferentes autores podem ser adicionados em outras páginas, pelo mesmo comando, para o caso no qual diferentes arquivos não tiverem sido projetados pela mesma pessoa. Esse comando também pode ser colocado em qualquer lugar da página.

Seu resultado pode ser visto, na imagem abaixo:

# "Título Inicial"

"Exemplo de briefing"

"Exemplo de Texto."

- "Tópico 1"
- "Tópico 2"

## "Título Seção"

"Texto Seção"

Autor

"Eric Sonagli Abbade"

"Outro Autor"

Figura 33: Exemplo de Comando de Seção.

Ainda existe um comando para a demonstração de imagens na documentação. É possível colocar quantas imagens for desejado, em qualquer lugar de qualquer página. Existem vários métodos de se colocar imagens, mas um exemplo de comando para fazer isso, é dado por:

```
* 
* <div style="clear: both"></div>
```

Nos comandos acima: `C:\Users\eric.abbade\Downloads\simar-sw-master\main\sirius.png` deve ser substituído pelo local da imagem que se deseja colocar. Pelos comandos `width` e `height` pode-se definir o comprimento e largura da imagem em pixels. Além disso, se tem a possibilidade de definir outros parâmetros comuns as linguagens HTML e CSS como o `align` (alinhamento).

Essa imagem e os “direitos de cópia” serão mostrados na página em breve, na representação da página completa. Uma prática comum dos usuários de Doxygen é a inclusão do termo de “copyright” (direitos de cópia) no final do documento. Pode ser feito pela inserção de um comando de texto comum (já mencionado antes):

```
* "Copyright (c) 2024 All rights reserved.."
```

## ii. Exemplo de Página principal por comentários (C)

```
/**
 * @mainpage "Inicial C"
 *
 * @brief "Exemplo de briefing"
 *
 * "Exemplo de Texto."
 * - "Tópico 1"
 * - "Tópico 2"
 *
 * @section main_page_section "Título Seção"
 * "Texto Seção"
 *
 * @author "Eric Sonagli Abbade"
 * @author "Outro Autor"
 *
 * 
 * <div style="clear: both"></div>
 *
 * "Copyright (c) 2024 All rights reserved.."
 */
```

Esse código gerou a seguinte página principal:

**Demonstração Doxygen 1.0**[Página Principal](#)[Páginas](#)  
Exemplo de documentação do Doxygen em

---

## "Inicial C"

---

"Exemplo de briefing"

"Exemplo de Texto."

- "Tópico 1"
- "Tópico 2"

## "Título Seção"

"Texto Seção"

Autor  
"Eric Sonagli Abbade"  
"Outro Autor"



"Copyright (c) 2024 All rights reserved.."

---

Figura 34: Página Principal Gerada por Comandos C.

### iii. Página principal por comentários (Python)

Agora, será explicado como gerar uma página principal por comentários em um código Python. Primeiramente, é importante ressaltar que os comandos para gerar a página (em Python) devem estar entre os seguintes caracteres:

```
##  
#  
#  
#  
#
```

Todos os comandos devem estar dentro as linhas com os caracteres de “hashtag”: #. O número de linhas com o “hashtag”: # é determinado pelo número de linhas com comandos, já que toda a linha com comando deve iniciar com esse “hashtag”. Pular uma linha (desde com “hashtag” no começo), não altera o funcionamento do processo de geração da página.

Dessa forma o “hashtag”: # funciona de forma análoga ao asterisco: \* em C, com a diferença que o texto deve começar com 2 “hashtags”: ## e terminar com um “hashtag”: #. Todos os outros comandos são iguais aos mostrados em C, com a troca do asterisco: \* em C por “hashtag”: # em Python. Por exemplo, o comando para gerar uma página principal, similar ao que foi mostrado em C, é dado por:

```
# @mainpage "Título Inicial"
```

Assim, os demais comandos podem ser escritos iguais aos retratados em C, trocando o \* por #, de modo que basta ler os comandos já mencionados acima em C para reproduzi-los em Python. Um exemplo do código completo em Python, com base nos comandos mostrados anteriormente, para a geração de uma página principal será mostrado a seguir.

### iv. Exemplo de Página principal por comentários (Python)

```
##  
# @mainpage "Inicial Python"  
#  
# @brief "Exemplo de briefing"  
#  
# "Exemplo de Texto."  
# - "Tópico 1"  
# - "Tópico 2"  
#  
# @section main_page_section "Título Seção"  
# "Texto Seção"  
#  
# @author "Eric Sonagli Abbade"
```



```
# @author "Outro Autor"
#
# 
# <div style="clear: both"></div>
#
# "Copyright (c) 2024 All rights reserved.."
#
```

Esse código gerou a seguinte página principal:



Figura 35: Página Principal Gerada por Comandos Python.

## v. Página principal por arquivos separados

Arquivos podem ser inseridos no campo de INPUT (conforme explicado na seção 6.V desse tutorial). Dessa forma, eles geram as páginas, desde que tais arquivos sejam de um formato compatível com essa tarefa. Para que seja gerada a Página Principal, basta adicionar a seguinte linha no começo do arquivo:

```
/*! \mainpage "Title" - "Briefing".
```

No comando acima, o termo: mainpage é obrigatório para que o Doxygen interprete que se está gerando a primeira página. "Title" e "Briefing" podem ser substituídos pelos respectivos títulos e breve resumo da página. Um exemplo é o arquivo no formato .md mostrado abaixo:

```

/#!/mainpage "Title" - "Briefing".
# "CNPEM"

[![Build](https://github.com/cnpem-iot/simar-software/actions/workflows/c-
cpp.yml/badge.svg)](https://github.com/cnpem-iot/simar-
software/actions/workflows/c-cpp.yml)
"Example text".

## MainPage
"This is an example of Doxygen mainpage."
- [x] "main"
- [ ] "page"

"Author: Eric Sonagli Abbade"

"Copyright (c) 2024 All rights reserved.."

```

O caminho dele foi colocado no campo INPUT mencionado, conforme a ilustração:

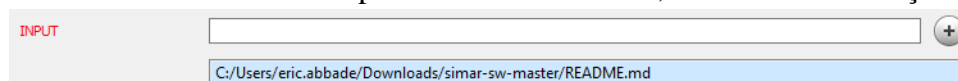


Figura 36: Arquivo no campo INPUT.

Deve-se lembrar que após colocar o caminho do arquivo no campo INPUT, deve-se clicar em: +. Esse código gerou a seguinte página:



Figura 37: Página Principal Gerada por Arquivo .md.

## II. Páginas Relacionadas

### i. Páginas relacionadas por comentários nos códigos

Além da página principal, existem outros tipos de páginas. Um exemplo de outros tipos de página são as páginas relacionadas.

Páginas relacionadas são basicamente todas as páginas que não referem a nenhum arquivo ou trecho em específico de um arquivo. Elas são totalmente projetadas pelo usuário, sem nenhum tipo de padrão pré-definido pelo Doxygen.

Tais páginas podem ser acessadas em uma aba que fica do lado da aba principal, no canto superior da tela (as vezes também na lateral esquerda, dependendo das configurações definidas na seção 6.III). Essa aba será denotada a seguir, pelo retângulo vermelho na figura abaixo:

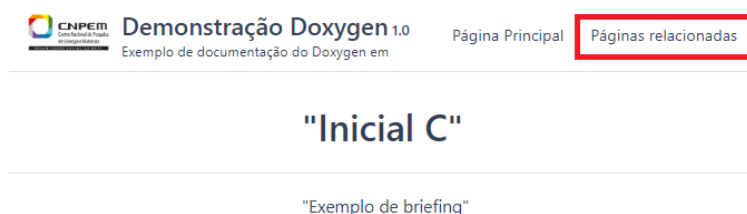


Figura 38: Aba das Páginas Relacionadas.

