# Qt Browser Manual

## Ben Becker (bbecker5)

## Rob Simari (rsimari)

*Note on Bookmarks and History/Auto Complete Feature:*
*These features use two txt files, visited.txt and bookmarks.txt, to store their items. Since Qt creates project builds in different directories depending on which operating system you use, you must edit the QString constant at the top of mainwindow.cpp & bookmarkdialog.cpp in order for these features to work. Since these are simple .txt files, you may edit them to add or remove history and bookmarks manually, using you favorite text editor.*

## Special Libraries

Since Qt stores its libraries in different places when using different operating systems, and since the Makefile generated by Qt Creator uses the location of these libraries, you will need Qt Creator to compile the project.

To install Qt Creator, go to qt.io in FIREFOX*→ Downloads → In-house deployment, private use or student use → No → Yes → Download.

* I have no idea why Chrome doesn't work

Installing Qt in this way will install the latest Qt binaries and version of Qt creator.

## Building Instructions

To build the Browser project, first open Qt Creator, and then open the Browser.pro project file. After the project has been loaded, click the green arrow in the bottom left corner (or press CTRL + R), and the project will automatically build and run (assuming all of the Qt libraries have been installed correctly).

## User Manual

Auto Complete Feature:

For increased browsing capabilities we created a feature that allows the browser to attempt to guess which address you are typing in the address bar and complete it automatically if you press the TAB key. This searches through your history for addresses containing the current typed word and will try to guess which address you are typing. To make sure that your session history gets written for future use, press CTRL + Q to close the browser.

HTTP Prefix Adder:

The browser automatically adds the “http://” prefix to any query that does not have one. This allows you to type websites such as “google.com” and have it resolve to “http://www.google.com”, similar to what would happen in a commercial browser.

Bookmarks:

Bookmarks allow you to save certain sites so you can easily access them from anywhere. To access your bookmarks, use the button labeled “B” on the top right. This will display all of your current bookmarks in a window. Click on a link to go to that site. If you want to add to your bookmarks use the shortcut CTRL + D and that will add the current page to your bookmarks. To edit your list of bookmarks, you can edit the bookmarks.txt file found in the Project directory.

Incognito Mode:

Incognito mode allows a user to browse sites without the sites being saved in the history. Any sites visited in incognito will not be able to be autocompleted with TAB. To toggle the incognito mode using the shortcut CTRL + I. If the browser is currently in incognito mode then the window will be titled “Browser (Incognito).

Invalid HTTP Address Entry:

If you enter an invalidly formatted HTTP address the browser will display custom html that says, “Invalid HTTP Address.” Note this is only done with the address format is invalid and not necessarily if the address is not found.

Short Cuts:

| | |
|---|---|
| CTRL + I | Toggle Incognito Mode |
| CTRL + Q | Exits the window |
| CTRL + R | Refresh Page |
| CTRL + L | Highlight the address bar |
| CTRL + T | New Tab |
| CTRL + Tab, CTRL + SHIFT+Tab | Switch Tab |
| CTRL + W | Delete Current Tab |
| CTRL + Left | Go Back to last page |
| CTRL + Right | Go Forward to next page |
| CTRL + D | Add current page to |

| | |
|---|---|
| | bookmarks |

Tabs:

Our browser supports multiple tabs, each with their own separate history. To create a new tab, simply press CTRL + T, and new tab displaying the homepage will pop up at the end of the tab window. To switch to another tab, click the individual tab, or press CTRL + Tab to shift one tab to the right or CTRL + SHIFT + Tab to shift one tab to the left. To delete a tab, press CTRL + W. If you only have one tab left and you close that tab, then the window will close as well. If you have multiple tabs open, you may press CTRL + Q to close the entire window.

Tab History:

Each tab you create has its own separate history that you are able to access through the back and forward button. Whenever you visit a new page, it is added to this temporary history. You can then use the back and forward buttons to move through this list of URLs. If you are in the middle of the list and click a different link, the front of your tab history is reset and your place tab history is now the end of the list. The tab history is implemented in the class HistStack using two stacks, front and back.

Other Buttons:

The “Go” button loads the address currently in the address bar. This button is connected to the enter key, so that when you press enter, it emits the click signal to the “Go” button.

The “Refresh” button loads the previously loaded URL, as read from the current tab’s HistStack.

## Potential Bugs

Occasionally there is a problem with writing to the visited.txt file. This happens when the user uses the autocomplete feature only. Sometimes it will not write the entire http address name to the file. It may result in invalid address names in the visited.txt file.

If you don’t have any bookmarks and try to load a blank bookmark, the browser will crash.

Sometimes on the deletion of a tab, the homepage will appear in the history of a different tab, but the rest of the history remains.

