# Simple Macro: Addon for World of Warcraft

by
Yemane Gebreyesus

Computer Engineering Department
College of Engineering
California Polytechnic State University
2015

Date Submitted: 6/10/2015
Advisor: Michael Haung

# Table of Contents

# Abstract

This senior project was started to try and solve a problem within World of Warcraft. That particular problem is the underdeveloped default macro interface, which Simple Macro aims to remedy by creating a more user-friendly interface that is accessible by a wider audience. It employs a click through method of accessing and editing data to reduce the amount of typing necessary. The addon also has a feature to specifically help players that want to change a target in certain groups of macros all at once. The project was developed in both Lua and XML.

# List of Figures

**Introduction**
Simple Macro is an addon for the video game World of Warcraft. World of Warcraft is an MMORPG (Massive Multiplayer Online Role Playing Game) made by Blizzard, where you control a single character and embark on adventures with other players. An addon is a modification tool that players can use to enhance their gameplay. There are a lot of types of addons, ones that alter placement and size of user interface elements (such as buttons or chat boxes), and others that send alerts to users during encounters (e.g. a sound and popup telling the player to move in order to dodge a boss' upcoming ability), and many others.

This addon focuses on altering a specific feature of World of Warcraft known as macros. Macros are user-made buttons that can improve a player's experience by grouping slash commands used by the game. Macros can be used to combine actions together like targeting an opponent, then using an ability. They can also be used to even click multiple buttons sequentially to make certain tasks like logging out or queuing yourself for a group slightly faster. Because they are the same as any other button that is already provided to a player they can also be key bound, allowing a player to truly speed up how they play the game. Macros are very powerful; however, the most evident issue with macros is that they are remarkably difficult for a newcomer to understand. A lot of players that use macros wouldn't even be able to make changes to their own macros if circumstances called for it. Most people usually find macros online or from other players and don't put a second thought into the actual making of the macro.

Simple Macro's focus is to bridge the gap for new players, while also providing tools that help veterans of the macro world. The motivation idea for this addon came to me when I constantly had to change a specific set of my macros every few days or so. Those addons specifically targeted players that I was playing with so that I wouldn't have to target them myself (which would require moving my mouse to click on them in game). The default macro interface made this process more tedious than it needed to be due to small text and a block cursor (which may not be confusing to some people). To solve my problem, I decided to make a simple addon that only took slash commands to change my macros quickly. It worked quite well but I realized quickly that it would be difficult to share this with others who were interested in its functionality. In order to make it easier for others to use it, the addon would require some kind of GUI (graphical user interface), something that can be done with buttons and visuals rather than some archaic command line interface.

This led to the creation of Simple Macro's most important feature, the macro editor. This feature mimics the default macro editor, which provides support for creating, deleting, and editing macros. While the previous function of editing many macros at once worked well, it targeted a very small portion of World of Warcraft players. The difference between Simple Macro's macro editor and the default one is that in Simple Macro, the text which contains all the commands for the macro is clickable. For example, a common slash command for a macro is "/cast". In the default editor the macro's content is just a text box, completely editable. Simple Macro does away with this completely by making the whole "/cast" portion a button. Once clicked, a menu with a few dropdown menus that allows the user to navigate through all predefined slash commands. To help newer users even further, this addon provides descriptions for each command. The click-through method reduces user error significantly, which can make a huge

difference. Most mistakes come from mistyping something or not using the command correctly, which will not be caught by the game. Errors can take a while to debug, so reducing the chance of them encourages users to use macros more. Simple Macro principally attempts to reduce how much a player has to type, and displays any relevant information about macro components so that they will understand their macros and how to make them.

**Application**

Upon opening the addon, the macro editor is shown, which looks very similar to the default one, both seen in Figure 1. The macro editor (everything within the selected tab, "Create") displays the user's macros, which can be clicked on to bring up information about that macro, its name, icon, and content. With a macro selected, a user will be able to change the content of their macro. In Simple Macro you are able to hover over each part of the macro and decide which one you want to edit independent of the rest. Once clicked, depending on the type of macro element you have chosen, a different menu will be displayed to assist in editing that specific part of the macro.


*Figure 1. The default macro edit (left) alongside Simple Macro's macro editor (right).*


*Figure 2. A command ("/cast") is selected in the macro editor. Once selected, the command menu opens up which can be seen on the right.*

In Figure 2, the command menu is shown after selecting "/cast". This menu can also be reached by clicking "+ add new line +" at the bottom of the macro's text box. Contained in this menu are two dropdown menus, one for category and one for the command that the player will be using. The categories and commands are taken from http://wow.gamepedia.com/Slash_command. Hovering over either of the dropdowns provides information on what the command does, shown in figure 3. If this menu was opened by the "+ add new line +" button, then a new entry in the macro will be added with the selected command. There is also an option to add an argument to this command. An argument is exactly like a parameter that would be passed into a function.