É importante ressaltar que **essa aba não é gerada em todas as documentações. Ela é gerada somente quando as páginas relacionadas forem criadas**, conforme as configurações que serão mencionadas nessa seção 7.II.

Quando essa aba é acessada é possível escolher entre diversas outras páginas relacionadas.

Esse processo pode ser visualizado abaixo:

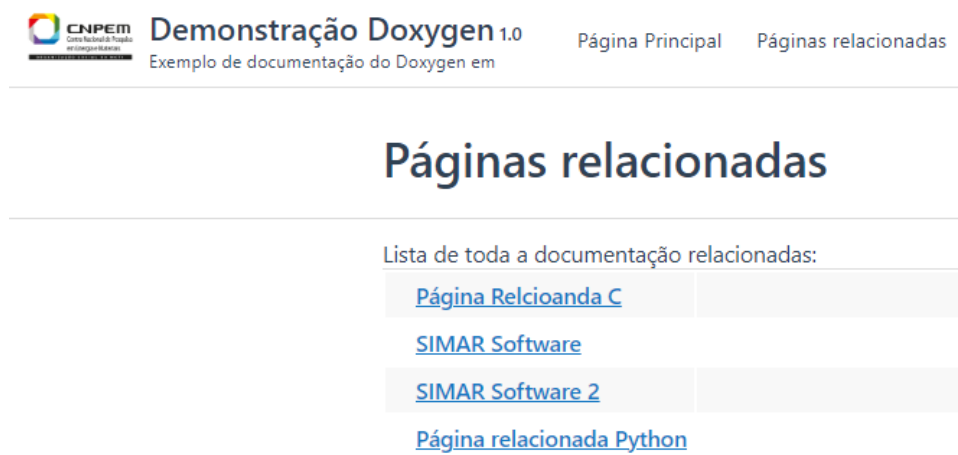


Figura 39: Escolha das Páginas Relacionadas.

Na figura acima, é possível acessar uma das 4 páginas relacionadas. As: “Página Relacionada C” e “Página Relacionada Python” foram geradas respectivamente por comentários de C e Python.

Páginas relacionadas podem ser geradas pelos exatos mesmos comandos mencionados da página principal, tanto para C, quanto para Python. A única diferença é que deve se substituir o comando inicial `@mainpage` por: `@page`. Dessa forma, os comandos iniciais para C e Python se tornam, respectivamente:

```
/**
 * @page "Página Relacionada C"
 *
 */
```

```
##
# @page "Página Relacionada Python"
#
#
```

Por fim serão mostrados exemplos da geração dessas páginas por comentários, tanto por C quanto por Python. Isso será feito, pelo uso de comandos similares aos usados na Página Principal.

## ii. Exemplo em C de páginas relacionadas

```
/**
 * @mainpage "Inicial C"
 *
 * @brief "Exemplo de briefing"
 *
 * "Exemplo de Texto."
 * - "Tópico 1"
 * - "Tópico 2"
 *
 * @section main_page_section "Título Seção"
 * "Texto Seção"
 *
 * @author "Eric Sonagli Abbade"
 * @author "Outro Autor"
 *
 * 
 * <div style="clear: both"></div>
 *
 * "Copyright (c) 2024 All rights reserved.."
 */
```

Esse código gerou a seguinte página principal:

**Demonstração Doxygen 1.0**  
Exemplo de documentação do Doxygen em

[Página Principal](#) [Páginas relacionadas](#)

---

## Página Relcioanda C

---

"Exemplo de Texto."

- "Esta é uma página relacionada."
- "Feita por comentários em C."

### "Título Seção"

"Texto Seção"

Autor  
"Eric Sonagli Abbade"  
"Outro Autor"



"Copyright (c) 2024 All rights reserved.."

Figura 40: Página Relacionada Gerada por Comandos C.

### iii. Exemplo em Python de páginas relacionadas

```
##
# @page "Página relacionada Python"
#
# "Exemplo de Texto."
# - "Esta é uma página relacionada."
# - "Feita por comentários em Python."
#
# @section main_page_section "Título Seção"
# "Texto Seção"
#
# @author "Eric Sonagli Abbade"
# @author "Outro Autor"
#
# 
# <div style="clear: both"></div>
#
# "Copyright (c) 2024 All rights reserved.."
#
```

Esse código gerou a seguinte página principal:

## Página relacionada Python

"Exemplo de briefing"

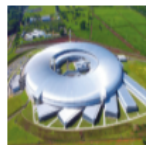
"Exemplo de Texto."

- "Esta é uma página relacionada."
- "Feita por comentários em Python."

## "Título Seção"

"Texto Seção"

Autor  
"Eric Sonagli Abbade"  
"Outro Autor"



"Copyright (c) 2024 All rights reserved.."

Figura 41: Página Relacionada Gerada por Comandos Python.

### iv. Páginas relacionadas por arquivos separados

Para realizar a geração de páginas relacionadas, basta seguir os mesmos passos da geração da página principal SEM o comando inicial:

```
/*! \mainpage "Title" - "Briefing".
```

Dessa forma, um exemplo de arquivo no formato .md, para a geração de uma página relacionada, pode ser visto logo abaixo:

```
# "Página Relacionada"

[![Build](https://github.com/cnpem-iot/simar-software/actions/workflows/c-cpp.yml/badge.svg)](https://github.com/cnpem-iot/simar-software/actions/workflows/c-cpp.yml)

"Example text".

## Exemplo: página relacioanda

"Example of Doxygen related page."
- [x] "related"
- [ ] "page"

"Author: Eric Sonagli Abbade"
"Copyright (c) 2024 All rights reserved.."
```

Tal código gerou a seguinte página (que pode ser acessada após entrar na aba de Páginas Relacionadas, da forma mencionada anteriormente):



Figura 42: Página Relacionada Gerada por Arquivo .md.

### III. Aba de arquivos

#### i. Início da aba de arquivos

“Aba dos arquivos” pode ser acessada na parte superior da tela, como na figura que será mostrada abaixo (ou, as vezes também na lateral esquerda, dependendo das configurações definidas na seção 6.III). O acesso dessa aba pode ser representado por:

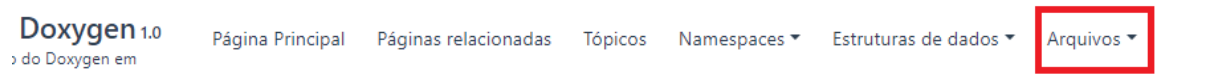


Figura 43: Acesso da Aba dos “Arquivos”.

Após, feito o clique no local mostrado, irão aparecer algumas opções para se escolher., conforme a figura abaixo:

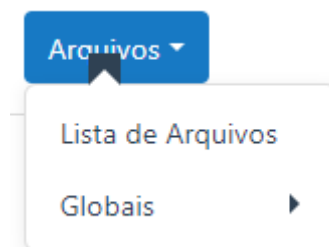


Figura 44: Escolha de opções dentro da Aba dos “Arquivos”.

