FTmodule

Author: Jeroen Regtien
Version: V1.0
December 2025

Arduino Library to control module parallel and serial fischertechnik computing interfaces. Original code by Jeroen Regtien, December 2023 with revisions since.

Supported Parallel interfaces: Universal (30520), CVK (66843), Centronics(30566)

Supported Serial interface: Intelligent (30402), ROBO (93293)

Supported Controllino PLCs: MINI, MAXI Automation, MICRO

Supported Shields: Dicacta UNO, Didacta MEGA, Adafruit Motorshield, ftNano shield

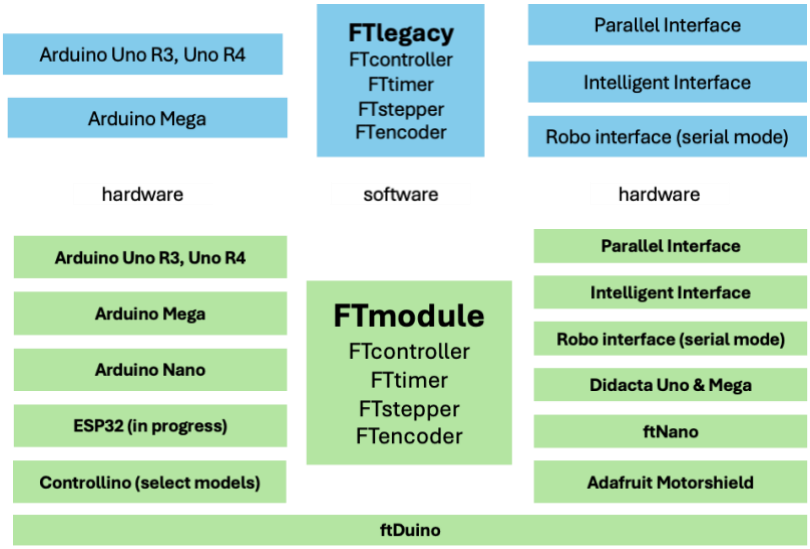
Supported Arduino's: UNO R3, UNO R4 (Minima/Wifi), MEGA, Nano

The library consists of two files: **FTmodule.h** and **FTmodule.cpp**

Table of Contents

Introduction

The FTmodule library is an extension of the FTlegacy library allows selected Arduino based microcontrollers to control one or more module fischertechnik interfaces or third part, such as the parallel (universal) or serial(intelligent) interfaces. It is also capable of using Arduino based Controllino PLCs as well as controlling the Didacta shields (Mega and Uno), the Adafruit motorshield and the ftNano shield. It will run fine on Arduino Mega's en Uno's R4, but probably not always on the Uno R3. A special case is the ftDuino, this is already a combination of an Arduino and an Input/Output shield totally designed for fischertechnik and a great solution for those who want to work in the Arduino environment with fischertechnik. The ftDuino has its own 'ecosystem' with excellent documentation and examples and was included in the scope of this project for the sake of portability of my software. The ftDuino can do much more though than the functionality offered in FTmodule.



The following Arduino's are supported: Uno R3, Uno R4, Mega, Nano, ftDuino

This library contains the follow classes:

FTmodule	*one interface*
FTcontroller	*the controller controlling one of more interfaces*
FTtimer	*timer utility functions*
FTstepper	*single stepper motor*
FTstepperXY	*two stepper motors in plotter or CNC configuration*
FTencoderMotor	*single encoder motor*

This library has been described in:

https://www.ftcommunity.de/ftpedia/2025/2025-4/ftpedia-2025-4.pdf

Hardware

fischertechnik parallel interface

https://www.ftcommunity.de/ftpedia/2014/2014-1/ftpedia-2014-1.pdf

https://www.ftcommunity.de/ftpedia/2017/2017-2/ftpedia-2017-2.pdf

https://www.ftcommunity.de/ftpedia/2017/2017-3/ftpedia-2017-3.pdf

https://www.ftcommunity.de/ftpedia/2017/2017-4/ftpedia-2017-4.pdf

https://www.ftcommunity.de/ftpedia/2023/2023-4/ftpedia-2023-4.pdf

fischertechnik serial interface

https://www.ftcommunity.de/ftpedia/2023/2023-4/ftpedia-2023-4.pdf

https://www.ftcommunity.de/ftpedia/2025/2025-2/ftpedia-2025-2.pdf

ftDuino

When using the ftDuino refer to the following documentation:

https://ftcommunity.de/ftpedia/2018/2018-1/ftpedia-2018-1.pdf#page=85
https://harbaum.github.io/ftduino/www/en/

ftNano

When using the ftNano refer to the following manual:

https://www.obd2-shop.eu/files/ftnano_benutzerhandbuch_1.2.pdf

Also not that often in the Arduino IDE Tools menu, under processor, the ‘old’ bootloader option has to be selected in case the program will not load.

