CSM Label Designer Documentation

Overview

This source code is used to build the Clinical Supplies Management, Inc. Label Designer end-user software, which interfaces with the On-Demand Label Cutting System, US Patent No. US20140238214. It provides a graphical user interface to the On-Demand Label Cutting System on Windows and Linux, allowing the user to create labels to size, preview their placement on the label paper, and send them to the Cutting System via a USB-RS232 connection.

Table of Contents

Version

The most up-to-date version of this source code can always be found at the CSM Label Designer Repository.

This software mainly is built on four main frameworks: the three of which - TinyThread++, SFML 2.2, and SFGUI 0.2.3 - are licensed under the zlib/png license. The fourth library, Serialib, is free of any licenses.

Directory Structure

Files in the project's root directory are organized into six sections for ease of navigation and to reduce clutter. Each is described in detail below.

Root Directory

The project root directory contains the project's **`main.cpp`** file, which defines the program's entry point, its main threads, and its globally defined variables and methods. It also contains the main class files for each thread, **`GUIWindow`** and **`SerialTerminal`** and a Readme file containing basic project information.

`Shares.h` is a file that can be included in any source file that requires access to global variables, as it externs each of them from **`main.cpp`**.

Finally, the project's Makefile is located in the root directory. Its functionality is described below in the Makefile section.

Config Directory

The ./config directory contains all files required for setting up libraries and drivers within the context of the CSM Label Designer software.

`rs232_definitions.cpp` helps setup Serialib use by defining the RS232 baudrate as well as COM ports for Linux and Windows, which are platform-dependent.

Doc Directory

The ./doc directory contains the files required for building this documentation, which relies on Doxygen, a piece of software that allows for the automatic generation of documentation files from specialized comments in the source code (the syntax for Doxygen is rather similar to Oracle's Javadoc). Here, it is used to generate an HTML manual - the very manual you're reading - that explains the code to those unfamiliar with it.

This HTML version of the documentation is built by the **Makefile Options** using the `make doc` command, which generates the new "main documentation page", relative to the main project directory, at `doc/html/index.html` . It is also available for browsing via an HTML embeded frame in the document `./doc/Documentation.html` , which prevents the user from having to sift through the Doxygen html directory, since it's a bit of a mess.

In order to compile the documentation, it is necessary to install Doxygen on the build machine.

Drivers Directory

The ./drivers directory contains code specifically written for the CSM Label Designer.

Here, the **`label_t`** structure, which houses data common to all labels, is defined, as well as the **`shared_data`** class, which incorporates TinyThread++ to allow for atomic access to and thread-safe use of data shared between threads.

The **`dynamicLabel`** class is also defined here, which uses **`label_t`**, SFML 2.2, and SFGUI 0.2.3, to create a drop-in solution to label data storage and access for **`GUIWindow`**.

Libraries Directory

The ./libraries directory houses all code that was obtained for the project from external sources. For more information about the libraries used in the CSM Label Designer software, refer to each library's landing page:

- TinyThread++ - Multi-threading library
- SFML 2.2 - OpenGL graphics library
- SFGUI 0.2.3 - GUI library for SFML
- Serialib - RS232 communication library

It is worth noting that the .so/.dll files contained in the SFML and SFGUI windows/linux directories are not statically included when the program is built, so they *must* be shipped with the CSM Label Designer software for distribution.

Resources Directory

The ./resources directory holds program resources that are required for styling and presentation at runtime. This directory must be shipped with the CSM Label Designer software for distribution.

Makefile Options

The project Makefile is capable of building the project on Windows and Linux systems as well as building this project documentation, cleaning the project directory of temp files, and running the built program. A table of these commands is shown below:

Command	Function
make	Builds the CSM Label Designer software
make run	Runs the software from the command line
make doc	Builds the project documentation (what you're reading)
make clean	Removes all project temp files and any built documentation

Developers

- Ivan Real, an original author of this program and developer of the CSM Label Cutting System's electrical systems
- Lucas Tintikakis, an original author of this program and developer of the CSM Label Cutting System's electrical systems

Acknowledgements

- Lorne Stoops, Nathan Cheadle, and Tony Wang, the Mechanical engineers responsible for the pioneering work on the CSM Label Cutting System

Class List

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

C dynamicLabel	The class that holds all information about a particular user-created label
C GUIWindow	This class handles the drawing of and user interaction with the graphical user interface
C label_t	The structure for storing all relevant label data
C SerialTerminal	The class that handles the opening and closing of, and sending and receiving of data over RS232 interfaces
C shared_data	This template allows for the thread-safe, atomic storage and retrieval of any data sent between two competing tasks

Generated on Sat Mar 17 2018 00:33:31 for CSM Label Designer by doxygen 1.8.11

dynamicLabel Class Reference

The class that holds all information about a particular user-created label. More...