Escolhendo a opção “Lista de Arquivos” pode-se observar todos os arquivos definidos nos campos de entradas mencionados nas seções anteriores. Uma representação dessa nova aba (incluindo tanto arquivos em C, quanto em Python), pode ser vista abaixo:



## Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:  
[nível de detalhes 123456]

▼ <a href="#">Users</a>	
▼ <a href="#">eric.abbade</a>	
▼ <a href="#">Downloads</a>	
▼ <a href="#">simar-sw-master</a>	
▼ <a href="#">bme280</a>	
<a href="#">bme2.c</a>	
<a href="#">bme2.h</a>	
<a href="#">bme2_defs.h</a>	
▼ <a href="#">i2c</a>	
<a href="#">common.c</a>	Common functions for I2C operations
<a href="#">common.h</a>	Common declarations for I2C operations
▼ <a href="#">main</a>	
<a href="#">bme.c</a>	Main starting point for BME280 sensor module
<a href="#">fan.c</a>	Main starting point for fan RPM sensor module
<a href="#">leak.c</a>	Main starting point for fan RPM sensor module
<a href="#">volt.c</a>	Main starting point for AC board module
<a href="#">wireless.c</a>	Main starting point for wireless SIMAR
▼ <a href="#">sht3x</a>	
<a href="#">arch_config.h</a>	
<a href="#">sht3x.c</a>	Sensirion SHT3x driver implementation
<a href="#">sht3x.h</a>	Sensirion SHT driver interface
▼ <a href="#">spi</a>	
<a href="#">common.c</a>	Common functions for SPI operations
<a href="#">common.h</a>	Common declarations for SPI operations
▼ <a href="#">tccin</a>	
<a href="#">main.py</a>	"Exemplo de briefing"

Figura 45: Imagem da lista de “Arquivos”.

Ao se clicar em cada um desses arquivos será possível observar a documentação de cada um deles. O arquivo main.py pode ser observado abaixo:

## Referência do Arquivo main.py

"Exemplo de briefing" [Mais...](#)

[Ir para o código-fonte desse arquivo.](#)

### Estruturas de Dados

class [teste](#)

### Namespaces

namespace [main](#)

### Funções

[auxiliar\\_docstring](#) (self)

[on connect](#) (client, userdata, flags, rc)

[Read AD](#) (channel)

[SensorTemperature](#) (sen)

[fun1](#) ()

[fun2](#) ()

[fun3](#) ()

[True](#) ()

[output](#) (out)

### Variáveis

int [CS](#) = 7

Figura 46: Documentação do arquivo main.py.

Conforme é possível de se observar na figura acima, pode-se olhar diversos parâmetros do arquivo nessa página, como estrutura de dados (classes do Python), namespace, funções e variáveis. Quando se clica em estrutura de dados, ou namespace ocorre o direcionamento para páginas similares as descritas nas seções 12 e 11, respectivamente. Por isso, basta olhar essas seções, para estudar essas opções.

Ao se clicar nas variáveis, é possível os nomes delas e as linhas do arquivo nas quais foram definidas. Como, essas são as informações necessárias para que se entenda como foram definidas, não será comentado mais sobre elas, porque são representadas apenas por essas simples informações.

Porém, as representações das funções na documentação são um pouco mais complexas, e por isso serão discutidas com mais detalhes nas seções 7.III.ii e 7.III.iii. Por fim, existe a seção 7.III.iv na qual se irá documentar algumas configurações padrões das páginas de arquivo.

## ii. Diagramas de funções

Por fim, se tem as representações das funções. Ao se clicar nelas, é possível ver alguns comentários sobre elas, o código delas (dependendo do que foi definido na seção .V) além do diagrama de funções.

O diagrama de funções é uma ferramenta do GraphViz que gera a ligação de blocos. As funções são representadas por blocos. A ação de uma função acionar outra é representada por um esquemático no qual uma flecha sai de um bloco (que representa uma função que aciona outra função) para outro bloco (que representa a função acionada). Esse processo é repetido várias vezes, para todos os casos em que uma função aciona a outra. Dessa forma, se tem um esquemático completo de quais funções acionam ou são acionadas.

É importante ressaltar que esse esquemático somente representa os acionamentos dentre funções. Caso, uma linha (fora de uma função) acione determinada função, esse acionamento não é representado no diagrama. Isso ocorre, devido ao fato que tecnicamente o acionamento foi feito pela rotina principal do código, o que não é representado por essa funcionalidade.

Esse diagrama é gerado de forma automática para todas as funções dos arquivos de entrada, desde que instalado o GraphViz e outras configurações tenham sido definidas da forma como mencionado. Como essa funcionalidade ajuda a visualizar o fluxo do código e é de fácil configuração (já que após uma vez configurada, pode ser reaproveitada para todas as documentações), ela acaba sendo uma ferramenta interessante.

Exemplos de como esse diagrama é representado serão mostrados a seguir, para um caso de um diagrama mais simples, no qual uma função somente é acionada por outra, enquanto no outro caso é representado o fluxo de acionamento de diversas funções entre si. Esses diagramas podem ser visualizados abaixo:

Este é o diagrama das funções utilizadas por essa função:



Esse é o diagrama das funções que utilizam essa função:



Figura 47: Diagrama da Função updtae\_open.

Este é o diagrama das funções utilizadas por essa função:

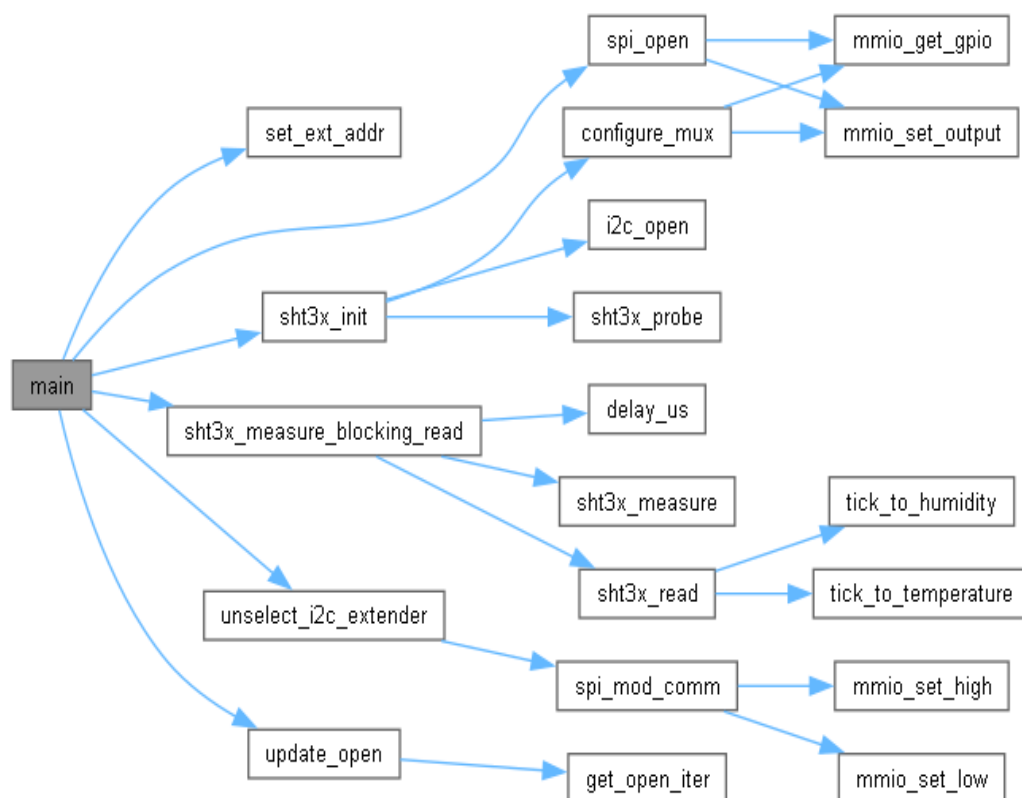


Figura 48: Diagrama da Função Main.

No diagrama acima da função `update_open` pode-se ver que ela é acionada pela função `main` e aciona a função `get_open_iter`. Já, no outro diagrama da função `main`, pode-se ver todas as funções acionadas por ela, bem como as funções acionadas por essas novas funções. Desse modo, é possível ver uma boa representação do fluxo do código.

Esse diagrama é gerado de forma correta e automática, na maioria dos casos, para todas as funções. Porém, em alguns raros casos, o diagrama vem com pequenos erros. Esses erros podem ser a não representação do acionamento de uma função, ou a representação de um acionamento não existente.