Adafruit

When using an Adafruit Motor Shield V2, more information can be found here: Adafruit Motorshield V2

Didacta

When using the Didacta Mega or Uno shield refer to the information on the Didacta website: http://www.didacta.hr/

Controllino

When using the Controllino Micro, Mini or Maxi Automatin refer to the information on the Controllino website: Controllino

Hierarchical Class Index

This inheritance list is sorted roughly, but not completely, alphabetically:

Class Index

Here are the classes, structs, unions and interfaces with brief descriptions:

FTcontroller Class Reference

`#include <FTmodule.h>`

Public Member Functions

FTcontroller (**ftDisplayTypes** display_type) *Constructor, creates controller.*
void **begin** (char *message) *Initialise Controller.*
void **addInterface** (**FTmodule** ftModule) *Add interface to the list of interfaces.*
int **getNumberInterfaces** () *Retrieve the number of interfaces connected.*
int **getBoard** () *Get the type of the controller board.*
bool **isLCDconnected** () *Checks whether an I2C device is attached, assumes it to be display.*
void **ftMessageToDisplay** (int x, int y, char *messsage, bool clear) *Send text message to LCD*

Detailed Description

FTcontroller class

Class to manage the (micro)controller

Author **Jeroen Regtien**

Date **2024-2025**

Version **1.0**

Depending on the type of controller one or more interfaces can be added.

Constructor & Destructor Documentation

FTcontroller::FTcontroller (ftDisplayTypes display_type) `[inline]`

Constructor, creates controller.

Parameters

display_type	type of display, one of enum **ftDisplayTypes**

Member Function Documentation

void FTcontroller::addInterface (FTmodule ftModule)

Add interface to the list of interfaces.

Parameters

instance	of **FTmodule** class

void FTcontroller::begin (char * message)

Initialise Controller.

Parameters

message	Program name

void FTcontroller::ftMessageToDisplay (int x, int y, char * messsage, bool clear)

Send a text message to the display (if connected)

Parameters

x-index	on LCD display
y-index	on LCD display
**message*	to display
Clear	clear the display before adding message

int FTcontroller::getBoard ()

Get the type of the controller board.

Returns **int** an integer that defines the microcontroller board.

int FTcontroller::getNumberInterfaces ()

Retrieve the number of interfaces connected.

Returns int The number of declared interfaces

bool FTcontroller::isLCDconnected ()

Checks whether an I2C device is attached, assumes it to be display.

Returns bool LCD connected (true or false)

FTencoderMotor Class Reference

Public Member Functions

FTencoderMotor (**FTmodule** *choice, int motorID, FTencoderMode modeChoice, int sensorID)
Constructor.

void **updateEncoder** ()	*updates the encoder counter*
void **begin** ()	*begin and initialise the encoder motor*
bool **setSteps** (int steps)	*set steps and do the accounting*
void **setOrigin** (int origin)	*set the origin*
int **getPosition** ()	*get the current position*
void **setRange** (int origin, int maximum)	*set the range of allowed positions*
bool **moveToOrigin** ()	*move to the origin*
bool **moveToPosition** (int position)	*move to a position*
bool **moveRelative** (int delta)	*incrementally move a number of steps*
bool **findHome** (int endPin, **motorDirection** dir)	*find the home position*
void **setMotorCW** ()	*activate the motor in clock-wise (CW) position*
void **setMotorCCW** ()	*activate the motor in counter-clock-wise (CCW) position*
void **setMotorSTOP** ()	*deactivate the motor*

Constructor & Destructor Documentation

FTencoderMotor::FTencoderMotor (FTmodule * choice, int motorID, FTencoderMode modeChoice, int sensorID)

Constructor.

Parameters

choice	the associated interface
motorID	the interface motor index
modeChoice	E_STD (counters) or E_INT (hardware interrupt)
sensorID	pin for encoder signal

Member Function Documentation

bool FTencoderMotor::findHome (int endPin, motorDirection dir)

find the home position

Parameters

the	pin of the end switch
dir	the idrection the motor has to move in

int FTencoderMotor::getPosition ()

get the current position

Returns

int the position

bool FTencoderMotor::moveRelative (int delta)

incrementally move a number of steps

Parameters

delta	the incremental steps

bool FTencoderMotor::moveToOrigin ()

move to the origin

Returns

bool true when done

bool FTencoderMotor::moveToPosition (int position)

move to a position

Parameters

position	

void FTencoderMotor::setOrigin (int origin)

set the origin

Parameters

origin	new origin

void FTencoderMotor::setRange (int origin, int maximum)

set the range of allowed positions

Parameters

origin	the minimum value
maximum	the maximum value

bool FTencoderMotor::setSteps (int steps)

set steps and do the accounting

Parameters

steps	the number of steps

FTmodule Class Reference

FTmodule class.

```
#include <FTmodule.h>
```

Public Member Functions

FTmodule (int, int) *simpler constructor when connected to consecutive Arduino pins.*
FTmodule () *zero constructor*
bool **begin** () *initialisation of interface instance*

void **getInputs** () *get digital input for all pins and store in buffer*
bool **getInput** (int pin) *get digital input for pin from buffer*
bool **getInput2** (int pin1, int pin2)
get digital input for two pins from buffer and return true if both are pressed
bool **getDirectInput** (int pin*)* *get direct digital input from Arduino pin, no buffer*
void **zeroInput** () *reset input buffer to zero*
void **getAnalogInputs** () *retrieve analog Ex, Ey inputs and store in buffer*
int **getAnalogX** () *return analog X (Ex) value from buffer*
int **getAnalogY** () *return analog Y (Ey) value from buffer*
int **getSerialAnalog** (int xory) *get serial analog input: 0=X, 1=Y*
long int **getParallelAnalog** (int xory) *get parallel analog input, 0=X, 1=Y*
int **getDirectAnalog** (int pin)
get direct analog value from Arduino or shields, if not interface is used
void **printInputBuffer** () *print input buffer to serial monitor*
int **connectedAnalog** () *returns >0 if problems with Ex, Ey*

