Google Summer of Code 2020 Proposal

GUI Revamp

Rewriting LibreCAD 3 GUI entirely in C++

Table of Contents

Personal Information

Name: Akhil J Nair

Major: Computer Engineering

University: Army Institute Of Technology, Pune, India

Expected Graduation Year: 2022

GitHub: Jedi18

ZulipChat: Akhil Nair

Email: targetakhil@gmail.com

Phone: (+91)7507632797

Time zone: Indian Standard Time (UTC +5:30)

Brief Project Summary :-

LibreCAD 3

LibreCAD is a free Open Source CAD application for Windows, Apple and Linux. LibreCAD 3 is a next generation 2D CAD application written to be modular, with a core independent from GUI toolkits. LibreCAD supports scripting with Lua.

LibreCAD 3 UI

LibreCAD 3 UI is currently written in both C++ and Lua, the widget classes are written in C++/Qt and Lua scripts are responsible for creating and initializing the GUI.

Motivation

The current system of using Lua to initialize the UI has the following problems: -

1) The Lua Qt bridge needs to be maintained as we need to bind the Qt methods to Lua ourselves.

2) Lua being a scripting language is difficult to debug, since there are no compiler checks or unit tests, which is a major problem for stability, especially during refactors.

3) No fixed API for Lua to interact with the UI, qtbridge is regularly changed and user needs to know Qt to make changes to the UI.

Project outcome

1) Every Lua GUI file will be rewritten in C++, window management of the UI will be shifted to C++.

2) Overhaul the toolbar to make it customizable by the user.

3) Remove Qt bindings to Lua, replacing it with the Lua API bridge.

4) Provide an API accessible to Lua plugins to change the GUI widgets and create GUI elements in the interface.

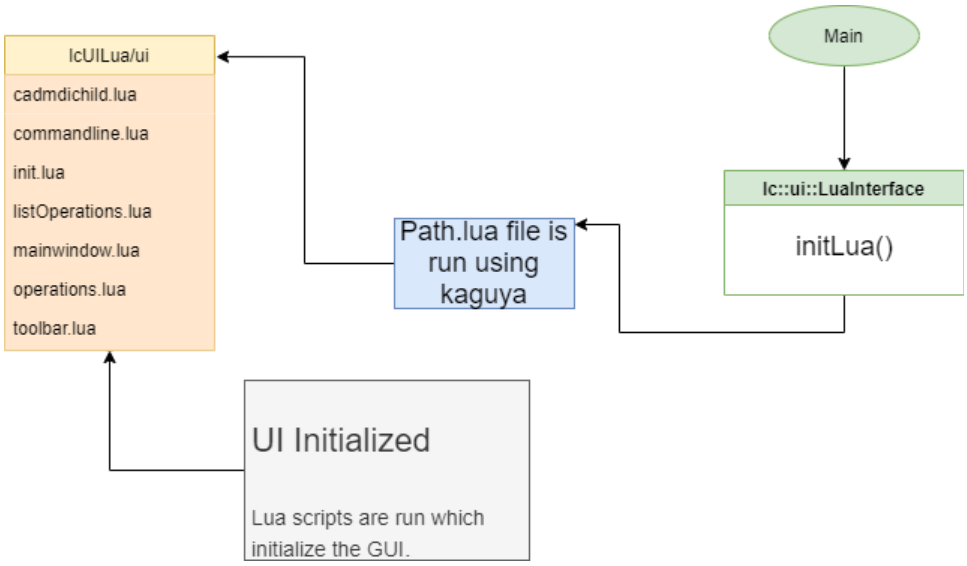
5) Write documentation for the new Lua GUI API.

6) Comprehensive unit tests for LibreCAD 3 GUI.

Detailed Project Description :-

Current Status :-

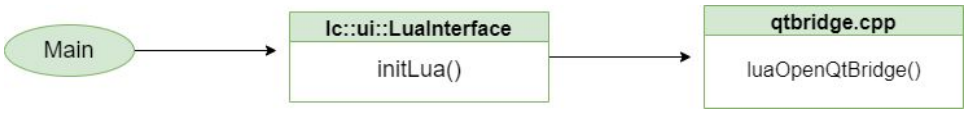
The diagram below shows how the Lua UI scripts are currently executed.



By the end of the project, the functionality of all these files in *lcUILua/ui* will be moved to C++ (with a few exceptions like *run_basic_operation()* in *operations.lua* which will remain in Lua and will be dealt with accordingly).

These Lua scripts access the widgets written in Qt and C++ through the Qt bindings to Lua maintained in the *qtbridge.cpp*.

LibreCAD 3 makes use of kaguya for C++ to Lua bindings. All the required bindings for Qt are made in the *qtbridge.cpp* file. It makes the Qt widgets as well as custom widgets available to Lua so that the Lua scripts can initialize them.



