My Project

Generated by Doxygen 1.8.17

Chapter 1

README.md

1.1 AIDoxygenCleaner

The `AIDoxygenCleaner` class handles the various phases of warning management, including detection, fixing, and application of fixes.

1.1.1 Directory Structure

```
ai_doxygen_cleaner
 README.md
 __init__.py
 ai_doxygen_cleaner.py
```

1.1.2 AIDoxygenCleaner Class

Located in the `AIDoxygenCleaner.py` file, the `AIDoxygenCleaner` class consists of the following methods:

1.1.2.1 <tt>detect_warnings_pre_fix()</tt>

This method is responsible for detecting Doxygen warnings before any fixes have been applied. It uses the `Doxy↩ WarningDetector` to detect warnings and saves the warnings to a JSON file.

1.1.2.2 <tt>insert_warnings_to_db()</tt>

This method is responsible for loading the pre-fix warnings from a JSON file and inserting these warnings into a database.

1.1.2.3 <tt>fix_doxygen_warning()</tt>

This method uses the `DoxygenWarningFixer` class to fix the detected Doxygen warnings.

1.1.2.4 <tt>apply_fix()</tt>

This method uses the `WarningPostProcessor` to apply the fixes to the warnings.

1.1.2.5 <tt>detect_warnings_post_fix()</tt>

This method detects any Doxygen warnings that still exist after the fixes have been applied. It uses the `Warning↩ PostProcessor` to detect post-fix warnings.

1.1.2.6 <tt>print_pre_post_fix_data_to_file()</tt>

This method prints the details of the warnings before and after the fixes were applied to a file. It uses the `File↩ Handler` for handling the printing operation.

1.1.2.7 <tt>generate_html_diff_page()</tt>

This method generates an HTML diff page to display the differences between the warnings before and after the fixes have been applied.

1.1.2.8 <tt>update_warnings_details_post_fix_in_db()</tt>

This method updates the warnings' details in the database after the fixes have been applied.
Each of these methods corresponds to a phase in the handling of Doxygen warnings. They are designed to be called in order, progressing through the detection, fixing, and application of fixes to Doxygen warnings.
Please note that this class is meant to be used as a part of a larger system, and its methods are not intended to be used standalone.

```
# Example of How to Use the AIDoxygenCleaner Functions
You can use the functions provided in the AIDoxygenCleaner class as follows:
"'python
from doxygen_warning_manager import AIDoxygenCleaner
# create an instance of the AIDoxygenCleaner
manager = AIDoxygenCleaner()
# perform the Doxygen configuration
manager.doxygen_configuration()
# detect warnings before applying any fixes
manager.detect_warnings_pre_fix()
# insert detected warnings into the database
manager.insert_warnings_to_db()
# fix the detected Doxygen warnings
manager.fix_doxygen_warning()
# apply the fixes
manager.apply_fix()
# detect warnings after the fixes have been applied
manager.detect_warnings_post_fix()
# print pre- and post-fix data to file
manager.print_pre_post_fix_data_to_file()
# generate an HTML page showing differences between the pre- and post-fix data
manager.generate_html_diff_page()
# update the warning details in the database after applying the fixes
manager.update_warnings_details_post_fix_in_db()
```

Make sure to import the `AIDoxygenCleaner` class from the `ai_doxygen_cleaner` module as shown above.
Note: Depending on your specific use case, you may need to use these functions in a different order or use only a subset of them. The example above shows how to use all the functions provided in the `AIDoxygenCleaner` class.

Chapter 2

README.md

2.1 AI Language Model

The `ai_language_model` module contains classes designed for interacting with AI language models. These classes support a variety of functionalities such as sending queries to the model, creating prompts and responses, and validating the content of the model's responses.

2.1.1 Structure

The module is structured as follows: \ ai_language_model\ ai_language_model.py\ models\ gpt_3_5_turbo.py

2.1.2 <tt>AILanguageModel</tt> class

This file contains the abstract base class `AILanguageModel` which outlines the fundamental methods and attributes that a derived AI language model should possess.
The `AILanguageModel` class, defined in the ai_language_model.py file, is an abstract base class that outlines the fundamental methods and attributes that derived AI language models should possess. More details can be found in the `file` itself.

2.1.2.1 Methods:

1. `send_query()`: Sends a query to the AI model, creates the response, and sets and validates the response content.
2. `get_response_content()`: Returns the content of the AI model's response.
3. `get_prompt_content()`: Returns the content of the prompt sent to the AI model.
4. `create_prompt_content()`: Creates the content of the prompt by concatenating the instruction and the input string.
5. `create_response(prompt_content)`: An abstract method for creating a response that should be implemented by a specific AI model class.
6. `set_response_content()`: An abstract method for setting the response content that should be implemented by a specific AI model class.
7. `get_num_tokens_from_response(string, encoding_name)`: An abstract method for getting the number of tokens from a response that should be implemented by a specific AI model class.
8. `validate_response_content()`: An abstract method for validating the response content that should be implemented by a specific AI model class.

2.1.3 models/<tt>GPT3_5Turbo</tt> class

The `GPT3_5Turbo` class, defined in the `gpt_3_5_turbo.py` file within the `models` folder, extends the `AI↵ LanguageModel` class. It is specifically designed to handle interactions with OpenAI's GPT-3.5 Turbo model.

This file contains the class `GPT3_5Turbo` which extends the abstract base class `AILanguageModel` and is specifically designed to handle interactions with the OpenAI's GPT-3.5 Turbo model. This class provides methods to create a response, set and validate the response content, count tokens in a response, and remove certain elements from the response. More details can be found in the `file` itself.

2.1.3.1 Methods:

`create_response(prompt_content)`: Creates a response from the GPT-3.5 Turbo model using the given prompt content. `validate_response_content()`: Validates the content of the model's response and performs necessary modifications. `set_response_content()`: Sets the processed content of the model's response. `get_num_tokens_from_response(string, encoding_name)`: Counts the number of tokens in a given string. `remove_back_quote()`: Removes backquote encapsulated strings from the model's response. `remove_cpp_directives()`: Removes C++ directive lines from the model's response. `remove↩_unwanted_line()`: Removes unwanted lines from the model's response.
This class enables the interaction with the model, the formatting and validation of the responses, and also contains utility functions for token counting and line removal.

2.1.4 How to Use

To use these classes, instantiate the appropriate class (`AILanguageModel` or `GPT3_5Turbo`), pass the required parameters during initialization, and call the desired methods. Note that the `AILanguageModel` class is abstract and is intended to be extended by concrete classes (like `GPT3_5Turbo`) that implement its abstract methods.
To use this code, you will also need to provide your `OpenAI API key` which is used to authenticate requests to the OpenAI API. The API key should be passed when creating an instance of the `GPT3_5Turbo` class.

Chapter 3

README.md

3.1 Configuration Module

The configuration module contains all the necessary configurations for the application, including API keys, environment variables, paths and other configuration settings.

3.1.1 Structure

Here is the structure of the module: config\ Doxyfile\ README.md\ api_key.env\ config_paths.py\ config_↩ variables.py\ db_local_variables.env

3.1.2 Description of Files

3.1.2.1 <a href="./Doxyfile"><tt>Doxyfile</tt></a>

The Doxyfile is a configuration `file` for Doxygen, a tool utilized for generating documentation from commented source code. It defines the settings and preferences for documentation generation. Each time the ai-doxygencleaner pipeline is triggered, this file is automatically generated and configured based on the `dox_↩ config_parameters` variable.

3.1.2.2 api_key.env

The api_key.env file stores the API keys required by the application.

3.1.2.3 <a href="./config_paths.py"><tt>config_paths.py</tt></a>

The config_paths.py file is a Python script that includes configurations related to file paths, specifically defining the locations of data files, logs, and other resources used by the application.

3.1.2.4 <a href="./config_variables.py"><tt>config_variables.py</tt></a>

The `config_variables.py` file is Python script that contains various application-wide settings. These settings may include flags to control application behavior, constants used throughout the application, and other types of configuration variables. The file houses two specific variables:

- `dox_config_parameters`: This variable houses settings that help configure the Doxyfile as needed.
- `prompt_instruction`: This variable contains the prompt instruction sent along with the query to the AI language model.

3.1.2.5 db_local_variables.env

The db_local_variables.env file stores local environment variables related to the database, including the database name, user, password, host, and port.

3.1.3 How to Use

To utilize these configurations, import the appropriate Python files (`config_paths.py` or `config_variables.py`) into your script, and access the configuration variables as necessary. As for the environment variables, they should be loaded into your environment prior to running your script. The Doxyfile is automatically generated and configured each time the pipeline starts. This file's configuration is dependent on the variable defined in the config_variables.py file. This variable can be modified to change the configuration if required.

Chapter 4

README.md

4.1 Data Manipulation Scripts

This directory contains SQL scripts used to manipulate data in the database. These scripts mainly include stored procedures for inserting and updating records in the database tables.

4.1.1 Files

4.1.1.1 <tt>insert_procedures.sql</tt>

- Author: Ismail Al Shuaybi
- Created on: 2023-07-01
- Last updated: 2023-07-02

This script contains stored procedures for inserting records into the tables created to analyze the Doxygen warnings before they are resolved by ChatGPT. These stored procedures simplify the process of adding new data to the database and maintain data consistency.
This script includes four main procedures:

1. **`InsertIntoPipelineTable`:** This procedure inserts a new record into the ci_cd_pipeline_table.
2. **`InsertIntoProjectTable`:** This procedure inserts a new record into the project_table.
3. **`InsertIntoHeaderFileTable`:** This procedure inserts a new record into the header_file_table.
4. **`InsertIntoWarningTable`:** This procedure inserts a new record into the warning_table.

4.1.1.2 <tt>update_procedures.sql</tt>

- Author: Ismail Al Shuaybi
- Created on: 2023-07-01
- Last updated: 2023-07-02

This script is comprised of stored procedures designed for updating records in the database tables used for the analysis of Doxygen warnings. These updates occur after the warnings have been addressed and resolved by ChatGPT. These procedures play a key role in maintaining data accuracy and consistency throughout the analysis. They also provide insights into the number of warnings resolved by ChatGPT and present the content of the associated file after resolution. Therefore, they offer a mechanism to evaluate the effectiveness of the fixes implemented by ChatGPT.
This script includes two main procedures:

1. `**UpdateHeaderFileContentPostFix**`: This procedure updates the field 'Header_file_content↩ _post_fix' in a specific header file associated with a given pipeline ID. It reflects the state of the header file post-resolution.

2. `**UpdateFixedStatus`:`**` This stored procedure is tasked with managing the 'Fixed' status of a specific Doxygen warning associated with a specified pipeline and header file. The procedure is designed to perform the following actions:

   - Checks if a warning exists in the `warning_table`.
   - If the warning doesn't exist, it adds the warning to the table, assigning it a `Fixed` status of 0 (indicating it has not been resolved) and the type `Post-Fix`. This signifies that the warning was identified following a resolution attempt by ChatGPT.
   - If the warning is found, it means the warning wasn't fixed by ChatGPT. Hence, we need to update the `Fixed` status to 0. Any warning not detected post ChatGPT's resolution attempt is considered fixed and retains `Fixed` status 1.
   - The procedure also manages the `Number_of_Warnings` field in the `header_file_table` by incrementing the count when a new warning is added.

4.1.2 Usage

To use these scripts, import them into your preferred SQL client and execute the procedures as needed, providing the necessary input parameters.

Chapter 5

README.md

5.1 Database Operations Module

The Database Operations module contains Python and SQL scripts that handle the insertion, manipulation, and setup of data in a MySQL database. The primary purpose is to manage data related to Doxygen warnings, from capturing details about pipelines, projects, and header files, to inserting and updating warnings in the database.

5.1.1 Structure

Here is the module's structure:

```
db_operations
 README.md
 __init__.py
 data_manipulation
    README.md
    insert_procedures.sql
    update_procedures.sql
 db_insertion_handler.py
 main.sql
 setup
     README.md
     create_table.sql
     drop_tables_procedures.sql
```

5.1.2 Description of Files

5.1.2.1 @ref ./data_manipulation/README.md "data_manipulation"

This directory contains SQL scripts used for manipulating data in the database. These scripts mainly include stored procedures for inserting and updating records in the database tables.

- `insert_procedures.sql`: This script contains stored procedures for inserting records into the various tables for Doxygen warnings analysis.
- `update_procedures.sql`: This script includes stored procedures designed for updating records in the database tables.

See More

5.1.2.2 @ref ./setup/README.md "setup"

This directory contains the scripts necessary for setting up the database.

- `create_table.sql`: This script creates tables to analyze Doxygen warnings.
- `drop_tables_procedures.sql`: This script removes any existing stored procedures or tables to avoid naming conflicts.

See More

5.1.2.3 <a href="./db_insertion_handler.py"><tt>db_insertion_handler.py</tt></a>

The `db_insertion_handler.py` is a Python class designed to manage connections to a MySQL database and provide methods for inserting information related to Doxygen warnings, such as pipeline, project, and warnings details.
This script contains the `DBInsertionHandler` class with the following methods:

- `connect_to_database()`: Connects to the MySQL database using environment variables.
- `create_cursor()`: Creates a new cursor to execute database commands.
- `set_pipeline_data()`: Sets pipeline data by getting information from environment variables.
- `update_last_project_id()`: Updates the `last_project_id` member variable with the most recent project ID.
- `update_last_header_file_id()`: Updates the `last_header_file_id` member variable with the most recent header file ID.
- `execute_sql(sql: str, values: tuple, optional)`: Executes a SQL command.
- `insert_into_pipeline_table()`: Inserts pipeline details into the pipeline table.
- `insert_into_project_table(project_folder_path: str)`: Inserts project details into the project table.
- `insert_into_header_file_table(file_name: str, file_content: str, number↩ _of_warnings: int)`: Inserts header file details into the header file table.
- `insert_into_warning_table_pre_fix(warning_content: str, line_number↩ _pre_fix: int, fixed_status: int = FIXED_STATUS_DEFAULT)`: Inserts warning details into the warning table.
- `insert_warnings_details(project_folder_path: str, file_name_list: list, file_content_list: list, number_of_warnings_list: list, warning_↩ content_llist: list, line_number_pre_fix_llist: list)`: Wrapper function that inserts pipeline, project, header file, and warning details into their respective tables.
- `update_post_fix_warning_details(self, pipeline_id, file_name_list, file↩ _content_list, line_number_llist, warnings_content_llist)`: Update the database with the details of post-fix warnings.

5.1.2.4 <tt>main.sql</tt>

The `main.sql` file is the main SQL script file that integrates and runs the SQL procedures defined in the `data↩ _manipulation` and `setup` folders.

5.1.3 How to Use

To use this module:

1. Ensure that the appropriate environment variables are set in your environment before running any scripts, particularly those related to database access.
2. Run the setup scripts first to set up the required tables in your database.
3. Use the Python class `DBInsertionHandler` in your Python scripts to insert or update data in the database.
4. The data manipulation scripts can be imported into your preferred SQL client and executed as needed.

5.1.4 Contributions

Contributions are welcome. If you find any bugs or issues, or if you want to enhance any functionality, feel free to contribute. Remember to follow the standard Python and SQL style guidelines and avoid committing sensitive information like API keys or database credentials.
Note: Ensure that you have the necessary permissions and clearances before accessing or manipulating any data.

