Code Workstation

Design Document

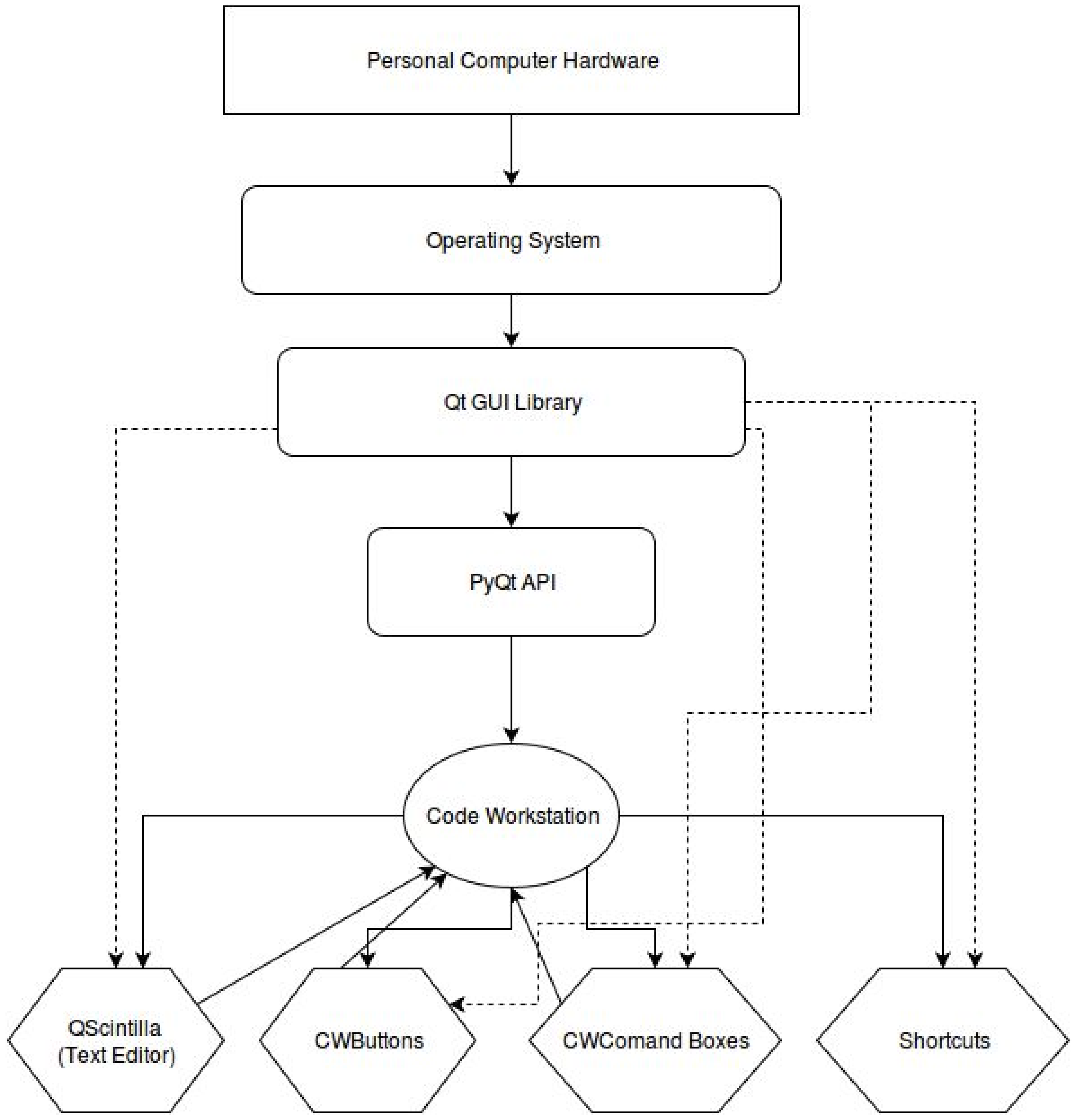
https://github.com/lrickard/CodeWorkstation/wiki

Lukas Rickard - Project Leader - l_rickard@u.pacific.edu

Jake MacMillan - Integrations Lead - j_macmillan1@u.pacific.edu

Last Revision: 5/9/2016

System Architecture



Personal Computer Hardware - Modern personal computer with i386 or x86 architecture
Operating System - Debian at first but hopefully available on all common operating systems
Qt GUI Library - Provides event handling, interface with OS, GUI elements, and windowing
PyQt API - Provides interface with Qt libraries through python
Code Workstations - Initialized window, loads settings, saves settings, and runs commands
QScintilla - Text editor build into Qt will allow users to edit text
CWButtons - Extending Qt Buttons. Button press events will run flag a command to be run by Code Workstation
CWCommand Boxes - Extending Qt Plain Text Edit boxes. Command Boxes display output from button commands and can be set to display output from their own commands.
Shortcuts - Activate existing Buttons and actions through the PyQt API

