#include “ nxcBoost/motor.nxc“

safecall inline void RunMotorFast (const byte out,

const char pwr,

const long angle)

Run the motor with maximum acceleration and breaking

Use this command only when it is necessary.

This function need a lot of battery(power).

The motor will be in break mode after this function.

Do not apply coast mode directly after this function. (use wait(200))

RotationCount is bugged in nxc (not recommended)

byte out ... Desired output port. Can be a constant or a variable,

see uutput port constants.

char pwr ... Output power, 0 to 100

long angle ... Angle limit, in degree. Can be negative to reverse direction.

safecall void ResetMotor(const byte out, const char pwr)

Reset the motor by turning until stalling

Be aware of using this function.

byte out ... Desired output port. Can be a constant or a variable,

see Output port constants.

char pwr ... Output power, -50 - 50, use the sign for direction

#include “ nxcBoost/algorithm.nxc“

unsigned int ArrayMaxPos(const float array[],

unsigned int idx = 0,

unsigned int len = NA)

This function calculates the position of the maximum

of all or a subset of the elements in the numeric array.

const long array[] ... the source numeric array

unsigned int idx = 0 ... The index of the start of the array subset to process. Pass NA to start with

the first element.

unsigned int len ... The number of elements to include in the calculation. Pass NA to include

the rest of the elements in the src array

unsigned int ArrayMinPos(const float array[],

unsigned int idx = 0,

unsigned int len = NA)

This function calculates the position of the minimum of all or a subset of

the elements in the numeric array.

const long array[] ... the source numeric array

unsigned int idx = 0 ... The index of the start of the array subset to process. Pass NA to start with

the first element.

unsigned int len ... The number of elements to include in the calculation. Pass NA to include

the rest of the elements in the src array

string strtok(const string str, const string del)

A sequence of calls to this function split str into tokens, which are sequences of contiguous characters separated by a string of a delimiter.

On a first call, the function expects the token-string as argument for str, whose first character is used as the starting location to scan for tokens. In subsequent calls, the function expects a "NULL" string and uses the position right after the end of last token as the new starting location for scanning.

The function returns in every single call one token.

If there are no more tokens left, the function returns a "NULL" string.

If str or del contains no character, the function returns also "NULL"

const string str ... the string containing the tokens or "NULL" continue with current string

const string del ... delimiter between tokens

string return ... current token

unsigned int tokenize(const string raw\_string,

const string delimiter,

string &tokens[])

This is a example how to use strtok This function searches for tokens in the raw\_string.

The tokens are stored in an array &tokens[]

const string raw\_string ... the string containing the tokens

const string delimiter ... delimiter between tokens

string &tokens[] ... array containing the tokens

int return ... number of tokens found

#include “nxcBoost/console.nxc“

void resetConsole(const bool clearScreen = true)

Resets the console buffer and sets the linepointer to the top line.

const bool clearScreen = true ... set false if you do not want to clear the console.

void initConsole(unsigned int topLine = 1, bottomLine = 8)

Initialize the console. Use the parameters to define the size of the console on the screen.

If you pass no values, the console uses the whole screen.

unsigned int topLine = 0 ... the console begin line 1-8

unsigned int bottomLine = 0 ... the console end line 1-8

void setConsoleTime(const unsigned long time)

You may set the default wait time. The wait time is used after every write console.

const unsigned long time

void setConsoleEndl(const bool endl)

You can set the standard parameter for the end line function.

const bool endl ... set to false if you do not want a standard word warp after ervery cout.

bool setConsoleTopLine(unsigned int line)

Use this to set the begin line of your console. See initConsole for more information

The console will be reseted with this function call

unsigned int line line from 0 to 8

bool setConsoleBottomLine(unsigned int line)

Use this to set the end line of your console. See initConsole for more information

The console will be reseted with this function call

unsigned int line line from 0 to 8

unsigned int getConsoleLinepointer()

Get the actual line

return … actual line

unsigned int getConsoleLinepointer()

Get the actual line

return … actual line

void cout(const string str, unsigned long time = NA, bool endl = NA)

console out

Print a text and wait for "time" milliseconds. If "endl" is true, the console will print the text and

end this line and begin a new line with the next function call. If endl is false, the console will stay in the current line and the next text will be attached to the current string in this line.

const string str ... the text to print

unsigned long time = NA ... wait time after printing

endl = NA ... end line after printing

void coutNum(const float number, unsigned long time = NA, bool endl = NA)

console out

Print a number and wait for "time" milliseconds. If "endl" is true, the console will print the number and end this line and begin a new line with the next function call. If endl is false, the console will stay in the current line and the next string will be attached to the current number in this line.

const float number ... the number to print

unsigned long time = NA ... wait time after printing

endl = NA ... end line after printing

coutFormat(const string format,

const float number,

unsigned long time = NA,

bool endl = NA)

console out format

Print a formated number and wait for "time" milliseconds. If "endl" is true, the console will print the format and end this line and begin a new line with the next function call. If endl is false, the console will stay in the current line and the next string will be attached to the current string in this line.

const string format ... format (see nxc api doc "Format" for more info)

const float number ... the number to print

unsigned long time = NA ... wait time after printing

endl = NA ... end line after printing

inline bool kbhit()

Returns true if any button is pressed. It will not wait for a button to be pressed!