Chapter 6

README.md

6.1 Setup Scripts

This directory contains the scripts necessary for setting up the database. It includes the following SQL scripts:

6.1.1 Files

6.1.1.1 <tt>create_table.sql</tt>

- Author: Ismail Al Shuaybi

- Created on: 2023-07-01

- Last updated: 2023-07-02

This script creates tables to analyze Doxygen warnings both before and after their resolution through ChatGPT. These tables are normalized to the third normal form and include:

1. `ci_cd_pipeline_table`: Stores information about each CI/CD pipeline. This information aids in identifying the pipeline and branch where the Doxygen warning was detected and resolved.
2. `project_table`: Stores information about each project present in the `doxygen_projects` folder. Each project is linked to a CI/CD pipeline. This table is particularly useful when dealing with multiple projects.
3. `header_file_table`: Stores information about each header file within a project. Each header file is associated with a project.
4. `warning_table`: Stores information about each Doxygen warning in a header file. Each warning is linked to a header file. Each warning has a `fix-status` that indicates if the warning has been successfully resolved (1) or not (0). The `warning type` indicates if the warning was detected before (`Pre-Fix`) or after (`Post-Fix`) the fixes applied by ChatGPT.

6.1.1.2 <tt>drop_tables_procedures.sql</tt>

- Author: Ismail Al Shuaybi

- Created on: 2023-07-01

- Last updated: 2023-07-02

This script removes any existing stored procedures or tables to avoid naming conflicts. The script does the following:

1. Drops stored procedures used to insert records into the PipelineTable, ProjectTable, HeaderFileTable, and WarningTable, if they exist.
2. Drops stored procedures used to update the `HeaderFileContentPostFix` field and the `Fixed←↩ Status` field, if they exist.
3. Drops tables used to store data related to CI/CD pipelines, projects, header files, warnings if they exist.

Chapter 7

README.md

7.1 doxy_warning_post_processor

This folder contains code for post-processing Doxygen warnings. It includes the following files:

- `README.md`: This file provides an overview of the project's structure, files, and functionalities.
- `doxy_warning_post_processor.py`: The main Python script for post-processing Doxygen warnings.

7.1.1 <tt>doxy_warning_post_processor.py</tt>

This Python script includes a class named `WarningPostProcessor` that is used to detect and process warnings after fixes have been applied to the codebase. Here's a brief overview of this class:

7.1.1.1 Class: <tt>WarningPostProcessor</tt>

The `WarningPostProcessor` class doesn't have any class-level attributes. It consists of the following methods:

7.1.1.1.1 1. <tt>apply_fix()</tt>

This method applies fixes to the header files and saves changes.
It works by:

- Loading pre-warning details from a JSON file (`preprocessed_warnings_data.json`).
- Loading post-fix file content from a JSON file (`postprocessed_warnings_data.json`).
- Writing the post-fix content back to the header files.

7.1.1.1.2 2. <tt>detect_warning()</tt>

This method detects and updates warning details post-fix.
It performs the following steps:

- Loading pre-warning details from a JSON file (`preprocessed_warnings_data.json`).
- Running the `DoxyWarningDetector` to detect warnings after fixes have been applied.
- Updating the warning lists with the post-fix warnings.
- Adding updated warning details to a JSON file (`postprocessed_warnings_data.json`).

This method doesn't take any parameters and doesn't return any value.

7.1.1.2 Key Functions Used In This Script

- `load_pre_fix_data_from_json()`: Loads pre-fix warning details from a JSON file.
- `load_data_from_json()`: Loads post-fix file content from a JSON file.
- `FileHandler().write_file()`: Writes post-fix content back to the header files.
- `DoxyWarningDetector().run_doxygen()`: Detects warnings after fixes have been applied.
- `add_data_to_json()`: Adds updated warning details to a JSON file.
- `print_project_details()`: Prints out project details post-fix.

For additional details, including the individual method's specific parameters and returns, refer to the inline comments and method docstrings within `doxy_warning_post_processor.py`.
Note: Remember to set up any necessary configuration and dependencies before running the script.

Chapter 8

README.md

This README provides a brief introduction to the various Python files present in the doxygen_management folder. These files contain important classes and methods that perform various functions like detecting warnings in Doxygen documentation generation, configuring Doxyfiles, and handling file operations.

8.1 doxygen_management

This folder contains scripts for handling and managing Doxygen documentation generation. It consists of the following files:

- `README.md`: This file provides an overview of the folder's structure and its functionalities.
- `__init__.py`: A Python initialization script.
- `doxy_warning_detector.py`: The main Python script to detect warnings during Doxygen documentation generation.
- `doxyfile_configurator.py`: The Python script to configure the Doxyfile used for Doxygen documentation generation.
- `file_handler.py`: The Python script for handling file operations.

8.1.1 doxy_warning_detector.py

This Python file contains the `DoxyWarningDetector` class, which serves the purpose of detecting warnings in Doxygen documentation generation.

8.1.1.1 Class: <tt>DoxyWarningDetector</tt>

The `DoxyWarningDetector` class has the following attributes and Key Methods:

- Attributes:
  - `warnings_list` : A list to hold the detected warnings (initialized as an empty list).
  - `doxyfile_configurator` : An object of the class `DoxyfileConfigurator` to configure the Doxyfile.
  - `file_handler` : An object of the class `FileHandler` to handle file operations.
- Key Methods:
  - `check_project_directory()` : Checks if there is only one subfolder in the project directory and returns its name. In case of multiple subfolders, the program exits with an error message.
  - `get_all_header_files(project_folder_path: str)` : Retrieves all the header files in the specified project folder path. If no header files are found, the program exits with an error message.

  - `run_doxygen_for_file(header_file_name: str, lines: list, input_line: str, project_folder_path: str, line_num: int)`: Prepares the Doxyfile with given parameters and runs Doxygen for the specified header file. The result of the Doxygen run is returned.
  - `extract_warnings(result: subprocess.CompletedProcess, header_file_name: str)`: Extracts warnings from the result of a Doxygen run. Returns lists of warning contents, warning line numbers and total number of warnings.
  - `run_doxygen()`: Executes the complete process to run Doxygen for all header files and extract warnings. The process includes preparing the Doxyfile, running Doxygen, and extracting warnings. A tuple containing the results of the process is returned.

8.1.2 How to Use

To use these classes, you should import the necessary Python files and create objects of the classes, as shown below:

```
from doxygen_management.doxy_warning_detector import DoxyWarningDetector
# Create an object of DoxyWarningDetector
detector = DoxyWarningDetector()
# Use the methods of the object
detector.check_project_directory()
detector.get_all_header_files(project_folder_path)
```

You can replace `project_folder_path` with the path of your project directory.
For any queries or further explanation regarding the classes and their methods, please refer to the respective Python files as they contain detailed docstrings for each class and method.

8.2 file_handler.py

The `FileHandler` class is responsible for managing file operations. It provides various functions to read, write, and find specific lines of text in files. Moreover, it offers the capability to redirect the output of certain functions into a file.

8.2.0.1 Class: <tt>FileHandler</tt>

The `FileHandler` class consists of the following methods:

8.2.1 Methods

- `read_file(filename: str) -> list`: This method reads a file and returns its contents as a list of lines.
- `write_file(filename: str, lines: list)`: This method writes a list of lines into a file.
- `has_single_line(list: list) -> bool`: This method checks if a list has only a single line. If yes, it returns True, else the program exits with an error message.
- `find_line_numbers(filename: str, searchText: str) -> int`: This method finds the line numbers of all occurrences of a specific text in a file. It returns the line number of the first occurrence of the search text if there is only one occurrence, else the program exits with an error message.
- `redirect_print_to_file(self, project_folder_path, header_file_name_list, warning_num_list, warning_content_llist, warning_line_number_llist, file_content_list, pre_or_post, output_file)`: `This method redirects the output of the` print_project_details_without_color` function to a specified file. It modifies the standard output (stdout) to a specified file, calls a function to print data without color, and then restores stdout to its original state.

8.2.2 Example

```
from doxygen_management.file_handler import FileHandler
file_handler = FileHandler()
# Read a file
lines = file_handler.read_file('example.txt')
# Write a list of lines into a file
```

```
file_handler.write_file('output.txt', lines)
# Check if a list has only a single line
single_line = file_handler.has_single_line(lines)
# Find the line numbers of a certain text in a file
line_number = file_handler.find_line_numbers('example.txt', 'searchText')
# Redirect the output of a function into a file
file_handler.redirect_print_to_file(project_folder_path, header_file_name_list, warning_num_list,
       warning_content_llist
```

8.3 doxyfile_configurator.py

The `DoxyfileConfigurator` class is used to configure a Doxyfile. It includes methods to create and modify a Doxyfile, add spaces to configuration parameters, and check for single subfolders in a directory. The class also includes an instance of the `FileHandler` class, used to manipulate files.

8.3.0.1 Class: <tt>DoxyfileConfigurator</tt>

The `DoxyfileConfigurator` class has the following attributes and Methods:

8.3.1 Attributes

- `file_handler : FileHandler`: An instance of the `FileHandler` class used to manipulate files.

8.3.2 Methods

- `__init__(self)`: This method initializes an instance of the `DoxyfileConfigurator` class.
- `remove_doxyfile_if_exists(self, directory_path)`: This method removes a Doxyfile at the given directory path.
- `create_doxyfile(self, directory_path)`: This method creates a Doxyfile at the given directory path.
- `add_spaces_to_string(self, config_parameters)`: This method adds specific spaces to configuration parameters.
- `get_single_subfolder(self, directory)`: This method checks if there is only one subfolder in a given directory.
- `get_header_files(self, directory)`: This method returns a list of .h and .hpp files in the given directory.
- `configure_doxyfile(self, filename, config_parameters)`: This method configures the Doxyfile with the given parameters.

8.3.3 Example

```
doxyfile_configurator = DoxyfileConfigurator()
# Remove a Doxyfile at a directory path
doxyfile_configurator.remove_doxyfile_if_exists('/path/to/directory')
# Create a Doxyfile at a directory path
doxyfile_configurator.create_doxyfile('/path/to/directory')
# Add spaces to configuration parameters
formatted_parameters = doxyfile_configurator.add_spaces_to_string([('EXAMPLE_PARAM', 3)])
# Check if there is only one subfolder in a directory
single_subfolder, subfolder_name = doxyfile_configurator.get_single_subfolder('/path/to/directory')
# Get .h and .hpp files in a directory
header_files_present, header_files = doxyfile_configurator.get_header_files('/path/to/directory')
# Configure the Doxyfile with given parameters
doxyfile_configurator.configure_doxyfile('Doxyfile', formatted_parameters)
```

Note: You will need to replace `/path/to/directory` and `Doxyfile` with your actual directory paths and filenames. The second value in tuples passed to `add_spaces_to_string` should represent the number of spaces you want to add to the corresponding parameter.\ **Note:** Remember to set up any necessary configuration and dependencies before running the scripts.

Chapter 9

DoxygenWarningFixer

The `DoxygenWarningFixer` folder is part of the larger `ai_doxygen_cleaner` project. It contains a script `doxygen_warning_fixer.py` that hosts the `DoxygenWarningFixer` class, which is designed to detect and fix Doxygen warnings in your code documentation.
This folder structure is as follows:

```
DoxygenWarningFixer
 README.md
 __init__.py
 doxygen_warning_fixer.py
```

The `doxygen_warning_fixer.py` file contains the logic for the Doxygen warning fixer.

9.1 DoxygenWarningFixer Class

The `DoxygenWarningFixer` class is the main class responsible for detecting and fixing Doxygen warnings in the code documentation. It uses the GPT-3.5-turbo model from OpenAI for fixing the warnings. The class has the following methods:

9.1.1 <tt>__init__()</tt>

This method is used to initialize the `DoxygenWarningFixer` object. It defines the GPT model instance (`self.gpt`) and initializes a list to hold colored prompt strings for output (`self.colored_prompt_input↩ _str_list`).

9.1.2 <tt>wait()</tt>

This method is used to print the elapsed waiting time every second as long as the thread is running. This is useful when you are waiting for a response from the OpenAI server.

9.1.3 <tt>print_query(pos)</tt>

This method prints the query that is sent to the OpenAI server and its corresponding response. It takes one argument `pos` which represents the position of the query in the sequence.

9.1.4 <tt>_concat_warnings_details(...)</tt>

This method concatenates warning details to generate the prompt_input_str_list. It accepts the following parameters:

- `header_file_name_pre_fix_list`: List of header file names before the fix.
- `file_content_pre_fix_list`: List of file contents before the fix.
- `warning_content_pre_fix_llist`: List of warning contents before the fix.
- `warning_line_number_pre_fix_llist`: List of warning line numbers before the fix.

It returns a list of warning details to be fixed.

9.1.5 <tt>_fix_warnings(prompt_input_str_list, prompt_instruction)</tt>

This method uses the GPT model to fix the warnings and return the post-fix file content list. It takes two arguments:

- `prompt_input_str_list`: List of concatenated warning details.
- `prompt_instruction`: Instruction for the GPT model.

It returns a list of fixed file content.

9.1.6 <tt>run()</tt>

This is the main method that orchestrates the warning fixing process. It loads the warning details from `preprocessed_warnings_data.json`, sends them to the GPT model, gets the fixed content, and saves the fixed content to `postprocessed_warnings_data.json`.
Remember, before you run the warning fixer, you need to set your OpenAI API key in the `config/api_key.env` file.

9.2 Usage

To use the `DoxygenWarningFixer`, you simply need to create an instance of the class and call the `run()` method. Before you do this, make sure your OpenAI API key is set and the warning details are properly loaded in `preprocessed_warnings_data.json`. The warning details include the file names, file contents, warning contents, and warning line numbers.

```
doxygen_fixer = DoxygenWarningFixer()
doxygen_fixer.run()
```

After running the warning fixer, you can find the fixed content in `postprocessed_warnings_data.json`.

Chapter 10

ErrorHandler Module

The error_handler module is part of the ai_doxygen_cleaner package. It contains the Error↩ Handler class which provides methods for handling common errors that may occur during program execution. It's a key tool for error handling and debugging in the system.

10.1 Structure

The error_handler folder contains the following files:

- __init__.py: This is an initialization file for the module.
- error_handler.py: This is the main script of the module that contains the ErrorHandler class.

10.2 ErrorHandler Class

The ErrorHandler class provides static methods to handle common types of errors. Specifically, it deals with subprocess errors, file not found errors, and SQL execution errors.

10.2.1 Static Methods

The class offers the following static methods:

10.2.1.1 handle_subprocess_error(e, custom_message)

This method handles subprocess errors by displaying a custom error message and terminating the program.
Parameters:

- e (Exception): The exception raised
- custom_message (str): The custom message to be displayed before the exception message

Example usage:
```
try:
    # some code that may raise a subprocess error
except Exception as e:
    ErrorHandler.handle_subprocess_error(e, "A subprocess error occurred.")
```

10.2.1.2 handle_file_not_found_error(e, custom_message)

This method handles file not found errors by displaying a custom error message and terminating the program.
Parameters:

- e (Exception): The exception raised
- custom_message (str): The custom message to be displayed before the exception message

Example usage:
```
try:
    # some code that may raise a FileNotFoundError
except Exception as e:
    ErrorHandler.handle_file_not_found_error(e, "A file not found error occurred.")
```

10.2.1.3 execute_sql(cursor, sql, values=None)

This method executes a SQL command. If an error occurs during the execution, it calls `handle_subprocess↩ _error` to handle it.
Parameters:

- `cursor` (Cursor): The cursor to execute the SQL command
- `sql` (str): The SQL command to be executed
- `values` (tuple, optional): The values to be inserted into the SQL command. Defaults to None.