*Figure 3. The list of entries within each dropdown menu of the command menu. It also displays the tooltips that are shown for the currently selected Category/Command.*


*Figure 4. The argument menu.*

Figure 4 shows the argument menu. It uses a editable text box for the user to enter any type of argument they want. The argument in the figure is “Flash of Light”, which means this macro will cause the player’s character to use the ability named “Flash of Light”. Similar to the command menu, it has a button for the next type of macro element, the conditional, adding any conditionals you’d like to this argument. Conditionals are sort of like if-else statements, they allow the command to be run if they are true.


*Figure 5. The conditionals menu.*

The conditional menu is shown in Figure 5. It contains every conditional currently available, a total of 36. Hovering over any of the conditionals will show a tooltip that describes the situation that must be true in order for the command to be run. It also tells the user how to format input for the conditional because some conditionals have their own parameters, these conditionals are designated by the text boxes next to them. On the bottom left there is a checkbox named “Use alternate text”. When checked, some conditionals are changed. These conditionals still have the same effect except the length of their text is different. Macros have a maximum character limit of 255 so a player may want to use an alternate version of a conditional to meet that limit. For example, “target=” and “@” both change the conditional target, but the “target=” makes the macro easier to read and “@” is shorter by 6 characters.


*Figure 6. The Groups tab. Three macros are inside the current group and the change target text box is set to "new target". Once the "Change Target" button is pressed the three macros will be edited to have the new target, "new target".*

In the "Groups" tab, shown in Figure 6, a user can change every argument of "target=" or "@" in a group of macros. Macros can be added into the group from the list of the user's macros with the "Add To Group" button, and of course removed from the group with the "Delete From Group" button once they are selected in the group. Once you have the macros you want to be altered added to the group, you enter in the new target that will be replaced in this group of macros and click "Change Target".

**Background**

This project requires a substantial amount of knowledge in two main areas, macros and addons. Macros are extremely variable so it is impossible to explain how they all work but this one will help in understanding some important parts.

```
#showtooltip
/cast [target=party2, exists] Leap of Faith
/target party2
/p “Using Leap of Faith on %t!”
/targetlasttarget
```

This macro is made up of five commands: “#showtooltip”, “/cast”, “/target”, “/p”, “/targetlasttarget”. The first command doesn’t have arguments. The second’s argument is “Leap of Faith” and it has the conditionals: “target=party2”, “exists”. The third’s is “party2”. The fourth’s is “Using Leap of Faith on %t”. The last command also doesn’t have arguments. Only one command, the second, has any conditionals: “target=party2” and “exists”.

There are a few interesting parts of this macro. The first is the “#showtooltip” command. This is a special system command that configures this macro’s icon and tooltip (text box that appears when you hover over the button) to match the ability or item that will be used when this macro is pressed. In this case the macro will look the exact same as the “Leap of Faith” ability’s button, as shown in Figure 7. The second is that the conditional “target=party2”, is a special conditional that changes the target for this command, it isn’t evaluated to see if it is true like every other conditional. The two commands, “/target” and “/targetlasttarget”, are used to quickly switch targets in order to perform a command that needs “party2” to be the target. In the “/p” command, “%t” is a special character that is replaced by the name of the player’s current target. Technically the “/target” command could be moved in front of the “/cast” command, allowing the “target=party2” conditional to be removed, but for the sake of this explanation it was not.


*Figure 7. The default editor containing the example macro from above alongside the player's Spellbook. Both the macro's icon and the ability Leap of Faith have the same icon.*

Upon execution this macro will attempt to cast the ability “Leap of Faith” on the second player in the party. If there is no second party member in the user’s party, the cast command will not run. Next, it will target the second player in the party, then, in party chat will say “Using Leap of Faith on <name of player in party 2 slot>”. And finally, the player’s target is set to their old target, making it seem as if nothing changed from the player’s perspective.