`#include <dynamic_label.h>`

Public Member Functions

	dynamicLabel (**label_t** &labelRef, sfg::Box::Ptr boxPtr, sfg::Entry::Ptr *&entryArrayRef, sf::Vector2f sheetVector)
	~dynamicLabel (void) Destructor for the `dynamicLabel` class.
void	**OnClickDelete** (void) Hide this label from the List of Labels in the GUI, then set it as ready for deletion so it is deleted when the Sheet View `GUIWindow` is opened next.
void	**OnClickEdit** (void) Fill the `GUIWindow` Designer tab entries with the `label_t` structure data stored in this label, then queue this label for deletion.
sfg::Frame::Ptr	**getFrame** (void) Returns a pointer to the List of Labels frame. More...
void	**getLabel** (**label_t** &labelRef) Places the internal label structure in the passed in label reference. More...
bool	**checkRenderPosition** (void) Returns whether or not the `dynamicLabel` render has a position set. More...
void	**unsetRenderPosition** (void) Unsets the render position for the `dynamicLabel`. More...
float	**getPadding** (void) Returns the offset used to space the label from other labels. More...
float	**getInchToPixels** (void) Returns the internal inch-to-pixels conversion factor. More...
void	**setRenderPosition** (float x_offset, float y_offset, float x_sheet_offset, float y_sheet_offset) Sets the draw position of the internal `dynamicLabel` render for when it is drawn in the `GUIWindow` Sheet View tab. More...
sf::Vector2f	**getRenderOffset** (void) Returns the offset used by `GUIWindow` for positioning the render of the label sheet in Sheet View. More...
sf::Vector2f	**getRenderPosition** (void) Returns the internal position for the `dynamicLabel` render. More...
sf::RectangleShape	**getRender** (void) Returns the internal render for the `dynamicLabel`. More...
bool	**checkDeletion** (void) Returns the readiness of the `dynamicLabel` for deletion. More...

Protected Attributes

const int	**InchToCountsFactor** = 2600 This is the conversion factor between encoder counts and inches. More...
const int	**XAxisLength** = 19500 This defines the length of the x-axis cutting area.
const int	**YAxisLength** = 42500 This defines the length of the y-axis cutting area.
float	**inchToPixels** This defines the internal conversion from inches to pixels. More...

Detailed Description

The class that holds all information about a particular user-created label.

It stores all information that the user entered in a `label_t` variable and creates an SFGUI representation of the label for visual presentation in the `GUIWindow` Sheet View tab.

Warning

This class is absolutely dependent on SFML and SFGUI, at the moment

Constructor & Destructor Documentation

```
dynamicLabel::dynamicLabel ( label_t &          labelRef,
                             sfg::Box::Ptr      boxPtr,
                             sfg::Entry::Ptr *& entryArrayRef,
                             sf::Vector2f       sheetVector
                           )
```

Create a table for the new label stuff ///

Member Function Documentation

```
bool dynamicLabel::checkDeletion ( void )
```

Returns the readiness of the `dynamicLabel` for deletion.

Returns

The `dynamicLabel` deletion status

```
bool dynamicLabel::checkRenderPosition ( void )
```

Returns whether or not the `dynamicLabel` render has a position set.

Returns

The label position's set/unset status (true = set, false = unset)

```
sfg::Frame::Ptr dynamicLabel::getFrame ( void )
```

Returns a pointer to the List of Labels frame.

Returns

Pointer to the List of Labels frame.

```
float dynamicLabel::getInchToPixels ( void )
```

Returns the internal inch-to-pixels conversion factor.

Units: inches/pixels

Returns

The conversion factor, a float (units: inches/pixels)

```
void dynamicLabel::getLabel ( label_t & labelRef )
```

Places the internal label structure in the passed in label reference.

Parameters

labelRef A reference `label_t` to store the `dynamicLabel` internal `label_t` data.

float dynamicLabel::getPadding (void)

Returns the offset used to space the label from other labels.

Units: inches

Returns

The padding value, a float (units: inches)

sf::RectangleShape dynamicLabel::getRender (void)

Returns the internal render for the `dynamicLabel`.

