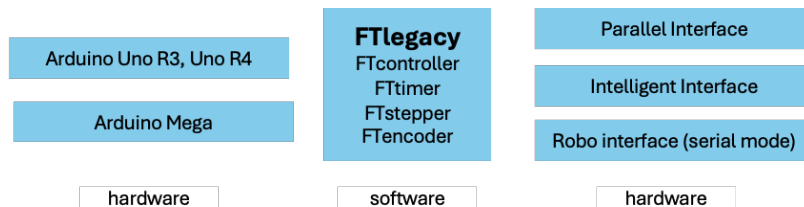


FTlegacy

Author: Jeroen Regtien

Version: V1.0

The FTlegacy library allows selected Arduino models to control one or more legacy fischertechnik interfaces, such as the parallel (universal) or serial(intelligent) interfaces. A description of how to connect these interfaces to an Arduino is provided in the appendices.



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

This library contains the follow classes:

FTlegacy	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>

Additional information:

<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>

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

Table of Contents

FTLEGACY	1
AUTHOR: JEROEN REGTIEN	1
VERSION: V1.0	1
TABLE OF CONTENTS	2
DETAILED DESCRIPTION	3
HIERARCHICAL CLASS INDEX	3
CLASS INDEX	3
FTCONTROLLER CLASS REFERENCE.....	4
<i>Public Member Functions</i>	4
<i>Detailed Description</i>	4
<i>Constructor & Destructor Documentation</i>	4
<i>Member Function Documentation</i>	4
FTENCODERMOTOR CLASS REFERENCE	6
<i>Public Member Functions</i>	6
<i>Constructor & Destructor Documentation</i>	6
<i>Member Function Documentation</i>	6
FTLEGACY CLASS REFERENCE.....	8
<i>Public Member Functions</i>	8
<i>Detailed Description</i>	9
<i>Member Function Documentation</i>	9
FTSTEPPER CLASS REFERENCE	13
<i>Public Member Functions</i>	13
<i>Constructor & Destructor Documentation</i>	13
<i>Member Function Documentation</i>	13
FTSTEPPERXY CLASS REFERENCE	14
<i>Public Member Functions</i>	14
<i>Constructor & Destructor Documentation</i>	14
<i>Member Function Documentation</i>	15
FTTIMER CLASS REFERENCE	18
<i>Public Member Functions</i>	18
<i>Detailed Description</i>	18
<i>Constructor & Destructor Documentation</i>	18
<i>Member Function Documentation</i>	18
MACROS.....	19
ENUMERATIONS	20
VARIABLES	20
ENUMERATION TYPE DOCUMENTATION	20
PARALLEL INTERFACES	21
SERIAL / INTELLIGENT INTERFACES	22
SERIAL (INTELLIGENT) INTERFACE WITHOUT EXTENSION MODULE	22
SERIAL INTELLIGENT INTERFACES WITH EXTENSION MODULE.....	23

Detailed Description

Arduino Library to control legacy 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 PLCs: none

Supported Shields: none

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

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

The MIT License (MIT)

Copyright (c) 2023-2025 Jeroen Regtien

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Hierarchical Class Index

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

FTcontroller	4
FTencoderMotor	6
FTlegacy	9
FTstepper	14
FTstepperXY	16
FTtimer	21

Class Index

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

FTcontroller	4
FTencoderMotor	6
FTlegacy (FTlegacy class)	9
FTstepper	14
FTstepperXY	16
FTtimer (FTtimer class)	21

FTcontroller Class Reference

```
#include <FTlegacy.h>
```

Public Member Functions

- FTcontroller (ftDisplayTypes display_type)

Constructor, creates controller.
- void begin (char *message)

Initialise Controller.
- void addInterface (FTlegacy ftLegacy)

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

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

<i>display_type</i>	type of display, one of enum ftDisplayTypes
---------------------	--

Member Function Documentation

void FTcontroller::addInterface (FTlegacy ftLegacy)

Add interface to the list of interfaces.

Parameters

<i>instance</i>	of FTlegacy class
-----------------	--------------------------

void FTcontroller::begin (char * message)

Initialise Controller.