Addons can be quite complex as well, so they will be explained using a small addon that explains some key components. All addons are created using three types of files, .toc, .lua and .xml files. The “toc” file stands for “table of contents” and the others refer to the language they are coded in (LUA and XML). The table of contents file is quite simple and just tells the game where to find the addon files, its version number, author of the addon, etc. This is an example of a .toc file’s contents:

```
## Interface: 60100
## Title: ExampleAddon
## Author: ExamplePerson
## Version: 1.0
## Notes: An example addon.

ExampleAddon.lua
ExampleAddon.xml
```

More importantly, the .lua and .xml files contain the programming for the addon itself. Lua is a scripting language and it controls the logic for a wow addon. This is where functions that respond to events during the game, such as dying or logging in exist. The XML for a wow addon is meant to display all the static user interface elements, including windows and buttons. XML is a markup language similar to HTML. So, while the .lua file contains functions and code that respond to events, the .xml file is used for displaying the addon’s results from those LUA functions and it also provides the user with ways to input data for the addon. This isn’t always the case though. Sometimes, programmers are more comfortable with programmatically creating all of their user interface elements, meaning that there is no need for .xml files.

The following code is actually sufficient for an addon on its own:

```
local Congrats_EventFrame = CreateFrame("Frame")
Congrats_EventFrame:RegisterEvent("PLAYER_LEVEL_UP")
Congrats_EventFrame:SetScript("OnEvent",
    function(self, event, ...)
      local arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9 = ...
      print(Congratulations on reaching level ' ..arg1.. ', ' ..
      UnitName("Player").. '! You gained ' ..arg2.. ' HP and ' ..arg3..
      ' MP!')
    end)
```

If copied into a file named ExampleAddon.lua, this file along with the .toc shown can be placed into a folder named “ExampleAddon”. Once this folder is added into the “World of Warcraft/Interface/AddOns” folder, it will be seen in game.

This addon is a bit more complex than a simple “Hello World!” tutorial but it isn’t that difficult to understand. The first line creates a frame which is World of Warcraft’s version of an object like in Java or C++. A frame is actually an instance of a table (the only lua data structure), but it has special features that allow it to interact with the World of Warcraft API. Frames have their name because they are used for all user interface elements, but don’t necessarily have to be shown. In this particular example, the second line registers the frame to the “PLAYER_LEVEL_UP” event. By registering this frame to this specific event, when the event is fired, World of Warcraft will check to see if the frame has a response set for the event and execute it. The third line does this by calling the SetScript() function. So when a player levels up the function provided to SetScript() will be called. The 9 local variables are being set to the variadic parameter. Many events send varying amounts of data when they are fired and in this case, the PLAYER_LEVEL_UP event. To figure out how much data, if any is sent, one must consult the WoW API, this specific event’s information can be found here: http://wowwiki.wikia.com/Events_A-Z_(Full_List)#PLAYER_LEVEL_UP. Lastly, the addon displays a few of these arguments in the chat log upon leveling up, shown in Figure 8.


*Figure 8. A message in the chat log informing the player that they have leveled up.*

**Design**

Much of the project required a lot of planning ahead. The frontend is the most important part of this project. The entire purpose is to provide a simpler view of the macro universe within WoW. That needs to be addressed in the user interface the most. On the other hand, the backend is important as well because it interacts with the frontend and should be designed to mimic the structure of the frontend. Most of the backend design followed the UI design while also paying attention to standard coding practices.

When deciding on how to layout the GUI, I decided that a few things needed to be accomplished. It needed to be, above all, straightforward and efficient in comparison to the default editor. Also, it felt necessary that the amount of memorization that is currently required to create a macro was diminished as much as possible. This led to taking the default macro editor’s view and only altering it slightly to incorporate the new features of Simple Macro. As seen in Figure 2, the player’s macros, macro content, and even menu buttons are all laid out in the same manner. This

choice falls under keeping the addon very straightforward. Users will be more used to this layout and be able to learn about the new features easily. If I used a completely different layout than the default one, players would be less comfortable with the layout, offsetting the benefits that the addon is supposed to provide. The last main design choice for the editor was to make buttons out of the different macro elements. This satisfies both goals by both removing the need for a user to remember how to type a command or which conditional they want to use, while also keeping the addon efficient by allowing you to click each part and instantly have a menu to change that part quickly.

The other feature of the addon, the group target changer, didn't require as much planning as the macro editor. This tab is a little more self-explanatory because the buttons pretty much speak for themselves, making it quite straightforward. A more preferable layout would be to have the two macro selection frames (the scroll areas that contain the macros) side by side just as many other types of interfaces do it, an example of this can be seen in Figure 9. Unfortunately, space was an issue so the vertical layout was chosen. This feature is still somewhat early in development, so most of the design is yet to come.