Returns

An SFML RectangleShape which reflects the label size entered by the user

sf::Vector2f dynamicLabel::getRenderOffset (void)

Returns the offset used by `GUIWindow` for positioning the render of the label sheet in Sheet View.

Returns

An SFML float vector which reflects the label sheet's offset in the `GUIWindow` Sheet View tab

sf::Vector2f dynamicLabel::getRenderPosition (void)

Returns the internal position for the `dynamicLabel` render.

This is the position of the **upper right corner of the label**.

Returns

An SFML float vector which reflects the label's position in the `GUIWindow` Sheet View tab

```
void dynamicLabel::setRenderPosition ( float  x_offset,
                                       float  y_offset,
                                       float  x_sheet_offset,
                                       float  y_sheet_offset
                                     )
```

Sets the draw position of the internal `dynamicLabel` render for when it is drawn in the `GUIWindow` Sheet View tab.

This takes in both the offset of the label on the label sheet render, as well as the offset that centers the label sheet render on its SFML canvas. It sets the position of the **upper right corner of the label**.

Parameters

x_offset	The label's x offset on the label sheet
y_offset	The label's y offset on the label sheet
x_sheet_offset	The label sheet's x offset on its SFML canvas
y_sheet_offset	The label sheet's y offset on its SFML canvas

void dynamicLabel::unsetRenderPosition (void)

Unsets the render position for the `dynamicLabel`.

This is necessary when the Sheet View has to be redrawn because a label has been removed from the List of Labels.

Member Data Documentation

const int dynamicLabel::InchToCountsFactor = 2600 protected

This is the conversion factor between encoder counts and inches.

The conversion factor for inches to encoder counts. It was calculated from a mean average of the amount of encoder counts required to push the motor an inch from its starting position.

float dynamicLabel::inchToPixels protected

This defines the internal conversion from inches to pixels.

Units: inches/pixels

The documentation for this class was generated from the following files:

- drivers/dynamic_label/**dynamic_label.h**
- drivers/dynamic_label/**dynamic_label.cpp**

GUIWindow Class Reference

This class handles the drawing of and user interaction with the graphical user interface. More...

`#include <gui_window.h>`

Public Member Functions

	GUIWindow (**SerialTerminal** *p_SerialTerminal)	The `GUIWindow` class constructor. More...
void	**OnButtonClick** (void)	This method is used to alert the user that a button's feature is unimplemented.
void	**OnClearClick** (void)	Clears the RS232 Terminal View of all text.
void	**OnClickNewLabel** (void)	Saves the entries in the Designer tab in a `dynamicLabel`. More...
void	**OnClickAddLabel** (void)	Saves the entries in the Designer tab in a `dynamicLabel` and adds it to the List of Labels. More...
void	**OnClickCut** (void)	Actions required when the user clicks the "Send to Cut" button. More...
void	**OnClickRefresh** (void)	Refreshes the COM port list and starts a serial connection with the first COM port found using `p_SerialTerminal`.
void	**OnClickSet** (void)	Sets the COM port and baud rate with the user's choices in `serial_port_combobox` and `serial_baud_combobox`, then attempts to start a serial connection with `p_SerialTerminal`.
void	**OnClickSerialEdit** (void)	Shows the serial connection menu and brings it to the front of the GUI windows.
void	**OnClickConnectionExit** (void)	Hide and remove the "no connection" warning popup.
void	**Run** (void)	This is the "main" equivalent method of a **GUIWindow** object. More...
void	**update** (float delta_time)	Update the GUI, assuming the amount of time passed in has elapsed since the previous update. More...

Protected Member Functions

void	**sendToTerminal** (std::string &string)	Takes an input string reference, sends it to the `outgoingTerminalThread()`, and waits for the outgoing terminal to signal that it has been sent before exiting. More...
bool	**printWaitingString** (void)	Prints an incoming string in the RS232 Terminal View. More...
void	**setupWindow** (void)	Sets up the SFML window used to render the GUI and loads necessary fonts.
void	**populateDesignerTab** (void)	Creates the tabbed interface's "Designer" tab.
void	**populateSheetViewTab** (void)	Creates the tabbed interface's "Sheet View" tab.
void	**populateOptionsTab** (void)	Creates the tabbed interface's "Options" tab.
void	**populateAboutTab** (void)	Creates the tabbed interface's "About" tab.
void	**packTabs** (void)	Adds all tabs to the `GUIWindow::notebook` object, which fills the tabbed interface with the actual UI.
void	**applyDesktopTheme** (void)	

Applies styling to the GUI, making is consistent across platforms and as readable as possible.

void	**createNoConnectionPopup** (void) Creates the No Connection popup displayed if "Send to Cut" button can't find an open serial connection.
void	**createTerminalPopup** (void) Creates the RS232 Terminal popup window.
void	**createSerialPopup** (void) Creates the Serial Connection Menu popup window.