Esses erros ocorrem, normalmente, devido ao fato de como o Doxygen se propõe a documentar qualquer arquivo de diferentes formatos de entrada, existem muitas possibilidades de escritas diferente de código. Essas possibilidades ocorrem, tanto pela existência de diversos comandos de um formato específico (incluindo suas bibliotecas), bem como a variabilidade do tipo de formatos de entrada existente.

Para corrigir esses erros, é necessário estudar as causas deles, nos casos específicos em que eles ocorrem. Como existem diversas possibilidades de escrita de códigos, em tantos formatos, é muito difícil de se prever todos os possíveis erros.

Apesar disso, se percebeu alguns erros específicos que ocorreram na geração do diagrama de funções em Python e C. Tais falhas foram explicadas na seção 7.II a seguir e as correções desses erros, foram também relatadas, para que se facilite a geração do diagrama de funções de forma correta, em tais casos. Vale ressaltar, que é possível que falhas similares ocorram para outros formatos de arquivos. Por isso é sugerido que se leia a seção 7.II a seguir em caso de erros semelhantes com diferentes formatos de arquivo.

### iii. Comentários de funções

Conforme explicado em outros tópicos, as funções já devem vir com algumas documentações, com base nas configurações mencionadas acima. Algumas dessas documentações são os digramas de funções (seção 7.III.ii), uma linha com as variáveis que são enviadas a função e as linhas de código da própria função (dependendo das definições da seção 5.E).

Além dessas documentações mencionadas, é possível adicionar documentações específicas as funções. Isso pode ser feito ao terminar a sequência de documentação da função pela linha similar de uma declaração de função terminada com ;

Um exemplo seria a função:

```
uint32_t bme280_cal_meas_delay(const struct bme280_settings* settings){  
    //Comandos da função.  
    return max_delay;
```

Essa função, teria o término de sua documentação dada por uma linha com o seu próprio nome, acrescido de um ponto e vírgula ; no final. Então, os outros comandos, podem ser dados pelas linhas abaixo:

```
/*!  
 * @brief "Exemplo de briefing"  
 *  
 * @param[in] settings : struct bme280_settings* settings. "Pointer variable  
 * which contains the settings to be set in the sensor."  
 *  
 * @return max_delay. Result of API execution status.  
 *  
 * @retval > Value_A -> Return_A  
 * @retval > Value_B -> Return_B  
 *  
 */  
uint32_t bme280_cal_meas_delay(const struct bme280_settings* settings);
```

Na sequência de comandos acima, o comando `@brief` "Exemplo de briefing" fornece uma breve explicação da função. `@param[in] settings` : se refere as variáveis de entrada (`struct bme280_settings* settings`). Além disso, fornece uma breve descrição dela: "Pointer variable which contains the settings to be set in the sensor." `@return max_delay` se refere a qual é a variável de saída (note que a função retratada acima, retorna essa variável). Também, apresenta uma breve descrição dela: `Result of API execution status`.

`@retval`, por fim, se refere a associação entre valores assumidos por variáveis de entrada e saída. Dessa forma, foi retratado que caso a variável de entrada assuma o valor `Value_A`, a variável de saída irá ser `Return_A`. Analogamente, foi retratado que a variável de saída irá ser `Return_B` no caso em que a variável de entrada assuma o valor `Value_B`.

Existem outros comandos para documentar funções, que podem ser encontrados em buscas na web, mas aqui focou-se nos principais. Pode-se visualizar a documentação exemplificada acima, na figura abaixo:

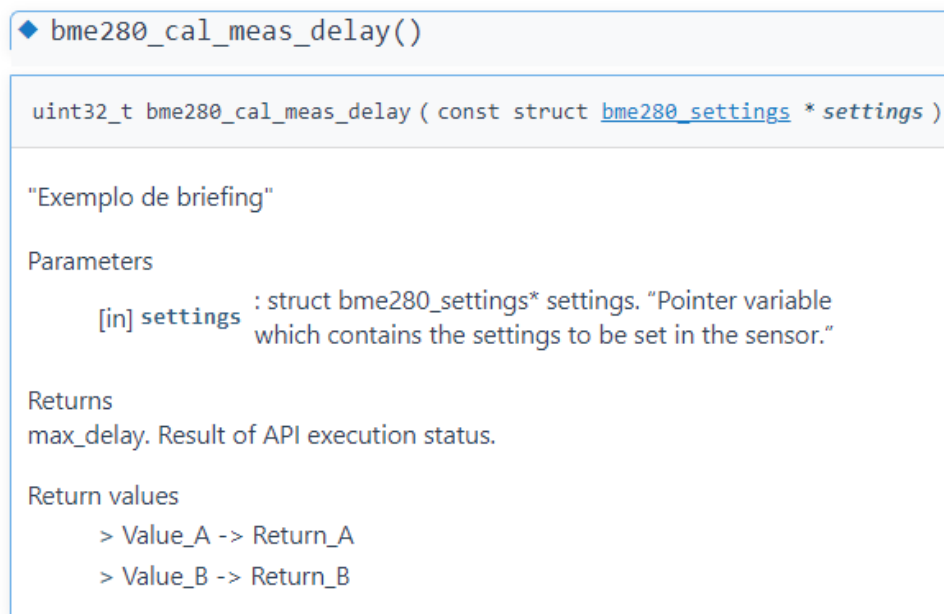


Figura 49: Exemplos de Documentação de função, em C.

É importante comentar sobre os casos relativos a uma função em C que começa com `static`. Nessa situação, deve-se escrever a linha com o comando de acionamento da função no término da sequência de comandos SEM o `static`. Dessa forma, a documentação irá sair da forma como desejado. Esse ponto será melhor detalhado na seção 8.I.

Também, pode se fazer as documentações em Python. Para isso, os comandos devem estar entre os caracteres de comentários, respectivos de sua linguagem (ex: `#` em Python), conforme mencionado anteriormente. Além disso, em Python deve-se usar os comandos dentro da função em que se quer comentar (sem a linha com o seu próprio nome, acrescido de um ponto e vírgula `;` no final). Feitas essas trocas, pode-se usar o resto dos comandos da mesma maneira que foram descritos para C. Um exemplo, em Python, é dado a seguir:

```
def F2(var_a):
```

```

##
# @brief "Exemplo de briefing"
#
# @param[in] var_a. Descrição da variável de entrada.
#
# @return result. Descrição da variável de saída.
# @retval > 0 -> 1.
# @retval > 1 -> 0.
#
time.sleep(5)
if var_a==0:
    result==1
elif var_a==1:
    result==0
return result

```

Esses comandos, geram a documentação abaixo:

F2()

F2 ( var\_a )

"Exemplo de briefing"

Parâmetros

[in] var\_a. Descrição da variável de entrada.

Retorna

result. Descrição da variável de saída.

Valores Retornados

> 0 -> 1.

> 1 -> 0.

Definição na linha 17 do arquivo [python\\_example.py](#).

```

17 def F2(var_a):
18
19
28
29     time.sleep(5)
30     if var_a==0:
31         result==1
32     elif var_a==1:
33         result==0
34     return result
35

```

Esse é o diagrama das funções que utilizam essa função:

F1

→

F2

Figura 50: Exemplos de Documentação de função, em Python.

55

Vale lembrar que as linhas dos comandos e os diagramas representados nas funções acima, são referentes a configurações de outras seções desse tutorial (seção 6.V e seção .II).