System Requirements

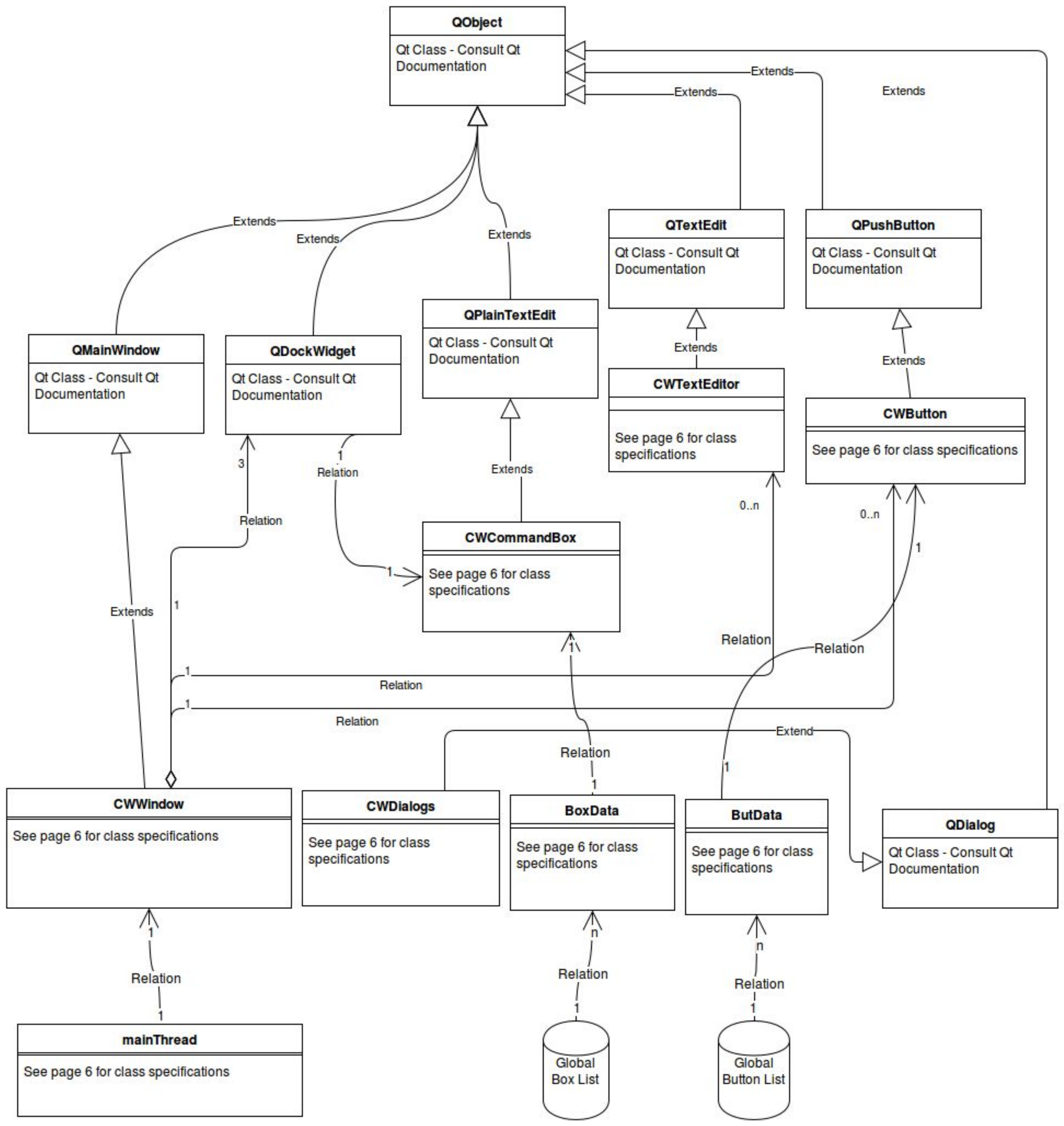
Hardware Requirements:
We suspect a single core 1 GHz CPU and 512MB of RAM will enable responsive use.
Software Requirements:
Python 3, Qt5, and PyQt5.
System Requirements:
32 or 64 bit operating system, Debian Linux and OSX are what we are developing for.

External Interfaces

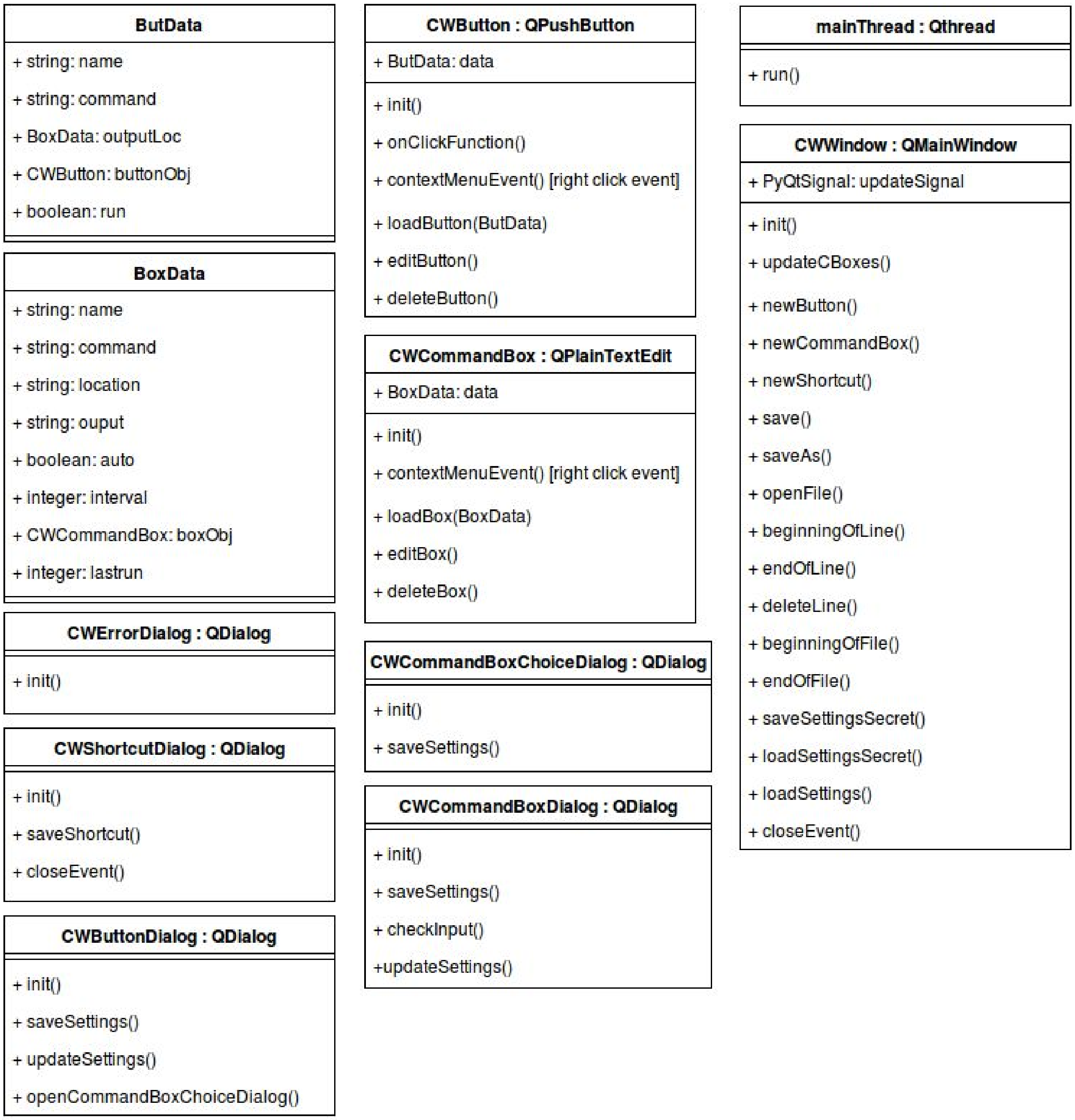
Qt - Will be used for generating windows, interfaces, and handling events.
http://doc.qt.io/
PyQt - A Python API to interface with Qt Libraries
https://riverbankcomputing.com/software/pyqt/intro