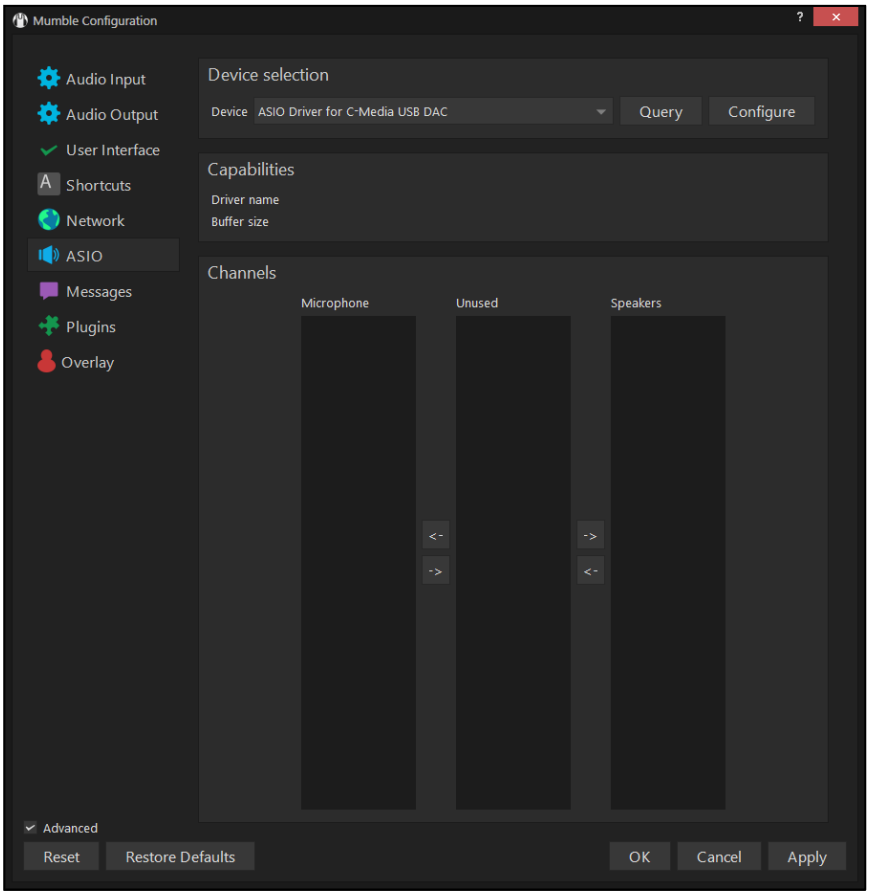


*Figure 9. An options menu in Mumble. It displays three groups that have buttons that move entries between them.*

In terms of code organization, I attempted to make the code easy to read and maintain. The structure is illustrated in Figure 10. The "SimpleMacro", "Create Tab", and "Group Tab" boxes all represent portions of the addon that can be displayed (consist of both .lua and .xml files). The

diamonds, “macroparser”, “WoW API”, and “localization” represent libraries that are used by the addon.