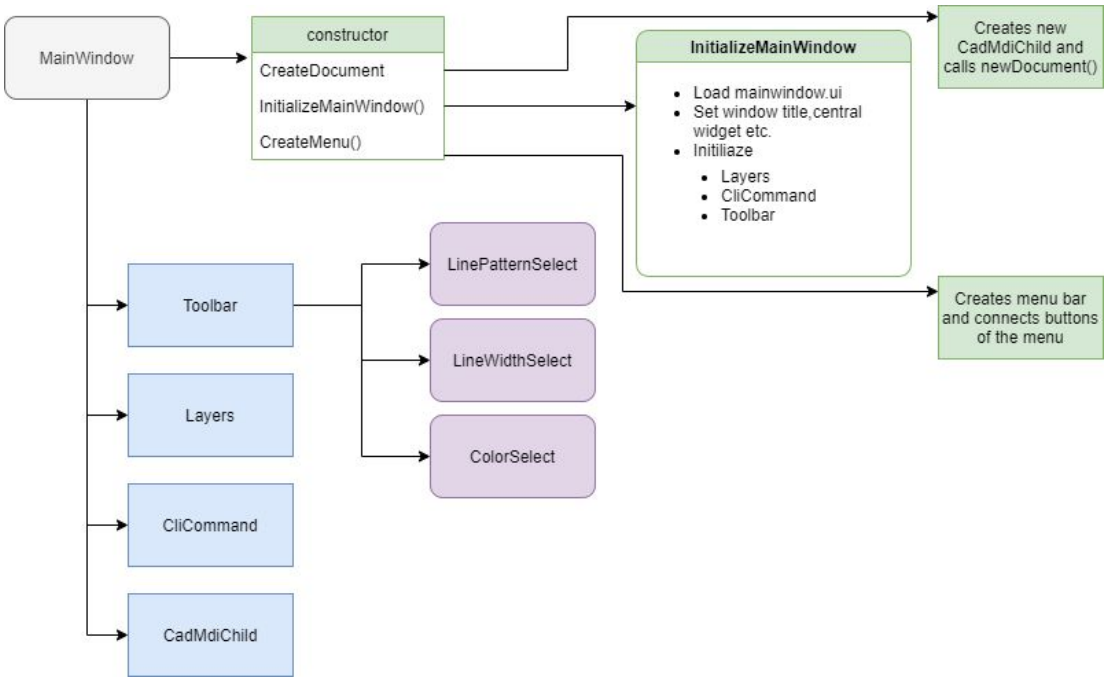
Rewriting the GUI in C++ :-

MainWindow class will be created which will initialize the GUI and contain the CadMdiChild, Toolbar, Layers and CliCommand as attributes. The constructor would do the following things.

A new CadMdiChild (MDI – Multiple Document Interface) will be created and a new document would be created.

Initializing the GUI would include the loading of the mainwindow.ui file in lcUI/ui, setting of window title, central widget, etc. and initialize and call the constructors of Toolbar, Layers and CliCommand.

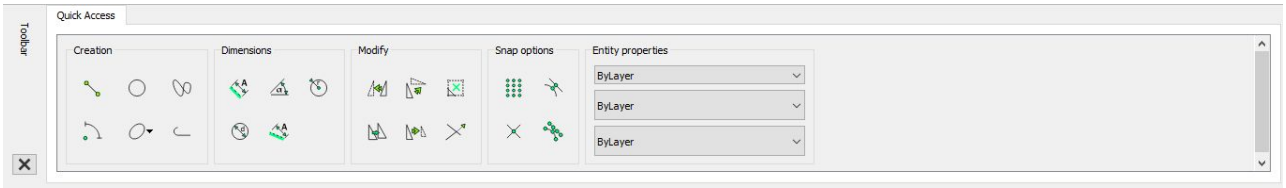
The menu bar would be created and the buttons will be added and connected (through a Lua script using the new Lua GUI API mentioned later in the proposal).



MainWindow class will also contain Lua API functions for the menu. Toolbar, Layers, CliCommand, and CadMdiChild already exist in *lcUI/widgets* however they will be updated for the GUI initialization and Lua API functions will be added. The connection of menus and the toolbar will be done via lua scripts using the new API.

Customizable Toolbar :-

The current toolbar has a fixed number of icons for operations, providing no customizability to the user. Some of the more specific create operations are not available in the toolbar and need to be accessed through the menu.



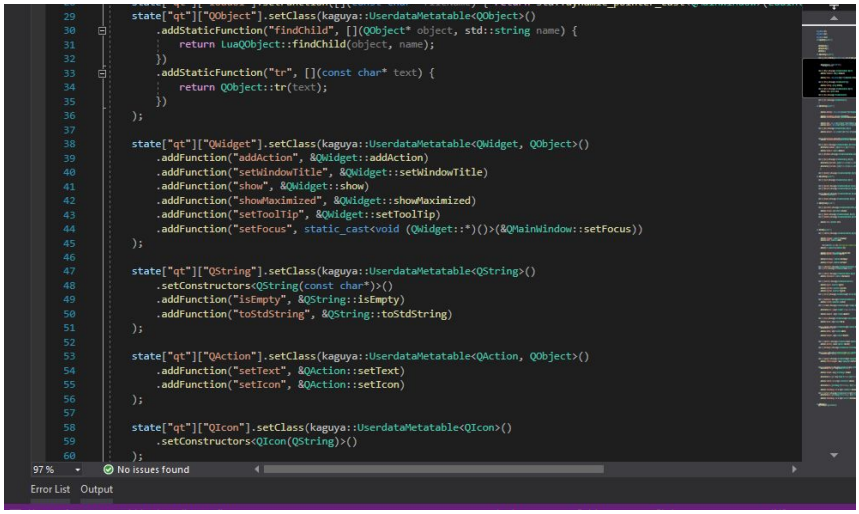
A customize icon would be added and clicking on this icon would either make the toolbar customizable, that is the user would be able to directly select and drag icons to the proper location, or as an alternative approach another customize widget would open which would allow the user to reorder and customize the icons. Whichever option would turn out to be more feasible would be selected.