Example usage:
```
sql = "SELECT * FROM some_table"
try:
    ErrorHandler.execute_sql(cursor, sql)
except Exception as e:
    ErrorHandler.handle_subprocess_error(e, "An error occurred while executing SQL command.")
```
Note: Error messages are printed in different colors (red for custom error messages and yellow for exception messages) to enhance readability and assist debugging.

10.3 Conclusion

The `error_handler` module is a helpful tool for managing and debugging common errors within the `ai↩ _doxygen_cleaner` package. By providing informative error messages and terminating the program when necessary, it helps to prevent silent failures and facilitates troubleshooting.

Chapter 11

README.md

11.1 print_data

The `print_data` folder is a module part of the `ai_doxygen_cleaner` project. It contains the `print_↩ data.py` script which is used to print details about projects, including file contents, Doxygen warnings, and lines where the warnings occurred.

11.1.1 Folder Structure

```
print_data
 README.md
 __init__.py
 print_data.py
```

11.1.2 Function Overview

`print_data.py` contains two main functions:

1. `print_project_details()`
2. `print_project_details_without_color()`

11.1.2.1 print_project_details()

This function is responsible for printing a colorful, detailed summary of a project, including file contents, Doxygen warnings, and lines where the warnings occurred. It requires parameters like project folder path, header file names, warning numbers, warning contents, and line numbers. Optionally, it can take file contents as well.

11.1.2.2 print_project_details_without_color()

Similar to `print_project_details()`, but this function prints the details without color codes.

11.1.3 Example Usage

Here's how you can use these functions in your Python script:

```
from print_data import print_project_details, print_project_details_without_color
# assuming these lists are populated with relevant data
project_folder_post_fix_path = "/path/to/project"
header_file_name_post_fix_list = ["header1.h", "header2.h"]
warning_num_post_fix_list = [5, 0]
warning_content_post_fix_llist = [["warning 1", "warning 2"], None]
warning_line_number_post_fix_llist = [[1, 2], None]
file_content_post_fix_list = ["file content 1", "file content 2"]
# Using print_project_details()
print_project_details(
    project_folder_post_fix_path,
    header_file_name_post_fix_list,
    warning_num_post_fix_list,
    warning_content_post_fix_llist,
    warning_line_number_post_fix_llist,
    "post",
    file_content_post_fix_list
)
```

```
# Using print_project_details_without_color()
print_project_details_without_color(
    project_folder_post_fix_path,
    header_file_name_post_fix_list,
    warning_num_post_fix_list,
    warning_content_post_fix_llist,
    warning_line_number_post_fix_llist,
    "post",
    file_content_post_fix_list
)
```

In the above examples, we are printing the details of a hypothetical project. The results will include details like file contents, Doxygen warnings, and lines where the warnings occurred, both with and without color coding.

Chapter 12

README.md

12.1 Overview

`main.py` serves as the main execution script for the AIDoxygenCleaner tool. This AI-based tool automates several tasks related to the management of Doxygen warnings. The following operations are automated:

- Configuring Doxygen
- Detecting warnings
- Inserting warnings into a database
- Fixing warnings using an AI language model
- Applying these fixes
- Re-detecting warnings after fixes have been applied
- Printing warning details to a file
- Generating a diff page in HTML format
- Updating warning data in the database post-fix

12.2 Structure

The file is structured into the importing of necessary modules, creating an `ArgumentParser` instance for handling command-line arguments, defining optional arguments for the script, and creating an instance of `AI↩ DoxygenCleaner`. Following this, the script processes each argument and calls the appropriate `AIDoxygen↩ Cleaner` method.

12.3 Command Line Arguments

The script provides several optional command line arguments that control the execution of the different functionalities of the tool:

- `-c`, `--configure-doxygen`: Configure Doxygen using predefined configuration parameters.
- `-dw`, `--detect-warnings-pre-fix`: Detect Doxygen warnings in the specified project.
- `-iwdb`, `--insert-warnings-db`: Insert detected Doxygen warnings into the database.
- `-fdw`, `--fix-doxygen-warnings`: Attempt to fix detected Doxygen warnings using an AI language model.
- `-af`, `--apply-fix`: Apply suggested fixes from the AI language model to the original header files.

- `-dwpf`, `--detect-warnings-post-fix`: Re-run Doxygen to detect warnings after the fixes have been applied.
- `-pwdtf`, `--print-warning-details-to-file`: Print details of pre and post-fix warnings to a file.
- `-gdhp`, `--generate-html-diff-page`: Generate an HTML diff page to visualize changes made by the fix application.
- `-uwd`, `--update-warning-data-in-db`: Update the warning data in the database after applying fixes.

12.4 Example Usage

The `main.py` script can be executed from the terminal using Python. The optional command line arguments can be provided to control the execution of the tool. Here are a few examples:

- To run the script with the `configure-doxygen` and `detect-warnings-pre-fix` options:

```
python main.py -c -dw
```

- To run the script with all options:

```
python main.py -c -dw -iwdb -fdw -af -dwpf -pwdtf -gdhp -uwd
```

Please ensure to replace `python` with the correct Python command based on your Python installation and system configuration.

Chapter 13

README.md

13.1 Directory Structure

```
transport_data
 README.md
 __init__.py
 pipeline_id.json
 postprocessed_warnings_data.json
 preprocessed_warnings_data.json
 transport_data.py
```

13.2 Overview

The `transport_data` directory is part of the `ai_doxygen_cleaner` project. The primary role of this module is to handle the saving, loading, and updating of data in JSON format. The data is used to manage preprocessed and postprocessed warning data from Doxygen.
The `transport_data.py` script contains five primary functions:

- `save_data_to_json()`
- `load_data_from_json()`
- `add_data_to_json()`
- `load_pre_fix_data_from_json()`
- `load_post_fix_data_from_json()`

13.3 <tt>save_data_to_json(file_path: str, data_dict: dict) -> None</tt>

This function takes a file path and a data dictionary as inputs and saves the dictionary in a JSON file at the specified file path.
Example usage:
`save_data_to_json("path/to/file.json", {"key": "value"})`

13.4 <tt>load_data_from_json(file_path: str) -> dict</tt>

This function loads data from a JSON file and returns a dictionary. The input is the path to the JSON file.
Example usage:
`data = load_data_from_json("path/to/file.json")`

13.5 <tt>add_data_to_json(file_path: str, new_data: dict) -> None</tt>

This function adds new data to an existing JSON file. It takes a file path and a dictionary of new data as inputs.
Example usage:
`add_data_to_json("path/to/file.json", {"new_key": "new_value"})`

13.6 <tt>load_pre_fix_data_from_json() -> tuple</tt>

This function loads preprocessed warning data from the `preprocessed_warnings_data.json` file and returns a tuple containing the project folder path, header file names, file contents, warning numbers, warning contents, and line numbers of the warnings.
Example usage:
```
data = load_pre_fix_data_from_json()
```

13.7 <tt>load_post_fix_data_from_json() -> tuple</tt>

This function loads postprocessed warning data from the `postprocessed_warnings_data.json` file and returns a tuple containing the project folder path, header file names, file contents, warning numbers, warning contents, and line numbers of the warnings.
Example usage:
```
data = load_post_fix_data_from_json()
```

13.8 JSON Files

The `transport_data` directory includes three JSON files:

- `pipeline_id.json`: Stores the pipeline ID data.
- `postprocessed_warnings_data.json`: Stores the postprocessed warning data.
- `preprocessed_warnings_data.json`: Stores the preprocessed warning data.

These files are used by the `transport_data.py` script to manage data across different stages of the Doxygen cleanup process.

13.9 Summary

The `transport_data` directory is an essential part of the `ai_doxygen_cleaner` project, responsible for data storage and retrieval before and after processing Doxygen warnings.

Chapter 14

README.md

14.1 <tt>utils</tt>

14.1.1 Directory Structure

```
utils
 README.md
 __init__.py
 utils.py
```

14.1.2 Overview

The `utils` directory is part of the `ai_doxygen_cleaner` project. The `utils.py` script within this directory contains utility functions to handle miscellaneous tasks, such as compiling warning details, performing file differences, and generating HTML from a diff file.
The `utils.py` script contains three primary functions:

- `compaine_warn_details()`
- `perform_diff_on_files()`
- `generate_html_diff()`

14.1.3 <tt>compaine_warn_details(warning_content_llist, warning_line_number_llist, file_content_list, header_file_name_list)</tt>

This function takes in four lists: warning content, warning line numbers, file content, and header file names. It compiles these details into a list of prompts that contain warnings and their respective file contents.
Example usage:

```
prompts = compaine_warn_details(warning_content_llist,
                                warning_line_number_llist, file_content_list,
                                header_file_name_list)
```

14.1.4 <tt>perform_diff_on_files() -> None</tt>

This function uses the `diff` command to compare two files and outputs the difference to a text file. It specifically compares `output_pre_fix.cpp` and `output_post_fix.cpp` in the `data/different_between↩ _pre_and_post` directory.
Example usage:

```
perform_diff_on_files()
```

14.1.5 <tt>generate_html_diff() -> None</tt>

This function generates an HTML file from a diff text file using the `diff2html` command. It takes the diff file generated by `perform_diff_on_files()` and creates an HTML representation.
Example usage:

```
generate_html_diff()
```

14.1.6 Summary

The `utils` directory contains utility functions that perform various tasks within the `ai_doxygen_cleaner` project. These tasks include compiling warning details into a list of prompts, comparing file differences, and generating HTML files from diff files. These utilities aid in the project's aim of cleaning and managing Doxygen warnings.

Chapter 15

AIDoxygenCleaner:

15.1 Description

AIDoxygenCleaner is a robust, AI-powered tool that simplifies and streamlines the management of Doxygen-generated documentation. This innovative project employs a comprehensive, step-by-step process to configure, detect, fix, and apply changes to Doxygen warnings, while meticulously documenting these changes for future review. The power of machine learning is harnessed to analyze, interpret, and resolve Doxygen warnings, providing developers with insightful and actionable solutions.
The tool is structured around one main Python file, `ai_doxygen_cleaner.py`, which carries out the primary operations. These operations encompass a wide range of functionalities from configuring Doxygen, detecting and inserting warnings into a database, to applying fixes using an AI language model. Following the application of fixes, the tool detects remaining warnings, prints detailed warning information to a file, generates a diff page in HTML format, and updates warning data in the database post-fix.
This comprehensive approach significantly improves the overall documentation quality and maintainability of software projects, reducing the need for manual intervention. This README file serves as a guide, detailing the structure of the project and the functionality of the classes and functions contained within, aiming to provide a clear understanding of the purpose and use of each component.

15.2 Directory Structure

Here's the project structure for your understanding:

```
ai-doxygencleaner
 Dockerfile
 README.md
 documentation
 logs
 main
    DoxygenWarningFixer
       README.md
       doxygen_warning_fixer.py
    README.md
    __init__.py
    ai_doxygen_cleaner
       README.md
       ai_doxygen_cleaner.py
    ai_language_model
       README.md
       ai_language_model.py
       models
           gpt_3_5_turbo.py
    config
       Doxyfile
       README.md
       api_key.env
       config_paths.py
       config_variables.py
       db_local_variables.env
    data
       different_between_pre_and_post
           diff.html
           diff.txt
           output_post_fix.cpp
           output_pre_fix.cpp
    db_operations
       README.md
       data_manipulation
          README.md
          insert_procedures.sql
          update_procedures.sql
       db_insertion_handler.py
       main.sql
       setup
           README.md
           create_table.sql
           drop_tables_procedures.sql
    doxy_warning_post_processor
       README.md
       doxy_warning_post_processor.py
    doxygen_management
       README.md
       doxy_warning_detector.py
       doxyfile_configurator.py
       file_handler.py
    error_handler
       README.md
       error_handler.py
    main.py
    print_data
       README.md
       print_data.py
    transport_data
       README.md
       pipeline_id.json
       postprocessed_warnings_data.json
       preprocessed_warnings_data.json
       transport_data.py
    utils
        README.md
        utils.py
 update_docker_image.sh
19 directories, 49 files
```

15.3 ai_doxygen_cleaner.py Overview

This file is the heart of the AIDoxygenCleaner. It contains the AIDoxygenCleaner class that performs all the operations related to the management of Doxygen-generated documentation.
The class consists of the following key methods:

1. **configure_doxygen_files:** This method starts the process by configuring the Doxygen files. It creates and configures a Doxyfile using a DoxyfileConfigurator object.

2. **detect_warnings_pre_fix:** This method is responsible for detecting Doxygen warnings before any fixes are applied. It uses a DoxyWarningDetector object to carry out the detection process. The warnings and associated details are saved to a json file for later use.

3. **insert_warnings_to_db:** This method inserts the details of the detected warnings into a database. The data for this process is loaded from the previously created json file.

4. **fix_doxygen_warning:** This method applies fixes to the Doxygen warnings using a DoxygenWarningFixer object.

5. **apply_fix:** This method applies the suggested fixes to the header files in the doxygen_projects folder.

6. **detect_warnings_post_fix:** This method detects Doxygen warnings after the fixes have been applied using a WarningPostProcessor object.

7. **print_pre_post_fix_data_to_file:** This method writes the warning details before and after the fix to a file, to give a clear comparison of changes made.

8. **generate_html_diff_page:** This method generates an HTML page displaying the differences between the Doxygen warning details before and after the fixes have been applied.

9. **update_warnings_details_post_fix_in_db:** This method updates the warning details in the database after the fixes have been applied.

15.3.1 Note

Each of the methods follows a similar pattern of operations: initializing a specific object related to the task at hand, executing the task, and then printing a confirmation message to the console indicating the successful completion of the task.

15.4 Getting Started

To utilize this tool, import the `AIDoxygenCleaner` class from the `ai_doxygen_cleaner.py` file. You can then create an instance of the class and call the desired methods on this instance.

```
from ai_doxygen_cleaner import AIDoxygenCleaner
# Create instance
cleaner = AIDoxygenCleaner()
# Call desired methods
cleaner.configure_doxygen_files()
cleaner.detect_warnings_pre_fix()
# ... and so on ...
```

Each method is meant to be called in sequence, as each one builds upon the actions of the previous method.
Please refer to the method documentation in the `ai_doxygen_cleaner.py` file for more detailed information about each function, its parameters, and what it does.
For a deeper understanding of the project and how to use it, please refer to the full documentation.

15.4.1 Key Directories and Files:

- **main**: This directory is the heart of the project where all the core functionality lies. It contains several subdirectories, each dedicated to a specific aspect of the Doxygen warning management.
  - **DoxygenWarningFixer**: This module is responsible for fixing the Doxygen warnings.
  - **ai_doxygen_cleaner**: This module handles the overall management of Doxygen warnings.
  - **ai_language_model**: This is where the AI model for language processing resides.
  - **config**: Contains configuration files for the application.
  - **data**: Stores data related to pre and post warning fix.
  - **db_operations**: Contains files related to database operations.
  - **doxygen_management**: This module is dedicated to managing Doxygen configurations and warnings.
  - **error_handler**: Handles errors throughout the application.
  - **print_data**: This module helps in printing warning details to a file.
  - **transport_data**: Used for storing and transporting data across the pipeline.
  - **utils**: Contains utility functions used across the application.
- **main.py**: The main script for executing the Doxygen Warning Manager tool.
- **Dockerfile**: Contains Docker instructions to build the application's image.
- **update_docker_image.sh**: A shell script to update the Docker image of the application.
- **docs**: Contains all the documentation related files.
- **logs**: Logs produced by the application will be stored here.

