ScStw shared libraries

Generated by Doxygen 1.9.4

Chapter 1

ScStw Libraries documentation

1.1 Introduction

This library is meant for usage with the Speed climbing stopwatch project. It contains some helper classes to build a client application for the ScStw basestation with Qt.

1.2 Installation

```
cd yourRepo
git submodule add https://git.itsblue.de/ScStw/shared-libraries/
git submodule update --init --recursive
```

And in your MyProject.pro include the .pri file:

```
include($$PWD/shared-libraries/ScStwLibraries/ScStwLibraries.pri)
```

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 ScStw Class Reference

The ScStw class provides some shared functions and enums for use in the ScStw project.

`#include <ScStw.hpp>`

Inheritance diagram for ScStw:

5.2 ScStwLibraries Class Reference

Inheritance diagram for ScStwLibraries:

Collaboration diagram for ScStwLibraries:

Static Public Member Functions

- static void **init** ()

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

- /drone/src/ScStwLibraries/headers/scstwlibraries.h
- /drone/src/ScStwLibraries/sources/scstwlibraries.cpp

5.3 ScStwRace Class Reference

The ScStwRace class can be used to measure timings of climbing races with multiple lanes at once.

`#include <scstwrace.h>`

Inheritance diagram for ScStwRace:

Collaboration diagram for ScStwRace:

Public Types

- enum **RaceState** {
  IDLE , **PREPAIRING** , **WAITING** , **STARTING** ,
  RUNNING , **STOPPED** , **INCIDENT** }

Public Slots

- virtual ScStw::StatusCode start (bool asyncronous=true)

  Function to start the race.
- virtual ScStw::StatusCode stop ()

  Function to stop the currently running race.
- virtual ScStw::StatusCode reset ()

  Function to reset a stopped race.
- virtual ScStw::StatusCode **cancel** ()
- virtual ScStw::StatusCode **setTimerDisabled** (int id, bool disabled)
- virtual Q_INVOKABLE bool **addTimer** (ScStwTimer ∗timer)
- RaceState **getState** ()
- virtual QVariantMap **getCurrentStartDelay** ()
- QList< ScStwTimer ∗ > **getTimers** ()
- QVariantList **getTimerDetailList** ()
- QVariantMap **getDetails** ()
- bool **getCompetitionMode** ()
- virtual bool **getReadySoundEnabled** ()
- ScStwSettings ∗ **getSettings** ()
- void **setSettings** (ScStwSettings ∗settings)
- bool **getAutoRefreshTimerText** ()
- void **setAutoRefreshTimerText** (bool autoRefresh)

Signals

- void **startTimers** ()
- void **stopTimers** (int type)
- void **resetTimers** ()
- void **stateChanged** (RaceState state)
- void **currentStartDelayChanged** ()
- void **timersChanged** ()
- void **isReadyForNextStateChanged** ()
- void **detailsChanged** ()
- void **competitionModeChanged** ()
- void **readySoundEnabledChanged** ()
- void **settingsChanged** ()
- void **autoRefreshTimerTextChanged** ()

Public Member Functions

- **ScStwRace** (QObject ∗parent=nullptr)
- **ScStwRace** (ScStwSettings ∗settings, QObject ∗parent=nullptr)

Protected Member Functions

- void **setState** (RaceState newState)

Protected Attributes

- QList< ScStwTimer ∗ > **timers**

Properties

- RaceState **state**
- QVariantList **timers**
- QVariantMap **currentStartDelay**
- bool **isReadyForNextState**
- bool **competitionMode**
- bool **readySoundEnabled**
- QVariantMap **details**
- ScStwSettings ∗ **settings**
- bool **autoRefreshTimerText**

Friends

- class **ScStwRemoteRace**

5.3.1 Detailed Description

The ScStwRace class can be used to measure timings of climbing races with multiple lanes at once.

The ScStwRace is a container to manage multiple timers at a time and introduces a propper start sequence with start commands ('At your Marks' and 'Ready') and the official IFSC start signal.

Basic usage:

```
ScStwRace race;
// add two timers
race.addTimer(new ScStwTimer());
race.addTimer(new ScStwTimer());
// start a race
race.start();
```

5.3.2 Member Function Documentation

5.3.2.1 reset

```
ScStw::StatusCode ScStwRace::reset ( )  [virtual], [slot]
```

Function to reset a stopped race.

Returns

5.3.2.2 start

```
ScStw::StatusCode ScStwRace::start (
            bool asyncronous = true )  [virtual], [slot]
```

Function to start the race.