This will include a list of all operations with checkboxes next to them so that the user can select which create operations the user wishes to have in the toolbar. An icon space for the most recently used operation will also be added.

Some of the create operations would require icons so these would either be taken from LibreCAD 2 or made in some photo editing application(most probably GIMP).

Remove Qt bindings to Lua (qtbridge.cpp) :-

The qtbridge.cpp file contains the Qt bindings to Lua that allows the creation and initialization of the Qt and custom widgets in the Lua scripts. Making changes to the GUI or adding new elements often requires checking this file to see which functions are available, and whether a new function needs to be added to it. Being 300 lines long, this file is also hard to maintain.



```
state["qt"]["QObject"].setClass(kaguya::UserdataMetatable<QObject>()
    .addStaticFunction("findChild", [](QObject* object, std::string name) {
        return LuaQObject::findChild(object, name);
    })
    .addStaticFunction("tr", [](const char* text) {
        return QObject::tr(text);
    })
);

state["qt"]["QWidget"].setClass(kaguya::UserdataMetatable<QWidget, QObject>()
    .addFunction("addAction", &QWidget::addAction)
    .addFunction("setWindowTitle", &QWidget::setWindowTitle)
    .addFunction("show", &QWidget::show)
    .addFunction("showMaximized", &QWidget::showMaximized)
    .addFunction("setToolTip", &QWidget::setToolTip)
    .addFunction("setFocus", static_cast<void (QWidget::*)()>(&QMainWindow::setFocus))
);

state["qt"]["QString"].setClass(kaguya::UserdataMetatable<QString>()
    .setConstructors<QString(const char*)>()
    .addFunction("isEmpty", &QString::isEmpty)
    .addFunction("toStdString", &QString::toStdString)
);

state["qt"]["QAction"].setClass(kaguya::UserdataMetatable<QAction, QObject>()
    .addFunction("setText", &QAction::setText)
    .addFunction("setIcon", &QAction::setIcon)
);

state["qt"]["QIcon"].setClass(kaguya::UserdataMetatable<QIcon>()
    .setConstructors<QIcon(QString)>()
);
```

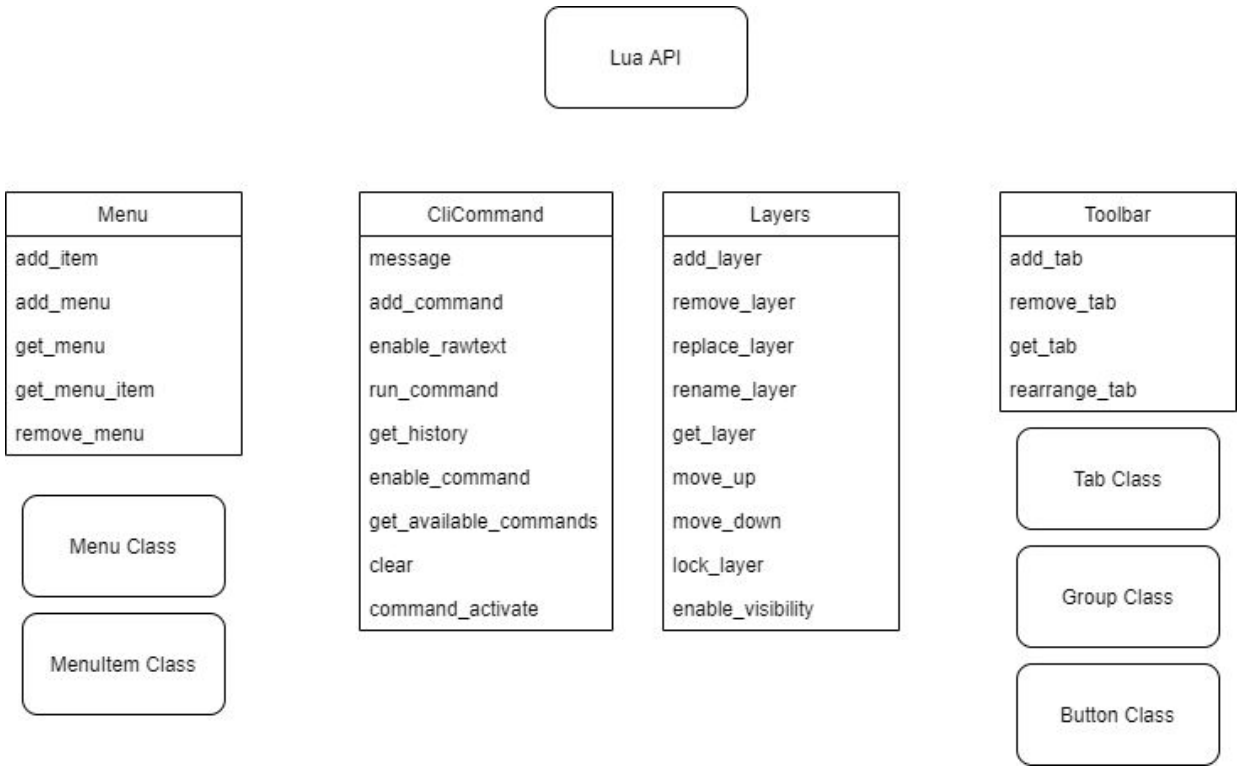

This file will be removed and in its place a bridge for the Lua GUI API will be added, which will expose the Lua API functions and will not require a lot of maintenance as the API will not undergo as many changes as the qtbridge did. Presence of a documentation for the API will also make it not necessary for the users to need to have a look at this file.