Example: until(kbhit()); <- wait until any button pressed

const string format ... format (see nxc api doc "Format" for more info)

const float number ... the number to print

unsigned long time = NA ... wait time after printing

endl = NA ... end line after printing

#include “nxcBoost/debug-console.nxc“

void initConsoleDebug(const long time = 1000, const bool endl = true)

Initialize the console debug module with standard parameters

const long time = 1000 ... default wait time after display debug message

const bool endl = true ... set default end line, see console doc

void setConsoleDebugTime(const unsigned long time)

Set standard wait time after output a debug message

const long time ... default wait time after sending debug message

void setConsoleDebugEndl(const bool endl)

Set standard end line see console for info

-

inline void coutDebug(const string &message,

unsigned int class = NA,

unsigned long time = NA,

bool endl = NA)

For message, time, endl parameters read console doc.

Read description “debuging with class“ for class parameter!

-

inline void coutDebugNum(const float &number,

unsigned int class = NA,

unsigned long time = NA,

bool endl = NA)

For message, time, endl parameters read console doc.

Read description “debuging with class“ for class parameter!

-

inline void coutDebugFormat(const string &format,

const float &number,

unsigned int class = NA,

unsigned long time = NA,

bool endl = NA)

For message, time, endl parameters read console doc.

Read description “debuging with class“ for class parameter!

-

#include “nxcBoost/com.nxc“

void initCom(const byte outbox = 0,

const byte inbox = 0)

Use this to initialize the communication module. The paramters are set to default values.

You can initialize the com-module with standard in/outbox.

const byte outbox = 0 ... set default outbox

const byte inbox = 0 ... set default inbox

void setComTime(const unsigned long sendTime)

Set standard wait time. (wait time after sending)

-

void setComOutbox(const byte outbox)

set standard outbox.

-

void setComInbox(const byte inbox)

Set standard inbox.

-

void setComTimeout(const unsigned long timeout)

set standard timeout for trying to receive messages

-

void setComPollingTime(const unsigned long pollingTime)

Set standard polling time in ms for receiving messages

-

inline char send(const string &message,

unsigned long sendTime = NA,

byte outbox = NA)

Send a string to specified outbox and wait for sendTime ms

const string &message ... any message

unsigned long sendTime = NA ... wait time in ms, not essential

byte outbox = NA ... see outbox const.

return … send status (see communication const in nxc api doc)

inline char sendNum(float &number,

unsigned long time = NA,

byte outbox = NA)

Send a number to specified outbox and wait for sendTime ms

const string &number ... any signed number

unsigned long sendTime = NA ... wait time in ms, not essential

byte outbox = NA ... see outbox const.

return … send status (see communication const in nxc api doc)

inline char sendFormat(const string &format, const float &number,

unsigned long time = NA,

byte outbox = NA)

Send a formatted number to specified outbox and wait for sendTime ms

const string &format ... search for "Format" in nxc api doc

const string &number ... any signed number

unsigned long sendTime = NA ... wait time in ms, not essential

byte outbox = NA ... see outbox const.

return … send status (see communication const in nxc api doc)

inline string receiveString(byte inbox = NA,

unsigned long timeout = NA,

unsigned int pollingTime = NA)

Wait until theres a string in the specified inbox.

You may set a timeout if you want to wait a longer or shorter period

byte inbox = NA ... see inbox const. (not essential)

unsigned long timeout = NA ... timeout if theres no message

unsigned int pollingTime = NA ... the time to wait between every request

string return ... the received message or

-1123 if theres a timeout

inline long receiveNum(byte inbox = NA,

unsigned int pollingTime = NA,

unsigned long timeout = NA)

Wait until theres a number in the specified inbox.

You may set a timeout if you want to wait a long or short period

byte inbox = NA ... see inbox const. (not essential)

unsigned long timeout = NA ... timeout if theres no message

unsigned int pollingTime = NA ... the time to wait between every request

long return ... the received message or

-1123 if theres a timeout

(you may set this to float=?)

#include “nxcBoost/debug-com.nxc“

void initComDebug(const byte outbox = 0, const unsigned long sendTime = 500)

Initialize the comunication debug module with standard parameters

const byte outbox = 0 ... see outbox const.

const unsigned long sendTime = 500 ... default wait time after sending debug message

void setComDebugSendTime(const unsigned long sendTime)

Set standard wait time after sending a debug message

-

void setComDebugOutbox(const byte outbox)

Set standard debug outbox

-

inline void sendDebug(const string &message,

unsigned int class = NA,

unsigned long time = NA)

Send a debug message and wait for "time".

Read class description!

-

inline void sendDebugNum(const float &number,

unsigned int class = NA,

unsigned long time = NA)

Send a debug number and wait for "time".

Read class description!

-

inline void sendDebugFormat(const string &format,

const float &number,

unsigned int class = NA,

unsigned long time = NA)

Send a debug formated string and wait for "time".

Read class description!

-

#include “nxcBoost/debug-„… “

(#include “nxcBoost/class.nxc“)

inline void initClass(const unsigned int instances = 0)

Initialize the class module.

Specifie how many classes you want to have in your project

const unsigned int instances = 0 ... number of classes you want to use. 1-max int

inline void setClassFlag(const unsigned int class, const bool flag)

Set a class flag so that the messages from this class are blocked or not blocked.

This function does nothing if the class is not a valid class!

const unsigned int class ... class number

const bool flag ... true for allow messages

inline void setClassScopeBegin(const unsigned int class)

All functions with standard parameters for class use from scope begin to scope end the specified class. Do not mix scopes! (this will lead to confusing code) This function do not check if the passed value is valid! If theres a file error -1, class value is maby to high!

-

inline void setClassScopeEnd()

All functions with standard parameters for class use from scope begin to scope end the specified class. Do not mix scopes! (this will lead to confusing code)

-

inline bool classHandler(unsigned int class)

You dont need this function if you use the nxcBoost debugger functions.

This function returns if the specified class is active.

-