Em Python, também pode se gerar essas documentações de funções, substituindo os # por """! no começo e """ no final das linhas dos comentários (removendo os # no início das linhas). Um exemplo pode ser dado abaixo pela reescrita dos mesmos comandos mostrados acima:

```
"""!  
    @brief "Exemplo de briefing"  
    @param[in] var_a. Descrição da variável de entrada.  
    @return result. Descrição da variável de saída.  
    @retval > 0 -> 1.  
    @retval > 1 -> 0.  
    """
```

É possível também gerar textos que irão aparecer nos comentários das funções. Um exemplo de texto em Python (deve-se fazer as mudanças já mencionadas para o uso dessa funcionalidade em C) é dado por:

```
#####  
#  
# DAT/EMI and the Brazilian Center for Research in Energy and Materials  
# (CNPEM) are not liable for any misuse of this material.  
#  
# @file waveform_controller.py  
#  
#####  
#
```

Outra observação importante, é que existem outras formas de gerar os comentários de funções. Apesar disso, a fim de se manter esse tutorial conciso, se focou somente em mostrar os métodos aqui descritos. É importante ressaltar que esses métodos são válidos para diversos casos, e por isso foram aqui relatados.



## iv. Páginas de arquivos

Páginas de arquivos são páginas que são geradas automaticamente pelo Doxygen (desde que feitas as configurações acima). Elas já têm um padrão pré-definido, mostrando algumas funcionalidades do arquivo. Apesar disso, é possível adicionar funcionalidades nela, como textos e imagens totalmente personalizados (conformes os comandos mostrados acima para a página principal). Também é possível comentar funções específicas dos arquivos.

O processo de geração dessas páginas de arquivos é semelhante aos processos de gerações das páginas principais e relacionadas, descritos acima. Porém, existe a diferença de que a primeira linha de comando, deve ser substituída por:

```
# @file file.py
```

Na linha acima, file.py seria o nome do arquivo do qual se está gerando a página. Ele deve estar dentro de um dos diretórios fontes (mencionados na seção 6) escolhidos. O comando acima foi escrito em Python, sendo que o comando análogo pode ser escrito em C. Pode-se ver abaixo a página gerada:

`diff = setpoint-Te`

### Descrição detalhada

"Exemplo de briefing"

## "Seção na página main.py"

"Exemplo de descrição com Doxygen."


Autor  
"Eric Sonagli Abbade"

"Exemplo de Texto da página main.py".

## "Nova Seção na página main.py"

"Texto da seção..."

- "Tópicos"



"Copyright (c) 2024 All rights reserved.."

Definição no arquivo [main.py](#).

Figura 51: Personalização da página de arquivo.

É possível notar na imagem acima um exemplo da personalização que foi feita na página de arquivos. Note que o conteúdo dela foi mostrado embaixo das outras funcionalidades, já explicadas, da página de arquivo. O trecho de código usado (Python) para a geração dessas características, pode ser visto abaixo:

```
##
# @file main.py
#
# @brief "Exemplo de briefing"
#
# @section new_page_section "Seção na página main.py"
# "Exemplo de descrição com Doxygen."
#
# @author "Eric Sonagli Abbade"
#
# "Exemplo de Texto da página main.py".
#
# @section new_page_new_section "Nova Seção na página main.py"
# "Texto da seção..."
# - "Tópicos"
#
#
# 
# <div style="clear: both"></div>
#
# "Copyright (c) 2024 All rights reserved.."
#
```

Pode-se notar no trecho de código acima, que essa foi a página referente ao arquivo: main.py , conforme o comando já mostrado. É importante lembrar que para escrever esse trecho em C, basta fazer as mudanças necessárias mencionadas anteriormente.

## IV. Tópicos

Será documentado sobre a aba de tópicos, nessa seção 7.IV. Ela pode ser acessada conforme a figura a seguir.



Figura 52: Acesso da Aba de Tópicos.

Após entrar nessa aba, se tem a opção de escolher entre vários tópicos. Um exemplo disso, é dado abaixo:

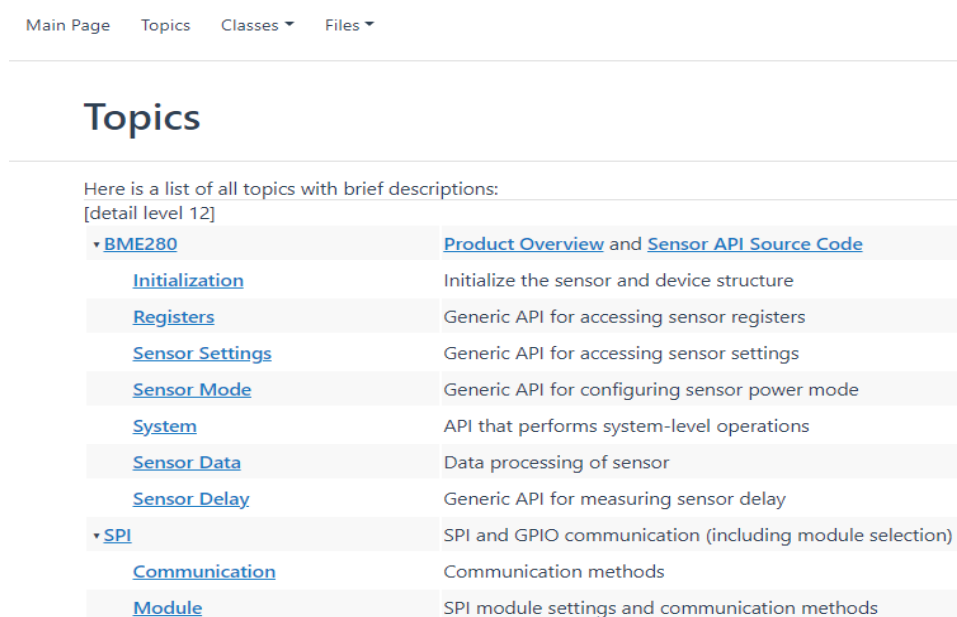


Figura 53: Exemplo de tópicos.

Esses tópicos da figura acima representam alguma funcionalidade do código. Ao se escolher algum deles é possível se observar diversas documentações referentes a funcionalidade escolhida. Note que dentro do tópico existem outro subtópicos. Um exemplo disso é que dentro de SPI, pode se escolher Module (esse exemplo será mostrado abaixo).

Criar tópicos organiza a documentação de funções em um mesmo tópico. Sem eles, as funções seriam documentadas nas páginas de arquivos. Portanto, a **aba dos tópicos tem como funcionalidade reorganizar as documentações das funções que apareceriam nas páginas de arquivos**. Portanto, pode-se escolher se as documentações das funções devem aparecer nas páginas de arquivos, ou em um tópico específico, conformes as preferências de cada projetista.

Sem configurações específicas, essa aba não é gerada pela documentação. Para definir um módulo basta usar os comandos:

```
# * \defgroup spi
```

Além disso, para definir um novo módulo dentro desse módulo antigo, basta usar os seguintes comandos:

```
/**
 * \ingroup spi
 * \defgroup spiComm Communication
 * @brief Communication methods
 */
```

Para gerar incluir funções dentro de um tópico (no caso: `Module`), basta acrescentar a linha:

```
* \ingroup Module
```

Esses comandos mostrados acima foram escritos em C (é necessário fazer as alterações mencionadas para Python). Ele deve ser acrescentado no início dos comentários de uma função (funcionalidade comentada na seção 7.III.iii), da qual se deseja adicionar em um tópico. Deve se substituir no lugar de: `Module`, o nome do módulo que se deseja criar.