## Internal Workings

Main Window Class:

The main window had many aspects built from ui file made my Qt. All of the buttons, text fields, and web views were placed by us and made possible by the Qt framework. The main window is the central class of our project. It contains instances of the history class and bookmark class in a vector, which was indexed in correspondence to the tab indices.  This class also contains a lot of the default variables that give the browser dynamic characteristics such as incognito mode. The constructor sets various default settings like the homepage as well as the shortcuts. The constructor also calls the functions load_bookmarks and load_visited. These functions both read from local text files that load their respective values to be use by the browser. The bookmarks are loaded into a list and the visited sites into a set, each for their own optimization. Main window also contains all of the functions that execute the actions corresponding to the button presses and shortcuts for the web browser. Additionally, Main window also contains a lot of SLOT functions, which are a special set of Qt member functions that can be connected to SIGNALs that are emitted by Qt classes. SIGNALS can be connected to SLOTS using the connect function, so that when a certain SIGNAL is emitted by the specified object, the connected object will respond by running the specified SLOT. An example of this is when the BookmarkDialog loadBookmark SIGNAL (signifies that the button has been clicked and a bookmark is sent to be loaded) is connected to the mainwindow SLOT load_bookmark (loads the sent bookmark to the current tab), so that clicking the button in the BookmarkDialog will load that url in the mainwindow. This SIGNAL/SLOT design pattern made it easier for us to communicate with all of the different classes within our project.

## History Class:

The history class, HistStack, is a class that keeps track of a tabs history and provides an interface through its methods to move back and forth through the history. At first I tried to implement this history class with a vector, but after running into a lot of seg faults I decided to reimplement the class with a two stacks instead. The CanGoForward and CanGoBackward return a boolean indicating if that action is permissible given the current state of the history. The getPresent function returns the top of the back stack, which is the current url that the tab is on. The stepForward and stepBack functions are called when the back or forward buttons are pressed, and step through the history by popping off one stack and pushing onto the other. Stacks are a good data structure for this action because you only have to move incrementally through the history, and the limitations of the stack prevent you from going out of order.

## Autocompletion

The autocompletion feature is made possible by first loading visited sites into a set in the load_visited function. I chose to use a set because of the fast lookup times. I knew that I would be searching through the data structure to find values and if a user has a lot of visited sites then fast lookup times is key to making a smooth interface. When a user is typing in an address and presses TAB, the autocompletion function will run. It takes the current string that is typed in already and searches through the set to find the substring. Once it finds the substring it will fill the text field with the function's guess.

Bookmarks Class

Due to the list of bookmarks being dynamic (adding them constantly), it was difficult to hard code a GUI dropdown menu. Instead, we created a separate dialog box class, called BookmarkDialog, that handled this functionality. The BookmarkDialog has its own ui, which contains a scrolling listview listing the bookmarks and a pushbutton. When the button is clicked, then the highlighted list member is loaded in the current mainwindow tab. This functionality was implemented using the SIGNAL SLOT design pattern in qt. On instantiatino, the BookmarkDialog loads up the bookmarks from the file into a list, writes that list into a special QAbstractModel called a QStringListModel, and then displays that list in the listview (BookmarkView). On the button click, the class emits the loadBookmark SIGNAL to the mainwindow with the url attached. Then it emits the close SIGNAL to close the dialog box so that you can interact with your loaded bookmark page.

## Other Information

Using the Qt framework has made working on this project much different than we had anticipated. At first, we thought that using Qt would give us a few useful functions for making a web browser, but Qt was an entire development environment, halfway to another language. It came complete with a whole new type of string, an IDE, extensive documentation, and even a designer app. Since we learned Qt along the way, some aspects of the project might not have made the best use out of all the Qt resources out there, but towards the end we were more efficient and could add features and complex triggers with a lot less confusion. One of the results of using this framework, however, was that we weren't sure exactly what to put in the deconstructors. We erred on the side of Qt taking care of its own allocated memory, and deleted the things that we allocated.

OOP & STL:

This project was extremely oop based, since there was an enormous hierarchy from the beginning due to the ui object hierarchy. We added several more objects, such as histories and tabs, onto that hierarchy, which made it a good way of organizing the many different parts of the browser. Since the built-in Qt classes had so many methods, I couldn't imagine doing this project without objects, just because it would be way too much to keep track of without the organization that oop provides. Within most of the non-Qt related code that we wrote, we used a lot of the STL to organize objects and other data. We used vectors to keep track of tabs and histories, a set to keep track of visited websites, stacks to keep track of local history, and a list to keep track of all the bookmarks. The STL was very helpful because it provided a much need way of accessing our data in different ways without us having to program a new interface every time.