15.5 How to Run

To run the main.py script, navigate to the 'main' directory and execute the following command in the terminal:

```
python3 main.py --configure-doxygen --detect-warnings-pre-fix --insert-warnings-db \
--fix-doxygen-warnings --apply-fix --detect-warnings-post-fix \
--print-warning-details-to-file --generate-html-diff-page --update-warning-data-in-db
```

Each option corresponds to a specific operation:

- `-c, --configure-doxygen`: Configure Doxygen using predefined configuration parameters.
- `-dw, --detect-warnings-pre-fix`: Detect Doxygen warnings in the specified project.

- `-iwdb, --insert-warnings-db`: Insert detected Doxygen warnings into the database.
- `-fdw, --fix-doxygen-warnings`: Attempt to fix detected Doxygen warnings using an AI language model.
- `-af, --apply-fix`: Apply suggested fixes from the AI language model to the original header files.
- `-dwpf, --detect-warnings-post-fix`: Re-run Doxygen to detect warnings after the fixes have been applied.
- `-pwdtf, --print-warning-details-to-file`: Print details of pre and post-fix warnings to a file.
- `-gdhp, --generate-html-diff-page`: Generate an HTML diff page to visualize changes made by the fix application.
- `-uwd, --update-warning-data-in-db`: Update the warning data in the database after applying fixes.

Please select the options that best meet your needs.
Note: The options `-iwdb, --insert-warnings-db` and `-uwd, --update-warning-data-in-db` aren't essential to run the AI Doxygen Cleaner tool. They exist solely for storing the resulting data in a database and managing it for the purposes of academic research. Consequently, installing MySQL isn't a requirement and isn't necessary for the operation of the tool. However, it's important to observe the specified order of the arguments to ensure the correct functioning of the tool.

15.6 Documentation

The project's documentation employs the `Numpydoc` style, a flexible and detailed documentation standard that enjoys broad use within the Python community. Every directory contains a `README` file that elucidates the functions, classes, and overall purpose of its contents. This project's comprehensive documentation is designed to guide you through each facet of the application, ensuring you comprehend the role and usage of each individual function and class.
In addition to this, the `documentation` folder contains both LaTeX and HTML versions of the `ai_doxygencleaner` Python code's documentation.

- **View the most recent LaTeX documentation**: `** click here**`
- **View the most recent HTML documentation**: `** click here**`

These resources, created from Numpydoc strings, can further assist your understanding of the code. This documentation was generated using the `Sphinx tool` and contains no warnings, thereby ensuring its accuracy and reliability.

15.7 Dockerization and Image Updating

AIDoxygenCleaner is dockerized and can be run within a Docker container. A `Dockerfile` is provided in the project directory for creating a Docker image. This Docker image is based on the latest version of Alpine Linux and includes all necessary dependencies such as Python3, Doxygen, pip, and other dependencies. It also installs Python and Node.js packages that are necessary for running AIDoxygenCleaner.
You can build and push the Docker image to your registry as follows:

1. Ensure Docker is installed and properly configured on your system.
2. Navigate to the project directory containing the Dockerfile.
3. Run the `update_docker_image.sh` script to build and update the Docker image:
   ```bash ./update_docker_image.sh ```

The script performs the following steps:

- It builds a new Docker image with the tag `doxycleaner`.

- It tags the new image with the tag `registry.gitlab.com/ismsh/ai-doxygencleaner/doxycleaner↩ :latest`.
- It logs in to your Docker registry (you will be prompted to enter your login credentials).
- It pushes the new Docker image to your Docker registry.

Make sure you have the appropriate permissions to push images to the specified Docker registry. You can customize the script to suit your specific requirements and environment by modifying the Docker registry path and other parameters.

15.8 Pipeline Configuration

The pipeline configuration can be found in the `.gitlab-ci.yml` file. This file includes instructions for different pipeline stages such as `build_fix`, `apply_fix`, `test_fix` and `pages`. All these stages help automate the Doxygen warning management process in CI/CD pipeline.

15.8.0.1 The tool requires:

- **Python 3.8** or later version to be installed in your system.
- **Doxygen** should be installed and configured appropriately in your
- system to run the Doxygen warning detection part of the pipeline. \

15.8.0.2 Optional:

- **GitLab** or any other CI/CD platform where you can run your CI/CD pipelines.
- **Docker** if you wish to run the application as a Docker container.
- **MySQL** Database Server to store Doxygen warning related information.

15.9 Note #TODO

Please make sure to install the necessary dependencies listed in the `requirements.txt` file.

15.9.1 Platform Compatibility

The AI Doxygen Cleaner tool is built to be platform-independent. Its compatibility extends to any operating system capable of supporting Python, Doxygen, and Node.js (required for installing diff2html). This makes it highly versatile, covering Windows, Linux, and macOS environments.
If your project is hosted on GitLab and follows a CI/CD pipeline, the AI Doxygen Cleaner tool offers additional convenience. To use it in this context, ensure Docker is installed in your environment. You can then run the `update_docker_image.sh` script, which updates the Docker image. Once these steps are done, every time you commit changes to your GitLab repository, the tool's operations will be automatically triggered via the provided `.gitlab-ci.yml` file.
Remember that appropriate permissions and settings should be in place on GitLab to allow for this automatic execution. The `.gitlab-ci.yml` and Docker settings can be customized to fit your project's specific requirements.

15.9.2 Installation of Required Libraries #TODO

To install the required libraries, you can use pip, the Python package manager. Use the following command:
```
pip install -r requirements.txt
```
This command installs all the dependencies listed in the `requirements.txt` file.
Ensure that you are using a Python version that is 3.8 or later.

15.9.3 Troubleshooting

If you run into any issues during the setup or operation of the tool, I recommend consulting the `ISSUES` tab in the project repository. Here, you can explore solutions to previous issues and pose new questions. If the issue persists, don't hesitate to open a new issue detailing the problem you're encountering. I will strive to resolve it as promptly as possible.
To facilitate easier navigation, issues are categorized with labels, simplifying the process of finding solutions for specific problems. This troubleshooting section is continuously updated to encompass solutions to all potential issues. If you're unable to find a solution to your problem, please feel free to reach out directly. I'm here to assist.

15.9.4 Contacts

Should you require additional assistance, or if you wish to share feedback, propose new ideas, or discuss potential collaborations, don't hesitate to contact me via the `ISSUES` tab in the project repository. Your contributions can significantly impact the continuous improvement of this project, and they are always appreciated.

Chapter 16

Namespace Index

16.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 17

Hierarchical Index

17.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 18

Class Index

18.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 19

Namespace Documentation

19.1 config_variables Namespace Reference

Variables

- list **dox_config_parameters**
- string **prompt_instruction**

19.1.1 Detailed Description

```
This module defines the configuration parameters for Doxygen.

The `dox_config_parameters` variable is a list of tuples. Each tuple contains two elements:
    1. The Doxygen configuration parameter as a string, followed by '{}' and '='.
    2. The number of spaces to insert between the parameter and the '=' sign.

In the actual Doxygen configuration file, each parameter is followed by either 'YES' or 'NO'.
```

19.1.2 Variable Documentation

19.1.2.1 dox_config_parameters

`list config_variables.dox_config_parameters`

Initial value:

```
=  [
     ["GENERATE_LATEX{}=", 9],
     ["GENERATE_HTML{}=", 10],
     ["WARNINGS{}=", 15],
     ["WARN_IF_UNDOCUMENTED{}=", 3],
     ["WARN_IF_DOC_ERROR{}=", 6],
     ["WARN_NO_PARAMDOC{}=", 7]
 ]
```

19.1.2.2 prompt_instruction

`string config_variables.prompt_instruction`

Initial value:

```
=  '''
1- Fix the following Doxygen-warnings.
2- Ensure that the revised comments are intelligible, detailed, comprehensive, meaningful and adhere to
   the Doxygen format.
3- Do not change the code, just add comments that resolve the Doxygen warnings, and then send me the
   complete code with the generated comments.
4- Do not add any new code to this code.
5- You are not allowed to add any code such as #ifndef or anything similar.
6- You are not allowed to change the code.
7- You are not allowed to add additional text like note other.
8- Provide the code without adding back-quotes at the beginning and end, so that I can copy it directly.
'''
```

19.2 main.main Namespace Reference

Variables

- **parser**
- **required**
- **False**
- **action**
- **help**
- **args** = parser.parse_args()
- **doxygen_warning_manager** = AIDoxygenCleaner()

19.2.1 Detailed Description

```
Main execution script for the Doxygen Warning Manager tool.

The AI Doxygen Cleaner is a tool that automates several tasks related to Doxygen warning management. This incl
configuring Doxygen, detecting warnings, inserting warnings into a database, fixing warnings using an AI langu
model, applying these fixes, detecting warnings after fixes have been applied, printing warning details to a f
generating a diff page in HTML format, and updating warning data in the database post-fix.
```

19.2.2 Variable Documentation

19.2.2.1 parser

```
main.main.parser
```

Initial value:

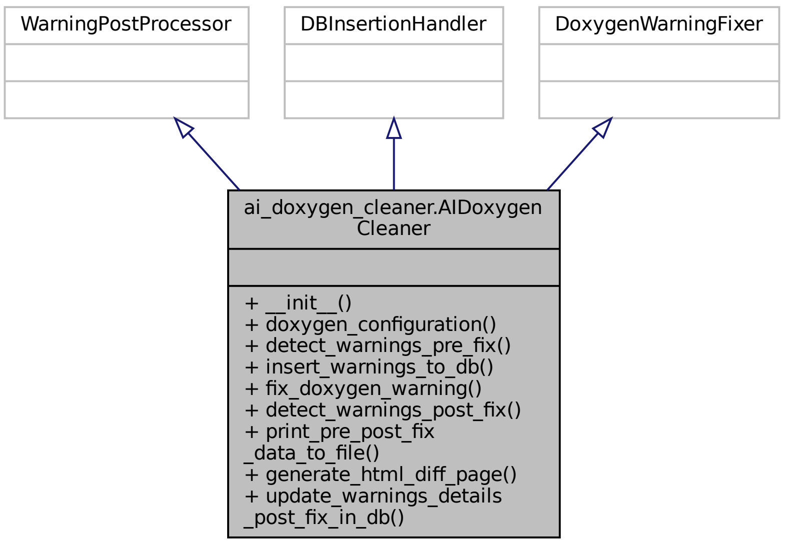
```
=  argparse.ArgumentParser(
       description="A script for configuring Doxygen, detecting and managing warnings.")
```

Chapter 20

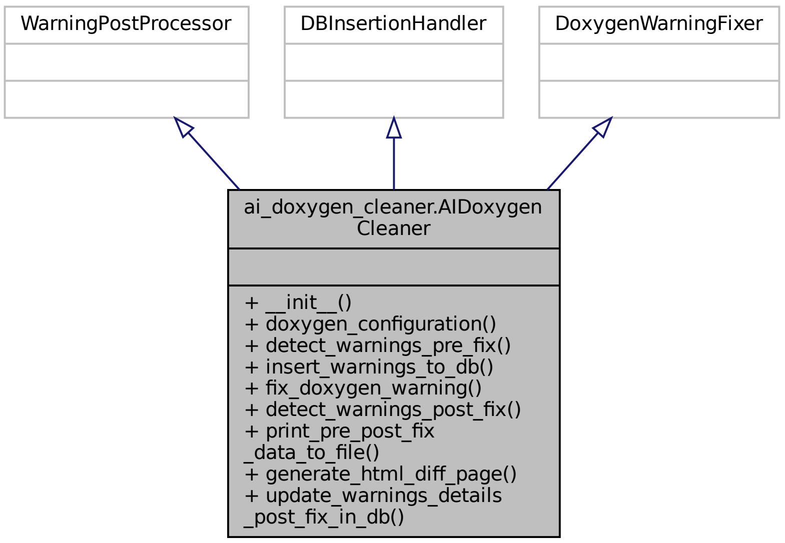
Class Documentation

20.1 ai_doxygen_cleaner.AIDoxygenCleaner Class Reference

Inheritance diagram for ai_doxygen_cleaner.AIDoxygenCleaner:


WarningPostProcessor
DBInsertionHandler
DoxygenWarningFixer
ai_doxygen_cleaner.AIDoxygen
Cleaner
+ __init__()
+ doxygen_configuration()
+ detect_warnings_pre_fix()
+ insert_warnings_to_db()
+ fix_doxygen_warning()
+ detect_warnings_post_fix()
+ print_pre_post_fix
_data_to_file()
+ generate_html_diff_page()
+ update_warnings_details
_post_fix_in_db()

Collaboration diagram for ai_doxygen_cleaner.AIDoxygenCleaner:


WarningPostProcessor
DBInsertionHandler
DoxygenWarningFixer
ai_doxygen_cleaner.AIDoxygen
Cleaner
+ __init__()
+ doxygen_configuration()
+ detect_warnings_pre_fix()
+ insert_warnings_to_db()
+ fix_doxygen_warning()
+ detect_warnings_post_fix()
+ print_pre_post_fix
_data_to_file()
+ generate_html_diff_page()
+ update_warnings_details
_post_fix_in_db()

Public Member Functions

- def **__init__** (self)
- def doxygen_configuration (self)
- def detect_warnings_pre_fix (self)
- def insert_warnings_to_db (self)
- def fix_doxygen_warning (self)
- def detect_warnings_post_fix (self)
- def print_pre_post_fix_data_to_file (self)
- def generate_html_diff_page (self)
- def update_warnings_details_post_fix_in_db (self)

20.1.1 Detailed Description

```
This class is responsible for managing the Doxygen warnings, it consists of several methods
for configuring Doxygen, detecting warnings, fixing them, and saving the fixed warnings.

...

Methods
-------
doxygen_configuration():
    Configures Doxygen.

detect_warnings_pre_fix():
    Detects warnings before fixing.

insert_warnings_to_db():
    Inserts detected warnings into the database.

fix_doxygen_warning():
    Fixes detected Doxygen warnings.

apply_fix():
    Applies the fix to the Doxygen warnings.

detect_warnings_post_fix():
    Detects any remaining warnings after applying the fix.

print_pre_post_fix_data_to_file():
    Prints warning details to a file before and after the fix.

generate_html_diff_page():
    Generates an HTML diff page to show differences before and after the fix.
update_warnings_details_post_fix_in_db():#TODO
```

20.1.2 Member Function Documentation

20.1.2.1 detect_warnings_post_fix()

```
def ai_doxygen_cleaner.AIDoxygenCleaner.detect_warnings_post_fix (
              self )
```

```
Detects Doxygen warnings after applying the fix.

Prints the phase and step of the process.
Runs the warning post processor to detect any remaining warnings.
```

20.1.2.2 detect_warnings_pre_fix()

```
def ai_doxygen_cleaner.AIDoxygenCleaner.detect_warnings_pre_fix (
              self )
```

```
This method is responsible for detecting Doxygen warnings before any fixes are applied.
It stores the warning data and saves it in a JSON file for further processing.