Essa aba de tópicos existe para que se agrupem as documentações de funções que realizem tarefas correlatas em um mesmo tópico. Para ilustração desse processo, será mostrado um exemplo no qual se colocou a documentação da função: `int read_data(int address, char* rx, int len)`, no tópico:

```
\ingroup spiModule:
/**
 * \ingroup spiModule
 * @brief Reads digital data at given address
 * @param[in] address address
 * @param[out] rx Buffer to write data to
 * @returns SPI transfer operation result
 * @retval >=0 Success
 * @retval <0 Failure
 */
int read_data(int address, char* rx, int len);
```

É importante ressaltar que foram usados vários comandos explicados na seção 7.III.iii e outras seções. No tópico, o trecho acima do código em C irá gerar a seguinte documentação:

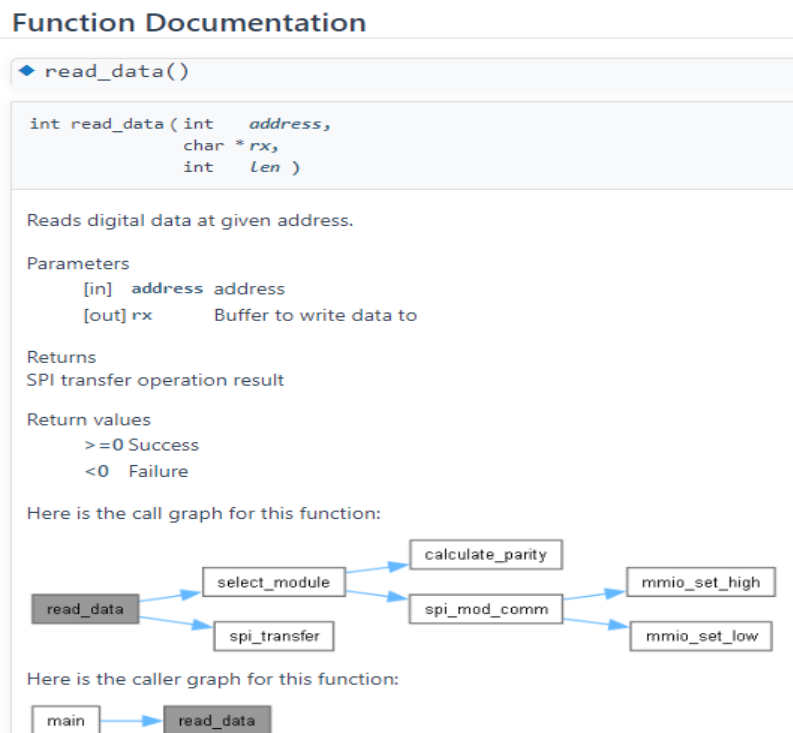


Figura 54: Exemplo de Documentação no Tópico.

Na figura acima, é possível se observar que diversos pontos da função foram documentados. Esses pontos são muito similares aos que foram citados na Aba de arquivo na seção 7.III.

## V. Namespace

Namespace (espaço de nome) é um conjunto de informações representadas pelo mesmo nome. No caso aqui se representa todas as informações representadas por: “main” que são justamente as próprias estruturas de dados, variáveis e funções que podem ser vistas nessa página. Por isso, não será comentado mais a respeito sobre o “namespace” já que seus parâmetros são os mesmos explicados dessa página.

Estrutura de dados, no caso, se referem as classes do Python. Ao se clicar nela, além de breves descrições, é possível ver suas funções. Como se explicará especificamente sobre as funções a seguir, tal explicação acaba também servindo para documentar os detalhes das classes. Na figura abaixo, pode se acessar a aba dos namespaces:

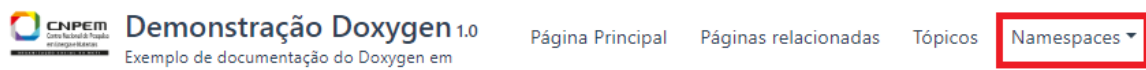


Figura 55: Acesso da Aba dos Namespaces.

Nessa aba pode-se escolher um arquivo, conforma a figura abaixo:

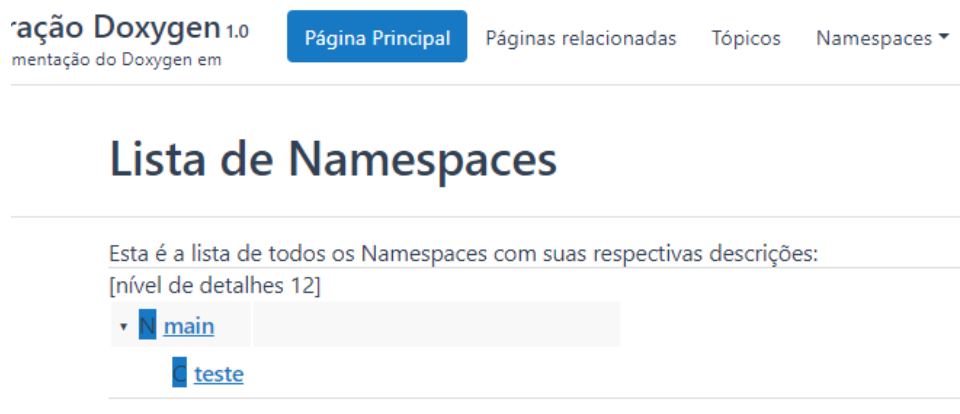


Figura 56: Escolha de Arquivos na Aba dos Namespaces.

Então, pode-se ver as classes do Python de tais arquivos, conforme pode ser visualizado na figura abaixo:

ina Principal   Páginas relacionadas   Tópicos   Namespaces ▾   Estruturas de dados ▾   Arquivos ▾

---

[Membros Públicos](#)

## Referência da Classe teste

---

### Membros Públicos

[funcao](#) (self)

---

### Descrição detalhada

Definição na linha [127](#) do arquivo [main.py](#).

### Documentação das funções

◆ funcao()

funcao ( self )

Definição na linha [128](#) do arquivo [main.py](#).

```
128 def funcao(self):
129     #Função dentro da class teste. Inserir comandos dentro da função.
130     pass
131
```

---

A documentação para essa classe foi gerada a partir do seguinte arquivo:

- C:/Users/eric.abbade/Downloads/tccin/[main.py](#)

Figura 57: Visualização das Classes na Aba dos Namespaces.

Na figura acima, pode-se perceber as principais configurações da classe “teste” e serve como exemplo da demonstração das configurações de classes na aba dos Namespaces. É importante ressaltar que a **geração das documentações das classes na aba dos Namespaces é feita de forma automática**, quando as classes já foram criadas.

## VI. Estrutura de Dados

Por fim, irá se comentar sobre a aba das Estrutura de Dados. Nela, é possível visualizar as classes do Python e as structs do C. Essa aba pode ser acessada conforme a figura abaixo:

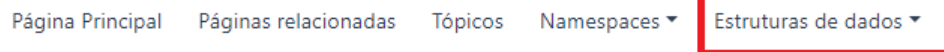


Figura 58: Acesso da Aba dos “Estrutura de dados”.

Após essa aba ter sido acessada é possível visualizar diferentes estruturas de dados (como classes do Python ou structs do C), conforme a figura abaixo:

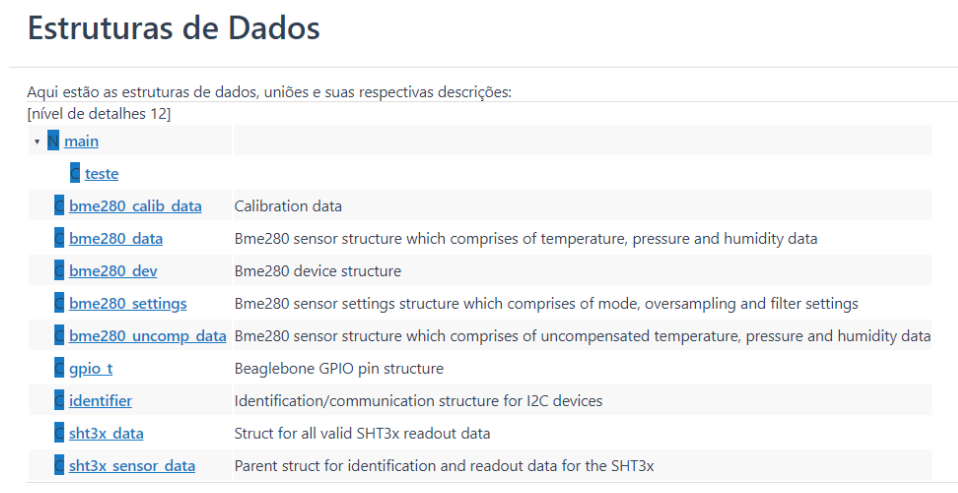


Figura 59: Escolha de Estrutra na Aba dos “Estrutura de Dados”.