Parameters

<i>message</i>	Program name
----------------	--------------

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

Send a text message to the display (if connected)

Parameters

<i>x-index</i>	on LCD display
<i>y-index</i>	on LCD display
<i>*message</i>	to display
<i>Clear</i>	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 (FTlegacy *choice, int motorID, FTencoderMode modeChoice, int sensorID)

Constructor.

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

Constructor & Destructor Documentation

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

Constructor.

Parameters

<i>choice</i>	the associated interface
<i>motorID</i>	the interface motor index
<i>modeChoice</i>	E STD (counters) or E INT (hardware interrupt)
<i>sensorID</i>	pin for encoder signal

Member Function Documentation

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

find the home position

Parameters

<i>the</i>	pin of the end switch
<i>dir</i>	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

<i>delta</i>	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

<i>position</i>	
-----------------	--

void FTencoderMotor::setOrigin (int origin)

set the origin

Parameters

<i>origin</i>	new origin
---------------	------------

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

set the range of allowed positions

Parameters

<i>origin</i>	the minimum value
<i>maximum</i>	the maximum value

bool FTencoderMotor::setSteps (int steps)

set steps and do the accounting

Parameters

<i>steps</i>	the number of steps
--------------	---------------------

FTlegacy Class Reference

FTlegacy class.

```
#include <FTlegacy.h>
```

Public Member Functions

FTlegacy (int, int)	<i>simpler constructor when connected to consecutive Arduino pins.</i>
FTlegacy ()	<i>zero constructor</i>
bool begin ()	<i>initialisation of interface instance</i>
void getInputs ()	<i>get digital input for all pins and store in buffer</i>
bool getInput (int pin)	<i>get digital input for pin from buffer</i>
bool getInput2 (int pin1, int pin2)	<i>get digital input for two pins from buffer and return true if both are pressed</i>
void zeroInput ()	<i>reset input buffer to zero</i>
void getAnalogInputs ()	<i>retrieve analog Ex, Ey inputs and store in buffer</i>
int getAnalogX ()	<i>return analog X (Ex) value from buffer</i>
int getAnalogY ()	<i>return analog Y (Ey) value from buffer</i>
int getSerialAnalog (int xory)	<i>get serial analog input: 0=X, 1=Y</i>
long int getParallelAnalog (int xory)	<i>get parallel analog input, 0=X, 1=Y</i>
void printInputBuffer ()	<i>print input buffer to serial monitor</i>
int connectedAnalog ()	<i>returns >0 if problems with Ex, Ey</i>
void setMotor (int M, motorDirection dir)	<i>sets motors M to motorDirection dir.</i>
void setMotorCW (int M)	<i>sets motor M to motorDirection Clockwise</i>
void setMotorCCW (int M)	<i>sets motor M to motorDirection Clockwise</i>
void setMotorSTOP (int M)	<i>sets motor M to STOP</i>
void setAllMotorsSTOP ()	<i>sets all motors stop.</i>
motorDirection getMotor (int M)	<i>gets motorDirection for motor M</i>
bool getMotorUntil (int M, int E, bool until, motorDirection dir)	<i>Tests whether target end switch was hit whilst moving motor.</i>
bool setMotorUntil (int M, int E, bool until, motorDirection dir)	<i>keeps motor running until end switch action</i>
bool setMotorUntilCount (int M, int E, bool until, motorDirection dir, int maxCount)	<i>keeps motor running until switch count is reached</i>
bool setMotorUntilOrCount (int M, int E1, bool until1, int E2, bool until2, motorDirection dir, int maxCount)	<i>keeps motor running until E1 switch count is reached or switch E2 is triggered</i>
void setOutputON (int O)	<i>sets output channel to ON</i>
void setOutputOFF (int O)	<i>sets output channel to OFF</i>
void setOutput (int O, int status)	<i>sets output channel ON or OFF.</i>
bool getOutputUntil (int O, int E, bool until)	<i>tests whether target end switch was hit whilst output is active</i>
bool setOutputUntil (int O, int E, bool until)	<i>keeps motor running until end switch action</i>
bool setOutputUntilCount (int O, int E, bool until, int maxCount)	<i>keeps motor running until switch count is reached</i>
bool setOutputUntilOrCount (int O, int E1, bool until1, int E2, bool until2, int maxCount)	<i>keeps motor running until E1 switch count is reached or switch E2 is triggered</i>