The entire process is printed in the console for visibility.
```

20.1.2.3 doxygen_configuration()

```
def ai_doxygen_cleaner.AIDoxygenCleaner.doxygen_configuration (
              self )
```

```
Configures Doxygen.

This method is responsible for configuring Doxygen.
It removes any existing Doxyfile in the directory and creates a new one.
The new Doxyfile is then configured using provided parameters.

The entire process is printed in the console for visibility.
```

20.1.2.4 fix_doxygen_warning()

```
def ai_doxygen_cleaner.AIDoxygenCleaner.fix_doxygen_warning (
              self )
```

```
Fixes detected Doxygen warnings.

Prints the phase and step of the process.
Runs the fixer to fix the warnings.
```

20.1.2.5 generate_html_diff_page()

```
def ai_doxygen_cleaner.AIDoxygenCleaner.generate_html_diff_page (
              self )
```

```
Prints warning details to a file before and after the fix.

Prints the phase and step of the process.
Loads warning details from JSON files and redirects the print to output files.
```

20.1.2.6 insert_warnings_to_db()

```
def ai_doxygen_cleaner.AIDoxygenCleaner.insert_warnings_to_db (
              self )
```

```
Inserts detected warnings into the database.

Prints the phase and step of the process.
Insert details of the warnings into the database.
```

20.1.2.7 print_pre_post_fix_data_to_file()

```
def ai_doxygen_cleaner.AIDoxygenCleaner.print_pre_post_fix_data_to_file (
              self )
```

```
Prints warning details to a file before and after the fix.

Prints the phase and step of the process.
Loads warning details from JSON files and redirects the print to output files.
```

20.1.2.8 update_warnings_details_post_fix_in_db()

```
def ai_doxygen_cleaner.AIDoxygenCleaner.update_warnings_details_post_fix_in_db (
              self )
```

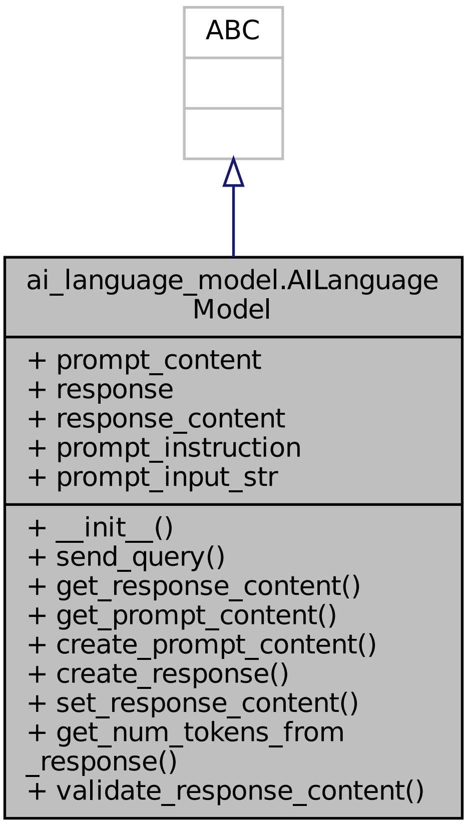
```
Update warning details in the database after applying fixes.

This method handles loading of the post fix data from JSON files, retrieving the pipeline ID and
then updating the warning details in the database with the help of the DBInsertionHandler instance.

Prints:
-------
Various status messages with timestamps indicating the start and end of the database update operation.
```

The documentation for this class was generated from the following file:

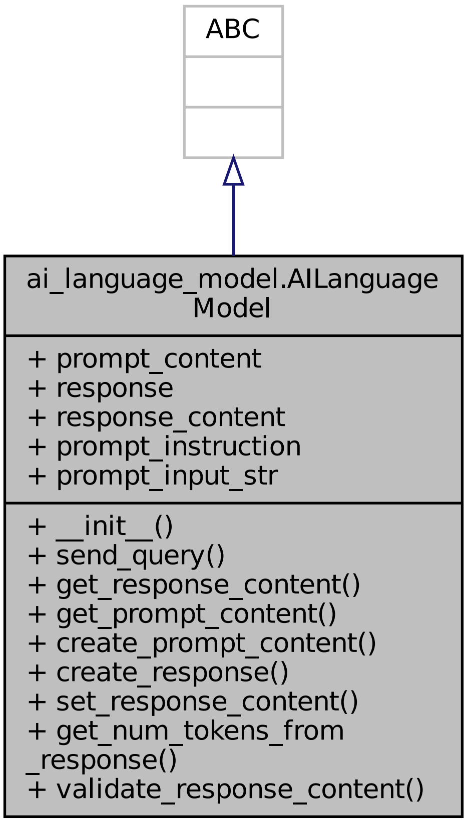
- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/ai_doxygen_cleaner/ai_doxygen_cleaner.py

20.2 ai_language_model.AILanguageModel Class Reference

Inheritance diagram for ai_language_model.AILanguageModel:



Collaboration diagram for ai_language_model.AILanguageModel:



Public Member Functions

- def __init__ (self, prompt_instruction, prompt_input_str)
- def send_query (self)
- str get_response_content (self)
- str get_prompt_content (self)
- def create_prompt_content (self)
- def create_response (self, prompt_content)
- def set_response_content (self)
- int get_num_tokens_from_response (self, str string, str encoding_name)
- def validate_response_content (self)

Public Attributes

- **prompt_content**
- **response**
- **response_content**
- **prompt_instruction**
- **prompt_input_str**

20.2.1 Detailed Description

```
AILanguageModel is an abstract class that serves as a base for AI Language Models.
This class outlines the fundamental methods and attributes derived AI language models should possess.

...

Attributes
----------
prompt_content : str
    The content of the prompt sent to the AI model.
response : any
    The raw response returned from the AI model.
response_content : str
    The extracted content from the AI model response.
prompt_instruction : str
    The instructions that guide the AI model's response.
prompt_input_str : str
    The input string that the AI model will respond to.

Methods
-------
send_query():
    Sends a query to the AI model, including creating the prompt, creating the response,
    and setting and validating the response content.
get_response_content() -> str:
    Returns the content of the response from the AI model.
get_prompt_content() -> str:
    Returns the content of the prompt that was sent to the AI model.
create_prompt_content():
    Creates the content of the prompt by concatenating the instruction and the input string.
create_response(prompt_content):
    Abstract method for creating a response, to be implemented by a specific AI model class.
set_response_content():
    Abstract method for setting the response content, to be implemented by a specific AI model class.
get_num_tokens_from_response(string: str, encoding_name: str) -> int:
    Abstract method for getting the number of tokens from a response, to be implemented by a specific AI model
validate_response_content():
    Abstract method for validating the response content, to be implemented by a specific AI model class.
```

20.2.2 Constructor & Destructor Documentation

20.2.2.1 __init__()

```
def ai_language_model.AILanguageModel.__init__ (
              self,
              prompt_instruction,
              prompt_input_str )
```

```
Initializes an AILanguageModel with given prompt instructions and input string.

Parameters
----------
prompt_instruction : str
    Instruction to guide the language model's response.
prompt_input_str : str
    Initial input for the language model.
```

20.2.3 Member Function Documentation

20.2.3.1 create_prompt_content()

```
def ai_language_model.AILanguageModel.create_prompt_content (
              self )
```

```
Creates the content of the prompt by concatenating the instruction and the input string.
```

20.2.3.2 create_response()

```
def ai_language_model.AILanguageModel.create_response (
              self,
              prompt_content )
```

```
Creates a response from the language model. This is an abstract method that must be implemented in a subclass.

Parameters
----------
prompt_content : str
    Content of the language model prompt.
```

20.2.3.3 get_num_tokens_from_response()

```
int ai_language_model.AILanguageModel.get_num_tokens_from_response (
              self,
             str string,
             str encoding_name )
```

```
Gets the number of tokens in the response. This is an abstract method that must be implemented in a subclass.

Parameters
----------
string : str
    String from which to count tokens.
encoding_name : str
    Name of the encoding method to use.

Returns
-------
int
    Number of tokens in the string.
```

20.2.3.4 get_prompt_content()

```
str ai_language_model.AILanguageModel.get_prompt_content (
              self )
```

```
Gets the content of the language model prompt.

Returns
-------
str
    the content of the prompt that was sent to the AI model.
```

20.2.3.5 get_response_content()

```
str ai_language_model.AILanguageModel.get_response_content (
              self )
```

```
Gets the processed content of the language model's response.

Returns
-------
str
    Processed content of the language model's response.
```

20.2.3.6 send_query()

```
def ai_language_model.AILanguageModel.send_query (
              self )
```

```
Sends a query to the AI model, including creating the response,
and setting and validating the response content.
```

20.2.3.7 set_response_content()

```
def ai_language_model.AILanguageModel.set_response_content (
              self )
```

```
Sets the processed content of the language model's response. This is an abstract method that must be implement
```

20.2.3.8 validate_response_content()

```
def ai_language_model.AILanguageModel.validate_response_content (
              self )
```

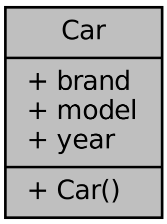
```
Method for validating the response content, to be implemented by a specific AI model class.
```

The documentation for this class was generated from the following file:

- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/ai_language_model/ai_language_model.py

20.3 Car Class Reference

The Car class represents a car with its brand, model, and manufacturing year.
`#include <car.h>`
Collaboration diagram for Car:



Public Member Functions

- Car (string x, string y, int z)
  Constructor for Car class.

Public Attributes

- string brand
- string model
- int year

20.3.1 Detailed Description

The Car class represents a car with its brand, model, and manufacturing year.

20.3.2 Constructor & Destructor Documentation

20.3.2.1 Car()

```
Car::Car (
            string x,
            string y,
            int z )  [inline]
```

Constructor for Car class.
This constructor initializes a Car object with the given brand, model, and manufacturing year.

Parameters

x	The brand of the car.
y	The model of the car.
z	The manufacturing year of the car.

20.3.3 Member Data Documentation

20.3.3.1 brand

```
string Car::brand
```

The brand of the car.

20.3.3.2 model

```
string Car::model
```

The model of the car.

20.3.3.3 year

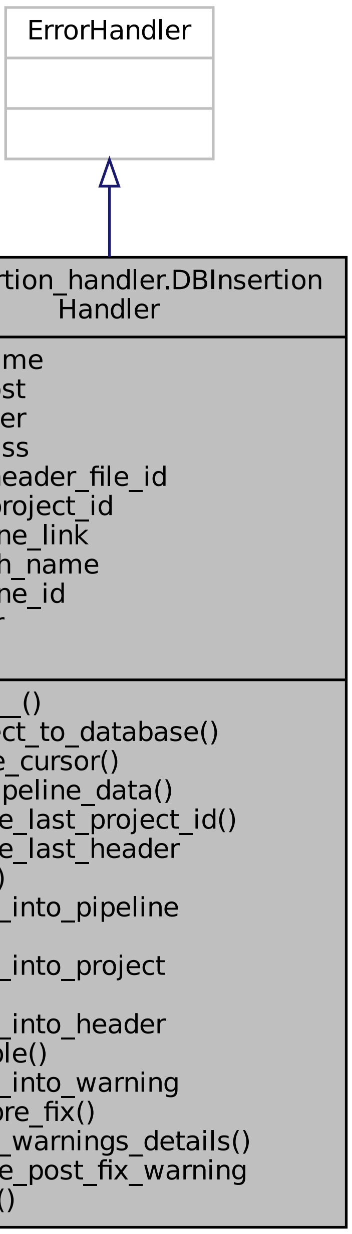
```
int Car::year
```

The manufacturing year of the car.
The documentation for this class was generated from the following file:

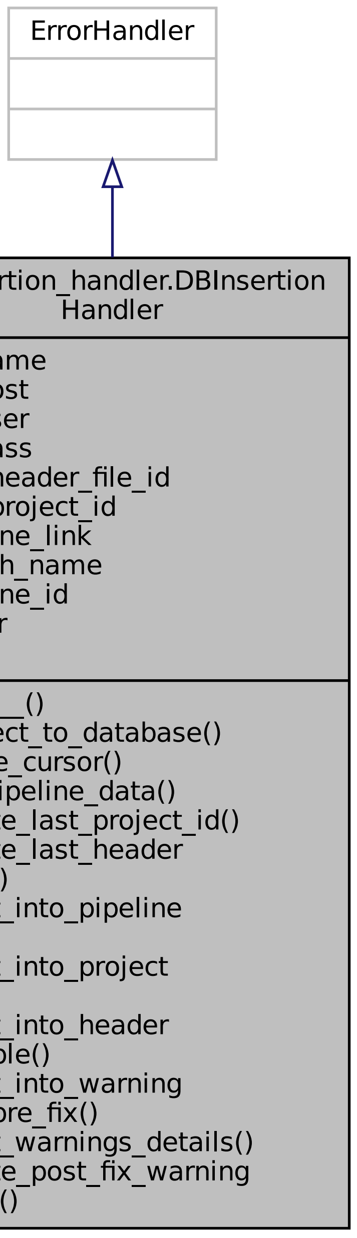
- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/doxygen_projects/chrono/car.h

20.4 db_insertion_handler.DBInsertionHandler Class Reference

Inheritance diagram for db_insertion_handler.DBInsertionHandler:


ErrorHandler
db_insertion_handler.DBInsertion Handler
+ db_name
+ db_host
+ db_user
+ db_pass
+ last_header_file_id
+ last_project_id
+ pipeline_link
+ branch_name
+ pipeline_id
+ cursor
+ db
+ __init__()
+ connect_to_database()
+ create_cursor()
+ set_pipeline_data()
+ update_last_project_id()
+ update_last_header _file_id()
+ insert_into_pipeline _table()
+ insert_into_project _table()
+ insert_into_header _file_table()
+ insert_into_warning _table_pre_fix()
+ insert_warnings_details()
+ update_post_fix_warning _details()

Collaboration diagram for db_insertion_handler.DBInsertionHandler:


ErrorHandler
db_insertion_handler.DBInsertion Handler
+ db_name
+ db_host
+ db_user
+ db_pass
+ last_header_file_id
+ last_project_id
+ pipeline_link
+ branch_name
+ pipeline_id
+ cursor
+ db
+ __init__()
+ connect_to_database()
+ create_cursor()
+ set_pipeline_data()
+ update_last_project_id()
+ update_last_header _file_id()
+ insert_into_pipeline _table()
+ insert_into_project _table()
+ insert_into_header _file_table()
+ insert_into_warning _table_pre_fix()
+ insert_warnings_details()
+ update_post_fix_warning _details()

Public Member Functions

- def __init__ (self)
- def connect_to_database (self)
- def create_cursor (self)
- def set_pipeline_data (self)
- def update_last_project_id (self)
- def update_last_header_file_id (self)
- def insert_into_pipeline_table (self)

- def insert_into_project_table (self, project_folder_path)
- def insert_into_header_file_table (self, file_name, file_content, number_of_warnings)
- def insert_into_warning_table_pre_fix (self, warning_content, line_number_pre_fix, fixed_status=FIXED_S↩ TATUS_DEFAULT)
- def insert_warnings_details (self, project_folder_path, file_name_list, file_content_list, number_of_↩ warnings_list, warning_content_llist, line_number_pre_fix_llist)
- def update_post_fix_warning_details (self, pipeline_id, file_name_list, file_content_list, line_number_llist, warnings_content_llist)

Public Attributes

- **db_name**
- **db_host**
- **db_user**
- **db_pass**
- **last_header_file_id**
- **last_project_id**
- **pipeline_link**
- **branch_name**
- **pipeline_id**
- **cursor**
- **db**

20.4.1 Detailed Description

```
Handles database insert operations for doxygen warnings.

