Doxygen

Doxygen
Common Use Cases For Doxygen
Basic Workflow
    1. Installation
    2. Commenting Code:
    3. Configuration
    4. Running Doxygen
    5. Review And Publish

Doxygen

- Doxygen is a documentation generation tool, primarily used for generating API documentation from annotated source code.
- *Need to be followed by every developer while adding static code.*
- Key Features of Doxygen:
    - **Automatic Documentation Generation**
    - **Output Formats**:
      Doxygen can generate documentation in multiple formats including HTML, LaTeX (for PDFs), RTF, XML, and man pages.
    - **Graphical Class Hierarchies**:
      Doxygen can generate inheritance diagrams and collaboration diagrams to visually represent the relationships between classes and other entities in your code.
    - **Customization**:
      The appearance and structure of the generated documentation can be customized using configuration files.
    - **Version Control Integration**:
      Doxygen can be integrated with version control systems like Git, SVN, or CVS to generate documentation that reflects the current state of the codebase.

Common Use Cases For Doxygen

- **API Documentation**:
  Developers use Doxygen to document APIs, making it easier for other developers to understand and use the code.
- **Class Documentation**:
  Doxygen helps in documenting class hierarchies, methods, and member variables, which is particularly useful for large object-oriented projects.
- **Collaborative Projects**:
  In open-source or team projects, Doxygen is used to keep everyone on the same page by maintaining up-to-date and easily accessible documentation.

Basic Workflow

1. Installation

Install the below packages in the setup to generate reports:

- ***sudo apt install doxygen***
- ***sudo apt install graphviz***
- ***Sudo apt install texlive-latex-base***

2. Commenting Code:

Header Files:

I. Add the doxy comment in top of the newly created header file

Example: For remotepowerexport.hpp

```
/*******************************************************************************
*
*  @copyright  (C) KPIT Technologies Limited
*              All rights reserved.
*              KPIT Technologies Limited owns all the rights to this work.This
*              work shall not be copied, reproduced, used, modified or its
information
*              disclosed without the prior written authorization of KPIT Technologies
*              Limited.
*   @file               RemotePowerExport.hpp
*   @brief
*   @details
*   @version   Revision 1
*   @date
*   @bug --
*   @warning --
*
* Version Histroy
*******************************************************************************
*SCR     | Author               | Version              | Description
*******************************************************************************
*Genesis
*******************************************************************************/
```

II. Add the doxy comments for the created class.

Example:

/ \class RemotePowerExport class*
\brief class to handle remote power export functionalities.

/

III. **Add the doxy comments for all the newly added parameters.**

Example:

*/** \brief string to store respStatus */*
std::string respStatus

Source Files:

- Add the doxy comment in top of the cpp file:

  Example:

  ```
  /**
   * @file RemotePowerExport.cpp
   * @brief Main file that includes Remote Power Export Related functions.
   */
  ```

- Add the Doxy Comments above Every Functions

  Example:

  ```
  /**
   * @brief Remote::PowerExport::RemotePowerExport::OnReceive()
   *
   * This Function Will receive the data.
   * @param rxtriggerdata
   * @return none
   */
  ```

3. Configuration

- Build and deploy the code after adding doxygen comments to ensure nothing wrong has been committed.
- Create One Folder and place the Doxyfile, mainpage.md and a png of component diagram there.
  -Folder_to_generate_doxygen _Document
    - Doxyfile
    - Mainpage.md
    - component.png
- Only do below changes in the doxyfile, refer some existing service:
  - PROJECT_NAME
  - PROJECT_BRIEF
  - OUTPUT_DIRECTORY
- Create two folders, inc and src.
  - Place all the feature headers along with other dependent header files in inc.
  - Place all the source files along with other dependent c++ files in src.
    - Ex: Power export files, along with common process For ECT service.

4. Running Doxygen

- Open the terminal in the same folder {Folder to generate doxygen}
  Run command:
  doxygen Doxyfile

5. Review And Publish

- In the generated document check for index.html which contains all the dataflow diag, classes and everything
- Verify the flow.
- Unzip the existing folder which contains the doxyfile. Zip yours and add it there.
- Push the changes to git.