void **setMotor** (int M, motorDirection dir) *sets motors M to **motorDirection** dir.*
void **setMotorCW** (int M) *sets motor M to **motorDirection** Clockwise*
void **setMotorCCW** (int M) *sets motor M to **motorDirection** Clockwise*
void **setMotorSTOP** (int M) *sets motor M to STOP*
void **setAllMotorsSTOP** () *sets all motors stop.*

void **setMotorSpeed** (int M, int speed) *sets speed of motor M*
motorDirection **getMotor** (int M) *gets **motorDirection** for motor M*
bool **getMotorUntil** (int M, int E, bool until, **motorDirection** dir)
tests whether target end switch was hit whilst moving motor.
bool **setMotorUntil** (int M, int E, bool until, **motorDirection** dir)
keeps motor running until end switch action
bool **setMotorUntilCount** (int M, int E, bool until, **motorDirection** dir, int maxCount)
keeps motor running until switch count is reached
bool **setMotorUntilOrCount** (int M, int E1, bool until1, int E2, bool until2, **motorDirection** dir, int maxCount)
keeps motor running until E1 switch count is reached or switch E2 is triggered

*void **setOutputON** (int O)* *sets output channel to ON*
*void **setOutputOFF** (int O)* *sets output channel to OFF*
*void **setOutput** (int O, int status)* *sets output channel ON or OFF.*
*bool **getOutputUntil** (int O, int E, bool until)*
tests whether target end switch was hit whilst output is active
*bool **setOutputUntil** (int O, int E, bool until)*
keeps motor running until end switch action

bool ***setOutputUntilCount*** *(int O, int E, bool until, int maxCount)*
keeps motor running until switch count is reached

bool ***setOutputUntilOrCount*** *(int O, int E1, bool until1, int E2, bool until2, int maxCount)*
keeps motor running until E1 switch count is reached or switch E2 is triggered

void ***magnetON*** *(int M)*	*switches magnet ON*
void ***magnetOFF*** *(int M)*	*switches magnet OFF*

void ***sendOutputs*** *()*	*sends output buffer to interface*
void ***printOutputBuffer*** *()*	*print output buffer to serial monitor*
void ***setInfo*** *(int level)*	*set information level towards Arduino IDE monitor.*
void ***ftUpdateDisplay*** *()*	*updates LCD display with in- and output status*

Detailed Description

FTmodule class.

Purpose: Class to manage one interface or shield

Version

1.0

Member Function Documentation

bool FTmodule::begin ()

initilisation of interface instance

int FTmodule::connectedAnalog ()

returns >0 if problems with Ex, Ey

void FTmodule::ftUpdateDisplay ()

updates LCD display with current in- and output status information as shown in the figure below

void FTmodule::getAnalogInputs ()

retrieve analog Ex, Ey inputs from interface and store in buffer

int FTmodule::getAnalogX ()

return analog X (Ex) value from buffer

int FTmodule::getAnalogY ()

return analog Y (Ey) value from buffer

int FTmodule::getDirectAnalog (int pin)

get direct analog value from Arduino or shields

Parameters

pin	digital input channel one of ft_E1 to ft_E16

bool FTmodule::getDirectInput (int pin)

get direct digital input from Arduino pin, no buffer

Parameters

pin	Arduino analog pin for Adafruit, E-pin for Didacta

bool FTmodule::getInput (int pin)

get digital input for pin from buffer

Parameters

pin	digital input channel one of ft_E1 to ft_E16

bool FTmodule::getInput2 (int pin1, int pin2)

get digital input for two pins from buffer and return true if both are pressed

Parameters

pin1	digital input channel one of ft_E1 to ft_E16
pin2	digital input channel one of ft_E1 to ft_E16

void FTmodule::getInputs ()

get digital input for all pins and store in buffer

motorDirection FTmodule::getMotor (int M)

gets **motorDirection** for motor M

Parameters

M	motor number

Returns

enum type **motorDirection**

bool FTmodule::getMotorUntil (int M, int E, bool until, motorDirection dir)

Tests whether target end switch was hit whilst moving motor.