This class manages connections to a MySQL database and provides methods for
inserting information related to doxygen warnings, such as details about the
pipeline, project, and warnings themselves.
...

Attributes
----------
db_name : str
    Name of the database.
db_host : str
    Host of the database.
db_user : str
    Username to connect to the database.
db_pass : str
    Password to connect to the database.
last_header_file_id : int
    The ID of the last inserted header file.
last_project_id : int
    The ID of the last inserted project.
pipeline_link : str
    The link to the GitLab pipeline.
branch_name : str
    The branch name in the repository.
pipeline_id : int
    The ID of the pipeline.
cursor : Cursor
    A MySQL database cursor to execute SQL commands.
db : Connection
    A MySQL database connection.

Methods
-------

connect_to_database():
    Connects to the MySQL database using environment variables.

create_cursor():
    Creates a new cursor to execute database commands.
```

```
set_pipeline_data():
    Sets pipeline data by getting information from environment variables. If no pipeline ID is provided, gener

update_last_project_id():
    Updates the last_project_id member variable with the most recent project ID.

update_last_header_file_id():
    Updates the last_header_file_id member variable with the most recent header file ID.

execute_sql(sql: str, values: tuple, optional):
    Executes a SQL command.

insert_into_pipeline_table():
    Inserts pipeline details into the pipeline table.

insert_into_project_table(project_folder_path: str):
    Inserts project details into the project table and updates the last project ID.

insert_into_header_file_table(file_name: str, file_content: str, number_of_warnings: int):
    Inserts header file details into the header file table and updates the last header file ID.

insert_into_warning_table_pre_fix(warning_content: str, line_number_pre_fix: int, fixed_status: int = FIXED_ST
    Inserts warning details into the warning table.

insert_warnings_details(project_folder_path: str, file_name_list: list, file_content_list: list, number_of_war
    Wrapper function that inserts pipeline, project, header file, and warning details into their respective ta

update_post_fix_warning_details(self, pipeline_id, file_name_list, file_content_list, line_number_llist, warni
    Update the database with the details of post-fix warnings.
```

20.4.2 Constructor & Destructor Documentation

20.4.2.1 __init__()

```
def db_insertion_handler.DBInsertionHandler.__init__ (
              self )
```

```
Initializes a new DBInsertionHandler instance.

During initialization, environment variables are loaded, a database
connection is established, and data related to the pipeline is set.
```

20.4.3 Member Function Documentation

20.4.3.1 connect_to_database()

```
def db_insertion_handler.DBInsertionHandler.connect_to_database (
              self )
```

```
Establishes a connection to the MySQL database using the details obtained from the environment variables.

Raises
------
Exception
    If there is an error while establishing the database connection.
```

20.4.3.2 create_cursor()

```
def db_insertion_handler.DBInsertionHandler.create_cursor (
              self )
```

```
Creates a new cursor to execute database commands.

Raises
------
Exception
    If there is an error while creating a cursor.
```

20.4.3.3 insert_into_header_file_table()

```
def db_insertion_handler.DBInsertionHandler.insert_into_header_file_table (
              self,
              file_name,
              file_content,
              number_of_warnings )
```

```
Inserts header file details into the header file table and updates the last header file ID.

Parameters
----------
file_name : str
    The name of the header file.
file_content : str
    The content of the header file.
number_of_warnings : int
    The number of warnings in the header file.

Raises
------
Exception
    If there is an error while executing the SQL command.
```

20.4.3.4 insert_into_pipeline_table()

```
def db_insertion_handler.DBInsertionHandler.insert_into_pipeline_table (
              self )
```

```
Inserts pipeline details into the pipeline table in the database.

Raises
------
Exception
    If there is an error while executing the SQL command.
```

20.4.3.5 insert_into_project_table()

```
def db_insertion_handler.DBInsertionHandler.insert_into_project_table (
              self,
              project_folder_path )
```

```
Inserts project details into the project table and updates the last project ID.

Parameters
----------
project_folder_path : str
    The path to the project folder.

Raises
------
Exception
    If there is an error while executing the SQL command.
```

20.4.3.6 insert_into_warning_table_pre_fix()

```
def db_insertion_handler.DBInsertionHandler.insert_into_warning_table_pre_fix (
              self,
              warning_content,
              line_number_pre_fix,
              fixed_status = FIXED_STATUS_DEFAULT )
```

```
Inserts the warning details into the warning table before the warnings are resolved by ChatGPT.

Parameters
----------
warning_content : str
    The content of the warning.
line_number_pre_fix : int
    The line number of the warning before fixing by ChatGPT.
fixed_status : int, optional
    The fixed status of the warning (default is FIXED_STATUS_DEFAULT).

Raises
------
Exception
    If there is an error while executing the SQL command.
```

20.4.3.7 insert_warnings_details()

```
def db_insertion_handler.DBInsertionHandler.insert_warnings_details (
              self,
              project_folder_path,
              file_name_list,
              file_content_list,
              number_of_warnings_list,
              warning_content_llist,
              line_number_pre_fix_llist )
```

```
Wrapper function that inserts pipeline, project, header file, and warning details into their respective tables
Once the data is inserted, it commits the changes and closes the database connection.

Parameters
----------
project_folder_path : str
    The path to the project folder.
file_name_list : list
    The list of file names.
file_content_list : list
    The list of file contents.
number_of_warnings_list : list
    The list of the number of warnings per file.
warning_content_llist : list
    The list of warning contents.
line_number_pre_fix_llist : list
    The list of line numbers of warnings before fixing.

Raises
------
Exception
    If there is an error while executing the SQL commands.
```

20.4.3.8 set_pipeline_data()

```
def db_insertion_handler.DBInsertionHandler.set_pipeline_data (
              self )
```

```
Sets pipeline data by getting information from environment variables. If no pipeline ID is provided, generates
Sets pipeline data by getting information from environment variables.
If no pipeline ID is provided, generates a pseudo-random one.
Raises
----------
Exception
    If necessary environment variables are not set.
```

20.4.3.9 update_last_header_file_id()

```
def db_insertion_handler.DBInsertionHandler.update_last_header_file_id (
              self )
```

```
Updates the last_header_file_id member variable with the most recent header file ID from the database.

Raises
------
Exception
    If there is an error while executing the SQL command.
```

20.4.3.10 update_last_project_id()

```
def db_insertion_handler.DBInsertionHandler.update_last_project_id (
              self )
```

```
Updates the last_project_id member variable with the most recent project ID from the database.

Raises
------
Exception
    If there is an error while executing the SQL command.
```

20.4.3.11 update_post_fix_warning_details()

```
def db_insertion_handler.DBInsertionHandler.update_post_fix_warning_details (
              self,
              pipeline_id,
              file_name_list,
              file_content_list,
              line_number_llist,
              warnings_content_llist )
```

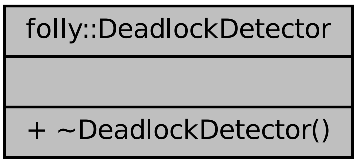
```
Update the database with the details of post-fix warnings.

This method updates the header file content and warning fixed status in the database. It loops
through the given list of file names, updates each file's content, and then checks if there
are any warnings. If there are, it sets the fixed status to 0 and updates this in the database.

Parameters
----------
pipeline_id : str
    The ID of the current pipeline.
file_name_list : list
    A list of the names of the files to update.
file_content_list : list
    A list containing the content of the files.
line_number_llist : list
    A list of lists, each sublist contains the line numbers where warnings occurred.
warnings_content_llist : list
    A list of lists, each sublist contains the warning content from a file.

Raises
------
Exception
    If there is an error while executing the SQL commands.
```

The documentation for this class was generated from the following file:

- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/db_operations/db_insertion_handler.py

20.5 folly::DeadlockDetector Class Reference

The DeadlockDetector class is an abstract base class for deadlock detectors.
`#include <DeadlockDetector.h>`
Collaboration diagram for folly::DeadlockDetector:



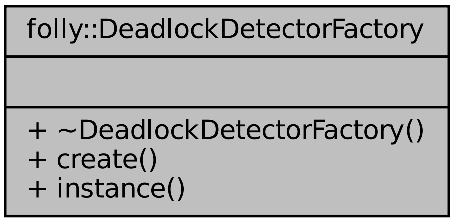
20.5.1 Detailed Description

The DeadlockDetector class is an abstract base class for deadlock detectors.
It provides a virtual destructor to allow derived classes to be deleted through a pointer to the base class.
The documentation for this class was generated from the following file:

- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/doxygen_projects/chrono/DeadlockDetector.↩ h

20.6 folly::DeadlockDetectorFactory Class Reference

The DeadlockDetectorFactory class is an abstract base class for deadlock detector factories.
`#include <DeadlockDetector.h>`
Collaboration diagram for folly::DeadlockDetectorFactory:



Public Member Functions

- virtual std::unique_ptr< DeadlockDetector > create (Executor ∗executor, const std::string &name, std↩ ::unique_ptr< WorkerProvider > threadIdCollector)=0

Creates a new instance of DeadlockDetector.

Static Public Member Functions

- static DeadlockDetectorFactory ∗ instance ()

  Returns an instance of DeadlockDetectorFactory.

20.6.1 Detailed Description

The DeadlockDetectorFactory class is an abstract base class for deadlock detector factories.
It provides a virtual destructor to allow derived classes to be deleted through a pointer to the base class. It also defines a pure virtual function create() that must be implemented by derived classes.

20.6.2 Member Function Documentation

20.6.2.1 create()

```
virtual std::unique_ptr<DeadlockDetector> folly::DeadlockDetectorFactory::create (
            Executor * executor,
            const std::string & name,
            std::unique_ptr< WorkerProvider > threadIdCollector )  [pure virtual]
```

Creates a new instance of DeadlockDetector.
This function creates and returns a new instance of DeadlockDetector using the provided executor, name, and threadIdCollector. The caller takes ownership of the returned object.

Parameters

executor	The executor used by the deadlock detector.
name	The name of the deadlock detector.
threadIdCollector	The worker provider used by the deadlock detector.

Returns

A unique_ptr pointing to the created DeadlockDetector object.

20.6.2.2 instance()

```
static DeadlockDetectorFactory* folly::DeadlockDetectorFactory::instance ( )  [static]
```

Returns an instance of DeadlockDetectorFactory.
This function returns an instance of DeadlockDetectorFactory. Each call to this function will return the same instance. This function can be used as a singleton pattern for accessing the deadlock detector factory.

Returns

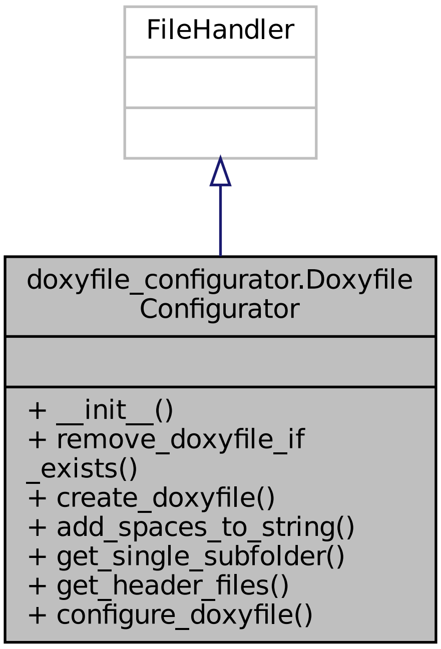
A pointer to the DeadlockDetectorFactory instance.

The documentation for this class was generated from the following file:

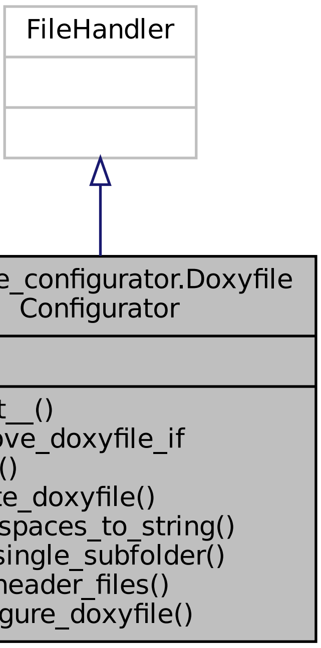
- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/doxygen_projects/chrono/DeadlockDetector.↩
h

20.7 doxyfile_configurator.DoxyfileConfigurator Class Reference

Inheritance diagram for doxyfile_configurator.DoxyfileConfigurator:



Collaboration diagram for doxyfile_configurator.DoxyfileConfigurator:



Public Member Functions

- def __init__ (self)
- def remove_doxyfile_if_exists (self, directory_path)
- def create_doxyfile (self, directory_path)
- def add_spaces_to_string (self, config_parameters)
- def get_single_subfolder (self, directory)
- def get_header_files (self, directory)
- def configure_doxyfile (self, filename, config_parameters)

20.7.1 Detailed Description

```
A class used to configure a Doxyfile.

...

Attributes
----------
file_handler : FileHandler
    an instance of the FileHandler class used to manipulate files

Methods
-------
create_doxyfile(directory_path, cmd):
    Tries to create a Doxyfile at the given directory path, using the provided command
add_spaces_to_string(config_parameters):
    Adds specific spaces to configuration parameters
get_single_subfolder(directory):
    Checks if there is only one subfolder in a given directory
configure_doxyfile(filename, config_parameters):
    Configures the Doxyfile with the given parameters
```

20.7.2 Constructor & Destructor Documentation

20.7.2.1 __init__()

```
def doxyfile_configurator.DoxyfileConfigurator.__init__ (
              self )
```

```
Initializes an instance of the DoxyfileConfigurator class.
```

20.7.3 Member Function Documentation

20.7.3.1 add_spaces_to_string()

```
def doxyfile_configurator.DoxyfileConfigurator.add_spaces_to_string (
              self,
              config_parameters )
```

```
Method to adds specific spaces to configuration parameters.

Parameters
----------
config_parameters : list
    A list of configuration parameters to which spaces are added

Returns
-------
list
    A list of configuration parameters with added spaces
```

20.7.3.2 configure_doxyfile()

```
def doxyfile_configurator.DoxyfileConfigurator.configure_doxyfile (
              self,
              filename,
              config_parameters )
```

```
Configures the Doxyfile with the given parameters.

Parameters
----------
filename : str
    The name of the file to be configured
config_parameters : list
    A list of configuration parameters

It modifies the lines in the Doxyfile according to the provided configuration parameters.
```

20.7.3.3 create_doxyfile()

```
def doxyfile_configurator.DoxyfileConfigurator.create_doxyfile (
              self,
              directory_path )
```

```
Method to create a Doxyfile at the given directory path, using the provided command.

Parameters
----------
directory_path : str
```

```
    The path of the directory where the Doxyfile will be created
cmd : str
    The command used to create the Doxyfile

Raises
------
subprocess.CalledProcessError
    If there is an error while executing the command
FileNotFoundError
    If the directory path does not exist
```

20.7.3.4 get_header_files()

```
def doxyfile_configurator.DoxyfileConfigurator.get_header_files (
              self,
              directory )
```

```
Returns a list of .h and .hpp files in the given directory.