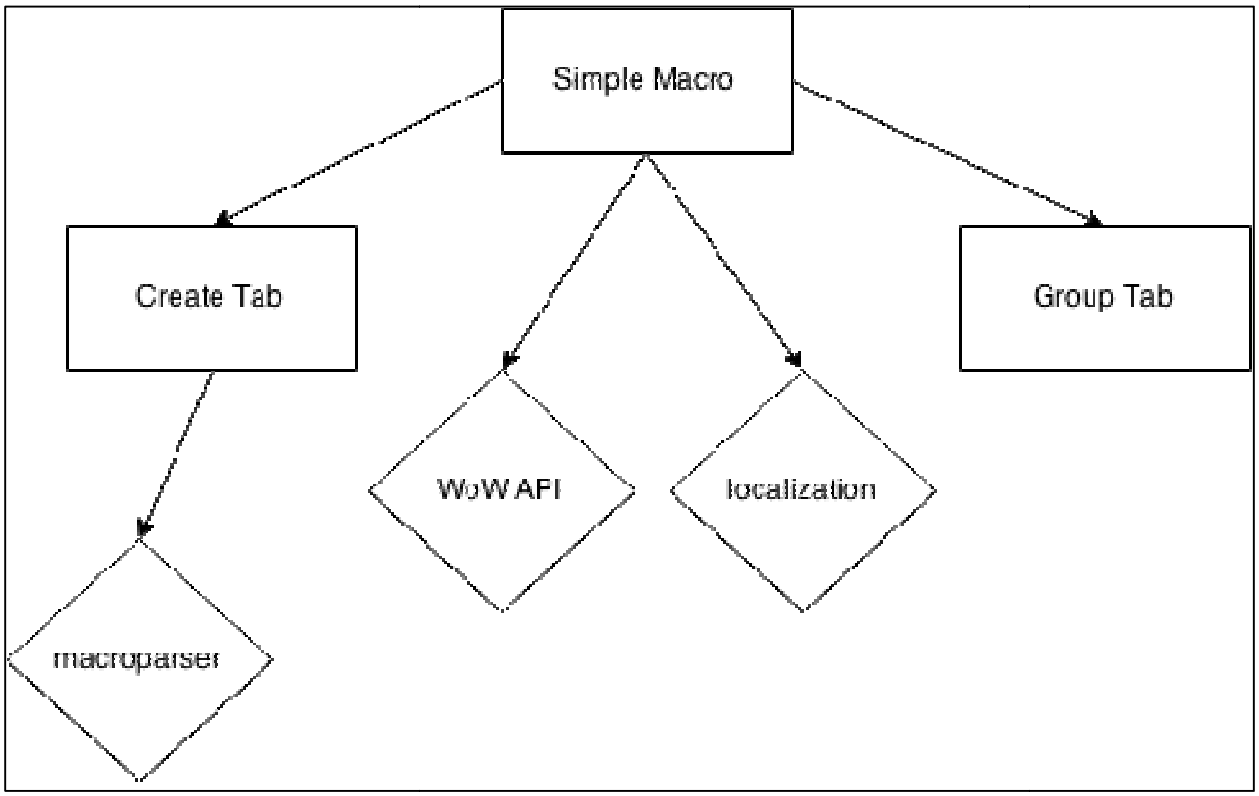


*Figure 10. A layout of Simple Macro's code structure.*

The SimpleMacro.lua file only exists to setup slash commands and the SimpleMacro.xml file only contains the layout for the frame that appears upon opening then addon. It contains the tabs but not any content within them and the close button that appears when either tab is open. Each tab is a separate entity, never interacting with the other tab. They are completely self-contained, making it quite easy to make changes to one and not worry about how or if it is interacting with anything else.

No third-party libraries were used for this project. The WoW API library is used by default (just making an addon and the other two are made specifically for this project. A localization library is used because it conforms to practices followed by most addons. This library allows any text within the code to be replaced by a table lookup. The table lookup result changes depending on what region the game being is played on, allowing more people to use the addon. Although the addon only has English localization so far, developing with this in mind opens up the opportunity for other languages to be supported in the future. Without implementing this early on, it would have required a lot of effort to incorporate this feature later on.

The last library is the macro parser. This library was created to take a macro and convert it into easily indexed parts. It is used by the macro editor to make changes to macros without having to constantly parse them to figure out where the second conditional is, for example. The macro

parser is the middle man between the addon and the macro. The addon is able to then make changes to the macro through an API. This was doable due to the predictable nature of macros. Almost every macro follows a specific hierarchy, shown in Figure 11. For the ones that don't follow this (which is only a few), there are functions in place to catch them and deal with them separately. An example of this is the "/focus" command. Conditionals can be applied to this command but it takes no arguments. These types of commands could have been incorporated into the design if it was less rigid, possibly if conditionals were a field within the command instead. The tradeoff here is that the current design allows parent elements to be deleted which automatically delete everything under it. Moving the conditionals up a level would require extra logic to find them and delete them when their matching argument is deleted. The current design makes the code a little simpler in general and is easier for another programmer to pick up.