Software Design

Class Diagrams (General):


QObject
Qt Class - Consult Qt Documentation
Extends
Extends
Extends
Extends
Extends
Extends
QMainWindow
Qt Class - Consult Qt Documentation
QDockWidget
Qt Class - Consult Qt Documentation
QPlainTextEdit
Qt Class - Consult Qt Documentation
QTextEdit
Qt Class - Consult Qt Documentation
QPushButton
Qt Class - Consult Qt Documentation
Extends
CWTextEditor
See page 6 for class specifications
Extends
CWButton
See page 6 for class specifications
3
Relation
1
Relation
Extends
CWCommandBox
See page 6 for class specifications
0..n
0..n
1
Extends
1
Relation
Relation
1
Relation
1
Relation
Extend
Relation
1
1
CWWindow
See page 6 for class specifications
CWDialogs
See page 6 for class specifications
BoxData
See page 6 for class specifications
ButData
See page 6 for class specifications
QDialog
Qt Class - Consult Qt Documentation
1
Relation
1
mainThread
See page 6 for class specifications
n
Relation
1
Global Box List
n
Relation
1
Global Button List

Class Specifications:

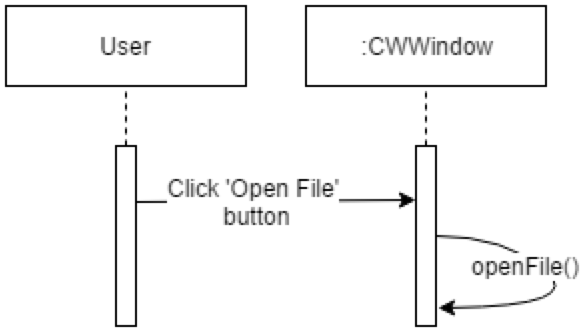


Additional Class information:

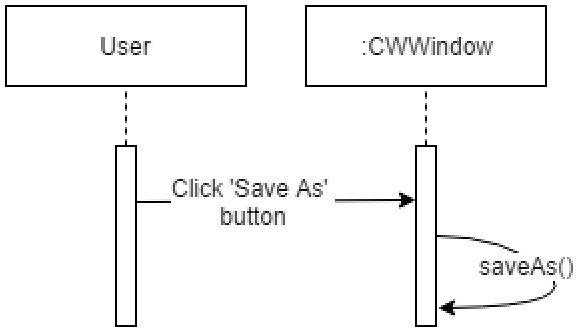
All CW classes which inherit from a Qt class use a user interface class created with Pyuic to set up their user interface. Thus, for every CW class there is also a correspinding UI.py file which is used solely within the corresponding class.
While this is the way we decided to create our interfaces, it is not necessary and we also implement some UI changes manually. Tracking all user interface elements in this document would be intractable, so we are just documenting that they exist and are created with Qt Designer and set up using python classes created automatically with Pyuic.

Interaction Diagrams:

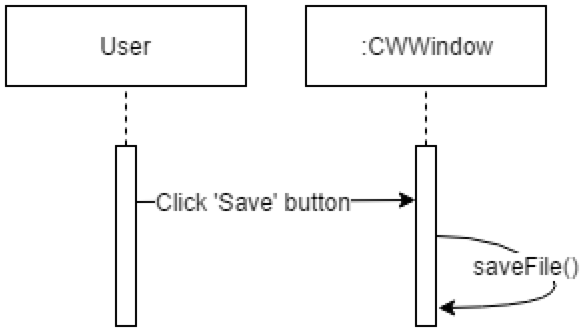
Open a file:



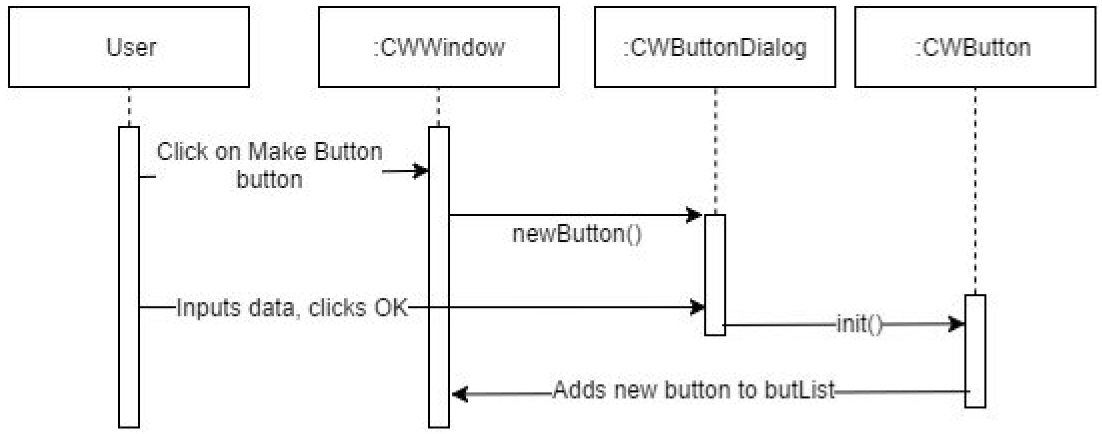
Save a new file:



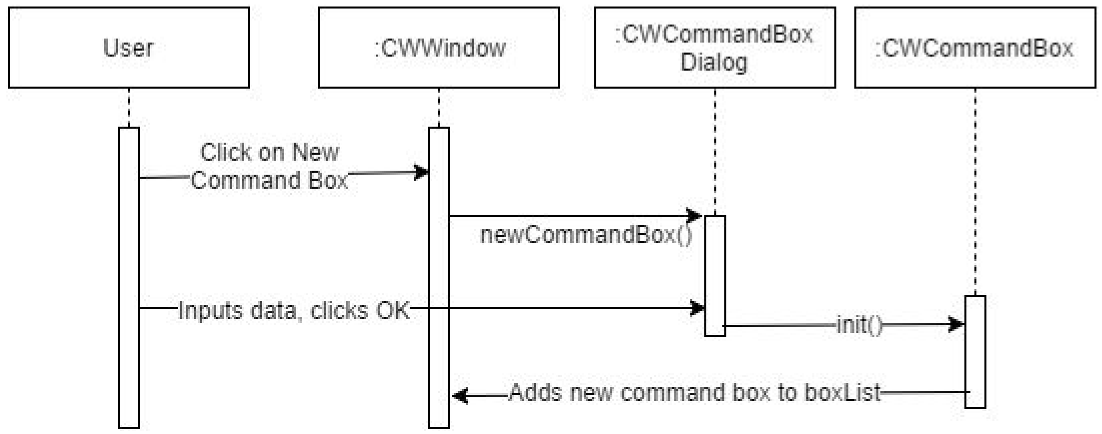
Save a previously saved file:



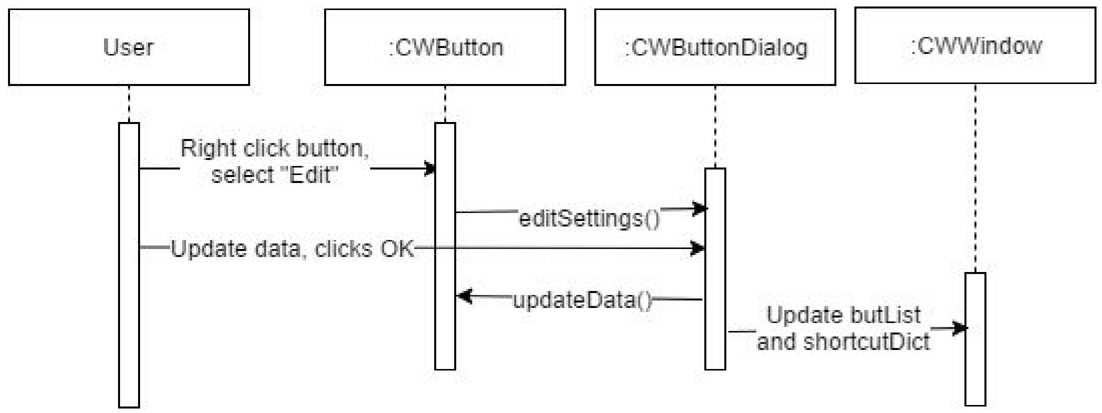
Create a new Button:



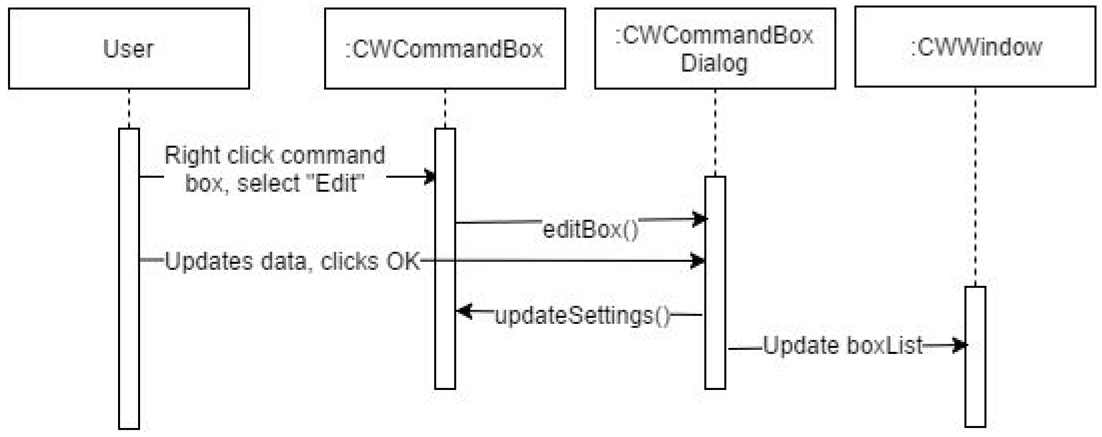
Create a new Command Box:



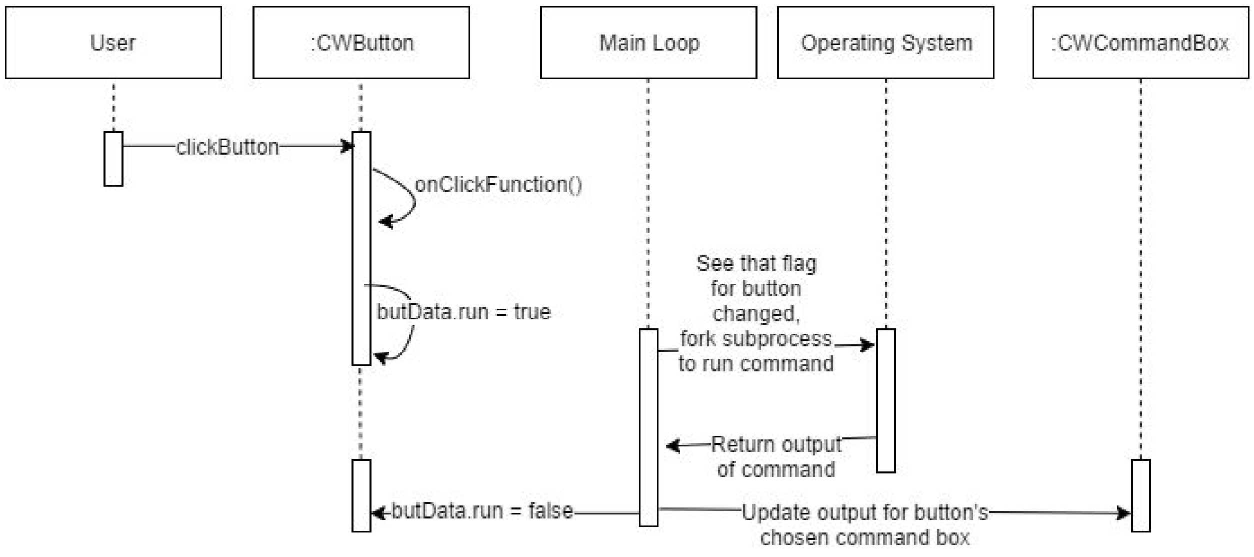
Edit a Button:



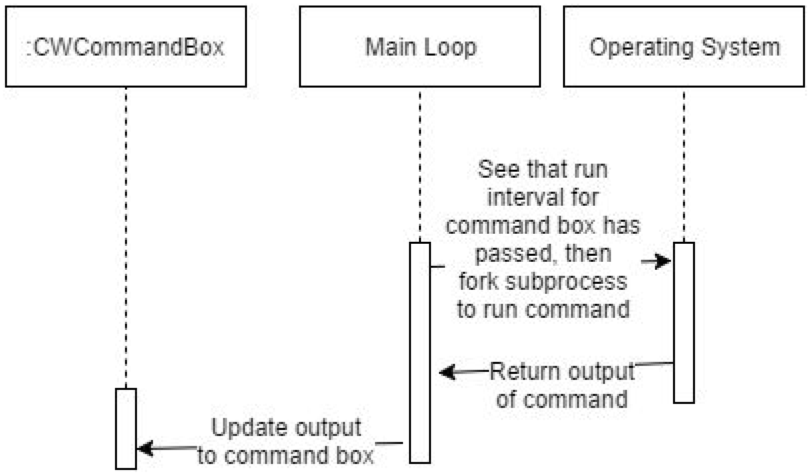
Edit a Command Box:



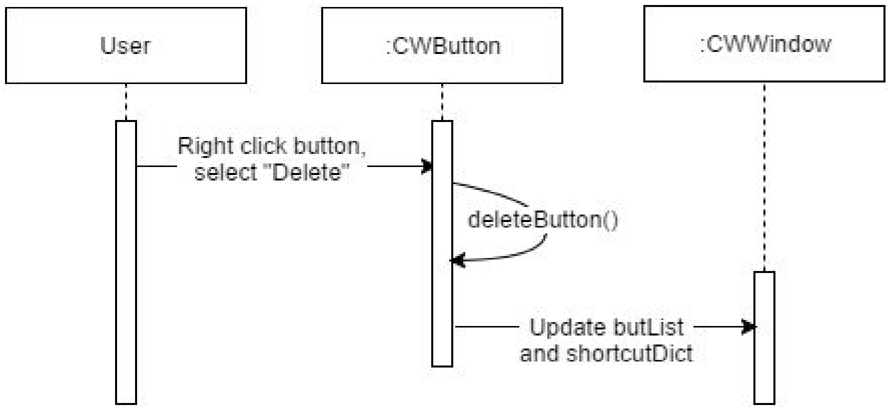
Run a command with a Button:



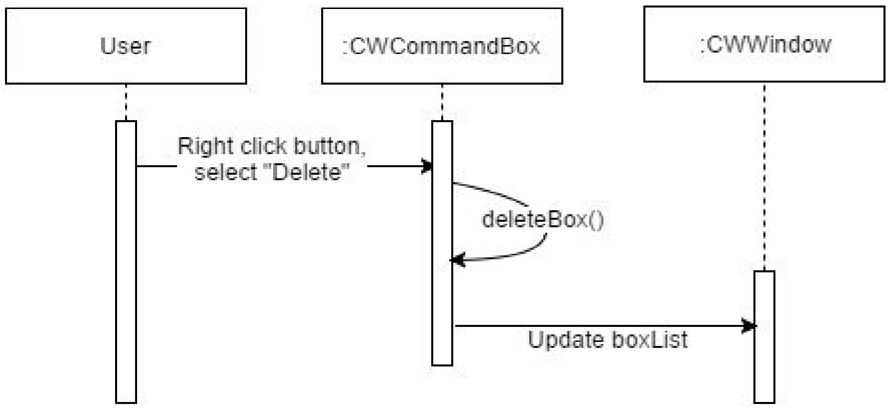
Run a command with a Command Box:


:CWCommandBox
Main Loop
Operating System
See that run interval for command box has passed, then fork subprocess to run command
Return output of command
Update output to command box

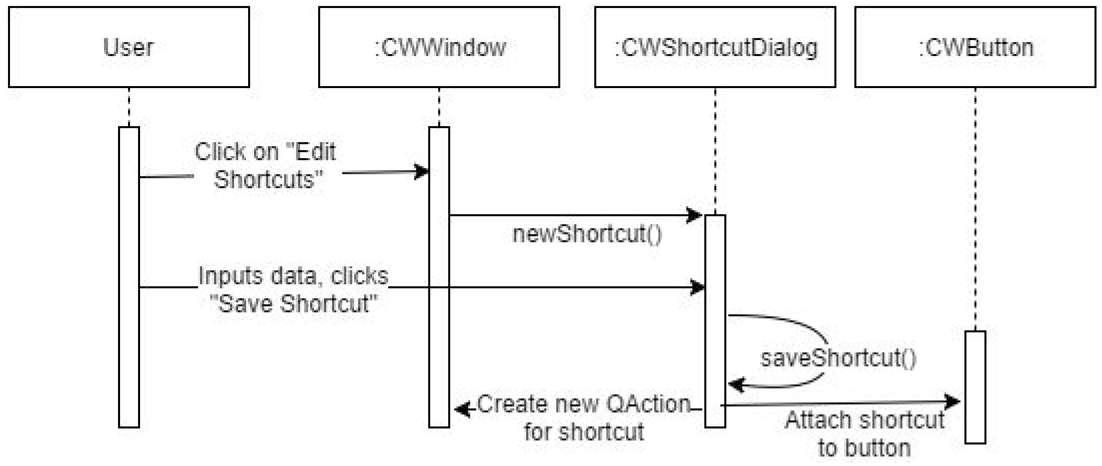
Delete a Button:


User
:CWButton
:CWWindow
Right click button, select "Delete"
deleteButton()
Update butList and shortcutDict

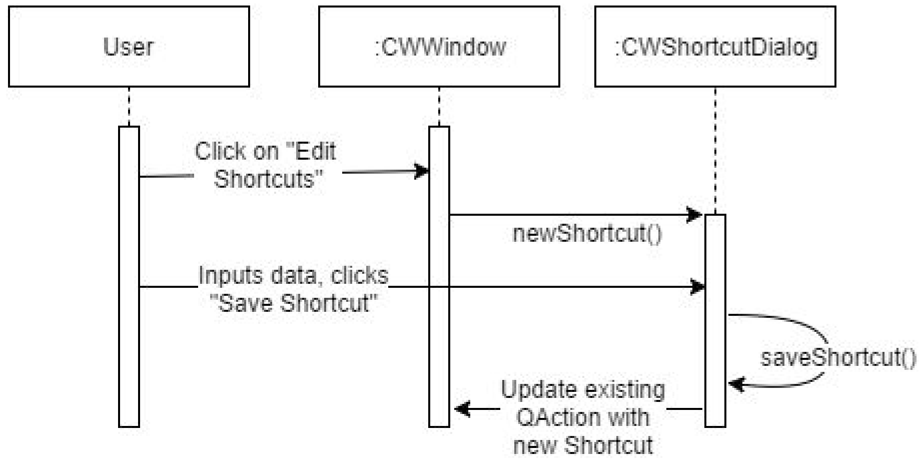
Delete a Command Box:


User
:CWCommandBox
:CWWindow
Right click button, select "Delete"
deleteBox()
Update boxList

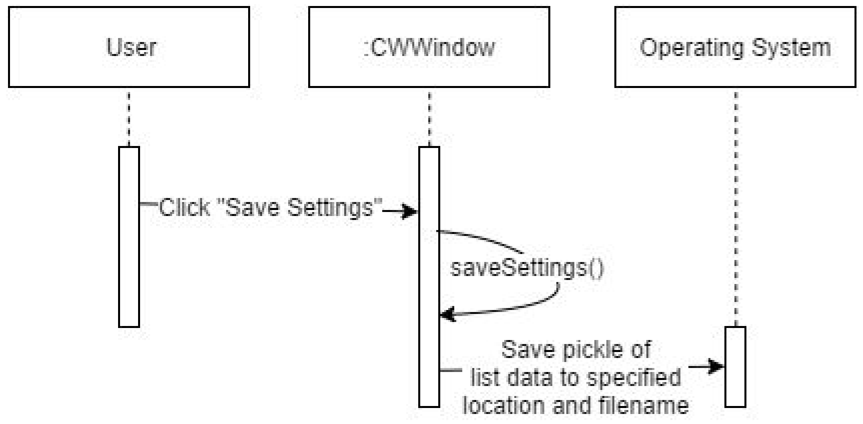
Create a shortcut for a Button:


User
:CWWindow
:CWShortcutDialog
:CWButton
Click on "Edit Shortcuts"
newShortcut()
Inputs data, clicks "Save Shortcut"
saveShortcut()
Create new QAction for shortcut
Attach shortcut to button