Parameters
----------
directory : str
    The directory to search

Returns
-------
list
    A list of .h and .hpp file paths
```

20.7.3.5 get_single_subfolder()

```
def doxyfile_configurator.DoxyfileConfigurator.get_single_subfolder (
              self,
              directory )
```

```
Checks if there is only one subfolder in a given directory.

Parameters
----------
directory : str
    The directory to check for subfolders

Returns
-------
bool
    A boolean value indicating the presence of a single subfolder
str
    The name of the subfolder (if any)
```

20.7.3.6 remove_doxyfile_if_exists()

```
def doxyfile_configurator.DoxyfileConfigurator.remove_doxyfile_if_exists (
              self,
              directory_path )
```

```
Method to remove a Doxyfile at the given directory path.

Parameters
----------
directory_path : str
    The path of the directory where the Doxyfile will be removed

Raises
```

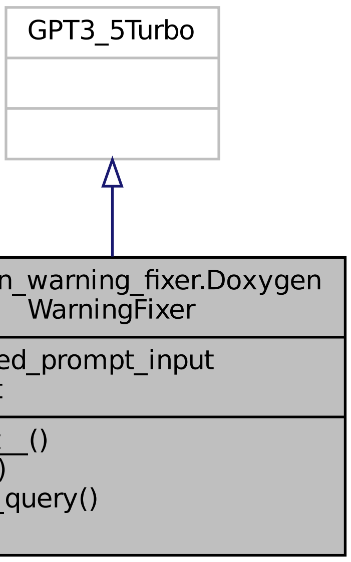
```
------
subprocess.CalledProcessError
    If there is an error while executing the command
FileNotFoundError
    If the directory path does not exist
```

The documentation for this class was generated from the following file:

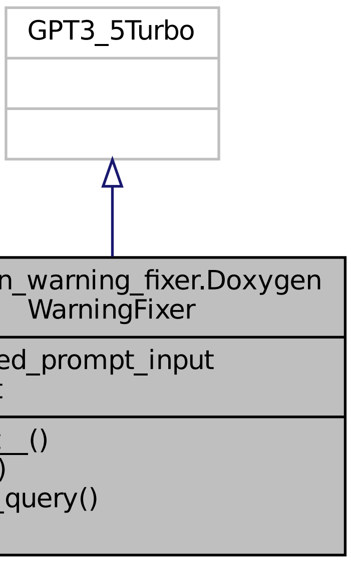
- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/doxygen_management/doxyfile_configurator.↩ py

20.8 doxygen_warning_fixer.DoxygenWarningFixer Class Reference

Inheritance diagram for doxygen_warning_fixer.DoxygenWarningFixer:


GPT3_5Turbo
doxygen_warning_fixer.Doxygen WarningFixer
+ colored_prompt_input _str_list
+ __init__()
+ wait()
+ print_query()
+ run()

Collaboration diagram for doxygen_warning_fixer.DoxygenWarningFixer:



Public Member Functions

- def __init__ (self, prompt_instruction="default_instruction", prompt_input_str="default_instruction", api_↩ key="default_instruction")
- def wait (self)
- def print_query (self, pos)
- def run (self)

Public Attributes

- **colored_prompt_input_str_list**

20.8.1 Detailed Description

```
DoxygenWarningFixer class for detecting and fixing Doxygen warnings.

Methods
-------
run(args)
    Run the warning fixer based on the provided arguments.
wait():
print_query(self, pos):
_concat_warnings_details(self, header_file_name_pre_fix_list, file_content_pre_fix_list,
                              warning_content_pre_fix_llist, warning_line_number_pre_fix_llist):
_fix_warnings(self, prompt_input_str_list, prompt_instruction):
```

20.8.2 Constructor & Destructor Documentation

20.8.2.1 __init__()

```
def doxygen_warning_fixer.DoxygenWarningFixer.__init__ (
              self,
              prompt_instruction = "default_instruction",
              prompt_input_str = "default_instruction",
              api_key = "default_instruction" )
```

```
Initialize the DoxygenWarningFixer object.

Attributes
----------
gpt : None
    To be defined GPT model instance.
colored_prompt_input_str_list : list
    List to hold colored prompt strings for output.
```

20.8.3 Member Function Documentation

20.8.3.1 print_query()

```
def doxygen_warning_fixer.DoxygenWarningFixer.print_query (
              self,
              pos )
```

```
Print the query that is sent to the OpenAI server and its corresponding response.

Parameters
----------
pos : int
    The position of the query in the sequence.
```

20.8.3.2 run()

```
def doxygen_warning_fixer.DoxygenWarningFixer.run (
              self )
```

```
Run the warning fixer based on the provided arguments.

This method loads the warning details from preprocessed_warnings_data.json, sends them to the GPT model,
gets the fixed content, and saves the fixed content to postprocessed_warnings_data.json.
```

20.8.3.3 wait()

```
def doxygen_warning_fixer.DoxygenWarningFixer.wait (
              self )
```

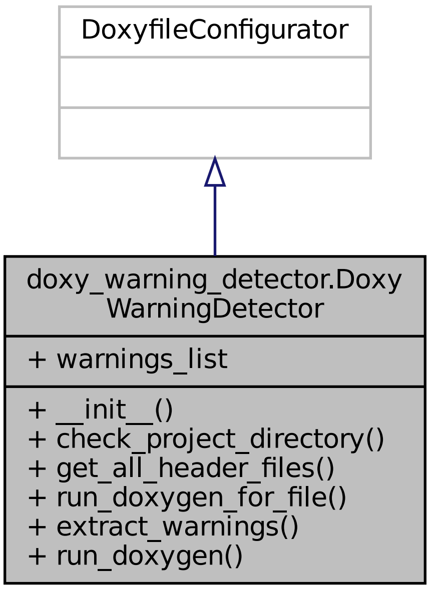
```
Print waiting time every second.

Continuously prints elapsed waiting time every second as long as the thread is running.
```

The documentation for this class was generated from the following file:

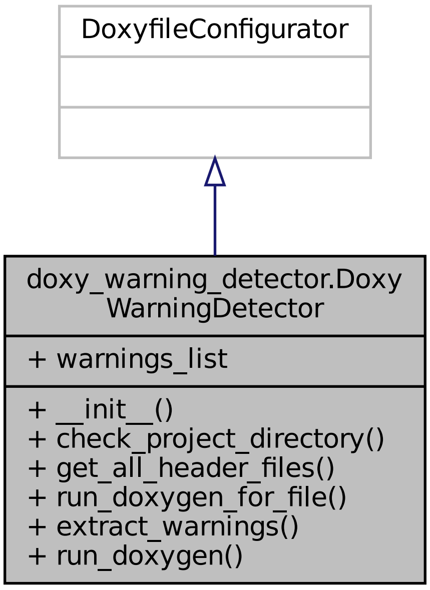
- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/DoxygenWarningFixer/doxygen_warning_↩ fixer.py

20.9 doxy_warning_detector.DoxyWarningDetector Class Reference

Inheritance diagram for doxy_warning_detector.DoxyWarningDetector:



Collaboration diagram for doxy_warning_detector.DoxyWarningDetector:



Public Member Functions

- def __init__ (self)
- def check_project_directory (self)
- def get_all_header_files (self, project_folder_path)
- def run_doxygen_for_file (self, header_file_name, list lines, input_line, project_folder_path, line_num)
- def extract_warnings (self, result, header_file_name)
- def run_doxygen (self)

Public Attributes

- **warnings_list**

20.9.1 Detailed Description

```
A class used to detect warnings in Doxygen documentation generation.

...

Attributes
----------
warnings_list : list
    a list to hold the detected warnings (initialized as an empty list)
doxyfile_configurator : DoxyfileConfigurator
    an object of the class DoxyfileConfigurator to configure the Doxyfile
file_handler : FileHandler
    an object of the class FileHandler to handle file operations

Methods
-------
check_project_directory():
    Checks the project directory for the existence of a single subfolder.

get_all_header_files(project_folder_path: str):
    Gets all the header files in the given project folder path.

run_doxygen_for_file(header_file_name: str, lines: list, input_line: str, project_folder_path: str, line_num:
    Runs Doxygen for the given header file.

extract_warnings(result: subprocess.CompletedProcess):
    Extracts warnings from the result of a Doxygen run.

run_doxygen():
    Executes the complete process to run Doxygen for all header files and extract warnings.
```

20.9.2 Constructor & Destructor Documentation

20.9.2.1 __init__()

```
def doxy_warning_detector.DoxyWarningDetector.__init__ (
              self )
```

```
Initializes DoxyWarningDetector with a DoxyfileConfigurator and a FileHandler.
```

20.9.3 Member Function Documentation

20.9.3.1 check_project_directory()

```
def doxy_warning_detector.DoxyWarningDetector.check_project_directory (
              self )
```

```
Checks if there is only one subfolder in the project directory.
If there are multiple subfolders, the program exits with an error message.
Returns the name of the single subfolder if it exists.

Returns
-------
str
    the name of the single subfolder
```

20.9.3.2 extract_warnings()

def doxy_warning_detector.DoxyWarningDetector.extract_warnings (
self,
result,
header_file_name)

```
Extracts warnings from the result of a Doxygen run.
Returns the extracted warnings.

Parameters
----------
result: subprocess.CompletedProcess
    The result of the Doxygen run
header_file_name: str
    The name of the header file

Returns
-------
list, list, int
    Lists of warning contents, warning line numbers and total number of warnings.
```

20.9.3.3 get_all_header_files()

def doxy_warning_detector.DoxyWarningDetector.get_all_header_files (
self,
project_folder_path)

```
Gets all the header files in the project directory.
If there are no header files, the program exits with an error message.
Returns the list of header files if they exist.

Parameters
----------
project_folder_path: str
    The path to the project directory

Returns
-------
list
    A list of all header files in the project directory.
```

20.9.3.4 run_doxygen()

def doxy_warning_detector.DoxyWarningDetector.run_doxygen (
self)

```
Executes the complete process to run Doxygen for all header files and extract warnings.
The process includes preparing the Doxyfile, running Doxygen, and extracting warnings.
Returns a list of results containing the header file name, project folder path, file content, and warnings.

Returns
-------
tuple
```

```
    The result of running doxygen and extracting warnings. It includes the project folder path,
    list of header file names, list of file contents, list of warning counts,
    list of warning contents, and list of warning line numbers.
```

20.9.3.5 run_doxygen_for_file()

```
def doxy_warning_detector.DoxyWarningDetector.run_doxygen_for_file (
              self,
              header_file_name,
             list lines,
              input_line,
              project_folder_path,
              line_num )
```

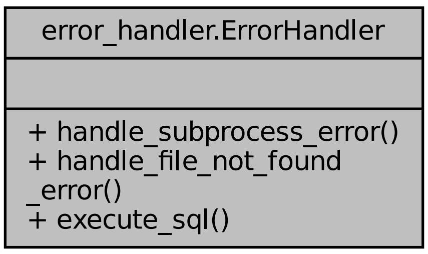
```
Prepares the Doxyfile with the given parameters and runs Doxygen for the header file.
Returns the result of the Doxygen run.

Parameters
----------
header_file_name: str
    The name of the header file
lines: list
    A list of lines in the Doxyfile
input_line: str
    The line to add to the Doxyfile
project_folder_path: str
    The path to the project directory
line_num: int
    The line number in the Doxyfile to modify

Returns
-------
subprocess.CompletedProcess
    The result of the Doxygen run
```

The documentation for this class was generated from the following file:

- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/doxygen_management/doxy_warning_↩ detector.py

20.10 error_handler.ErrorHandler Class Reference

Collaboration diagram for error_handler.ErrorHandler:



Static Public Member Functions

- def handle_subprocess_error (e, custom_message)
- def handle_file_not_found_error (e, custom_message)
- def execute_sql (cursor, sql, values=None)

20.10.1 Detailed Description

```
A class used to handle common errors that may occur during program execution

...

Static methods
--------------
handle_subprocess_error(e, custom_message):
    Handles subprocess errors by displaying the error message in a custom format and terminating the program
handle_file_not_found_error(e, custom_message):
    Handles file not found errors by displaying the error message in a custom format and terminating the progr
execute_sql(cursor, sql, values=None):
```

20.10.2 Member Function Documentation

20.10.2.1 execute_sql()

```
def error_handler.ErrorHandler.execute_sql (
              cursor,
              sql,
              values = None )  [static]
```

```
Executes a SQL command.

Parameters
----------
sql : str
    The SQL command to be executed.
values : tuple, optional
    The values to be inserted into the SQL command (default is None).

Raises
------
Exception
    If there is an error while executing the SQL command.
```

20.10.2.2 handle_file_not_found_error()

```
def error_handler.ErrorHandler.handle_file_not_found_error (
              e,
              custom_message )  [static]
```

```
Handles file not found errors by displaying the error message in a custom format and terminating the program

Parameters
----------
e : Exception
    The exception raised
custom_message : str
    The custom message to be displayed before the exception message
```

20.10.2.3 handle_subprocess_error()

```
def error_handler.ErrorHandler.handle_subprocess_error (
              e,
              custom_message )  [static]
```

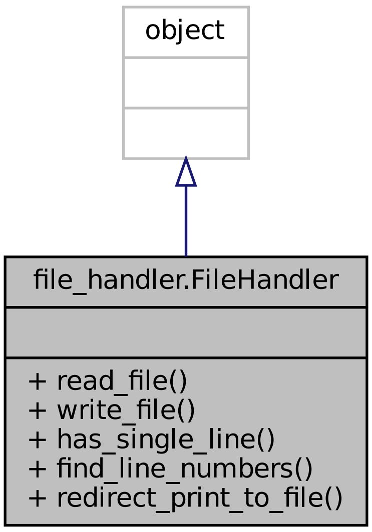
```
Handles subprocess errors by displaying the error message in a custom format and terminating the program

Parameters
----------
e : Exception
    The exception raised
custom_message : str
    The custom message to be displayed before the exception message
```

The documentation for this class was generated from the following file:

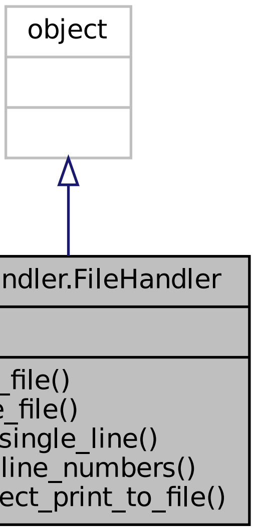
- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/error_handler/error_handler.py

20.11 file_handler.FileHandler Class Reference

Inheritance diagram for file_handler.FileHandler:



Collaboration diagram for file_handler.FileHandler:



Public Member Functions

- def read_file (self, filename)
- def write_file (self, filename, lines)
- def has_single_line (self, list list)
- def find_line_numbers (self, filename, searchText)
- def redirect_print_to_file (self, project_folder_path, header_file_name_list, warning_num_list, warning_↩ content_llist, warning_line_number_llist, file_content_list, pre_or_post, output_file)

20.11.1 Detailed Description

```
A class used to handle file operations.

...