Detailed Description

This class handles the drawing of and user interaction with the graphical user interface.

It uses SFML 2.2 and SFGUI 0.2.3 to draw a user interface that supports rendering visual representations of the designed labels during creation. It also draws each label on a scaled representation of the label sheet to show placement before the user sends any cuts to the Label Cutter.

Todo:

In the future, make sure cut radius is limited < (1/2) of the shortest label side, otherwise they won't be even

Add in range limits for text boxes

Tab-able entries would be nice, too

Constructor & Destructor Documentation

GUIWindow::GUIWindow (SerialTerminal * p_SerialTerminal)

The `GUIWindow` class constructor.

It takes in `p_SerialTerminal` in order to access the data stored in the other thread's `SerialTerminal` object.

Parameters

p_SerialTerminal Pointer to the other main thread's `SerialTerminal` object

Member Function Documentation

void GUIWindow::OnClickAddLabel (void)

Saves the entries in the Designer tab in a `dynamicLabel` and adds it to the List of Labels.

This method reads the Designer tab's entry fields, then creates a new `dynamicLabel` object, pushing it to the back of the std::vector `v_dynamicLabel`.

void GUIWindow::OnClickCut (void)

Actions required when the user clicks the "Send to Cut" button.

This method is the meat of the communication between the Label Designer software and the CSM On-Demand Label Cutter. It pulls in all items from the std::vector `GUIWindow::v_dynamicLabel` and converts them into a format that the Label Cutter's serial connection can read.

Essentially, the routine can be explained as follows:

1. Pop up a progress bar
2. Calibrate head position (currently, this is a static movement from the motor zeroes because the printer isn't attached)
3. Read in the **`dynamicLabel`** position information (upper right corner of the label)
4. Convert the pixel position to an inch position
5. Read the label length and width
6. Construct a string that cuts the label
7. Load the next **`dynamicLabel`** position
8. Compute the distance between the old and new **`dynamicLabel`** positions
9. Construct a string to move the cutting head to the new position
10. Repeat 3 - 8 until (v_dynamicLabel.end())
11. Re-initialize v_dynamicLabel (call destructors in each label, then call v_dynamicLabel.clear())
12. Close the progress bar

Warning
This method pauses the rest of the GUI for as long as it takes to run

Note
There is a commented-out section in this method that will destruct all objects stored in `GUIWindow::v_dynamicLabel`, but it's not included because it's not strictly necessary(?).

void GUIWindow::OnClickNewLabel (void)

Saves the entries in the Designer tab in a **`dynamicLabel`**.

This method reads the Designer tab's entry fields,

bool GUIWindow::printWaitingString (void) protected

Prints an incoming string in the RS232 Terminal View.

It takes an incoming message from the CSM Label Cutting System, parses out the Linux return character, which prints as a box, and sends it to the incoming terminal, then moves down the terminal view to the newest message.

This method also returns whether or not it had any work to do, which can be used to signal if a thread can move on from printing incoming strings.

Note
This is a likely stall point if mutexes and condition_variables break.

void GUIWindow::Run (void)

This is the "main" equivalent method of a GUIWindow object.

It currently creates the entire GUI, then begins the refresh loop, which checks for events and graphical changes on every run through.

This is the method that checks and handles all "OnClick" methods and key presses.

void GUIWindow::sendToTerminal (std::string & string) protected

Takes an input string reference, sends it to the `outgoingTerminalThread()`, and waits for the outgoing terminal to signal that it has been sent before exiting.

This method sends an outgoing message to the `outgoingTerminalThread()` as well as the GUI's RS232 Terminal, which gives the user visual feedback.

Note

If mutexes and condition_variables break, the program is pretty likely to stall here.

void GUIWindow::update (float delta_time)

Update the GUI, assuming the amount of time passed in has elapsed since the previous update.

We use this in the main loop for `GUIWindow::Run()` and to cheat and update the GUI when we're locked in click methods, like `GUIWindow::OnClickCut()`.

