Generating Documentation with Doxygen

CS 2ME3/SE 2AA4

Steven Palmer

Department of Computing and Software
McMaster University

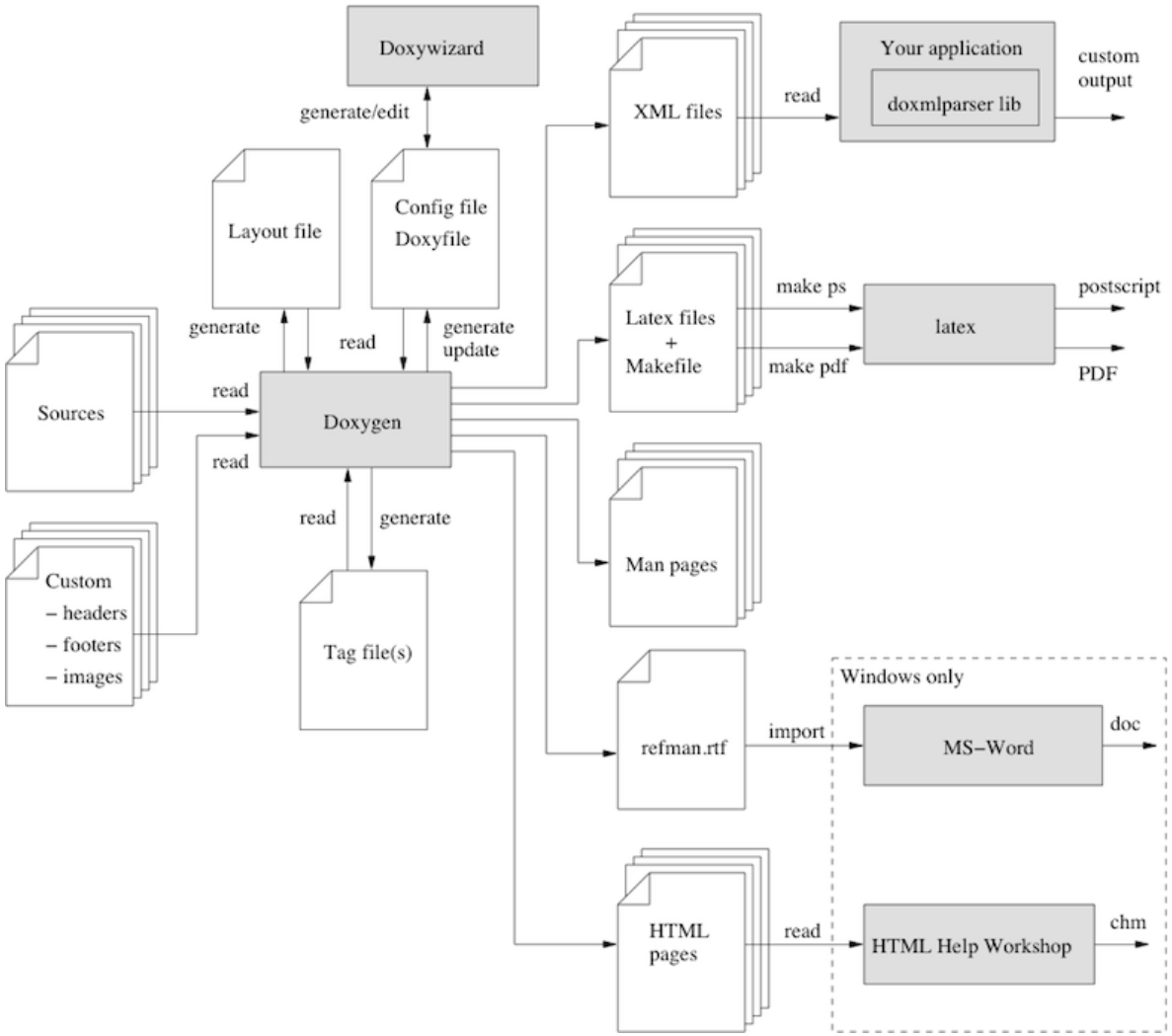
January 8 - 12

Outline

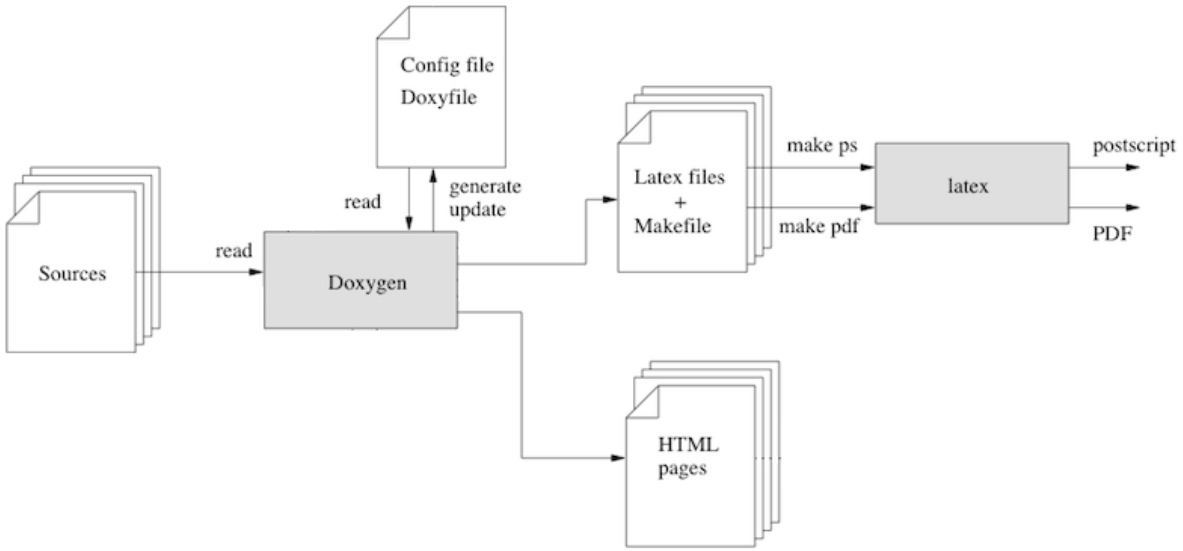
What is Doxygen?

- Doxygen is a tool used to generate documentation for code modules/classes.
- Comments with special syntax are used in source files to mark information that Doxygen should use.
- When Doxygen is run, it extracts the marked information from the source file(s) and compiles it into selected output formats.

Doxygen Information Flow



Doxygen Information Flow



- We will focus on the generation of LaTeX and HTML documentation.

Installing TeX Distribution

- You will need a TeX distribution.
  - TeX Live is recommended.
  - For Ubuntu-based Linux distributions you can install via `apt-get install texlive-full`.
  - For other operating systems see installation details.

Installing Graphviz

- Doxygen uses `dot`, which is part of the graphviz suite of graph visualization tools.
  - For Ubuntu-based Linux distributions you can install via `apt-get install graphviz`.
  - For other operating systems see installation details.
  - Depending on your operating system, you may need to add the graphviz bin folder to your path so that `dot` can be accessed via the command-line.

Installing Make

- You will need `make` to build a pdf from the Doxygen LaTeX output.
  - `make` should be available by default on Linux systems.
  - If you are using OS X and `make` is not available, you will need to install the Command Line Tools package provided by Apple.
  - If you are using Windows, you should install the MinGW environment. Make sure to add the MinGW bin folder to your path so that `make` is runnable from the command-line.

Installing Doxygen

- Finally, you will of course need to install `Doxygen` itself.
  - For Ubuntu-based Linux distributions you can install via `apt-get install doxygen`.
  - For other operating systems see further installation instructions.

Doxygen Comments in Python

- Doxygen comments in Python use the following simple structure:

```
## @command1 args
#  @command2 args
#  @command3 args
        ⋮
#  @commandn args
```

- In general, a Doxygen comment block directly precedes either a class definition, a function definition, or a field definition.