Methods
-------
read_file(filename: str) -> list:
    Reads a file and returns its contents as a list of lines.
write_file(filename: str, lines: list):
    Writes a list of lines into a file.
has_single_line(list: list) -> bool:
    Checks if a list has only a single line.
find_line_numbers(filename: str, searchText: str) -> int:
    Finds the line numbers of all the occurrences of a specific text in a file.

redirect_print_to_file(self, project_folder_path, header_file_name_list,
                           warning_num_list, warning_content_llist,
                           warning_line_number_llist, file_content_list, pre_or_post,
                           output_file):
    Redirects the output of the 'print_project_details_without_color' function to a specified file.
```

20.11.2 Member Function Documentation

20.11.2.1 find_line_numbers()

```
def file_handler.FileHandler.find_line_numbers (
              self,
              filename,
              searchText )
```

```
Finds the line numbers of all the occurrences of a specific text in a file.

Parameters
----------
filename : str
    The name of the file to be searched.
searchText : str
    The text to be found.

Returns
-------
int
    The line number of the first occurrence of the search text
    if there is only one occurrence, else it exits the program.
```

20.11.2.2 has_single_line()

```
def file_handler.FileHandler.has_single_line (
              self,
             list list )
```

```
Checks if a list has only a single line.

Parameters
----------
list : list
    The list to check.

Returns
-------
bool
    True if the list has only a single line, else it exits the program.
```

20.11.2.3 read_file()

```
def file_handler.FileHandler.read_file (
              self,
              filename )
```

```
Reads a file and returns its contents as a list of lines.

Parameters
----------
filename : str
    The name of the file to be read.

Returns
-------
list
    A list of lines from the file.
```

20.11.2.4 redirect_print_to_file()

```
def file_handler.FileHandler.redirect_print_to_file (
              self,
              project_folder_path,
              header_file_name_list,
              warning_num_list,
              warning_content_llist,
              warning_line_number_llist,
              file_content_list,
              pre_or_post,
              output_file )
```

```
Redirects the output of the 'print_project_details_without_color' function to a specified file.

This function modifies the standard output (stdout) to a specified file,
calls a function to print data without color, and then restores stdout to its original state.

Parameters
----------
project_folder_path : str
    The path of the project folder.
header_file_name_list : list
    List of header file names.
warning_num_list : list
    List of warning numbers.
warning_content_llist : list
    List of warning content.
warning_line_number_llist : list
    List of warning line numbers.
file_content_list : list
    List of file content.
pre_or_post : str
    String indicating whether it's pre or post data.
output_file : str
    Path to the output file where print statements are to be redirected.
```

20.11.2.5 write_file()

```
def file_handler.FileHandler.write_file (
              self,
              filename,
              lines )
```

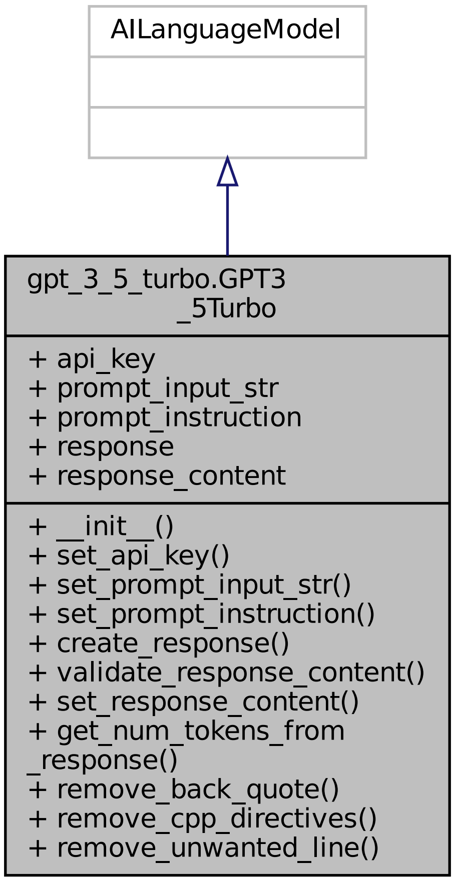
```
Writes a list of lines into a file.

Parameters
----------
filename : str
    The name of the file to write to.
lines : list
    A list containing the lines to be written to the file.
```

The documentation for this class was generated from the following file:

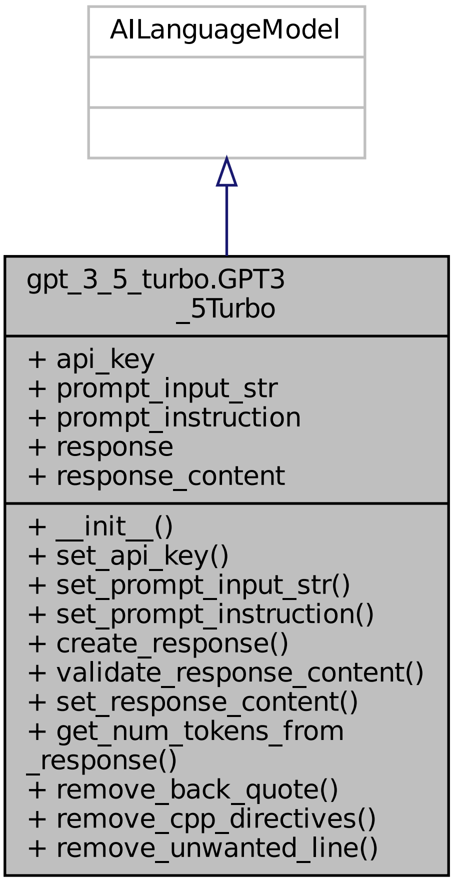
- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/doxygen_management/file_handler.py

20.12 gpt_3_5_turbo.GPT3_5Turbo Class Reference

Inheritance diagram for gpt_3_5_turbo.GPT3_5Turbo:



Collaboration diagram for gpt_3_5_turbo.GPT3_5Turbo:



Public Member Functions

- def __init__ (self, prompt_instruction, prompt_input_str, api_key)
- def **set_api_key** (self, api_key)
- def **set_prompt_input_str** (self, prompt_input_str)
- def **set_prompt_instruction** (self, prompt_instruction)
- def create_response (self, prompt_content)
- def validate_response_content (self)
- def set_response_content (self)
- int get_num_tokens_from_response (self, str string, str encoding_name)
- def remove_back_quote (self)
- def remove_cpp_directives (self)
- def remove_unwanted_line (self)

Public Attributes

- **api_key**
- **prompt_input_str**
- **prompt_instruction**

- **response**
- **response_content**

20.12.1 Detailed Description

```
This class extends the AILanguageModel class and is specifically designed to
handle interaction with the GPT-3.5 Turbo model provided by OpenAI. It enables
communication with the model, formatting and validation of the responses, and
also contains utility functions for token counting and line removal.
...

Attributes
----------
api_key : str
    The OpenAI API key used to authenticate requests to the API.

Methods
----------
create_response(prompt_content):
    Creates a response from the GPT-3.5 Turbo model using the given prompt content.
validate_response_content():
    Validates the content of the model's response and performs necessary modifications.
set_response_content():
    Sets the processed content of the model's response.
get_num_tokens_from_response(string: str, encoding_name: str) -> int:
    Counts the number of tokens in a given string.
remove_back_quote():
    Removes backquote encapsulated strings from the model's response.
remove_cpp_directives():
    Removes C++ directive lines from the model's response.
remove_unwanted_line():
    Removes unwanted lines from the model's response.
set_api_key(self, api_key):

set_prompt_input_str(self, prompt_input_str):

set_prompt_instruction(self, prompt_instruction):
```

20.12.2 Constructor & Destructor Documentation

20.12.2.1 __init__()

```
def gpt_3_5_turbo.GPT3_5Turbo.__init__ (
              self,
              prompt_instruction,
              prompt_input_str,
              api_key )
```

```
Constructs all the necessary attributes for the GPT3_5Turbo object.

Parameters
----------
prompt_instruction : str
    The instruction to guide the language model's response.
prompt_input_str : str
    The initial string input for the language model.
api_key : str
    The OpenAI API key to be used for making API requests.
```

20.12.3 Member Function Documentation

20.12.3.1 create_response()

```
def gpt_3_5_turbo.GPT3_5Turbo.create_response (
              self,
              prompt_content )
```

```
Calls the OpenAI API and generates a response using the GPT-3.5 Turbo model.

Parameters
----------
prompt_content : str
    Content of the language model prompt.

References
----------
1. OpenAI. (no date). API Documentation: Making Requests. OpenAI Platform. Available at:
    https://platform.openai.com/docs/api-reference/making-requests. Accessed on July 8, 2023.
```

20.12.3.2 get_num_tokens_from_response()

```
int gpt_3_5_turbo.GPT3_5Turbo.get_num_tokens_from_response (
              self,
             str string,
             str encoding_name )
```

```
Counts the number of tokens in a given string using OpenAI's tiktoken library.

Parameters
----------
string : str
    The string from which tokens are to be counted.
encoding_name : str
    The name of the encoding method to be used for token counting.

Returns
-------
int
    The number of tokens in the input string.

openai-cookbook/examples
References
----------
1. OpenAI Cookbook. (no date). How to count tokens with tiktoken. GitHub. Available at:
    https://github.com/openai/openai-cookbook/blob/main/examples/How_to_count_tokens_with_tiktoken.ipynb Acces
2. OpenAI. (no date). tiktoken. GitHub. Available at:
    https://github.com/openai/tiktoken/blob/main/tiktoken/model.py Accessed on July 8, 2023.
3. OpenAI. (no date). Tokenizer. OpenAI Platform. Ac Available at:
    https://platform.openai.com/tokenizer Accessed on July 8, 2023.
```

20.12.3.3 remove_back_quote()

```
def gpt_3_5_turbo.GPT3_5Turbo.remove_back_quote (
              self )
```

```
Removes lines containing backquote encapsulated strings from the model's response.
```

20.12.3.4 remove_cpp_directives()

```
def gpt_3_5_turbo.GPT3_5Turbo.remove_cpp_directives (
              self )
```

```
Removes C++ directive lines from the model's response.
```

20.12.3.5 remove_unwanted_line()

```
def gpt_3_5_turbo.GPT3_5Turbo.remove_unwanted_line (
              self )
```

```
Removes unwanted lines from the model's response.
```

20.12.3.6 set_response_content()

```
def gpt_3_5_turbo.GPT3_5Turbo.set_response_content (
              self )
```

```
Sets the content of the model's response by extracting the relevant portion
from the model's response.
```

20.12.3.7 validate_response_content()

```
def gpt_3_5_turbo.GPT3_5Turbo.validate_response_content (
              self )
```

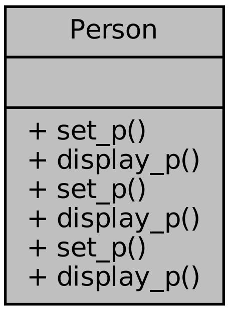
```
Validates the content of the model's response and performs necessary modifications like removing backquotes ar
unwanted lines, and C++ directives.
```

The documentation for this class was generated from the following file:

- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/ai_language_model/models/gpt_3_5_turbo.py

20.13 Person Class Reference

Collaboration diagram for Person:



Public Member Functions

- void set_p (int id, char n[ ])

  Sets the ID and name of the person.
- void display_p ()

  Displays the ID and name of the person.
- void set_p (int id, char n[ ])

Sets the ID and name of the person.

- void display_p ()

  Displays the ID and name of the person.
- void set_p (int id, char n[ ])

  Sets the ID and name of the person.
- void display_p ()

  Displays the ID and name of the person.

20.13.1 Member Function Documentation

20.13.1.1 display_p() [1/3]

```
void Person::display_p ( )
```

Displays the ID and name of the person.
This function displays the ID and name of a Person object on standard output.

20.13.1.2 display_p() [2/3]

```
void Person::display_p ( )
```

Displays the ID and name of the person.
This function displays the ID and name of a Person object on standard output.

20.13.1.3 display_p() [3/3]

```
void Person::display_p ( )
```

Displays the ID and name of the person.
This function displays the ID and name of a Person object on standard output.

20.13.1.4 set_p() [1/3]

```
void Person::set_p (
             int id,
             char n[ ] )
```

Sets the ID and name of the person.
This function sets the ID and name of a Person object.

Parameters

id	The ID of the person.
n	The name of the person.

20.13.1.5 set_p() [2/3]

```
void Person::set_p (
             int id,
             char n[ ] )
```

Sets the ID and name of the person.
This function sets the ID and name of a Person object.

Parameters

id	The ID of the person.
n	The name of the person.

20.13.1.6 set_p() [3/3]

```
void Person::set_p (
             int id,
             char n[] )
```

Sets the ID and name of the person.
This function sets the ID and name of a Person object.

Parameters

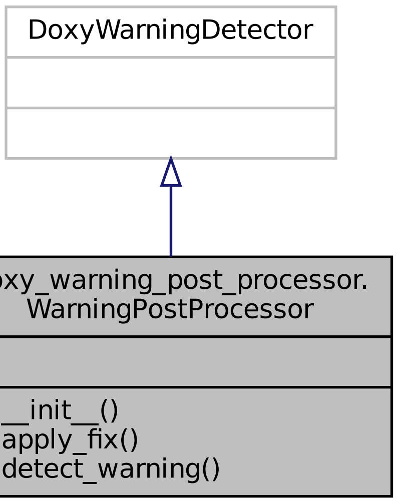
id	The ID of the person.
n	The name of the person.

The documentation for this class was generated from the following files:

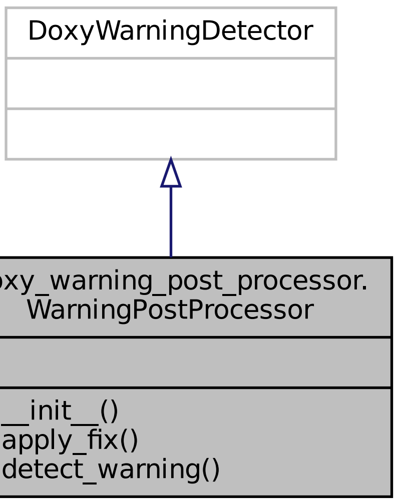
- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/data/different_between_pre_and_post/output←_post_fix.cpp
- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/data/different_between_pre_and_post/output←_pre_fix.cpp
- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/doxygen_projects/chrono/person.h

20.14 doxy_warning_post_processor.WarningPostProcessor Class Reference

Inheritance diagram for doxy_warning_post_processor.WarningPostProcessor:



Collaboration diagram for doxy_warning_post_processor.WarningPostProcessor:



Public Member Functions

- def **__init__** (self)
- def apply_fix (self)
- def detect_warning (self)

20.14.1 Detailed Description

```
The WarningPostProcessor class handles detection and processing of warnings after fixes have been applied.

Attributes
----------
None

Methods
-------
apply_fix():
    Applies fixes to the header files and saves changes.
detect_warning():
    Detects and updates warning details post-fix.
```

20.14.2 Member Function Documentation

20.14.2.1 apply_fix()

```
def doxy_warning_post_processor.WarningPostProcessor.apply_fix (
              self )
```

```
Applies fixes to the header files and saves changes.

This method loads pre-warning details, loads post-fix file content, and writes the post-fix file
content back to the header files.
```

20.14.2.2 detect_warning()

```
def doxy_warning_post_processor.WarningPostProcessor.detect_warning (
              self )
```

```
Detects and updates warning details post-fix.

This method loads pre-warning details, detects warnings after fixes, updates warning lists,
and finally adds the updated warning details to the postprocessed_warnings_data.json file.

Parameters
----------
None

Returns
-------
None
```

The documentation for this class was generated from the following file:

- /home/ismail/Desktop/bachelorarbeit/ai-doxygencleaner/main/doxy_warning_post_processor/doxy_↩ warning_post_processor.py