Por fim, pode-se escolher uma das opções da figura acima para se visualizar os parâmetros da classe Python, ou da struct em C. Essa visualização é muito semelhante à da aba “Namespaces” (seção 7.V). Por isso, sugere-se a visualização dos parâmetros da classe Python, mostrada na seção 7.V, para a compreensão do que é mostrado na aba das Estrutura de Dados.

É importante ressaltar que após a criação das classes Python, ou das structs C, **a documentação dessas estruturas de dados é gerada de forma automática.**



## 8. Observações para documentação de funções (Python e C)

Devem ser tomados alguns cuidados para a geração dos diagramas de funções para Python. Portanto, nessa seção irá se retratar alguns dos procedimentos necessários para que o processo de geração do diagrama de funções, ocorra de forma adequada.

É possível que algumas outras linguagens apresentem características semelhantes. Por isso, caso se perceba que o diagrama de funções de um arquivo em outro formato não esteja sendo gerado da forma como planejado, sugere-se a leitura desses tópicos a fim de averiguar a possibilidade de o arquivo necessitar configurações similares ao que será descrito abaixo.

### I. Doxygen e C na documentação de funções

Funções que começam com `static` não são acionadas. Para corrigir isso, basta escrever o comentário delas no começo, como descrito na seção 7.III.iii. O começo do comentário delas deve ser escrito SEM `static`. Dessa forma, uma função:

```
static void parse_device_settings(const uint8_t* reg_data, struct
bme280_settings* settings) {
//comandos da função.
}
```

Essa função irá aparecer documentada, caso se adicione o seguinte trecho no código (que pode estar em qualquer lugar, desde que em uma linha anterior em relação a função acima):

```
void parse_device_settings(const uint8_t* reg_data, struct bme280_settings*
settings);
/*!
 * @brief This internal API reloads the already existing device settings in
the
 * sensor after soft reset.
 *
 * @param[in] dev : Structure instance of bme280_dev.
 * @param[in] settings : Pointer variable which contains the settings to
 * be set in the sensor.
 *
 * @return Result of API execution status
 *
 * @retval 0 -> Success.
 * @retval > 0 -> Warning.
 * @retval < 0 -> Fail.
 */
```

## II. Doxygen e Python no acionamento de funções

Um problema ocorre quando funções no Python as quais tem o acionamento de outra função (fora dessa função), logo após seu término. Isso ocorre quando não existe nenhum comando intermediário entre o término da função e o novo comando de acionamento.

Para corrigir isso, basta criar um comando intermediário entre o término da função e o novo comando de acionamento.

Esse problema, pode ser ilustrado pelo exemplo a seguir:

```
def fun1():
    teste1=0 #Comando qualquer que pode ser substituído por outro(s)
comando(s) qualquer(is).

def fun2():
    teste2=0 #Comando qualquer que pode ser substituído por outro(s)
comando(s) qualquer(is).

def fun3():
    fun1() #Comando qualquer que pode ser substituído por outro(s) comando(s)
qualquer(is).

fun2() #Acionamento de fun2() que está depois de fun3() e em tese, não deveria
ser mostrado no diagrama da função fun3.
```

O resultado desse trecho de código no diagrama de funções é tal que:

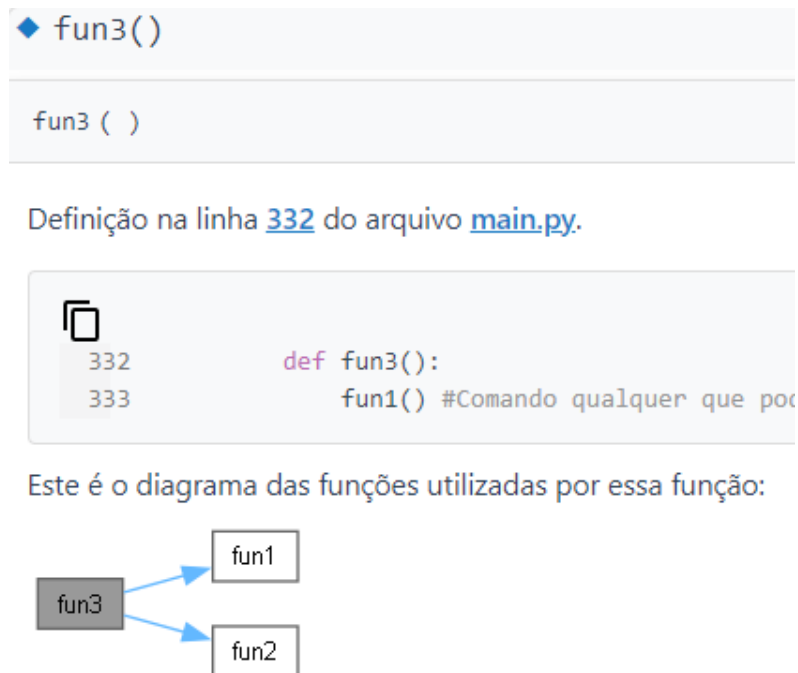


Figura 60: Diagrama da Função fun3 mostrando um acionamento inexistente.

No exemplo acima, pode-se observar o diagrama de função mostrando a função fun3 acionando a fun2, mesmo que isso não ocorra na prática. Para corrigir isso, basta colocar um comando intermediário entre o final da fun3 e o acionamento da fun2. Isso pode ser feito mesmo que esse comando extra não realize nenhuma funcionalidade no código.

Um exemplo do código acima, corrigido, é tal que:

```
def fun1():
    teste1=0 #Comando qualquer que pode ser substituído por outro(s)
comando(s) qualquer(is).

def fun2():
    teste2=0 #Comando qualquer que pode ser substituído por outro(s)
comando(s) qualquer(is).

def fun3():
    fun1() #Comando qualquer que pode ser substituído por outro(s) comando(s)
qualquer(is).

exemplo=0 #Comando qualquer que pode ser substituído e não precisa ter
funcionalidade no código.
fun2() #Acionamento de fun2() que está depois de fun3() e em tese, não deveria
ser mostrado no diagrama da função fun3.
```

Dessa forma, o diagrama do fluxo irá aparecer corrigido, de forma que:

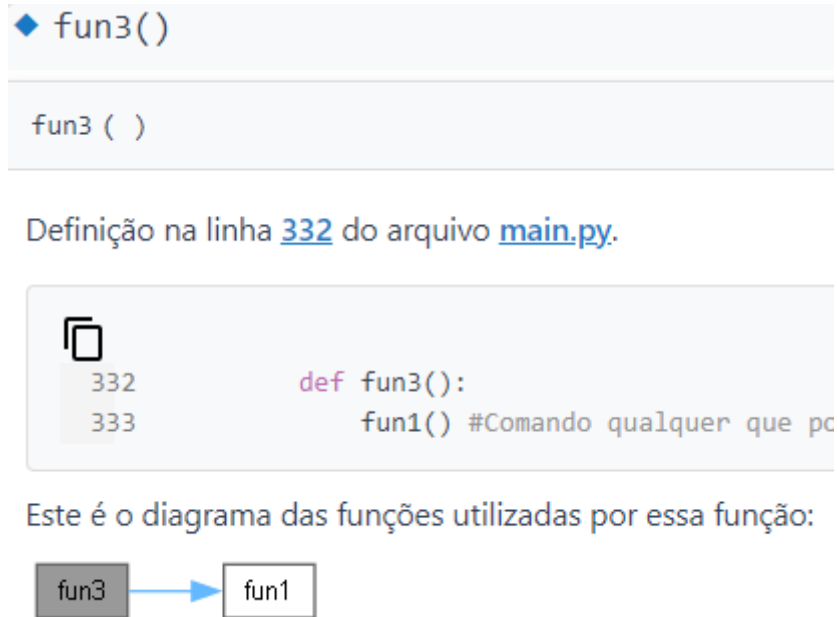


Figura 61: Diagrama da Função fun3 mostrando somente o acionamento correto.