Parameters

M	motor number
E	switch number
until	ON/true or OFF/false
dir	direction of type **motorDirection**

Returns

bool returns true of target not hit.

bool FTmodule::getOutputUntil (int O, int E, bool until)

tests whether target end switch was hit whilst output is active

Parameters

O	motor number
E	switch number
until	ON/true or OFF/false

Returns

bool returns true of target not hit.

long int FTmodule::getParallelAnalog (int xory)

get parallel analog input, 0=X, 1=Y

Parameters

xory	choose between X (=0) or Y (=1)

void FTmodule::magnetOFF (int M)

switches magnet OFF

Parameters

M	motor number for magnet, one of ft_M1 to ft_M4

void FTmodule::magnetON (int M)

switches magnet ON

Parameters

M	motor number for magnet, one of ft_M1 to ft_M4

void FTmodule::printInputBuffer ()

print input buffer to serial monitor: Interface number, the digital input data and analog values for EX and EY.

```
16:22:58.685 -> IN: 1 signal: 0 0 0 0 0 0 1 0  Analog x: 110 Analog y: 151
16:22:58.783 -> IN: 1 signal: 0 0 0 0 0 0 1 0  Analog x: 110 Analog y: 151
16:22:58.880 -> IN: 1 signal: 0 0 0 0 0 0 1 0  Analog x: 110 Analog y: 151
16:22:59.008 -> IN: 1 signal: 0 0 0 0 0 0 1 0  Analog x: 110 Analog y: 151
16:22:59.105 -> IN: 1 signal: 0 0 0 0 0 0 1 0  Analog x: 110 Analog y: 151
16:22:59.235 -> IN: 1 signal: 0 0 0 0 0 0 1 0  Analog x: 110 Analog y: 151
16:22:59.331 -> IN: 1 signal: 0 0 0 0 0 0 1 0  Analog x: 110 Analog y: 151
16:22:59.427 -> IN: 1 signal: 0 0 0 0 0 0 1 0  Analog x: 110 Analog y: 151
16:22:59.558 -> IN: 1 signal: 0 0 0 0 0 0 1 0  Analog x: 110 Analog y: 151
16:22:59.655 -> IN: 1 signal: 0 0 0 0 0 0 1 0  Analog x: 110 Analog y: 151
```

void FTmodule::printOutputBuffer ()

print output buffer to serial monitor

void FTmodule::sendOutputs ()

sends output buffer to interface

void FTmodule::setAllMotorsSTOP ()

sets all motors stop.

void FTmodule::setInfo (int level)

set information level towards Arduino IDE monitor.

Parameters

level,can	be 0-3, with 0 meaning no input and 3 maximum information. Impacts performance

void FTmodule::setMotor (int M, motorDirection dir)

sets motors M to **motorDirection** dir.

Parameters

M	motor number
dir	direction of type **motorDirection**

void FTmodule::setMotorCCW (int M)

sets motor M to **motorDirection** Clockwise

Parameters

M	motor number

void FTmodule::setMotorCW (int M)

sets motor M to **motorDirection** Clockwise

Parameters

M	motor number

void FTmodule::setMotorSpeed (int M, int speed)

sets motor M to STOP

Parameters

M	motor number
speed	motor speed 0-255

void FTmodule::setMotorSTOP (int M)

sets motor M to STOP

Parameters

M	motor number

bool FTmodule::setMotorUntil (int M, int E, bool until, motorDirection dir)

keeps motor running until end switch action

Parameters

M	motor number
E	switch number
until	ON/true or OFF/false
dir	direction of type **motorDirection**

Returns

bool returns true of target not hit.

bool FTmodule::setMotorUntilCount (int M, int E, bool until, motorDirection dir, int maxCount)

keeps motor running until switch count is reached

Parameters

M	motor number
E	switch number
until	ON/true or OFF/false
dir	direction of type **motorDirection**
int	maxCount maximum numer of counts

Returns

bool returns false when end point reached

bool FTmodule::setMotorUntilOrCount (int M, int E1, bool until1, int E2, bool until2, motorDirection dir, int maxCount)

keeps motor running until E1 switch count is reached or switch E2 is triggered

Parameters

M	motor number
E1	switch number
until1	ON/true or OFF/false
E2	switch number
until2	ON/true or OFF/false
dir	direction of type **motorDirection**
int	maxCount maximum numer of counts

Returns

bool returns false when end point reached

void FTmodule::setOutput (int O, int status)

sets output channel ON or OFF.

Parameters

O	output number
status	ON (1) or OFF (0)

void FTmodule::setOutputOFF (int O)

sets output channel to OFF

Parameters

O	output number , one of ft_O1 to ft_O16

void FTmodule::setOutputON (int O)

sets output channel to ON

Parameters

O	output number , one of ft_O1 to ft_O16

bool FTmodule::setOutputUntil (int O, int E, bool until)

keeps motor running until end switch action

Parameters

O	motor number
E	switch number
until	ON/true or OFF/false

Returns

bool returns true of target not hit.

bool FTmodule::setOutputUntilCount (int O, int E, bool until, int maxCount)

keeps motor running until switch count is reached

Parameters

O	motor number
E	switch number
until	ON/true or OFF/false

Returns

bool returns false when end point reached

bool FTmodule::setOutputUntilOrCount (int O, int E1, bool until1, int E2, bool until2, int maxCount)

keeps motor running until E1 switch count is reached or switch E2 is triggered

Parameters

O	motor number
E1	switch number
until1	ON/true or OFF/false
E2	switch number
until2	ON/true or OFF/false

Returns

bool returns false when end point reached

FTstepper Class Reference

Public Member Functions

FTstepper *(**FTmodule** &choice, int MA, int MB)*	*the constructor*
*void **setOrigin** (int newOrigin)*	*sets origin*
*void **setRange** (int newMinimum, int newMaximum)*	*sets range for valid position*
void **moveToOrigin** ()	*move to origin*
*void **moveToPosition** (int position)*	*move to position*
*void **moveRelative** (int delta)*	*make relative move of steps*
*void **setStepperSTOP** ()*	*stop motor*

