Refactoring Goal

The goal of the refactoring was to simplify and compartmentalize the interface half of the code, i.e. the code relative to the console interface, the parsing, the displaying and variable storage scheme. The second half of the software is mostly untouched, being the processing of JSL callback and sending of KB/M events.

Refactoring Resume

The refactoring is meant to decouple “model” code from “controller” code, and as such two base classes are added: JSMVariable<T> and JSMCommand.

The variable class responsibilities are limited to filtering assigned values, notifying listeners of changes (observer pattern) and resetting to a default value upon request. Most variables in JSM are chorded however, and this functionality is captured in a descendent: ChordedVariable<T>. This kind of variable additionally holds a mapping from ButtonID to another JSMVariable instance, holding the alternate value for the chorded variable. Chorded variables also offer getters for the value of the variable depending on chord or not. Finally, ChordedVariables come in two kinds: JSMSettings and JSMButtons. JSMSettings contain a SettingID value and some logic for removing chords-modeshifts. JSMButtons similarly have a ButtonID but also simultaneous press mappings, which is handled similarly to chords. It also requires some methods to provide string name for itself and its alternative values for use in the button handling function. It contains chord and simpress removal logic.

On the other hand, a JSMCommand’s responsibilities are holding a name to recognize from the command line and a parsing function to run upon call. It also contains a help string. A command also be requested to provide a “modified command” instance. This is used to provide a command for alternate values such as sim presses and chord presses. Commands can also run a task on destruction. This is currently used to remove chords and modeshifts. The JSMCommand is the basic interface used by the command registry that will process the command line, look up valid commands to run, and get that command to continue processing with its own specialized parser function. Unknown to the Command Registry, commands come into two descendants: MacroCommand and JSMAssignment<T>. The first one is simple, run this functor when called. The second one binds to a singular JSMVariable of the same type on construction. It has a powerful default parser that makes use of streaming operators to read and write the value to and from string, and as such, those may need to be defined by the dev at global scope. It also adds its own listener to the variable in order to display the value upon change, thus fulfilling the Model-View-Controller loop. It’s also the entity that implements the ability to generate “modified commands” such as sim press parser and chord press parser.

Finally some JSMAssignment derivatives are created for specific cases, such as gyro button assignment coming in two blends : GYRO_ON and GYRO_OFF, and GYRO_SENS modifying two variables who have their own assignments already.

Where did X go?

X	Where X went	File	Comment
#define for JSM specific actions	Constexpr in header file instead of cpp	JoyShockMapper.h	
#define MAPPING_[BTN]	Header file as part of a typed enum		All names have been changed to the string match with
#define [CMD_NAME]			
Any enums	Header file		
Struct FloatXY	Header file		
Struct GyroSettings	Header file		Using typed enum is greatly valuable, especially for debugging
struct Mapping	This structure is basically replaced by JSMButton, since it holds all possible mappings for a value. See comment		Take note, there is a Mapping structure which simply is a pair of press and hold Keycodes. It is the base variable type of a button.
WORD nameToKey(string)	OS specific file	Win32/PlatformDefinitions.cpp	In Linux, KeyCode can be defined using a type best suited for itself
Const char *version = “A.B.C”	The version string is now generated by cmake from the last git tag	Include/JSMVersion.h.in CMakeList.txt	Editing this string is handled automagically now!
struct ComboMap	Is now a pair of ButtonID and JSMVariable<Mapping>	JSMAssignment.hpp	This typedef also happens to be an item of the chorded map which is what it was before also.
Optional	Gone.		Use std::optional now
struct JSMSettings	Gone. Each JSMSetting now contains all possible alternate values.		
int keyToBitOffset(ButtonID index)	JoyShock member function	Main.cpp	Only used for JoyShock::IsPressed. It could be replaced by the chord stack
keyToMappingIndex	Replaced by ostream &operator << (ostream &out, ButtonID rhv)	Operators.cpp	Making used of magic_enum’s magic to minimize that code

nameTo[Enum]	Replaced by ostream &operator << (ostream &out, E rhv)	JoyShockMapper.h	There are a few exceptions that have their own specialization
Strtrim() loadMappings()	Member of CmdRegistry	CmdRegistry.cpp	Only place where it is used
parseCommand	Atomized between all command and variablees: all macro/unary commands have their own global scope static function called do_[Name] All settings are processed by the DefaultParser, filtering function and streaming operators and help string.	Do_[Name] -> Main.cpp DefaultParser -> JSMAssignment.hpp Filtering functions -> main.cpp Streaming operators -> JoyShockMapper.h and operators.cpp	Paradigm shift from procedural to object oriented. Sorry for the headaches. ;-P