Lua GUI API :-

A new Lua GUI API will be implemented to allow the user to interact with the GUI and create GUI elements in the interface.

This API will be much easier for the user to use and would also not require the user to know Qt as was previously required. The API has been designed with extensibility in mind and provides enough functions for most needs of the user.

The MainWindow, Layers, CliCommand and Toolbar classes will be extended to contain the required functions, and these functions will then be exposed to Lua through the Lua GUI bridge.



MenuItem class is for the items in a menu(Open, Save as etc.) which are contained in a Menu class (like File, Create etc). The specification for this class is shown in the diagram below.

Menu API

MenuItem
+ callbacks: list<LuaRef&>
+ item_name: string
+ position: int (default - last position in menu)
+ parent_menu: Menu&
+ constructor(name, LuaRef& cb, Menu&)
+ enable(bool): void
+ add_callback(LuaRef& callback): void
+ change_menu(Menu& new_menu): void
+ set_position(int new_position): void
+ get_position(): int
+ change_name(string name): void

MenuAPI
add_item(item_name, menu_name, callback)
add_menu(menu_name, position=last)
get_menu(string menu_name) get_menu(int position)
get_menu_item(item_name, menu_name)
get_menu_item(int item_pos, menu_name)
remove_menu(string menu_name) remove_menu(Menu& menu)

Menu
+ items: list<MenuItem>
+ menu_name: string
+ position: int
+ constructor(string name, int pos=last)
+ enable(bool): void
+ add_item(item_name,callback): MenuItem& + add_item(MenuItem item): MenuItem&
+ change_item_position(int old,int new): void + change_item(MenuItem&, int new): void
+ set_menu_position(int new_pos): void
+ get_menu_position(): int
+ remove_item(string item_name) : void + remove_item(MenuItem& item) : void
+ change_name(string name): void

Toolbar API includes Button, Group and Tab classes which would allow the user to add tabs to the Toolbar. This API would be used to create the Quick Access Tab in the toolbar during program startup, and plugins using this API could easily add more tabs.

Toolbar API

Tab
+ groups: list<Group>
+ tab_name: string
+ constructor(string name)
+ enable(bool): void
+ add_group(string group_name): Group& + add_group(Group group): Group&
+ get_group(string group_name): Group&
+ get_groups(): list<Group&>
+ remove_group(string group_name): void
+ change_name(string name): void

Button
+ callbacks: list<LuaRef&>
+ button_name: string
+ constructor(name, icon_link, Group&)
+ enable(bool): void
+ add_callback(LuaRef&): void
+ change_icon(string icon_link) : void
+ set_tooltip(string tip) : void
+ change_name(string name): void

Group
+ buttons: list<Button>
+ group_name: string
+ constructor(string name, Tab&)
+ enable(bool): void
+ add_button(button_name, cb,icon): Button& + add_button(Button button): Button&
+ get_button(string button_name): Button&
+ get_buttons(): list<Button&>
+ remove_button(string button_name): void
+ change_name(string name): void

The CliCommand and Layer API consist of functions which allow the user to interact with these widgets.

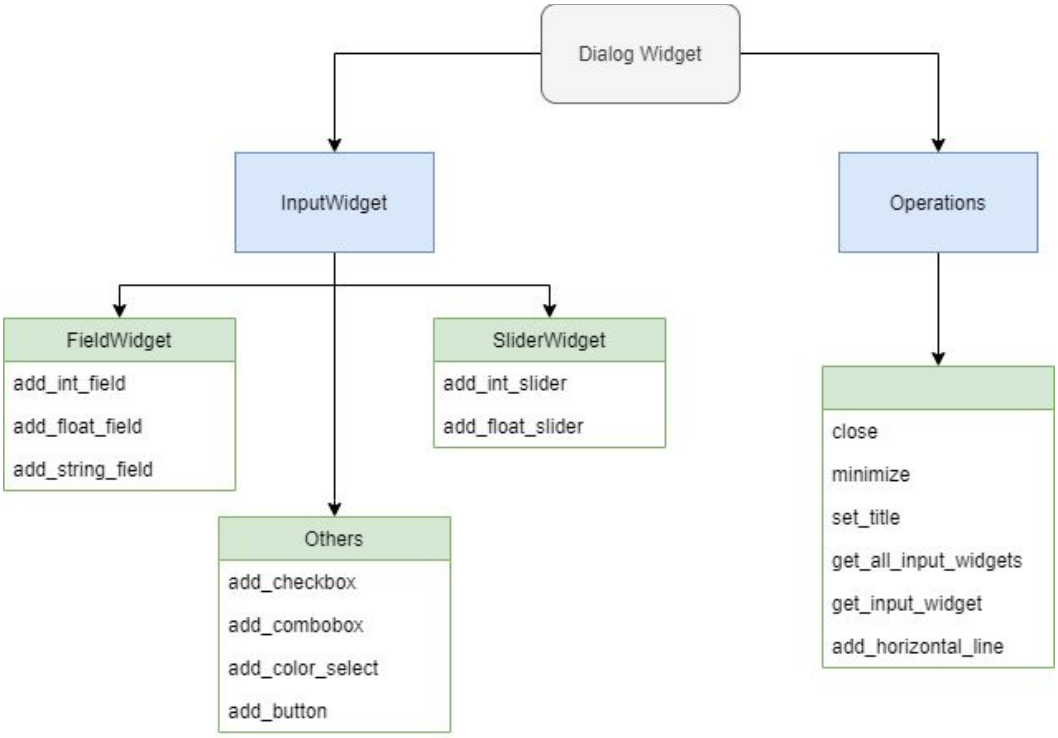
CliCommand
message(string message)
add_command(string command_name, LuaRef callback)
enable_rawtext(bool enable)
run_command(string command_name)
get_history (int last_n_elements=all)
enable_command(string command_name, bool enable)
get_available_commands()
clear()

