Software documentation - Doxygen

Andreas Seidl

What to expect:

- What is software documentation?
  - Types of software documentation
- Doxygen
  - How does it work?
  - What are the benefits?
  - How do you use it?
  - Examples

Software documentation

- All written documents and materials dealing with a software product´s development and use
- Provides information about a software system to users, developers and other stakeholders
- Easy to understand
- Maintenance

Types of software documentation

- User documentation
- Technical documentation
- Operations documentation
- Marketing documentation
- **API documentation**

Doxygen

- Documentation generation tool
- Generates output formats like HTML, PDF, LaTex etc.
- Idea:
  - You should write comments anyway
  - Enhance those comments with special Syntax
  - Automatically generate Documentation from those coments

Doxygen

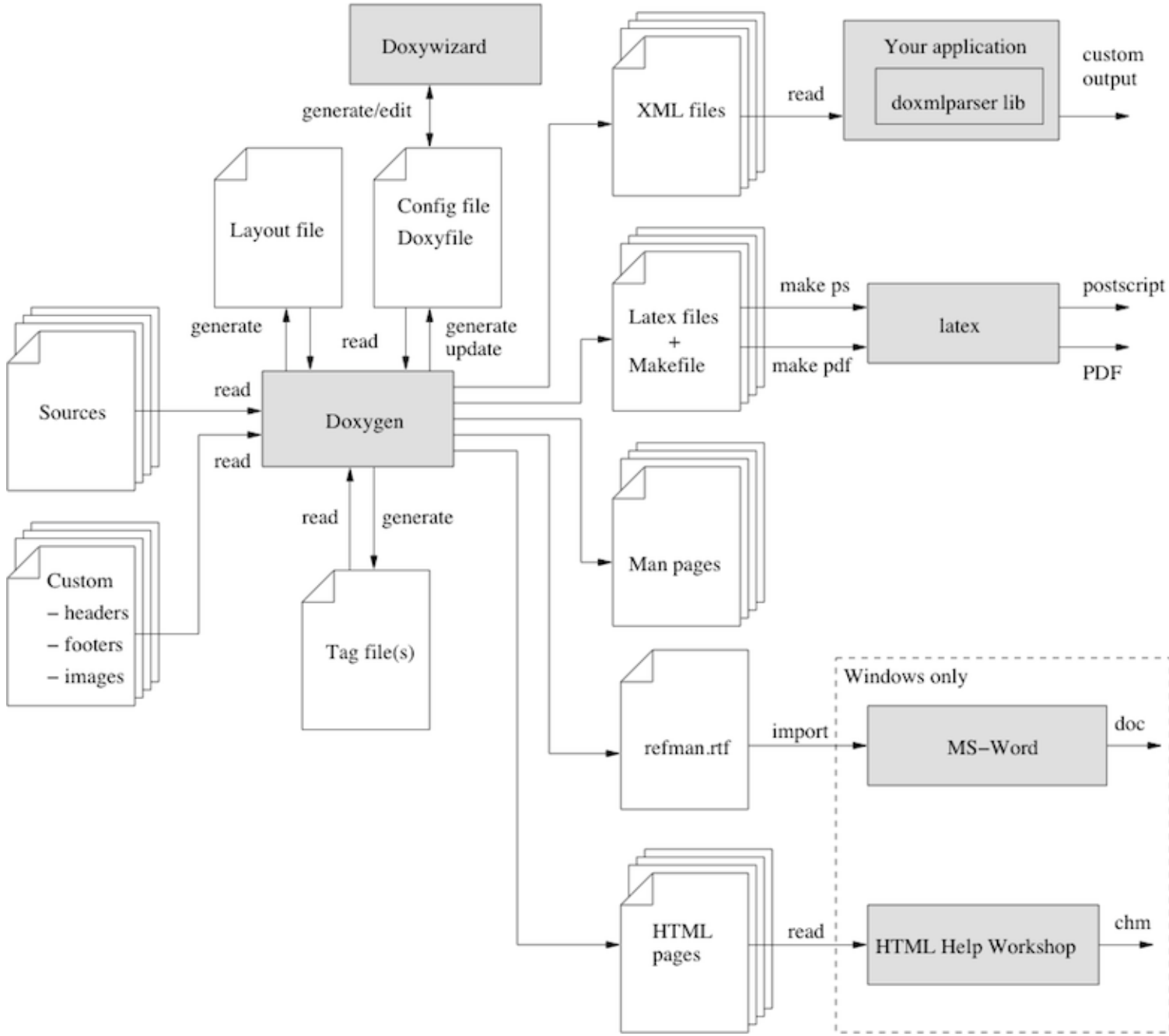
- Standard for C++
  - C, Objective-C, C#, PHP, Java, Python among others
- Open-Source
- Windows, Linux, macOS