Isso mostra que o uso da linha extra: exemplo=0 fez com que o diagrama de funções fosse gerado de forma correta.

### III. Doxygen e Python para programação orientada a objeto

Não é recomendado o uso do mesmo nome para métodos em diferentes classes do Python no Doxygen. Por vezes, isso gera os diagramas de funções que deveriam ser feitos somente com um método de uma classe, para todos os métodos com o mesmo nome de classes diferentes, ou até mesmo a exclusão de métodos do diagrama. Portanto, ao invés de escrever:

```
class Primeira_Classe:
    def funcao(self):
        #Insira os comandos da função
        pass

class Segunda_Classe:
    def funcao(self):
        #Insira os comandos da função
        pass
```

, é importante que se nomeie de forma, distinta as funções (“funcao”) das classes diferentes (Primeira\_Classe e Segunda\_Classe), o que pode ser feito ao se nomear a função da segunda classe para: funcao2, de modo que:

```

class Primeira_Classe:
    def funcao(self):
        #Insira os comandos da função
        pass

class Segunda_Classe:
    def funcao2(self):
        #Insira os comandos da função
        pass

```

Uma outra solução, seria a procura de parâmetros de configurações e de comandos de escrita no código que ocasionem na solução do problema mencionado acima no uso de funções de mesmo nome em classes diferentes.

Os acionamentos de métodos por meio de funções específicas de bibliotecas podem aparecer nos diagramas de funções, caso se use uma função no começo da classe com todos os comandos dos acionamentos de métodos por meio de funções específicas de bibliotecas. Para melhor explicação do que foi descrito, imagine que se use uma biblioteca (`import clock`) que acione funções (criadas pelo próprio desenvolvedor do código que está sendo documentado, nesse exemplo será denotada por: `def funcao_final(self)`), pelo comando: `clock(funcao_final)`. Esse comando: `clock(funcao_final)` está dentro de uma função: `def funcao_inicial(self)`, sendo que o objetivo da documentação é gerar no diagrama de funções que a função: `funcao_inicial(self)` aciona a função: `funcao_final(self)`.

Nesse caso, basta escrever o comando: `clock(funcao_final)` dentro de uma função de nome qualquer (no caso, denotada por: `def auxiliar_doxygen(self)`), de modo que esta função `auxiliar_doxygen(self)` esteja antes do comando `clock(funcao)` que tem funcionalidade no código e deseja-se representar no diagrama de funções.

A função `auxiliar_doxygen(self)`, e seus comandos (no caso: `clock(funcao_final)`) não terão nenhuma finalidade no código, mas servem de auxílio, para que o acionamento da função `funcao_final(self)` pelo comando `clock(funcao_final)` apareça no diagrama de funções do Doxygen.

Caso, se deseje acionar mais funções pelo comando `clock`, como por exemplo uma nova função: `def outra_funcao_final(self)`, basta colocar o comando: `clock(outra_funcao_final)`, na mesma função: `auxiliar_doxygen(self)`. Esse comando: `clock(outra_funcao_final)`, pode estar em qualquer outra função (no exemplo será usado que ela está dentro de: `def funcao_inicial2(self)`).

Além disso, caso se use um outro comando para acionar funções, como: `new_clock()` que foi baixado de outra biblioteca: `import new_clock`, basta colocar todos os comandos `new_clock()` que se deseja usar também na mesma função: `auxiliar_doxygen(self)`.

Se por exemplo, `new_clock()` acionar uma função: `def funcao_final_new_clock()`, por meio do comando: `new_clock(funcao_final_new_clock)` (que está dentro de uma função: `def funcao_inicial3(self)`) e todo o procedimento descrito acima estiver dentro da classe: `class biblioteca`, o código completo de todos os comandos, é dado por:

```
#Importando as bibliotecas:
import clock
import new_clock

class biblioteca:
    #Função somente usada para que o Doxygen gere os diagramas de função:
    def auxiliar_doxygen(self):
        clock(funcao_final)
        clock(outra_funcao_final)
        new_clock(funcao_final_new_clock)

    #Funções que serão acionadas pelos comandos das bibliotecas:
    def funcao_final(self):
        #Insira os comandos da função
        pass

    def outra_funcao_final(self):
        #Insira os comandos da função
        pass

    def funcao_final_new_clock(self):
        #Insira os comandos da função
        pass

    #Funções que irão acionar outras funções pelos comandos das bibliotecas
    def funcao_inicial(self):
        clock(funcao_final)

    def funcao_inicial2(self):
        clock(outra_funcao_final)

    def funcao_inicial3(self):
        new_clock(funcao_final_new_clock)
```

Dessa forma, é importante ressaltar que o diagrama de funções será gerado, somente por causa da função: “auxiliar\_doxygen” que não realiza nenhuma outra função no código, além de auxiliar na geração do esquemático.

Funções só com `pass` não costumam aparecer no diagrama. Para corrigir isso, basta adicionar um comando (mesmo que não faça nada) antes do `pass`. Um exemplo de função que costuma não aparecer no diagrama é dado por:

```
def funcao():  
    pass
```

Para fazer com que ela apareça no diagrama, basta adicionar (no caso: `exemplo=0`) uma linha de comando qualquer:

```
def funcao():  
    exemplo=0  
    pass
```

## 9. Conclusão

Doxygen é uma ferramenta para a geração de documentação de códigos. Esse tutorial focou em documentações geradas no formato HTML, para códigos em diversas linguagens (especialmente C e Python). Nesse tutorial se descreveram diversos detalhes da instalação e uso dessa ferramenta. É importante lembrar que esse tutorial é apenas uma introdução consideravelmente detalhada de algumas de suas principais funcionalidades. Esse tutorial não explica todas as suas configurações, já que existem diversos parâmetros para se configurar.

Inicialmente, após uma introdução (seção 3) das características do Doxygen, foi explicado como é feita a sua instalação (além do GraphViz, que é importante para auxiliá-lo), na seção 4. Na seção 5, se teve uma breve explicação do uso do Doxygen pelo terminal, via o arquivo de configuração. Depois, se explicou como configurar o Doxywizard para que se utilize alguns de seus parâmetros, na seção 6.

Doxygen é capaz de gerar documentações muito detalhadas e customizadas. Essa ferramenta tem um número muito grande de funcionalidades, o que dificulta que seja utilizado com seu pleno potencial por usuários sem uma certa experiência. Apesar disso, é possível fazer um uso mais simples e rápido dela (usando até a seção 5 do tutorial), de modo a se ter uma documentação razoavelmente detalhada. Visando o desenvolvimento de uma documentação ainda mais complexa, é possível se baser nos tópicos 6, 7, 8 e 9 desse relatório.

Como o Doxygen se propõe a documentar diversos formatos de arquivos (sendo que cada um tem diversos comandos), isso gera uma dificuldade na previsão de como ele irá lidar em cada caso específico. Por isso, é necessário realizar um teste, para que se observe como tais comandos interagem com o Doxygen. Caso, o Doxygen gere uma documentação diferente da esperada é necessário um estudo específico para o caso, de modo a evitar tais erros. Exemplos disso, foram relatados na seção 8, nos quais se explicou como corrigir alguns desses erros para o Python e C.

Por tudo isso, pode-se concluir que esse tutorial foi capaz de descrever as principais funcionalidades do Doxygen, para seu uso e instalação. Foram explicados os tópicos detalhados no objetivo e na introdução. Portanto, é possível afirmar que esse tutorial atingiu seu objetivo com sucesso.