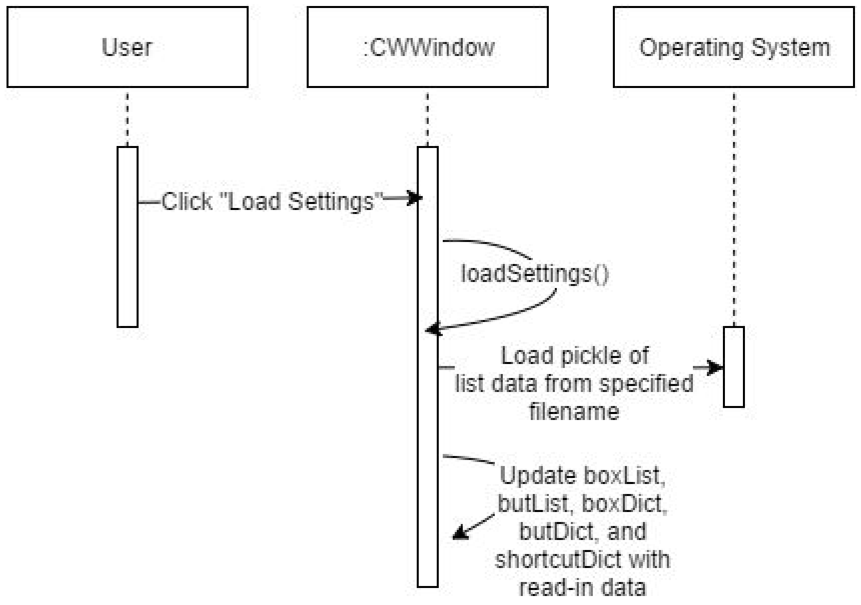
Create a shortcut for Save, Save as, or Open:


User
:CWWindow
:CWShortcutDialog
Click on "Edit Shortcuts"
newShortcut()
Inputs data, clicks "Save Shortcut"
saveShortcut()
Update existing QAction with new Shortcut

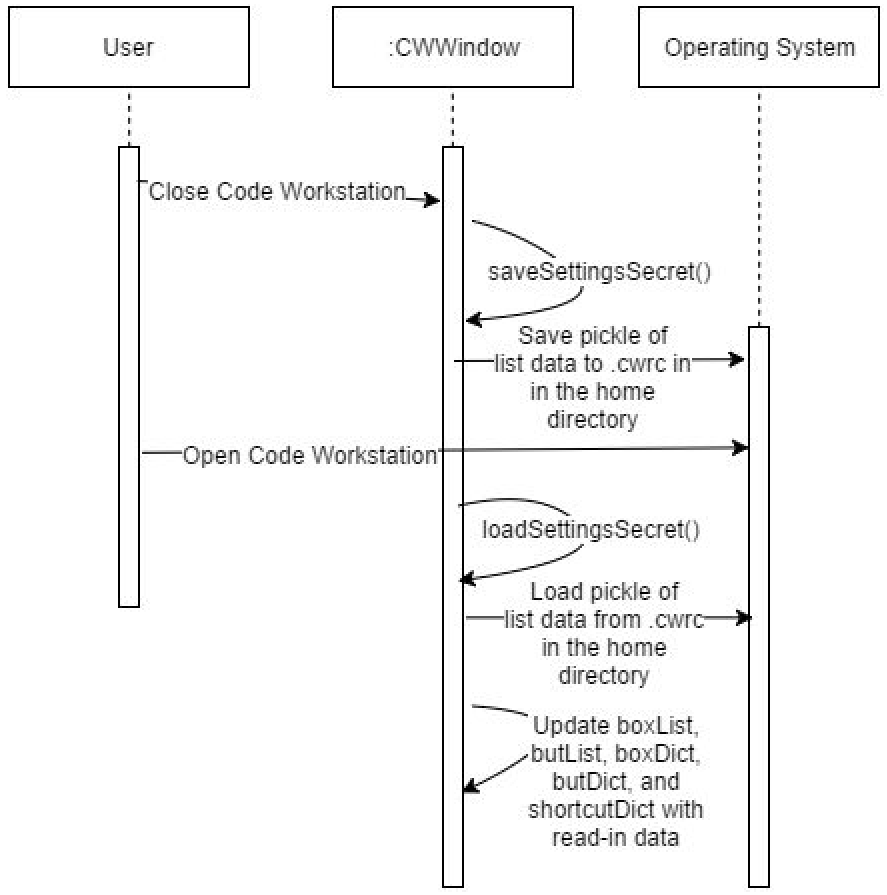
Save Settings:



Load Settings:


User
:CWWindow
Operating System
Click "Load Settings"
loadSettings()
Load pickle of list data from specified filename
Update boxList, butList, boxDict, butDict, and shortcutDict with read-in data

Close and Open Code Workstation:


User
:CWWindow
Operating System
Close Code Workstation
saveSettingsSecret()
Save pickle of list data to .cwrc in in the home directory
Open Code Workstation
loadSettingsSecret()
Load pickle of list data from .cwrc in the home directory
Update boxList, butList, boxDict, butDict, and shortcutDict with read-in data