Public Member Functions inherited from FTmodule

See public member functions in FTmodule

Constructor & Destructor Documentation

FTstepper::FTstepper (FTmodule & choice, int MA, int MB) `[inline]`

the constructor

Parameters

choice	the associated interface
MA	motor for coil A
MB	motor for coil B

Member Function Documentation

void FTstepper::moveRelative (int delta)

make relative move of steps

Parameters

delta	number of steps to be made

void FTstepper::moveToPosition (int position)

move to position

Parameters

position	number of steps to be made

void FTstepper::setOrigin (int newOrigin)

sets origin

Parameters

newOrigin	new origin

void FTstepper::setRange (int newMinimum, int newMaximum)

sets range for valid position

Parameters

newMinimum	minimum of range
newMaximum	maximum of range

FTstepperXY Class Reference

Public Member Functions

FTstepperXY *(**FTmodule** &choice, int M1, int M2, int M3, int M4, int pen)*
the constructor

*void **setStepX** (int steps)* — *sets steps in x-direction.*
*void **setStepY** (int steps)* — *sets steps in y-direction*
*void **setStepXY** (int stepX, int stepY)* — *sets step in x- and/or y-direction*
*void **setOrigin** (int origX, int origY)* — *set the origin coordinates*
*bool **findOrigin** (int stopX, int stopY)* — *find the origin using end switches*
*void **setPosition** (int posX, int posY)* — *overwrite the current plotter position*
*void **setArea** (int origX, int origY, int maxX, int maxY)* — *set the range for plotter (step) position*
*void **moveToPosition** (int posX, int posY)* — *move to position*
*void **moveRelative** (int deltaX, int deltaY)* — *relative number of steps*
*void **setSteppersSTOP** ()* — *stop all stepper motors*
*void **penDown** ()* — *activate actuator*
*void **penUp** ()* — *deactivate actuator*
*void **line** (int posX, int posY)* — *draw line from current position*
*void **lineRelative** (int posX, int posY)* — *Send a text message to the display (if connected)*
*void **circle** (int orX, int orY, int radius)* — *draw circle with center (orX,orY) and r=radius*
*void **ellips** (int orX, int orY, int radiusX, int radiusY)*
draw ellips with center (400,200) and radii 200 and 100
*void **box** (int posX, int posY, int sizeX, int sizeY, int hatch)*
draw box at (posX, posY) with size sizX and sizeY
*void **axis** (int xPB, int xPE, int yPB, int yPE, int xAB, int xAE, int yAB, int yAE, bool label, char *xLabel, char *yLabel, char *title, int xInterval, int yInterval, int ticklength)*
draw an x- and y-axis with scales, tickmarks and legends
*void **curve** (int numberOfPoints, int curveX[], int curveY[])* — *draw curve*
*void **plotChar** (int PosX, int PosY, int scale, int direction, char c)* — *plot a character*
*void **plotText** (int PosX, int PosY, int scale, int direction, char *string)*
plot a string of characters

Public Member Functions inherited from FTmodule

See public member functions in FTmodule

Constructor & Destructor Documentation

FTstepperXY::FTstepperXY (FTmodule & choice, int M1, int M2, int M3, int M4, int pen)`[inline]`

the constructor

Parameters

choice	the associated interface
M1	motor for coil A1
M2	motor for coil A2
M3	motor for coil B1
M4	motor for coil B2
pen	motor for actuator (eg pen for plotter)

Member Function Documentation

void FTstepperXY::axis (int xPB, int xPE, int yPB, int yPE, int xAB, int xAE, int yAB, int yAE, bool label, char * xLabel, char * yLabel, char * title, int xInterval, int yInterval, int ticklength)

draw an x- and y-axis with scales, tickmarks and legends

Parameters

int	xPB plot x-origin in plotter coordinates
xPE	plot x-maximum in plotter coordinates
yPB	plot y-origin in plotter coordinates
yPE	plot y-maximum in plotter coordinates
xAB	plot x-origin in scaled graph coordinates
xAE	plot x-maximum in scaled graph coordinates
yAB	plot y-origin in scaled graph coordinates
yAE	plot y-maximum in scaled graph coordinates
label	label yes or no
xlabel	label for x-axis
ylabel	label for y-axis
title	title above the plot
xInterval	x-interval for tickmarks
yInterval	y-interval for tickmarks
ticklength	length of the tickmarks in plotter steps

void FTstepperXY::box (int posX, int posY, int sizeX, int sizeY, int hatch)

draw box at (posX, posY) with size sizX and sizeY

Parameters

orX	origin X
orY	origin Y
radiusX	ellipse x radius
radiusY	ellipse x radius
hatch	not yet implemented

void FTstepperXY::circle (int orX, int orY, int radius)

draw circle with center (orX,orY) and r=radius

Parameters

orX	origin X
orY	origin Y
radius	circle radius

void FTstepperXY::curve (int numberOfPoints, int curveX[], int curveY[])

draw curve

Parameters

numberOfPoints	number of points
curveX	array of length numberOfPoints with x-coordinates
curveY	array of length numberOfPoints with y-coordinates

void FTstepperXY::ellips (int orX, int orY, int radiusX, int radiusY)

draw ellips with center (400,200) and radii 200 and 100

Parameters

orX	origin X
orY	origin Y
radiusX	ellipse x radius
radiusY	ellipse x radius

bool FTstepperXY::findOrigin (int stopX, int stopY)

find the origin using end switches

Parameters

stopX	the input pin for the x end switch
stopY	the input pin for the y end switch

void FTstepperXY::line (int posX, int posY)

draw line from current position

Parameters

posX	target x-position
posY	target y-position

void FTstepperXY::lineRelative (int posX, int posY)