Layers
add_layer(Layer layer)
remove_layer(int index) remove_layer(Layer& layer)
replace_layer(int index, Layer& newLayer)
rename_layer(int index, string new_name) rename_layer(string old_name, string new_name)
get_layer(int index) get_layer(string layer_name)
move_up(Layer& layer)
move_down(Layer& layer)
lock_layer(Layer& layer / string layer_name / int layer_index)
enable_visibility(int layer_index / string layer_name)

The layer API can be used to do various operations to interact with the layer widget and the layers for the document. The Layer class is already exposed to Lua. The CliCommand API will replace the functions in the *clicommand.lua* along with a few new functions. This will allow the user to display messages and get user input through the command console.

Apart from these functions, the Lua GUI API will also include custom widget classes that will make it easier for the user to create new elements in the interface. Among these are the DialogWidget and Toolbar API mentioned above which would allow users to create their own tab using plugins.

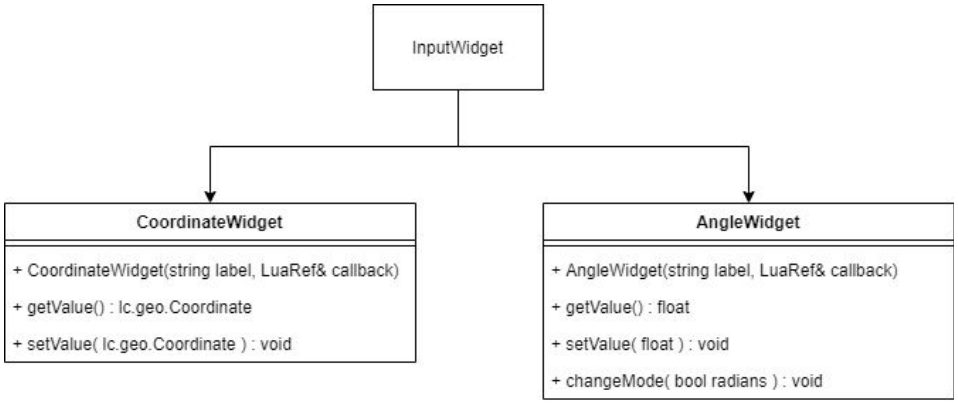
Dialog Widget: -

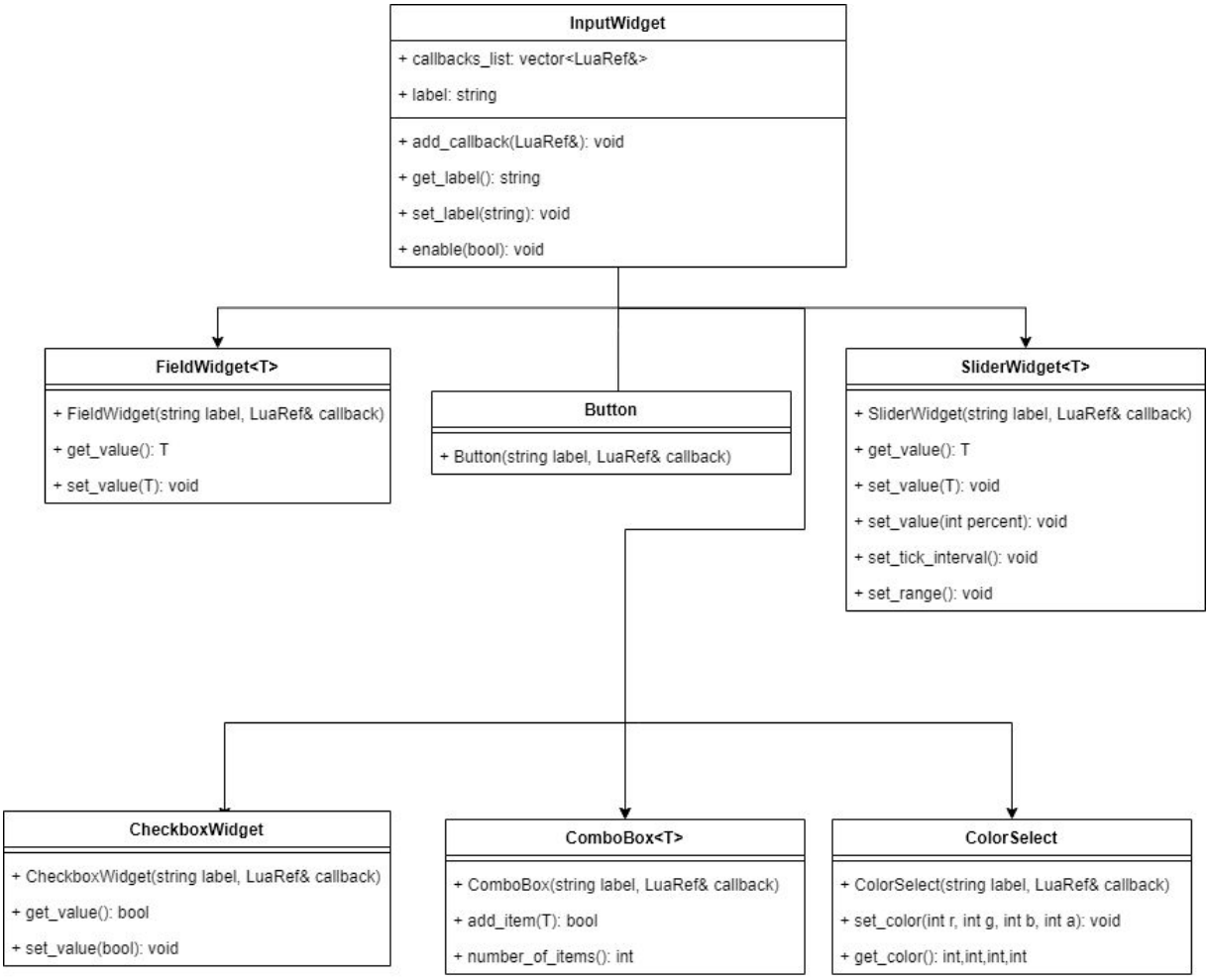