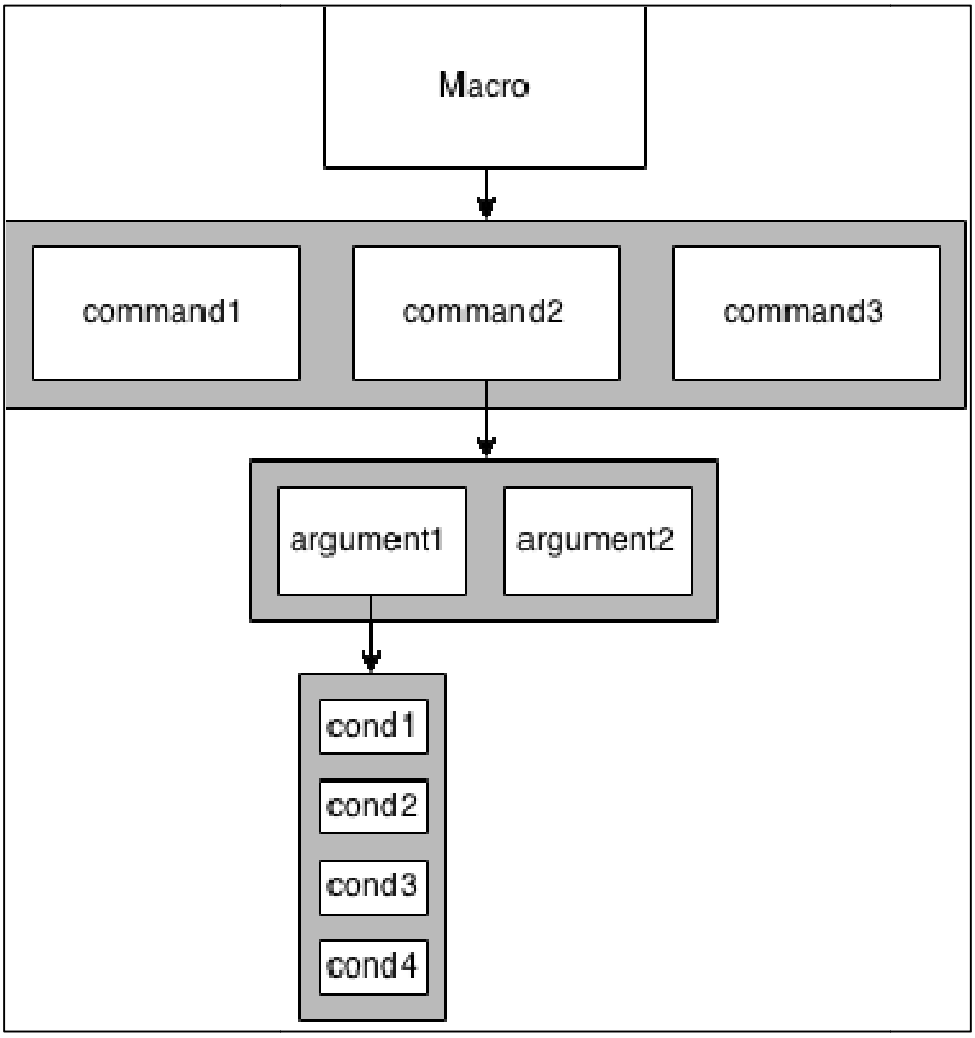


*Figure 11. The macro hierarchy.*

**Implementation**

For the macro editor to work it needed to be able to display the macro text normally but make the commands, arguments, and conditionals separately clickable. This called for them to be

represented as buttons, specifically “CheckButtons” as they are called in the World of Warcraft API. So at the core, these buttons are actually the same as the ones in the conditional menu from Figure 5, with a few differences of course. They use a different CheckButton template than those from the conditional menu and have an added text field added to their frame to display their content. These buttons also need to be placed dynamically within the scroll area that represents the macro’s content. This is done by using the macro parser library’s object to figure out which command/argument/conditional is being added and where it needs to be placed relative to anything else. An important thing to note is that in WoW, all frames are global. Making a new frame can be dangerous if unique names aren’t used, also reusing old frames is highly recommended. Especially in this case of dynamically created frames, a frame for the first command’s text can be reused for every macro instead of making a new one each time which could eventually cause memory problems. Not all macros may have a second or third command though. These frames will be hidden until they are needed again.

The editor also does something similar with the macro list to prevent user error. When switching between Account and Character macros, the number of user macros is usually different, and even more so the amount of max macros for each type is always different (120 for account and 18 for character). When this happens, there is a function that actually has to loop through every macro button that exists. It will hide the excess ones that the Character tab doesn’t need so that the user won’t even be able to scroll down past the 18 macros. The function also disables any macros that are within the max limit of the tab but aren’t being used. This leaves a box that cannot be selected, visible in Figure 6 where the three macros have been added to the group but the rest are empty.

The macro parser is actually the most intricate part of the addon even though it is all behind the scenes. It involves a lot of pattern matching, which is very similar to regular expressions used by other languages but not as expansive. It is able to take a whole macro and actually turn it into a table of other tables that end up representing the macro structure in Figure 11. It was necessary to pay close attention to which delimiters were responsible for determining if there is another conditional coming, when the conditionals end, or when another argument is next in the macro’s body, etc. A specific case arises when a conditional requires a parameter of its own. This happens in the case of the “target=” conditional mentioned near the end of the Application section. The following code snippet shows exactly how this is done:

```
if string.match(temp, ".-[@:=](.*)") then
   cond.cond, cond.input = string.match(temp, "(.-[@:=])(.*)")
else
   cond.cond = temp
end
```

The current value of temp at this part of the code would be “target=exampletarget” or “exists”, the full name of a conditional. The first string.match() function call will check this string to see if it has any of the three symbols in the brackets: “@”, “:”, or “=”. These symbols are the only ones that indicate if a conditional will have an argument, which can be discovered by examining the list of macro conditionals. If this returns a non-nil (nil is the same as NULL in lua) value then the next line will set both the name and input field for the conditional because it contains one of the special characters. Some other aspects of the parser include: whitespace needs to be trimmed at

every step in case of user error. The parser also has to be able to recreate the macro's body so that it can be read by the game to perform its intended function.