Doxygen

- Extracts information from source code:
  - Function descriptions
  - Variable declarations
  - Comments
  - Doxstrings
- API documentation
  - Document code as you write it
- User & Technical documentation
  - Including markdown files, or other

Benefits

- Automatically update Documentation, if changes to the source code is made
- Easily integrated
- Hard to document something old
- Little effort
  - Plain text
  - Automaticall generates class and collaboration diagram in HTML


Doxywizard
generate/edit
Layout file
Config file
Doxyfile
generate
read
generate
update
Sources
read
Doxygen
read
Custom
– headers
– footers
– images
read
generate
Tag file(s)
XML files
read
Your application
doxmlparser lib
custom
output
Latex files
+
Makefile
make ps
make pdf
latex
postscript
PDF
Man pages
Windows only
refman.rtf
import
MS–Word
doc
HTML
pages
read
HTML Help Workshop
chm

How to use Doxygen

- Configuration file
  - Text editor or doxywizard
  - Similar to Makefile
    - TAGNAME = VALUE
  - In large projects:
    - assign root directory to INPUT tag
    - Use FILE_PATTERNS
    - EXCLUDE, EXCLUDE_PATTERNS can be used

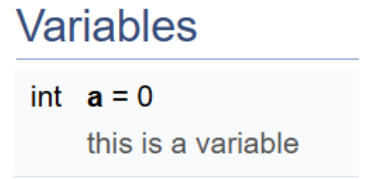
Configure Doxygen

- INPUT
- OUTPUT_DIRECTORY
- FILE_PATTERNS
- EXTRACT_ALL
- Diagrams
  - HAVE_DOT
  - CLASS_DIAGRAMS
  - UML_LOOK

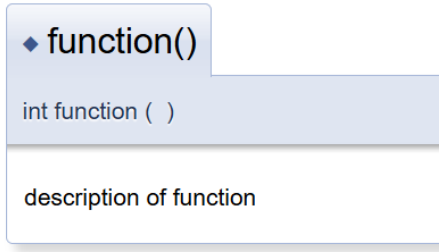
```
# Doxygen Configuration File

# Project Information
PROJECT_NAME = "My Project"
PROJECT_NUMBER = "1.0"
PROJECT_BRIEF = "My Project Documentation"

# Source Files
INPUT = src/
FILE_PATTERNS = *.cpp *.h

# Output Format
GENERATE_HTML = YES
GENERATE_LATEX = NO

# Styles and Templates
HTML_HEADER = header.html
HTML_FOOTER = footer.html
HTML_STYLESHEET = doxygen.css

# Miscellaneous
EXTRACT_ALL = YES
EXTRACT_PRIVATE = NO
```

Important syntax

```
 //this is a variable
int a = 0;
```




```
/**

description of function

*/
int function(){
}
```



Where to put it

- Usually in front of member
- Structural commands inside the documentation block

```
/**

 \brief description of function

 */
int function(){
}

/**
 \fn function
 \brief description of function

 */
```

Structural commands

```
\struct
\namespace
\union
\enum
\fn
\var
\def
\typedef
\file
\namespace
\package
```

Special commands

- \author
- \version
- \code
- \note
- \warning
- \todo
- \bug
- \addindex
- \param
- \throws
- \pre
- \post
- \link
- \ref
- \page
- \section

```c
/**
 * \file test.c
 * \brief A simple program to demonstrate Doxygen
 *
 * This program prints "Hello, World!" to the console.
 *
 * \author Andreas
 * \date 22.12.2022
 */

#include <stdio.h>

 char * hello = "Hello, World!";
/**
 * \brief The main function
 * \return 0 on success, non-zero on error
    \param char* h
 */
int main(char * h) {
    printf("%s\n",h);
    return 0;
}
```

test.c File Reference

A simple program to demonstrate Doxygen. More...

`#include <stdio.h>`

Functions

int **main** (char *h)
The main function. More...

int **function** ()

Variables

char * **hello** = "Hello, World!"

int **a** = 0
this is a variable

Detailed Description

A simple program to demonstrate Doxygen.

This program prints "Hello, World!" to the console.

Author
Andreas

Date
26.01.2023

Function Documentation

function()

int function ( )

description of function

main()

int main (char * h)

The main function.

Returns
0 on success, non-zero on error

Parameters
char* h

```cpp
/**
   \file point.cpp
 * \class MyClass
 * \brief This class performs some operations
 *
 * This class provides the following functionality:
 * - Addition of two numbers
 * - Subtraction of two numbers
 * - Multiplication of two numbers
 *
 * To use this class, you need to do the following:
 * 1. Create an instance of the class
 * 2. Call the appropriate member function
 * 3. Use the returned value
 *
 * \note this is a note
 * \bug this dose not work
 * Here is an example of how to use the class:
 * \code
 * MyClass mc;
 * int result = mc.add(3, 4);
 * std::cout << "Result: " << result << std::endl;
 * \endcode
*/
class MyClass {
  public:
```

Detailed Description

This class performs some operations.