Documenting Classes

- Python classes are documented as follows:

```
## @brief A brief description of ClassX
#  @details A more detailed description of ClassX
class ClassX:
    ...
```

- Depending on the complexity of the class, @details may not be necessary.

Documenting Functions

- Python functions are documented as follows:

```
## @brief A brief description of methodX
#  @details A more detailed description of methodX
#  @param p1 A description of parameter p1
#  @param p2 A description of parameter p2
#  @return A description of the returned value
def methodX(p1, p2):
    ...
    return x
```

- Depending on the complexity of the function, @details may not be necessary.
- There should be an @param entry for every parameter of the function (possibly none). The parameter **self** in class functions should be omitted.
- @return is not necessary for void functions.

Documenting Fields

- Python fields are documented as follows:

```
    ## A brief description of fieldX
    fieldX = ...
```

Documenting Source Files

- At the top of each source file it is good practice to add some information about authors, date, etc.:

```
## @file File name
#  @title Title of source (module name?)
#  @author The author(s)
#  @date Date of last revision
```

Example

- See Box3D.py for a small example of Python code with Doxygen style comments.

Exercise 1

Exercise 1: Annotate Circle.py for doxygen

- In the T01b exercise folder you will find a small Python file called Circle.py.
- Add comments for doxygen to this file (you should make annotations for the file itself, the Circle class, and all of the methods in the Circle class).
- Using the Box3D.py example as a guide might be helpful.

Additional Commands

- The Doxygen snippets given in this tutorial as well as the Box3D.py example file provide the basics for documenting your code.
- Sometimes you may want to use additional commands to capture more details in your documentation.
- Consult the command documentation for the full listing of available documentation commands and descriptions.

Configuration File

- Producing documentation for a given set of source files with Doxygen requires a configuration file.
- A default configuration file can be generated via the command line with:

```
doxygen -g configFileName
```

- configFileName can be whatever you want.
- The configuration file needs to be edited to define your project.

Configuration File

- Of particular importance for a new configuration are the PROJECT_NAME (line 35) and INPUT (line 774) fields.
- PROJECT_NAME should be replaced with the name of the program you are documenting.
- INPUT should list all of the source files you wish to be included in the documentation. Alternatively, you can list a directory as INPUT and use FILE_PATTERNS (line 799) to determine which files will be included.
- There are several other options you can use to customize your generated documentation (refer to config documentation).

Document Generation

- Documentation is generated using the following command:

```
doxygen configFileName
```

- HTML and LaTeX documentation are default generated outputs.
- By default, the HTML documentation will be found in a new directory called html: look for index.html.
- The LaTeX documentation will be found in a new directory called latex. This folder will contain a makefile – you must call make to generate a pdf which will be called refman.pdf by default.

Exercise 2

Exercise 2: Generate doxygen documentation for Circle.py

Using the information given in the last few slides, create doxygen documentation for the Circle.py file you annotated in Exercise 1. This will require the following steps:

1. Generate a new configuration file.
2. Fill in the title and source file information in the configuration file.
3. Run doxygen to generate documentation.

Exercise 3

Exercise 3: Add file annotations to A1 source files

A GitLab repository for assignments has been created for each student in the course (you should have received an invitation to join). You will find this repo on GitLab under the name se2aa4_cs2me3_assignments_2018/<macid>.

1. Clone your assignment repository using git
2. Add file annotations (see slide 14) to SeqADT.py and CurveADT.py in the A1 src folder
3. Stage your changes using git
4. Commit your change using git
5. Push your changes to GitLab using git

Exercise 4

Exercise 4: Clone or pull your updates on Mills

1. Connect with mills.mcmaster.ca using ssh
2. If you've already cloned your project on Mills, then pull the changes. If not, clone your project.
3. You should see the changes you pushed in Exercise 3 in your files on Mills.

Reminder: it is always a good idea to check that your code compiles on Mills!

Reference

- Refer to the Doxygen documentation for further details about using Doxygen.