Send a text message to the display (if connected)

Parameters

param

void FTstepperXY::moveRelative (int deltaX, int deltaY)

relative number of steps

Parameters

deltaX	incremental x-steps
deltaY	incremental y-steps

void FTstepperXY::moveToPosition (int posX, int posY)

move to position

Parameters

posX	target x-position
posY	target y-position

void FTstepperXY::plotChar (int PosX, int PosY, int scale, int direction, char c)

plot a character

Parameters

posX	target x-position
posY	target y-position
scale	size of the character (1-4)
direction	direction in which the text is written (1-4)
c	the ascii character

void FTstepperXY::plotText (int PosX, int PosY, int scale, int direction, char * string)

plot a string of characters

Parameters

posX	target x-position
posY	target y-position
scale	size of the character (1-4)
direction	direction in which the text is written (1-4)
string	the ascii text string

void FTstepperXY::setArea (int origX, int origY, int maxX, int maxY)

set the range for plotter (step) position

Parameters

origX	x origin position
origY	y origin position
maxX	maximum x plotter position
maxY	maximum y plotter position

void FTstepperXY::setOrigin (int origX, int origY)

set the origin coordinates

Parameters

origX	new x-origin
origY	new y-origin

void FTstepperXY::setPosition (int posX, int posY)

overwrite the current plotter position

Parameters

posX	new x-position
posY	new y-position

void FTstepperXY::setSteppersSTOP ()

stop all stepper motors

Parameters

deltaX	incremental x-steps
deltaY	incremental y-steps

void FTstepperXY::setStepX (int steps)

Sets steps in x-direction.

Parameters

steps	number of steps to be made

void FTstepperXY::setStepXY (int stepX, int stepY)

sets step in x- and/or y-direction

Parameters

stepX	steps in x-direction
stepY	steps in y-direction

void FTstepperXY::setStepY (int steps)

sets steps in y-direction

Parameters

steps	number of steps to be made

FTtimer Class Reference

FTtimer class.
`#include <FTmodule.h>`

Public Member Functions

- ***FTtimer*** *(unsigned long* ***interval****=0)* — *Constructor, initializes timer.*
- *bool* ***ready*** *()* — *Check if timer is ready.*
- *void* ***interval*** *(unsigned long interval)* — *Set the time interval.*
- *void* ***reset*** *()*

Detailed Description

FTtimer class.

```
\details   Class for simple user defined elegant timers
\author    Kiryanenko
\date     05.10.19
\version   1.0
```

EXAMPLE CODE

FTtimer firstTimer(5000); // Create a timer and specify its interval in milliseconds

secondTimer.interval(3000); // Re-Set an interval to 3 secs for a timer

if (firstTimer.ready()) {...} // check whether a timer is ready second

Timer.reset(); // reset a timer

Constructor & Destructor Documentation

FTtimer::FTtimer (unsigned long interval = 0)`[explicit]`

Constructor, initializes timer.

Parameters

interval	An timing interval in msec

Member Function Documentation

void FTtimer::interval (unsigned long interval)

Set the time interval.

Parameters

interval	An interval in msec

bool FTtimer::ready ()

Check if timer is ready.

Returns

True if is timer is ready

void FTtimer::reset ()

Reset a timer

Macros

- *#define **VERSION** "FT_L V1.0" software library release version*

- *#define **SCREEN_WIDTH** 128*
- *#define SCREEN_HEIGHT 64*

- *#define **ft_E1** 1 standardised E1 input channel*
- *#define **ft_E2** 2 standardised E2 input channel*
- *#define **ft_E3** 3 standardised E3 input channel*
- *#define **ft_E4** 4 standardised E4 input channel*
- *#define **ft_E5** 5 standardised E5 input channel*
- *#define **ft_E6** 6 standardised E6 input channel*
- *#define **ft_E7** 7 standardised E7 input channel*
- *#define **ft_E8** 8 standardised E8 input channel*
- *#define **ft_E9** 9 standardised E1 input channel optional extension interface*
- *#define **ft_E10** 10 standardised E2 input channel optional extension interface*
- *#define **ft_E11** 11 standardised E3 input channel optional extension interface*
- *#define **ft_E12** 12 standardised E4 input channel optional extension interface*
- *#define **ft_E13** 13 standardised E5 input channel optional extension interface*
- *#define **ft_E14** 14 standardised E6 input channel optional extension interface*
- *#define **ft_E15** 15 standardised E7 input channel optional extension interface*
- *#define **ft_E16** 16 standardised E8 input channel optional extension interface*