The InputWidget parent class would hold a list of callbacks so that each GUI element can be associated with more than one callback.

The Input widgets would consist of a Field, Button, Slider, Checkbox, ComboBox and ColorSelect.

Special widgets like CoordinateWidget and AngleWidget would also be added which would contain extra features (like switching between radians and angle by clicking on a switch).Upon further discussion before implementation, more widgets may be added depending on the requirement(eg. a label widget).



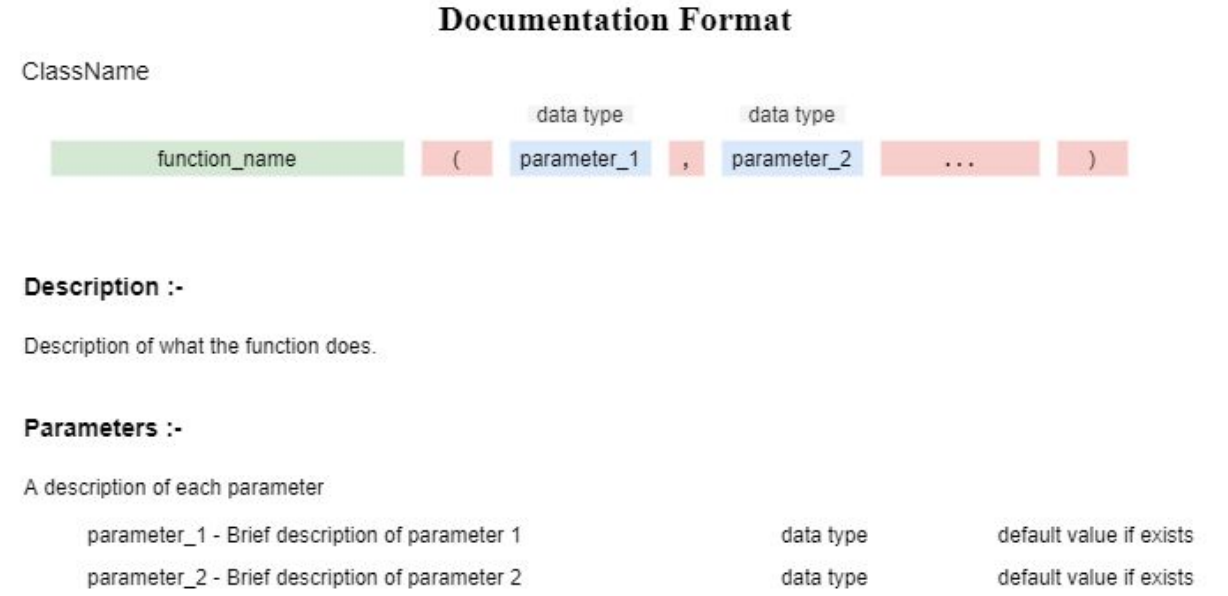


Aside from making it easier for the user to create GUI widgets without having any knowledge of Qt, the plugins would not be required to be updated whenever UI style changes are made to LibreCAD 3.

Documentation for the Lua GUI API:-

Documentation and tutorials for the Lua GUI API will be added to the LibreCAD wiki.

Documentation format:-



Each function will be documented in the format specified above, explaining the various parameters and return type. A very simple example will be given for most functions so that users have a good idea on how to use that particular function.

A link to the class containing the respective C++ function will be also linked for more advanced users to have a look at should they want to.