Parameters

asyncronous	if the function should just start the start sequence and then quit (true) or if if should wait until the start sequence is over and quit after that (false)

Returns

200: OK; 904: state not matching

5.3.2.3 stop

```
ScStw::StatusCode ScStwRace::stop ( )  [virtual], [slot]
```

Function to stop the currently running race.

Returns

200: OK; 904: state not matching

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

- /drone/src/ScStwLibraries/headers/scstwrace.h
- /drone/src/ScStwLibraries/sources/scstwrace.cpp

5.4 ScStwSetting Class Reference

Inheritance diagram for ScStwSetting:

Collaboration diagram for ScStwSetting:

Public Slots

- QVariant **getValue** ()
- void **setValue** (QVariant value)

Signals

- void **valueChanged** ()

Protected Slots

- void **handleSettingChange** (int key, int keyLevel, QVariant value)

Protected Member Functions

- **ScStwSetting** (int key, int keyLevel, ScStwSettings ∗scStwSettings, QObject ∗parent)

Protected Attributes

- int **key**
- int **keyLevel**
- bool **hasToReload**

Properties

- QVariant **value**
- QVariant **readonlyValue**

Friends

- class **ScStwSettings**

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

- /drone/src/ScStwLibraries/headers/scstwsetting.h
- /drone/src/ScStwLibraries/sources/scstwsetting.cpp

5.5 ScStwSettings Class Reference

Inheritance diagram for ScStwSettings:

Collaboration diagram for ScStwSettings:

Public Types

- enum BaseStationSetting {
  InvalidSetting = -1 , **ReadySoundEnableSetting** , **ReadySoundDelaySetting** , **AtYourMarksSound**↩
  EnableSetting ,
  AtYourMarksSoundDelaySetting , **SoundVolumeSetting** , **CompetitionModeSetting** }

  The BaseStationSetting enum contains all settings of the base station that can be changed by a client.
- enum **KeyLevelEnum** { **KeyLevel** = 0 }
- typedef QString(∗ **keyToStringConverter**) (int)
- typedef QVariant::Type(∗ **keyToTypeConverter**) (int)

Signals

- void **settingChanged** (int key, int keyLevel, QVariant value)

Public Member Functions

- **ScStwSettings** (QObject ∗parent=nullptr, bool overwriteFileOnErrors=true)
- virtual QVariant **readSetting** (BaseStationSetting key)
- virtual Q_INVOKABLE QVariant **readSetting** (int key, int keyLevel)
- virtual bool **writeSetting** (BaseStationSetting key, QVariant value)
- virtual Q_INVOKABLE bool **writeSetting** (int key, int keyLevel, QVariant value)
- virtual bool **setDefaultSetting** (BaseStationSetting key, QVariant defaultValue)
- virtual Q_INVOKABLE bool **setDefaultSetting** (int key, int keyLevel, QVariant defaultValue)
- Q_INVOKABLE ScStwSetting ∗ **getSetting** (int key, int keyLevel)

Static Public Member Functions

- static BaseStationSetting **keyFromInt** (int i)
- static QString **keyToString** (int key)
- static QVariant::Type **keyToType** (int key)

Protected Member Functions

- virtual QVariant **readSetting** (QString key, int keyInt=-1, int keyLevel=-1)
- virtual bool **writeSetting** (QString key, QVariant value, int keyInt=-1, int keyLevel=-1)
- virtual bool **setDefaultSetting** (QString key, QVariant defaultValue, int keyInt, int keyLevel=-1)
- bool **registerKeyLevelConverters** (int keyLevel, keyToStringConverter, keyToTypeConverter)

5.5.1 Member Enumeration Documentation

5.5.1.1 BaseStationSetting

```
enum ScStwSettings::BaseStationSetting
```

The BaseStationSetting enum contains all settings of the base station that can be changed by a client.