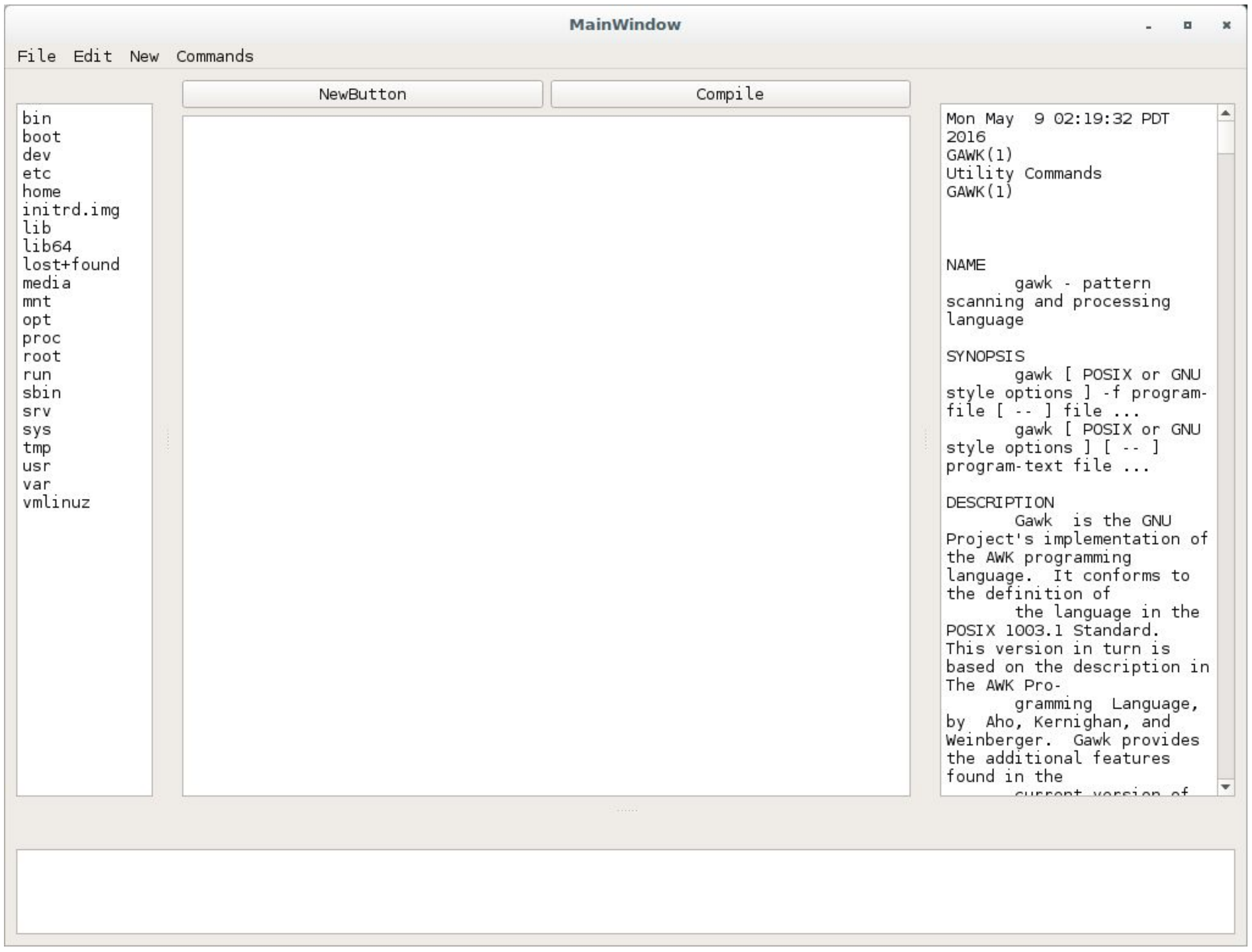
Design Considerations:

Upon consideration of performance and scalability, we decided to integrate all our features and event handling by extending Qt classes and rewriting their primary functions. This seemed to be the most efficient way to do event handling and helps to keep things organized.

In order to minimize threading and increase caching we decided to store all of our user interface and command data in separate global lists.

The use of an update signal was necessary because Qt events cannot spawn child threads. We decided after dealing with this that we would have a main loop which would run all the commands on a thread completely independant from Qt threads. Allowing Qt to update with the update signal.

User Interface Design



```
bin
boot
dev
etc
home
initrd.img
lib
lib64
lost+found
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
vmlinuz
```

```
Mon May  9 02:19:32 PDT
2016
GAWK(1)
Utility Commands
GAWK(1)



NAME
       gawk - pattern
scanning and processing
language

SYNOPSIS
       gawk [ POSIX or GNU
style options ] -f program-
file [ -- ] file ...
       gawk [ POSIX or GNU
style options ] [ -- ]
program-text file ...

DESCRIPTION
       Gawk  is the GNU
Project's implementation of
the AWK programming
language.  It conforms to
the definition of
       the language in the
POSIX 1003.1 Standard.
This version in turn is
based on the description in
The AWK Pro-
       gramming  Language,
by  Aho, Kernighan, and
Weinberger.  Gawk provides
the additional features
found in the
       current version of
```

Buttons:

- Created with New->Button
- Left Click: This should run the command stored in the button
- Right Click: This should display options for editing and deleting

Text Editor:

- Right Click: This should expose a list of well know commands such as copy, paste, select all, etc.
- Shortcuts: While most linux systems have keyboard command shortcuts, we are have well known and used text interactions mappable to shortcuts.

Command Boxes:

- Created with New->Command Box
- Left Click: This should do nothing. The text should be scrollable.
- Right Click: This should display an edit and delete option.
- There are grabable areas between the Command Boxes and the text editor which can be used to resize the command boxes

File Dropdown:

- Click Save Settings: This should save the current commands and window settings to the default location
- Click Save File: This should save the currently editing file
- Click Save File As: This should launch a save dialog/file explorer(part of Qt) which will allow the user to save the text in the main text box anywhere on the system where they have permissions

Keyboard Shortcuts:

- Edit->Shortcuts shows the user a list of existing custom shortcuts and allows them to create new ones

Glossary of Terms

API - Application Programming Interfaces (APIs) allow a programmer to access functionality of a library or previously created code with a specifically exposed functions and variables.
Python - A high level interpreted scripting/programming language.
OS - Operating Systems (OSs) interact with hardware and firmware to form a base for other software to interact with hardware and display things to a user. Examples: Windows 7, Debian Linux, OSX
OSX - A Macintosh operating system (OS)
UI/GUI - User Interface or Graphical User interface
UML - Unified Modeling Language is used to create class diagrams in a consistent manner
Command - Throughout this document this generally refers to a word, phrase, or set of words and phrases which can be executed in an operating system’s shell/terminal.

References

Qt - http://www.qt.io/
PyQt - https://riverbankcomputing.com/software/pyqt/intro
Python - https://www.python.org/