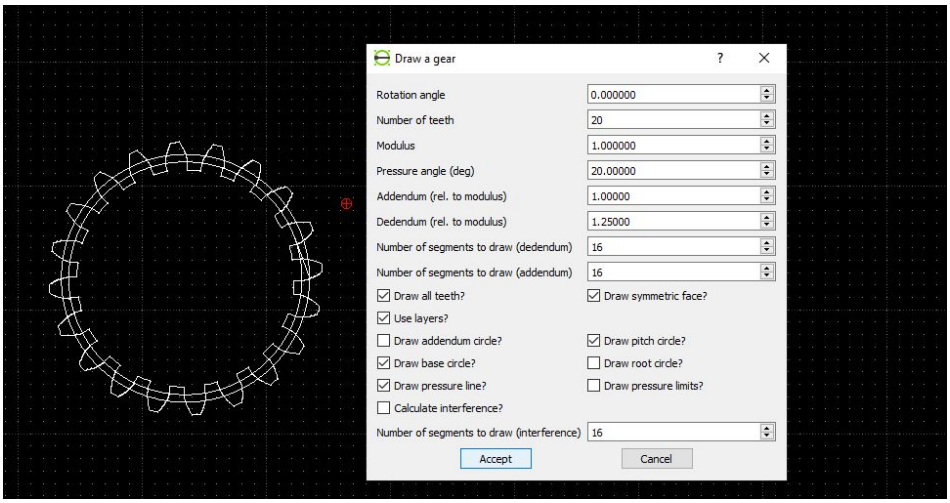
A diagram of the widget will also be shown along with the available member functions for the widget classes.

Tutorials for Lua GUI API :-

Step by step tutorials with clear diagrams and well written, commented code will be written on using the Lua GUI API to perform some useful tasks and on creating some simple widgets.

This will make it easy for the user to get started on using the Lua GUI API and give a clear idea on how the API is to be used.

A tutorial for making the gear plugin from LibreCAD 2 will be made to show users how to use the Lua GUI API to create custom widgets for plugins.



Unit tests for LibreCAD 3 GUI:-

Unit tests will be added for the LibreCAD 3 GUI. I will try to add these along with the implementation to ensure better and comprehensive unit tests.

The current code coverage can be seen here.

LibreCAD makes use of gtest for unit testing. The unit tests folder contains tests for different modules like lckernel, lcviewernoqt, rendering and the UI.

I shall be focusing on the ui module for my proposal.

Google Test (gtest):-

Google Test is a unit testing library for the C++ programming language. The library is released under the BSD 3-clause license. It can be compiled for a variety of POSIX and Windows platforms, allowing unit-testing C++ with minimal source modification. The tests themselves could be run one at a time, or even be called to run all at once. This makes the debugging process very specific.

Update and add tests for GUI widgets :-

Updating and adding more unit tests to existing ones like commandline, toolbar and layers. Adding unit tests for mainwindow.

Adding unit tests for Lua GUI API :-

Unit tests for all functions of the Lua GUI API would be added to ensure that changes to the API later would not break any of the existing functionality. Files like menuapi_test.cpp, clicommandapi_test.cpp, layersapi_test.cpp, toolbarapi_test.cpp, dialogwidgetapi_test.cpp, dockablewidgetapi_test.cpp would be added.

Brief Timeline

- (Phase 0) Till May 4: Pre-GSoC Period
- (Phase 1) 4 May - 1 June: Community Bonding Period
- (Phase 2) 01 June - 29 June: Coding Period 1
- (Phase 3) 29 June - 03 July: Phase 1 Evaluations
- (Phase 4) 03 July - 27 July: Coding Period 2
- (Phase 5) 27 July - 31 July: Phase 2 Evaluations
- (Phase 6) 31 July - 24 August: Coding Period 3
- (Phase 7) 24 August - 31 August: Final work submission

Detailed Project Timeline

Phase 0 [Pre-GSoC Period]

5 Weeks (31 March - 4 May)

My exams would possibly either be in the community bonding period or in the beginning of the first phase. I would use this period to finish preparing for my exams so that I could give enough hours to GSoC even during my exams. (This is just preparing for the worst case, I'll most probably cover up the loss in the community bonding period).

I'll also work on preparing and planning for the tasks for the first coding period trying out a few sample implementations on a test branch. I'll also continue fixing bugs and working on minor features like I have been in the past few months.

Phase 1 [Community Bonding Period]

4 Weeks (4 May - 1 June)

I will most likely have my end semester exams for around 10 days in this period. I will still be available to chat on ZulipChat and start coding for my first phase task.

In case my exams are to be held during the Coding Period 1, I would immediately start coding and treat the community bonding period as my coding period 1.

I shall discuss the implementation of the first phase that is the rewriting of the GUI in C++ with my mentors.

Phase 2 [Coding Period 1]

2 Weeks (1 June – 14 June)

I will implement the rewriting of the GUI from Lua to C++ in this period. This shall include writing of the MainWindow class, adding the necessary functions and attributes to the MainWindow class and getting it to run.

Addition of buttons and connecting these buttons to their appropriate slots will be done for the buttons in the menu and the toolbar class. Layers, CliCommand and Toolbar will be updated to ensure that all the existing operations function normally. I shall update the existing Lua scripts in lcUILua/ui to function properly.

This step being a major refractor will most likely break a couple of things, so the entire first period is given to this task so I could fix any unforeseen bugs that might come up.