In the other component of the addon, the target changer, it is necessary to be able to make a change to multiple macros within a group. This could be done by using the macro parser API to parse each macro and check conditionals that match "target=" or "@" but the overhead from parsing each macro in the first place may turn out to actually affect performance for a large group. To remedy this, a more specialized version of the parser exists in the form of a single function in the Group tab's .lua file. However, this version only searches the macro for any instances of "target=" or "@" and will replace whatever argument is following it with the new one. It doesn't store any data and deals with the macro text directly.

**Analysis**

The addon is able to perform in a variety of circumstances but has not yet been equipped to deal with a few situations. Most existing macros can be edited and added to in the editor. The only case that isn't covered right now is any macro with a command that can take conditionals without arguments. Unfortunately, it doesn't contain all the features needed that would allow it to replace the default editor. It is missing a menu for creating a new addon, which involves naming the addon and selecting an icon. This menu would also be used for editing the name/icon, another feature that is yet to be added. It also needs both a "Save" and a "Revert changes" button. And even though the new way to edit macros is superior in a lot of ways, the most advanced users don't need to click through menus to make macros that they are comfortable with. The last feature needed would be a way to switch between the ways Simple Macro represents the macro's content (highlighted buttons) with the default editor's user text entry box. Once these features are added, Simple Macro can be used over the default editor without missing any key functionality.

Previously, the addon had a different user interface that used the same parser and split the macro's components into separate lists. After surveying a few testers of the addon, it turned out that the method was tough to understand at first glance. Separating the commands, arguments and conditionals into different lists led to confusion as to what an Argument or Conditional was, as these were the titles on the lists. An experienced user could figure it out after some fiddling but newer macro users weren't able to. After receiving feedback from testers, I discovered the current design. It was much simpler for users to understand, they felt that keeping the display similar made it significantly easier to comprehend. Only the interaction with the macro content changed instead of how it was shown to the user. They don't have to worry about figuring out a new layout because it is the same as the default one.

In terms of speed, the addon doesn't have any noticeable lag. Even though there are some situations where code is repeated or called unnecessarily, everything runs very fast. I don't expect it to every run slow unless some kind of database is implemented. Addons that are slow typically have to deal with converting large amount of in game data into a database.

The responses from testers are quite varied. Some players have conveyed interest in using the addon in the future, and that it may actually help them when making a new macro. Others have found a use with the target changer just like myself, and use it occasionally. Unfortunately, there

are players that still don't find a use for the addon because they don't use any macros or just use a few that they never change. A few just don't find a need for it because they are already considerably proficient in making macros.

**Related Work**

The top downloaded macro-related addon is Macro Toolkit. It has a lot of options for building macros. The addon provides syntax highlighting, error checking, a sharing interface, and a lot of other features. This addon has a ton of quality of life changes in it, making the editor feel like a typical IDE (integrated development environment) like Eclipse. Macro Toolkit addresses a lot of the shortcomings with the default editor and goes even further to allow users to customize almost anything. While it has a lot of upsides, it misses out on what this project is aiming to solve which is the barrier to entry for new WoW players attempting to use and make macros.

Another macro-related addon is MacroForm. This addon allows a user to write macros with conditionals without typing anything. It does this by allowing a user to drag abilities from their spellbook onto the addon to use as an argument. Then it adds in conditionals through drop down menus to the macro. It focuses on providing support for commonly used conditionals and only supports creating macros with one "/cast" or "/castsequence" command. Simple Macro is basically an extension of this addon in terms of functionality. While Simple Macro doesn't implement the dragging of an ability onto the menu to use as an argument, Simple Macro can create multiline macros with any conditional in the game. MacroForm is more useful for users who only create macros of a specific type. The addon doesn't support enough for it to be widely useable.

**Future Work**

There are a lot of features to be added and bugs to be fixed before Simple Macro is complete. First of all, the user interface isn't completed. It needs a few touches to be more relatable to other menus so that users are comfortable with it. The border will most likely be changed to match other menus (like the default editor); its current border resembles the borders of typical popup menus. It also needs an exit button on the upper right like most windows have. The command/argument/conditional buttons in the macro's content area need some updates to make them more appealing. Some changes already in mind for the buttons would be to make the highlight more rigid in shape, have it encompass the complete amount of text it refers to, and keep its opacity uniform throughout but at a lower level. Right now it looks a bit tacky and it can be hard to click some of the entries that are less than a few characters long. These are all essentially quality of life changes, which are more or less so a key component of Simple Macro's goal.