This class provides the following functionality:

- Addition of two numbers
- Subtraction of two numbers
- Multiplication of two numbers

To use this class, you need to do the following:

1. Create an instance of the class
2. Call the appropriate member function
3. Use the returned value

Note

this is a note

Bug:

this dose not work Here is an example of how to use the class:

```cpp
MyClass mc;
int result = mc.add(3, 4);
std::cout << "Result: " << result << std::endl;
```

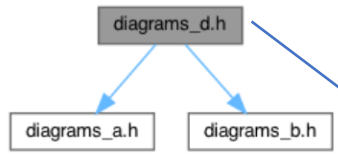
Diagrams

Main Page | Classes | Files

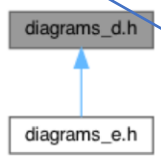
diagrams_d.h File Reference

```
#include "diagrams_a.h"
#include "diagrams_b.h"
```

Include dependency graph for diagrams_d.h:



This graph shows which files directly or indirectly include this file:



Go to the source code of this file.

Classes

class D

diagrams_a.h

```
#ifndef DIAGRAMS_A_H
#define DIAGRAMS_A_H
class A { public: A *m_self; };
#endif
```

diagrams_b.h

```
#ifndef DIAGRAMS_B_H
#define DIAGRAMS_B_H
class A;
class B { public: A *m_a; };
#endif
```

diagrams_c.h

```
#ifndef DIAGRAMS_C_H
#define DIAGRAMS_C_H
#include "diagrams_c.h"
class D;
class C : public A { public: D *m_d; };
#endif
```

diagrams_d.h

```
#ifndef DIAGRAM_D_H
#define DIAGRAM_D_H
#include "diagrams_a.h"
#include "diagrams_b.h"
class C;
class D : virtual protected  A, private B { public: C m_c; };
#endif
```

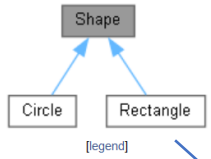
diagrams_e.h

```
#ifndef DIAGRAM_E_H
#define DIAGRAM_E_H
#include "diagrams_d.h"
class E : public D {};
#endif
```

Shape Class Reference abstract

The base class for shapes. More...

Inheritance diagram for Shape:



Public Member Functions

virtual	**~Shape** () Virtual destructor to allow for polymorphism.
virtual double	**Area** () const =0 Virtual function to calculate the area of the shape. More...
virtual double	**Perimeter** () const =0 Virtual function to calculate the perimeter of the shape. More...

Detailed Description

The base class for shapes.

Member Function Documentation

◆ Area()

virtual double Shape::Area ( ) const

Virtual function to calculate the area of the shape.

Returns

The area of the shape.

Implemented in **Rectangle**, and **Circle**.

◆ Perimeter()

virtual double Shape::Perimeter ( ) const

Virtual function to calculate the perimeter of the shape.

Returns

The perimeter of the shape.

Implemented in **Rectangle**, and **Circle**.

```cpp
/**
 * @class Shape
 * @brief The base class for shapes.
 */
class Shape {
 public:
  /**
   * @brief Virtual destructor to allow for polymorphism.
   */
  virtual ~Shape() {}

  /**
   * @brief Virtual function to calculate the area of the shape.
   * @return The area of the shape.
   */
  virtual double Area() const = 0;

  /**
   * @brief Virtual function to calculate the perimeter of the shape.
   * @return The perimeter of the shape.
   */
  virtual double Perimeter() const = 0;
};

/**
 * @class Rectangle
 * @brief A class for rectangles that inherits from Shape.
 */
class Rectangle : public Shape {
 public:
  /**
   * @brief Constructor for Rectangle class.
   * @param width The width of the rectangle.
   * @param height The height of the rectangle.
   */
  Rectangle(double width, double height) : width_(width), height_(height) {}

  /**
   * @brief Overridden version of Area from Shape.
   * @return The area of the rectangle.
   */
  double Area() const override {
    return width_ * height_;
  }

  /**
   * @brief Overridden version of Perimeter from Shape.
   * @return The perimeter of the rectangle.
   */
  double Perimeter() const override {
    return 2 * (width_ + height_);
  }

 private:
  double width_;  ///< The width of the rectangle.
  double height_; ///< The height of the rectangle.
};
```

Grouping

- Group things together or on a seperate page
  - Files, namespaces, classes, functions, variables..
- \defgroup
- \ingroup
- Compound entities can be in more than one group

In conclusion

- Powerful documentation generation tool
- Easy to create accurate and up-to-date documentation