- *#define **ft_M1** 1 bidirectional Motor 1*
- *#define **ft_M2** 2 bidirectional Motor 2*
- *#define **ft_M3** 3 bidirectional Motor 3*
- *#define **ft_M4** 4 bidirectional Motor 4*
- *#define **ft_M5** 5 bidirectional Motor 1 optional extension interface*
- *#define **ft_M6** 6 bidirectional Motor 2 optional extension interface*
- *#define **ft_M7** 7 bidirectional Motor 3 optional extension interface*
- *#define **ft_M8** 8 bidirectional Motor 1 optional extension interface*

- *#define **ft_O1** 1 standardised output channel 1, using half of motor 1*
- *#define **ft_O2** 2 standardised output channel 2, using the other half of motor 1*
- *#define **ft_O3** 3 standardised output channel 3, using half of motor 2*
- *#define **ft_O4** 4 standardised output channel 4, using the other half of motor 2*
- *#define **ft_O5** 5 standardised output channel 5, using half of motor 3*
- *#define **ft_O6** 6 standardised output channel 6, using the other half of motor 3*
- *#define **ft_O7** 7 standardised output channel 7, using half of motor 4*
- *#define **ft_O8** 8 standardised output channel 8, using the other half of motor 4*
- *#define **ft_O9** 9 standardised output channel 9, using half of extension motor 1*
- *#define **ft_O10** 10 standardised output channel 10, using the other half of extension motor 1*
- *#define **ft_O11** 11 standardised output channel 11, using half of extension motor 2*
- *#define **ft_O12** 12 standardised output channel 12, using the other half of extension motor 2*
- *#define **ft_O13** 13 standardised output channel 13, using half of extension motor 3*
- *#define **ft_O14** 14 standardised output channel 14, using the other half of extension motor 3*
- *#define **ft_O15** 15 standardised output channel 15, using half of extension motor 4*
- *#define **ft_O16** 16 standardised output channel 16, using the other half of extension motor 4*

- *#define **ON** true standardised global ON for interface outputs*
- *#define **OFF** false standardised global OFF for interface outputs*

Enumerations

- *enum **ftDisplayTypes** { **NONE** =0, **D1604** =1, **D1602** =2, **D2004** =3, **D1306L** =4, **D1306S** =5}*
  supported LCD or OLED displays
- *enum **motorDirection** { **CCW** = -1, **STOP** = 0, **CW** = 1 }*
  define type for motor direction
- *enum **typeIFace** { **PAR** = 1, **PAREX** = 2, **SER** = 3, **SEREX** = 4, **ROBO** = 5, **DID_UNO** = 6, **DID_MEGA** = 7, **ADA_UNO** = 8, **CONT_MINI** = 9, **CONT_MAXOUT** = 10, **CONT_MICRO** = 11, **FT_NANO** = 12, **FT_DUINO** = 13 }*
  define interface type
- *enum **FTencoderMode** { **E_STD** =0, **E_INT** =1 }*

Variables

- *const int **UNO** = 1*
- *const int **MEGA** = 2*
- *const int **UNO4** =3*
- *const int **NANO** = 4*
- *const int **ESP32** = 5*
- *const int **OTHER** = 0*
- *const int **board** = OTHER*
- *const int **maxBoards** = 0*
- *const int **lcdColumns** = 20*
- *const int **lcdRows** = 4*

- *hd44780_I2Cexp **lcd*** *define global lcd object*
- *SoftwareSerial **mySerial*** *define global softserial port*
- ***ftDisplayTypes ftDisplayType*** *define global display type*

Enumeration Type Documentation

enum ftDisplayTypes

supported LCD or OLED displays

D1306L : 128x64 pixel 1306 LCD display

D1306S : 128x32 pixel 1306 LCD display

D1604 : 16x4 character LCD display with I2C

D1602 : 16x2 character LCD display with I2C

D2004 : 20x4 character LCD display with I2C

enum motorDirection

define type for motor direction

CCW : CounterClockWise

STOP: stop

CW : ClockWise

enum typeIFace

define interface type

PAR : parallel or universal interface.

PAREX : parallel or universal interface with extension unit.

SER : serial or intelligent interface.

SEREX : serial or intelligent interface with extension unit.

ROBO : Robo interface.

DID_UNO : Didacta Arduino Uno Shield (supported in FTmodule library).

DID_MEGA : Didacta Arduino Mega Shield (supported in FTmodule library).

ADA : Adafruit Motor Shield (supported in FTmodule library).

CONT_MINI: Controllino PLC mini (supported in FTmodule library).

CONT_MAXOUT: Controllino PLC maxi automation (supported in FTmodule library).

CONT_MICRO: Controllino PLC micro (supported in FTmodule library).

FT_NANO : ftNano shield (supported in FTmodule library).

FT_DUINO : ftDuino shield (supported in FTmodule library).

PARALLEL INTERFACES

20-pin ribbon cable pinout

Signal directions relative to Arduino. The pin numbers on the PCB in the 30520 interface are printed backwards (20<->1, 19<->2 etc).

