GSoC 2024 Final Report

Description of the goals of the project

InVesalius is a cross-platform (Windows/ linux/ Apple OS) 3D Medical visualization and neuro-navigation tool, developed since 2001 by Centro de Tecnologia da Informação Renato Archer (CTI), Brazil. The software is mainly used for rapid prototyping, teaching, and forensics in the medical field. (https://invesalius.github.io/about.html).

My goal towards this project was to develop tools to facilitate users to activate the capture of logs and errors. The deliverables were as follows:
1. Graphic interface integrated to InVesalius that allows the user to activate the tool and save the logs, supporting respective log levels (debug, info, warn, error or critical). It should be possible to save the sequence of events in a text file.
2. Log support to all InVesalius functions and functionalities.
3. A separate text window for console stream output.
4. Add provision/tools for error catching.
https://summerofcode.withgoogle.com/programs/2024/projects/zYrNyRVw

What I did

After going through and getting an understanding of the code, I concluded to work towards fulfilling my goal in following three steps:

a. Front end user interface for accepting user preferences for logging: After few iterations, we have now the following interface for accepting user preferences for logging:

b.
c. Backend tools for setting up logging support as per user preferences,
d. Propagating logging message appropriately as per need, and
e. Tools for error catching support.

The work done by me are as follows:

a.
.

- A

- What you did.
- The current state.
- What's left to do.
- What code got merged (or not) upstream.
- Any challenges or important things you learned during the project.

Feature: Interface for logging parameters to invesalius3.
Additions to the code:

1. Preferences.py

a. Class Logging: Added a new class for managing logging pane for parameter input.
Four parameters:
i. Logging: To do or not to do
ii. Logging Level:

iii.Append: To append in the log file or start afresh
iv.File Name:

b. Class Preferences: Added a pane for editing logging parameters

2. Frames.py:

a. Class Frame:

i.Method ShowPreferences: Additions to load parameter values from the configuration file.

3. Session.py:

a. Class Session:

i.Added method __set_default_logfile

ii.Additions to methods: __init__, CreateConfig, _read_config_from_ini, _read_config_from_json,

b. Adds following parameters to the configuration file: 'do_logging', 'logging_level', 'append_log_file', 'logging_file'

4. Constants.py:

Adds: LOGGING, LOGGING_LEVEL, APPEND_LOG_FILE, LOGFILE

==========

I want to do it through defining a decorator function. They are like a function of a function.
Please look at my function call_tracking_decorator in file log.py (line 274-279), which adds log message to the input function before calling the input function.
To add this decorator to the function OnModifyButton, we simply 'decorate' the function declaration by
@log.call_tracking_decorator.
This can be done with every function we want to log calling the function.
The advantages I see here are as follows:
1. The log message code is not repeated, and localised.
2. There is no risk of introducing error in existing code as existing code is not touched.
2. If there is an error or change is needed in logging, it can be rectified in the decorator alone.

==========

I have ma de few changes in the decorator function to use. I am now passing file name, name of the class (if any) and function name to the decorator as parameters. File name is automatically passed, but I am figuring if class name and function name could be obtained automatically. Function name was possible in the earlier version when parameters weren't passed.
I have added the decorator to functions in image/bitmap reader functions I felt appropriate. I have also added decorator to the segmentation functions. I have run few times with no errors. Please suggest any test code to try.