void magnetON (int M)	<i>switches magnet ON</i>
void magnetOFF (int M)	<i>switches magnet OFF</i>
void sendOutputs ()	<i>sends output buffer to interface</i>
void printOutputBuffer ()	<i>print output buffer to serial monitor</i>
void setInfo (int level)	<i>set information level towards Arduino IDE monitor.</i>
void ftUpdateDisplay ()	<i>updates LCD display with in- and output status</i>

Detailed Description

FTLegacy class.

Purpose: Class to manage one interface or shield

Version

1.0

Member Function Documentation

bool FTLegacy::getInput (int pin)

get digital input for pin from buffer

Parameters

<i>pin</i>	digital input channel one of ft_E1 to ft_E16
------------	--

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

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

Parameters

<i>pin1</i>	digital input channel one of ft_E1 to ft_E16
<i>pin2</i>	digital input channel one of ft_E1 to ft_E16

motorDirection FTLegacy::getMotor (int M)

gets **motorDirection** for motor M

Parameters

<i>M</i>	motor number
----------	--------------

Returns

enum type **motorDirection**

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

Tests whether target end switch was hit whilst moving motor.

Parameters

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

Returns

bool returns true of target not hit.

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

tests whether target end switch was hit whilst output is active

Parameters

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

Returns

bool returns true of target not hit.

void FTlegacy::magnetOFF (int M)

switches magnet OFF

Parameters

<i>M</i>	motor number for magnet, one of ft M1 to ft M4
----------	--

void FTlegacy::magnetON (int M)

switches magnet ON

Parameters

<i>M</i>	motor number for magnet, one of ft M1 to ft M4
----------	--

void FTlegacy::printInputBuffer ()

Prints status information to the LCD as follows:

Interface number, the digital input data and analog values for EX and EY

Parameters

	motor number for magnet, one of ft M1 to ft M4
--	--

void FTlegacy::setInfo (int level)

set information level towards Arduino IDE monitor.

Parameters

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

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

sets motors M to **motorDirection** dir.

Parameters

<i>M</i>	motor number
<i>dir</i>	direction of type motorDirection

void FTlegacy::setMotorCCW (int M)

sets motor M to **motorDirection** Clockwise

Parameters

<i>M</i>	motor number
----------	--------------

void FTlegacy::setMotorCW (int M)

sets motor M to **motorDirection** Clockwise

Parameters

<i>M</i>	motor number
----------	--------------

void FTLegacy::setMotorSTOP (int M)

sets motor M to STOP

Parameters

<i>M</i>	motor number
----------	--------------

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

keeps motor running until end switch action

Parameters

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

Returns

bool returns true of target not hit.

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

keeps motor running until switch count is reached

Parameters

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

Returns

bool returns false when end point reached

bool FTLegacy::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

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

Returns

bool returns false when end point reached

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

sets output channel ON or OFF.

Parameters

<i>O</i>	output number
<i>status</i>	ON (1) or OFF (0)

void FTLegacy::setOutputOFF (int O)

sets output channel to OFF

Parameters

<i>O</i>	output number , one of ft O1 to ft O16
----------	--

void FTlegacy::setOutputON (int O)

sets output channel to ON

Parameters

<i>O</i>	output number , one of ft O1 to ft O16
----------	--

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

keeps motor running until end switch action

Parameters

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

Returns

bool returns true of target not hit.