2 Week (14 June - 29 June)

LuaAPIBridge will be added and the necessary functions to transfer all the functionality of the existing Lua UI scripts will be implemented in the classes and added to the bridge.

Lua UI scripts will be updated to use these new functions while the unnecessary and redundant Lua scripts will now be removed.

I shall also add unit tests for the C++ part of the GUI.

Phase 3 [GSoC Phase 1 Evaluations]

This period will be used to write a detailed report on the work done in Coding Period 1. All the work done will be sent as PRs.

Deliverables

- LibreCAD working with GUI code now shifted to C++.
- MainWindow class along with updated Layer, CliCommand and Toolbar class
- Approx. 90% Unit Test Coverage for the GUI part
- Partly implemented Lua GUI API

Phase 4 [Coding Period 2]

3 Weeks (03 July - 24 July)

This period will be spent on adding the remaining Lua GUI API functions for the GUI widget management and most functions for the widget creation. I shall also begin work on the customizable toolbar in this period.

Implement all the remaining functions as specified in the widget management specification for the Menu, Layers, CliCommand and Toolbar in their respective C++ classes and add these to the Lua GUI API Bridge. Add documentation for the added functions in the LibreCAD wiki. Menu and toolbar button connection will be shifted to the GUI API.

1 Week (24 July - 31 July)

The work from the previous two weeks will be finished this week. Unit tests for the Lua API will be added.

Phase 5 [Phase 2 Evaluations]

A detailed report on the working of the second coding period will be written during this time.

Deliverables

- Functions for interacting and adding to the existing GUI elements like menu, layer, clicommand and toolbar will be available through the Lua GUI API.
- Unit tests for the Lua GUI API.

Phase 6 [Coding Period 3]

2 Weeks (31 July - 14 August)

Finish up Lua GUI API and add tutorials for the Lua GUI API to the LibreCAD wiki. Also add the remaining documentation for the Lua GUI API.

Finish the customizable toolbar.

qtbridge will be removed, with the Lua GUI Bridge now fully functional.

2 Weeks (14 August - 31 August)

Finishing up, any leftover unit tests (so that code coverage is more than 90% for the UI and API), documentation updates, bug fixes, and other fixations will be completed.

This will also be used as a buffer period in case of any lag in the schedule.

Phase 7 [Final Evaluation]

All documentation, improvements, GUI updates and unit tests will be provided in PRs.

All the deliverables promised for GSoC will be provided by this stage.

Time Availability:-

I can give around 6-7 hours daily (even more if required) and have a fairly good amount of flexibility in increasing my timings should it be required.

As mentioned in the project timeline, my end semester exams which will probably last a week or two would be held either during the community bonding period or in the first coding period. I shall know which soon so I can start coding earlier if required to offset any loss of time during the first coding period.

Even after college reopens I can give 40+ hours a week (I can easily avail some holidays if required to keep up with my deadlines should I be falling behind).

Why LibreCAD ?:-

I tried my hand at a few other organizations before LibreCAD but the helpful nature of the community at LibreCAD and their prompt and helpful reply to my every query made me decide to choose LibreCAD to be the organization I wanted to contribute to and I'm sure I've made the right decision.

What also appeals to me is that LibreCAD 3 is a relatively newer project and I could feel my contributions impacting the project, something which is hard to feel with other older projects with massive codebases.

This also means that LibreCAD 3 has plenty of things to work on and I've found contributing to LibreCAD 3 to be a very fun and satisfying activity.

My experience with LibreCAD has been very enriching and fun so I will **not** be applying to any other organization except LibreCAD for GSoC.

I plan to continue contributing to LibreCAD well after GSoC is over and hope to stay at least till the day LibreCAD 3 is ready to be released to the public.

Why Me ?

I came across LibreCAD in the starting of January and have been contributing actively to it since then. I have also been in active communication with the mentors and I now have a good idea at what times they are active.

Before contributing I did not know Qt and Lua but over the course of these three months I have learnt enough to take on this task.

I believe that I have enough command over the prerequisites of the project to be confident in my ability to complete the task with the help of my mentors..

About Me :-

I am currently a second year undergraduate pursuing Computer Engineering at Army Institute Of Technology, Pune.

I am a member of the Open Source Software club of my college and have been interested in programming and computers since my childhood.

My contributions so far :-

- **Pull Requests Merged**

  Major PRs (> 200 lines of code) :-

  - **Fixed LWPolyline and Connected buttons under Create and Draw menu #254**
  - **Added custom titlebar to widget #258**
  - **Added Circle tangent to three other circles and tangent to two circle feature #270**

  Minor PRs :-

  - **Added Qt5 svg options to CMakeLists of lcUI #251**
  - **Removed settings menu bar #259**
  - **Added tooltips to buttons in quick access tab #261**
  - **3 minor bug fixes #265**
  - **Fixed 2 command line bugs #266**
  - **Delete button gets disabled when only one layer is left #267**
  - **Better display of temporary entities for three tan circle operation #272**
  - **Bug fixes for temporary circle creation in 3 tan function #275**

- **Connected select options #279**
- **Add entity constraint checks #283**
- **Ellipse operations feature #285**

References :-

A big thank you to the mentors for taking the time to reply to my various queries in the making of this proposal.

Draw.io was used to create all the diagrams in this proposal.

The timeline and methods mentioned in the proposal are tentative, and a more robust discussion will be done when beginning each part of the task.