Parameters

delta_time	The time that has passed since the previous call

The documentation for this class was generated from the following files:

- gui_window.h
- gui_window.cpp

label_t Struct Reference

The structure for storing all relevant label data. More...

```
#include <label.h>
```

Public Attributes

std::string	**name** The label's name / ID.
std::string	**length** Label length.
std::string	**width** Label width.
std::string	**radius** Label radius.
std::string	**patient** Prescription info: patient name.
std::string	**doctor** Prescription info: doctor name.
std::string	**drug** Prescription info: drug name.
std::string	**instructions** Prescription info: instructions.
std::string	**prescription** Prescription info: prescription #.

Detailed Description

The structure for storing all relevant label data.

The documentation for this struct was generated from the following file:

- drivers/label/**label.h**

SerialTerminal Class Reference

The class that handles the opening and closing of, and sending and receiving of data over RS232 interfaces. More...

`#include <serial_terminal.h>`

Public Member Functions

	SerialTerminal (void) This is the constructor for `SerialTerminal` objects. More...
void	**Run** (void) This is the "main" equivalent method for a `SerialTerminal` object. More...
void	**setPort** (std::string port) Sets the current serial port value, `serial_Port`. More...
std::string	**getPort** (void) Gets the current serial port value, `serial_Port`. More...
void	**setBaud** (int baudrate) Sets the current serial baud rate, `serial_Baud`. More...
int	**getBaud** (void) Gets the current serial baud rate, `serial_Baud`. More...
void	**testPorts** (void) This method runs through the COM port array to test each for a connection. More...
int	**startConnection** (void) This method simply tries to open a serial connection on the requested port and sets the `b_serial_connected` flag. More...

Detailed Description

The class that handles the opening and closing of, and sending and receiving of data over RS232 interfaces.

This class relies heavily on Serialib to do the heavy-lifting in terms of accessing the COM ports.

It contains methods for setting port parameters (`SerialTerminal::setPort()`, `SerialTerminal::setBaud()`) and retrieving port parameters (`SerialTerminal::getPort()`, `SerialTerminal::getBaud()`), as well as finding available COM ports and opening a connection to a particular port.

All handling of closing open ports is done automatically within `SerialTerminal::startConnection()`.

Currently, the methods use a stored list of COM ports and baudrates that are known to be compatible with Serialib. The list of COM ports is platform-specific. Each list is viewable in `rs232_definitions.cpp`.

An example usage of an object of this class is shown below:

```
// Store the default baudrate for connections
SerialTerminal::setBaud(9600);

// Cycle through the default list of COM ports and store any that are viable
// This stores ports in the global std::vector useable_ports
SerialTerminal::testPorts();

// If we don't find any ports, display a message
if ( useable_ports.empty() == true )
{
  // Let the user know the connection failed
  std::cout << "No useable COM ports!" << std::endl;
}

// If we found useable COM ports, continue
else
{
  // Let the user know the connection succeeded and connect to the first port
  std::cout << "Found at least one COM port." << std::endl;

  // Set the serial port with the first value in the COM port list
  SerialTerminal::setPort(useable_ports.front());

  // Open the first serial port listed
  Ret=SerialTerminal::startConnection();

  // If we couldn't open a serial connection, (Ret != 1)
  if (Ret != 1)
  {
    // Let the user know that the port failed to open.
    std::cout << "Error while opening port. Permission problem?" << std::endl;
  }

  // If startConnection returns 1, the port opened and connected successfully
  else
  {
    // Continue on because the connection was successful
  }
}
```

Constructor & Destructor Documentation

SerialTerminal::SerialTerminal (void)

This is the constructor for `SerialTerminal` objects.

It doesn't really do anything because we want control over when the "main" loop, `SerialTerminal::Run()` actually starts.

Member Function Documentation

int SerialTerminal::getBaud (void)

Gets the current serial baud rate, `serial_Baud`.

This method returns the current value of `serial_Baud`.

std::string SerialTerminal::getPort (void)

Gets the current serial port value, `serial_Port`.

This method converts `serial_Port` to a std::string and returns it.

void SerialTerminal::Run (void)

This is the "main" equivalent method for a `SerialTerminal` object.

It checks for COM ports, and opens a connection to the first one it finds (if it finds one - otherwise it waits), then starts the incoming and outgoing terminal threads, `incomingTerminalThread()` and `outgoingTerminalThread()`.

void SerialTerminal::setBaud (int baudrate)

Sets the current serial baud rate, `serial_Baud`.