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

keeps motor running until switch count is reached

Parameters

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

Returns

bool returns false when end point reached

bool FTlegacy::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

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

Returns

bool returns false when end point reached

FTstepper Class Reference

Public Member Functions

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

Public Member Functions inherited from FTlegacy

See public member functions under FTlegacy

Constructor & Destructor Documentation

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

the constructor

Parameters

<i>choice</i>	the associated interface
<i>MA</i>	motor for coil A
<i>MB</i>	motor for coil B

Member Function Documentation

void FTstepper::moveRelative (int delta)

make relative move of steps

Parameters

<i>delta</i>	number of steps to be made
--------------	----------------------------

void FTstepper::moveToPosition (int position)

move to position

Parameters

<i>position</i>	number of steps to be made
-----------------	----------------------------

void FTstepper::setOrigin (int newOrigin)

sets origin

Parameters

<i>newOrigin</i>	new origin
------------------	------------

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

sets range for valid position

Parameters

<i>newMinimum</i>	minimum of range
<i>newMaximum</i>	maximum of range

FTstepperXY Class Reference

Public Member Functions

FTstepperXY (FTlegacy &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 FTlegacy

See public member functions under FTlegacy

Constructor & Destructor Documentation

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

the constructor

Parameters

<i>choice</i>	the associated interface
<i>M1</i>	motor for coil A1
<i>M2</i>	motor for coil A2
<i>M3</i>	motor for coil B1
<i>M4</i>	motor for coil B2
<i>pen</i>	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

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

<i>orX</i>	origin X
<i>orY</i>	origin Y
<i>radiusX</i>	ellipse x radius
<i>radiusY</i>	ellipse x radius
<i>hatch</i>	not yet implemented

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

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

Parameters

<i>orX</i>	origin X
<i>orY</i>	origin Y
<i>radius</i>	circle radius

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

draw curve

Parameters

<i>numberOfPoints</i>	number of points
<i>curveX</i>	array of length numberOfPoints with x-coordinates
<i>curveY</i>	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

<i>orX</i>	origin X
------------	----------

<i>orY</i>	origin Y
<i>radiusX</i>	ellipse x radius
<i>radiusY</i>	ellipse x radius

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

find the origin using end switches

Parameters

<i>stopX</i>	the input pin for the x end switch
<i>stopY</i>	the input pin for the y end switch

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

draw line from current position

Parameters

<i>posX</i>	target x-position
<i>posY</i>	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

<i>deltaX</i>	incremental x-steps
<i>deltaY</i>	incremental y-steps

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

move to position

Parameters

<i>posX</i>	target x-position
<i>posY</i>	target y-position

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

plot a character

NOTE: this routine will not work with the Arduino UNO R3 because of the memory requirements to generate the character set.

Parameters

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

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

plot a string of characters

NOTE: this routine will only draw lines and not plot text the Arduino UNO R3 because of the memory requirements to generate the character set.

Parameters

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

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

set the range for plotter (step) position

Parameters

<i>origX</i>	x origin position
<i>origY</i>	y origin position
<i>maxX</i>	maximum x plotter position
<i>maxY</i>	maximum y plotter position

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

set the origin coordinates

Parameters

<i>origX</i>	new x-origin
<i>origY</i>	new y-origin

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

overwrite the current plotter position

Parameters

<i>posX</i>	new x-position
<i>posY</i>	new y-position

void FTstepperXY::setSteppersSTOP ()

stop all stepper motors

Parameters

<i>deltaX</i>	incremental x-steps
<i>deltaY</i>	incremental y-steps

void FTstepperXY::setStepX (int steps)

Sets steps in x-direction.

Parameters

<i>steps</i>	number of steps to be made
--------------	----------------------------

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

sets step in x- and/or y-direction

Parameters

<i>stepX</i>	steps in x-direction
<i>stepY</i>	steps in y-direction

void FTstepperXY::setStepY (int steps)

sets steps in y-direction

Parameters

<i>steps</i>	number of steps to be made
--------------	----------------------------

FTtimer Class Reference

FTtimer class.

```
#include <FTlegacy.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

<i>interval</i>	An timing interval in msec
-----------------	----------------------------

Member Function Documentation

void FTtimer::interval (unsigned long **interval**)

Set the time interval.

Parameters

<i>interval</i>	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 }`
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 }`
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

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).

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.