**Coding Style Guide**

**Background**

C++ will be the programming language used to build the annotations tool program. C++ is very good for making complex computer applications and graphical interfaces. Although this feature is great, this makes coding more difficult and bug prone compared to other languages like Java.

The purpose of this document is to provide a standard way of programming the annotation application to maintain simplicity and consistency in the code.

**C++ Version**

Code should be written in C++17 but optionally implementation of C++20 is readily available.

**Header Files**

Every .cc file should have an associated .h file, with some exceptions like unit tests and small cc files containing just a main() function.

**The #define Guard**

All header files should have #define guards to prevent multiple inclusion. The format of the symbol name should be <PROJECT>\_<PATH>\_<FILE>\_H\_.

**Self-contained Headers**

Header files should be self-contained and end in .h

**Names and Order of Includes**

Include headers in the following order: Related header, C system headers, C++ standard library headers, other libraries' headers, your project's headers.

**Scoping**

**Local Variables**

Define local variables as close to first use as possible. This will include initializing variables when its declared.

**Member Functions**

Access modifiers should be used appropriately to limit the scope of functions. Add these accordingly so functions are used only where needed.

**Classes**

**Structs vs Classes**

Use a struct for passive objects only that carry data, otherwise, the rest should be a class.

All fields must be public and accessed directly rather through getter/setter method.

Use of class is more suitable when more functionalities are required.

**Inheritance**

When inheritance is in use, it should be public. In the case of private inheritance, it should include an instance of the base class as a member instead.

**Access Control**

Unless data members are constants, otherwise the classes’ data members must be private.

**Order of Declaration**

It should be done by placing public parts first and all similar declarations grouped together. Preferred in the following order: types (including typedef, using, and nested structs and classes), constants, factory functions, constructors, assignment operators, destructor, all other methods, data members.

A class definition starts with public:, then protected: and then private:

**Functions**

Functions that are not declared ‘void’ should have a return value.

Functions should also be small, clear and concise. So that it is easier for members to understand the code and easier for the Software Tester to test them.

**Output Parameters**

Output of C++ function should be done via a return value or output parameters. Return values are preferred since it would provide better readability and performance. In the case that output parameters are used, they should appear after input parameters.

**Naming**

**File Name**

File names should be written with should begin with capital letters and use underscores to differentiate words. Examples include:

* My\_Annottations.cc

**Variable Name**

Hungarian naming format should be used for all variables. E.g.

* int getAge
* string fileName

**Function Name**

Mixed caseShould start with a lowercase letter and have a capital letter for each word. E.g.

* openFileDirectory()
* writeFile()

To distinguish between functions that are attached to button clicks, names should start with ‘on\_’ and finish with ‘\_clicked’. E.g.

* on\_objectName\_clicked

**Class names**

Should start with a capital letter and have a capital letter for each word.

**Formatting**

**Tabs**

The tab function will be used to indent functions and nested functions. This will increase the readability of the code.

**Line length**

80 characters per line is the maximum

**Boolean Expressions**

When Boolean expression is longer than set line length, break up the code in a readable manner

**Variable and Array**

We use either =, (), and {}.

**Comments**

**Comment Style**

Use // for single line comments and /\* \*/ for multiple line comments

**Function Comments**

Before each new function, describe what it does using comments. For non-obvious, complex lines of code comments should be written before them.