Cable PIN	Aduino PIN	Direction	Function
20 / 1			GND
19 / 2			GND
18 / 3	2	IN	DATA/COUNT IN
17 / 4			Not connected
16 / 5			Connect to pin 14/7
15 / 6			Connect to pin 13/8
14 / 7			Connect from pin 16/5
13 / 8			Connect from pin 15/6
12 / 9	3	OUT	TRIGGER X
11 / 10	4	OUT	TRIGGER Y
10 / 11	5	OUT	DATA OUT
9 / 12	6	OUT	CLOCK
8 / 13	7	OUT	LOAD OUT
7 / 14	8	OUT	LOAD IN
6 / 15			Not connected
5 / 16			Not connected
4 / 17			Not connected
3 / 18			Not connected
2 / 19	19		GND
1 / 20	20		GND

The analog inputs (EX and EY) run from the Fischertechnik side of the interface to the computer and then are looped back to two 556 timers on the interface. Two loopback wires must be installed (between pins 5 and 7, and between pins 6 and 8) on the Arduino side of the gray ribbon cable to make the analog inputs work as expected. It may be possible to connect those pins directly to two analog input pins on the Arduino, but this is not supported by the library.

Serial / Intelligent Interfaces

SERIAL (INTELLIGENT) INTERFACE WITHOUT EXTENSION MODULE

Interface setup: Baudrate=9600, Databits=8, Parity=none, Stopbits=1

Arduino Serial ports:

UNO: TX, RX MEGA: Serial 1 (TX1=18,RX1=19), Serial 2 (TX2=16, RX2=17), Serial 3 (TX3=14, RX3=15), TX0, RX0

To control the interface, two bytes should be sent. The first byte is the interface command (see below) and the second byte contains the motor state.

The motor state describes which motors should be started in which direction. Depending on the interface command, the interface replies with one or three bytes.

First byte: Interface command

Binary	Hex	Decimal	Description
11000001	C1	193	Only I/O state
11000101	C5	197	I/O state and analog value EX
11001001	C9	201	I/O state and analog value EY

Second byte: Motor state

Bits	Description
1	Motor 1 counter-clockwise (CCW)
2	Motor 1 clockwise (CW)
3	Motor 2 counter-clockwise
4	Motor 2 clockwise
5	Motor 3 counter-clockwise
6	Motor 3 clockwise
7	Motor 4 counter-clockwise
8	Motor 4 clockwise

Motors can be controlled in parallel by setting the required bits, however, the CW and CCW modes should not be set at the same time, this invalid command will be ignored. Replies from the interface:

CMD Dec	CMD Hex	CMD bin	Reply bytes	Description
193	0xC1	11000001	one byte	Each bit represents the value of an input
197	0XC5	11000101	three bytes	Byte 1 see above, Byte 2&3 analog Ex
201	0XC9	11001001	three bytes	Byte 1 see above, Byte 2&3 analog Ey

In general, programming should be done in threads. If the interface doesn't get a command every 300 ms, the motors are turned off automatically.

SERIAL INTELLIGENT INTERFACES WITH EXTENSION MODULE

Interface setup: Baudrate=9600, Databits=8, Parity=none, Stopbits=1

To control the extension module, basically three bytes should be sent. The first byte is the interface command (see below), the second byte contains the motor state of the Intelligent Interface (interface 1), and the third byte contains the motor state of the extension module (interface 2).

The motor state describes which motors should be started in which direction. Depending on the interface command, the interface replies with two or four bytes.

The interface commands (193, 197, 201) which control only interface 1, can still be used

First byte: Interface command for interface and extension module.

Binary	Hex	Decimal	Description
11000001	C2	194	Only I/O state of interface 1 and 2
11000101	C6	198	I/O state interface 1 and 2, analog value EX
11001001	CA	202	I/O state interface 1 and 2, analog value EY

Second byte: Motor state for interface 1

Bits	Description
1	Motor 1 counter-clockwise (CCW)
2	Motor 1 clockwise (CW)
3	Motor 2 counter-clockwise
4	Motor 2 clockwise
5	Motor 3 counter-clockwise
6	Motor 3 clockwise
7	Motor 4 counter-clockwise
8	Motor 4 clockwise

Third byte: Motor state for the extension

Bits	Description
1	Motor 1 counter-clockwise (CCW) (motor ft_M5 in the software)
2	Motor 1 clockwise (CW) (motor ft_M5 in the software)
3	Motor 2 counter-clockwise (motor ft_M6 in the software)
4	Motor 2 clockwise (motor ft_M6 in the software)
5	Motor 3 counter-clockwise (motor ft-M7 in the software)
6	Motor 3 clockwise (motor ft_M7 in the software)
7	Motor 4 counter-clockwise (motor ft_M8 in the software)
8	Motor 4 clockwise (motor ft_M8 in the software)

Motors can be controlled in parallel by setting the required bits, however, the CW and CCW modes should not be set at the same time, this invalid command will be ignored. Replies from the interface

Cmd Dec	Cmd Hex	CMD bin	Reply bytes	Description
194	0xC2	11000001	two byte	Byte 1 = Interface 1, Byte 2 = extension.
198	0XC6	11000101	four bytes	Byte 1,2 see above, Byte 3&4 analog Ex interface 1
202	0XCA	11001001	four bytes	Byte 1,2 see above, Byte 3&4 analog Ey interface 1

In general, programming should be done in threads. If the interface doesn't get a

command every 300 ms, the motors are turned off automatically.