The Create tab could use a few functional additions as well. As mentioned earlier, a way to switch to a normal text editor view of the macro's content should be available. For one, it is useful for users who don't want to go through the click-through process to setup a command they know by memory. Also, it makes it so any bugs that users run into can be circumvented using this editor. Having a failsafe is always useful.

Another feature for this tab would be one to move commands up and down within a macro. Right now, adding a line to a macro will always append it. This isn't a desirable result because there is

always a chance that you need to place a new command within a macro instead of only at the end.

In the argument menu, the user has to type in the name of the argument to be added to the command. This implementation works but is prone to error and a new user may not know exactly what kind of arguments they can even use. The goal for this menu is to have an auto-completion mechanism that will grab a subset of abilities, items, or other valid arguments that are compatible with the current command. The mechanism will then filter out arguments based on what the user is typing. This feature covers a lot of what Simple Macro is made for. It helps new users by suggesting arguments that they may have never guessed were applicable. It also speeds up the typing process because a user will be able to, hopefully most of the time, select the desired suggestion before they even finish typing the argument. Of course, there are some considerations that need to be addressed for this feature. Speed can definitely be an issue in terms of loading and filtering the data that will be displayed. Finding the right balance between showing relevant suggestions compared and speed will definitely play a factor in implementing such a system.

For the Group tab, a few things need to be added. Switching between Account and Character macros needs to be implemented. Also, support for having multiple groups will be added. Though, making multiple groups work necessitates the ability to save data across different logins. An issue arises with this feature because of many things that may happen. When a macro is deleted, it will need to be reflected in any group it belongs to. If not, the addon will attempt to alter a macro that doesn't exist, or even worse a macro that wasn't originally in the group will be changed by accident. There are a few fixes to this, but they all have their pros and cons. The delete button in Simple Macro's editor could also call a function within the Group tab to delete the macro from any groups it is in, which would achieve the desired result. Unfortunately, this solution doesn't solve the problem where a user could delete a macro from the default editor, which wouldn't call the function to make changes to groups. A different solution would store all data about the addon, its name, icon, and body. When a group is loaded it would then search through all the macros and check to make sure that each of its macros still exists. The main problem with this is speed, as searching through every single macro could take a long time.

Another Group tab related feature to add would be a slash command or context menu for editing the groups. A context menu is a menu that appears after clicking on something, usually a right-click. The menu typically contains actions that can be performed in relation to the clicked item. The most user-friendly method would be to allow a user to right-click a person and select something like "Change Group 2 targets to this target". This way a user never even has to type anything or open the menu once they have their groups set up. Another backup method would be to use a slash command. The slash command, while slower than the context menu and even needing the user to type, can change the targets in a group to any specified name, without the player having any current target.

Further down the road, other features similar to the group target changer will be added to Simple Macro. These would include any features suggested by fellow players that they'd want to help them deal with their macros more efficiently.

**Conclusion**

This project has a large focus on the user experience. From the beginning the goal has been to create a product that is straightforward, plain and simple. Simple Macro takes everything that felt sluggish about the old editor and attempts to streamline it. The ability to click on buttons rather than type commands is basically the idea behind an operating system. Simple Macro reinforces the idea that the consumer experience is the most important concern when developing a product. This idea creates products that will be used and reused due to their accessibility and efficiency. Simple Macro's interface opens the door for beginners to interact with simplified tools that help them to understand the more difficult concepts behind the macros, such as conditionals.

As technology becomes more advanced, more companies are starting to realize how much the customer's view of their product matters. This project is just one item in a long line of products that are trying to make some difficult idea reachable by the more general public. The idea for this project was even a result of being a consumer of the default editor. The frustration from having to deal with its shortcomings led to this. A close consideration of the user experience is crucial to the success of any product.

**Source Code Link**
https://github.com/crudos/SimpleMacro

**References**
1. Macro API: http://wow.gamepedia.com/Slash_command

2. Event API (PLAYER_LEVEL_UP): http://wowwiki.wikia.com/Events_A-Z_(Full_List)#PLAYER_LEVEL_UP

3. Tutorial on patterns in Lua: http://lua-users.org/wiki/PatternsTutorial

4. Macro conditionals list: http://wow.gamepedia.com/Macro_conditionals

5. Tutorial for making an addon: http://www.dev-hq.net/posts/2--create-world-of-warcraft-addon

6. Macro Toolkit addon page: http://www.curse.com/addons/wow/macro-toolkit

7. MacroForm addon page: http://www.curse.com/addons/wow/macroform