This method takes an incoming integer and assigns it to `serial_Baud`.

void SerialTerminal::setPort (std::string port)

Sets the current serial port value, `serial_Port`.

This method takes an incoming std::string and converts it to a C string before assigning it to `serial_Port`.

int SerialTerminal::startConnection (void)

This method simply tries to open a serial connection on the requested port and sets the `b_serial_connected` flag.

If a COM port is open before trying to start a new connection, the current connection is closed, which solves some issues with connections in Windows. The terminal is then notified to wake up because it should receive the Label Cutter's main menu upon a successful connection.

void SerialTerminal::testPorts (void)

This method runs through the COM port array to test each for a connection.

Running this method wipes the global std::vector `useable_ports`, then checks through each string in the `com_port_array` listed in **`rs232_definitions.cpp`**.

Before the first test and after all subsequent tests, the COM port of interest is closed and then opened. This solves some issues with connections in Windows.

The documentation for this class was generated from the following files:

- **serial_terminal.h**
- **serial_terminal.cpp**

shared_data< dataType > Class Template Reference

This template allows for the thread-safe, atomic storage and retrieval of any data sent between two competing tasks. More...

`#include <shared_data.h>`

Public Member Functions

	shared_data (void)
	The constructor for the the **shared_data** template/class. More...
bool	**set** (dataType &data)
	Sets the contents of the **shared_data** object. More...
bool	**get** (dataType &data)
	Gets the contents of the **shared_data** object. More...
bool	**peek** (dataType &data)
	Peeks at the contents of the **shared_data** object without waking up (tthread::condition_variable::notify()) anything. More...

Protected Attributes

tthread::mutex	**m**
	A TinyThread++ mutex, used to lock the shared data from multiple simultaneous accesses.
bool	**b_writing**
	A flag that signified a writing operation is taking place.
tthread::condition_variable	**cond_writing**
	A TinyThread++ condition_variable, which is used in conjunction with `shared_data::m` to lock access attempts while data writing is taking place.
bool	**b_reading**
	A flag that signified a reading operation is taking place.
tthread::condition_variable	**cond_reading**
	A TinyThread++ condition_variable, which is used in conjunction with `shared_data::m` to lock access attempts while data reading is taking place.
dataType	**the_shared_data**
	The actual data being stored in the **shared_data** type. This is set by the <> argument to the template.

Detailed Description

template<class dataType>
class shared_data< dataType >

This template allows for the thread-safe, atomic storage and retrieval of any data sent between two competing tasks.

Given that it's a template, a new `shared_data` object can be created with the command:

```
shared_data<some_type> the_share_name;
```

A usage example with a string is shown below:

```
// Create a string to reference and a shared_data object to store it
std::string string = "The String.";
shared_data<std::string> shared_string;

// Set the data in the shared string
shared_string.set(&string);

// ...

// Retrieve the possibly changed string
shared_string.get(&string);
```

Because `shared_data::set()`, `shared_data::get()`, and `shared_data::peek()` return booleans that indicate their success or failure, they can be used for execution control:

```
// Act on the success/failure of a get command
if (shared_string.get(&string) == true)
```

```
{
  // Action on successful get
}
else
{
  // Action on failed get
}
```

```
// If a set is unsuccessful, keep trying until it works
bool b_try_set = shared_string.set(&string);
while (b_try_set == false)
{
  b_try_set = shared_string.set(&string);
}
```

Constructor & Destructor Documentation

template<class dataType >

shared_data< dataType >::shared_data (void)

The constructor for the the **shared_data** template/class.

Template Parameters

dataType The input data type

Member Function Documentation

template<class dataType >

bool shared_data< dataType >::get (dataType & data)

Gets the contents of the **shared_data** object.

Parameters

data A reference to the storage location for the data being retrieved

Returns

The success (true) or failure (false) of the get operation

template<class dataType >

bool shared_data< dataType >::peek (dataType & data)

Peeks at the contents of the **shared_data** object without waking up (tthread::condition_variable::notify()) anything.

Parameters

data A reference to the storage location for the data being retrieved

Returns

The success (true) or failure (false) of the get operation

template<class dataType >

bool shared_data< dataType >::set (dataType & data)

Sets the contents of the **shared_data** object.

Parameters

data A reference to the data being stored

Returns

The success (true) or failure (false) of the set operation

The documentation for this class was generated from the following file:

- drivers/shared_data/**shared_data.h**