1 User manual

1.1 Generate option stub

You can have the options c and h files created automatically for you by running the script *generate_option_template.py* located in the *sm_options* folder inside the g*oldwing* folder.

The script will ask for the name of the option to be created and generates the files from it, according to the Goldwing template.

1.2 Create option stub manually

To create the option files manually, create the options h and c files in the sm_options folder: <*your_option.c/h*>. In the *your_option.c* file, make sure to add `#include "goldwing.h"`. You do not need to include *your_option.h*, as this is taken care of by g*oldwing.h*.

1.3 Option requirements

In order for the options to function, you must meet the following criteria:

- The name of the option routine function should be identical to the file name (except for .c/.h)
  E.g. if the file is named warning_lights.c/h the main function of the option should be called warning_lights
- Your .c-file must include g*oldwing.h*
- In your .h-file, be sure to declare your main function of the option under *GW_PRIVATE*, as it must be accessible from other parts of the system

The script *generate_option_template.py* does all this for you and is the recommened method.

To add an option to the *run queue*, you add the function pointer to *OptionArray* defined in *goldwing/sm_mcu/mcu.h/c* file. Increase the *NUMBER_OF_OPTION* constant in *mcu.h* and add the function pointer to the *OptionArray[n].run*-member.

All should now be configured to start implementing the option. All the tools made available by the ICH are listed in section 2. Use these to access signals and function calls. In the case of a vital signal needed by your option not present in the list, check if the signal is already available on the CAN-bus. These signals can be acquired by the sniffing-interface (the signals can occur in un-scaled format) without any penalty in CAN-traffic.

Remember to *make clean* if an option routine is removed manually.

2 Toolbox

Function	Description
`UWord get_currWeight(void);`	Gets the current weight on forks in mV
`Bool get_height1(void);`	Returns TRUE if forks are higher than height sensor 1
`Bool get_height2(void);`	Returns TRUE if forks are higher than height sensor 2
`SWord get_currSpeed(void);`	Gets current speed in engine rpm
`SWord get_currSteerAngle(void);`	Gets current steer angle as analog value
`SWord get_bflyRamped(void);`	Gets Butterfly value
`SByte get_adLift1(void);`	Gets adlift value
`SByte get_adLift2(void);`	Gets adlift value
`UByte get_digitalButtonBitfield(void);`	Returns bitfield with pressed digital buttons. 1 indicates button press
`UByte get_optionButtonBitfield(void);`	Returns bitfield with pressed option buttons. 1 indicates button press
`void write_display(UByte, UByte, UByte, UByte, UByte);`	Make a write to display call to ICH. If first byte is zero, ICH takes control of display. Use constants defined in jura_display_const.h (included in goldwing.h)
`void restrict_hydraulic(UByte);`	Restrict hydraulic function
`void request_hydraulic(UByte, SByte);`	Request hydraulic function
`void restrict_steer(SWord);`	Restrict to given steering angle
`void request_steer(SWord);`	Request given steering angle
`void restrict_drive(SByte);`	Restrict to given drive speed
`void request_drive(SByte);`	Request given drive speed
`void request_power(void);`	Request power (main contactor)
`tCanGoldwingPdo* get_CanMsg(UByte CanIndex);`	Used to get a can object to manipulate freely. Use with caution.