See also

ScStw::baseStationSettingFromInt()

ScStw::baseStationSettingToString()

ScStw::baseStationSettingFromString()

ScStw::baseStationSettings

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

- /drone/src/ScStwLibraries/headers/scstwsettings.h
- /drone/src/ScStwLibraries/sources/scstwsettings.cpp

5.6 ScStwSoundPlayer Class Reference

The ScStwSoundPlayer class is used for ultra low latency sound playback of the speed clibing start tones and commands.

```
#include <scstwsoundplayer.h>
```

Inheritance diagram for ScStwSoundPlayer:

Collaboration diagram for ScStwSoundPlayer:

Public Types

- enum **StartSound** {
  None = -1 , **AtYourMarks** = 0 , **Ready** = 1 , **Start** = 2 ,
  FalseStart = 3 }
- enum **PlayResult** { **Success** = 0 , **Cancelled** = -1 , **Error** = -2 }

Public Slots

- ScStwSoundPlayer::PlayResult play (StartSound sound, double volume, double ∗timeOfStart=nullptr)

  Function to begin playing the sound of a certain state.
- ScStwSoundPlayer::PlayResult waitForSoundFinish (double ∗timeOfStop=nullptr)

  Function to wait for the playback to finish.
- bool cancel ()

  Function to cancel the current playback.
- bool **isPlaying** ()

Signals

- void **playbackStarted** ()

  Emitted whenever a playback started.

Public Member Functions

- ScStwSoundPlayer (QObject ∗parent=nullptr)

  ScStwSoundPlayer constructor.

5.6.1 Detailed Description

The ScStwSoundPlayer class is used for ultra low latency sound playback of the speed clibing start tones and commands.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 ScStwSoundPlayer()

```
ScStwSoundPlayer::ScStwSoundPlayer (
            QObject * parent = nullptr )  [explicit]
```

ScStwSoundPlayer constructor.

Parameters

parent	

5.6.3 Member Function Documentation

5.6.3.1 cancel

```
bool ScStwSoundPlayer::cancel ( )  [slot]
```

Function to cancel the current playback.

Note that this function will automatically play the false start tone if the currently playing action is 2

Parameters

volume	the volume to play the false start sound at

Returns

true if the playback was successfully stopped, false otherwise

5.6.3.2 play

```
ScStwSoundPlayer::PlayResult ScStwSoundPlayer::play (
            ScStwSoundPlayer::StartSound sound,
            double volume,
            double * timeOfStart = nullptr )  [slot]
```

Function to begin playing the sound of a certain state.

Parameters

action	The action to play (0: AtYourMarks, 1:Ready, 2:Start)
volume	The volume to play at
timeOfStop	The time the playback actually started (msecs since epoch)

Returns

TODO true if the playback was successfully started, false otherwise

5.6.3.3 waitForSoundFinish

```
ScStwSoundPlayer::PlayResult ScStwSoundPlayer::waitForSoundFinish (
            double * timeOfStop = nullptr )  [slot]
```

Function to wait for the playback to finish.

Parameters

timeOfStop	the point in time when the plyback actually stopped (msecs since epoch)

Returns

false if there was any error (eg. there was no playback currently), true otherwise

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

- /drone/src/ScStwLibraries/headers/scstwsoundplayer.h
- /drone/src/ScStwLibraries/sources/scstwsoundplayer.cpp

5.7 ScStwStartSoundPlayer Class Reference

Inheritance diagram for ScStwStartSoundPlayer:

Collaboration diagram for ScStwStartSoundPlayer:

Public Slots

- bool **play** (double volume, double ∗timeOfStop=nullptr)

Public Member Functions

- **ScStwStartSoundPlayer** (QObject ∗parent=nullptr)

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

- /drone/src/ScStwLibraries/headers/scstwstartsoundplayer.h
- /drone/src/ScStwLibraries/sources/scstwstartsoundplayer.cpp

5.8 ScStwTimer Class Reference

The ScStwTimer class is used for advanced time measurement.

`#include <scstwtimer.h>`

Inheritance diagram for ScStwTimer:

Collaboration diagram for ScStwTimer:

Public Types

- enum TimerState {
  IDLE , STARTING , RUNNING , WAITING ,
  WON , LOST , FAILING , WILDCARD ,
  FAILED , CANCELLED , INCIDENT , DISABLED }

  The TimerState enum contains all state the timer can be in.
- enum ReadyState {
  IsReady = 0 , NotInIdleState , IsDisabled , ExtensionIsNotConnected ,
  ExtensionBatteryIsCritical , ClimberIsNotReady }

  The ReadyStatus enum contains all possible reasons for a timer not to be ready (>0) and the case that it is ready (0)

Public Slots

- bool start ()

  Function to start the timer.
- virtual bool start (double timeOfStart)

  Function to start the timer at a given point in time (present or future)
- bool cancel ()

  Function to cancel the timer.
- bool stop ()

  Function to stop the timer.
- bool stop (double timeOfStop)

  Function to stop the timer at a given point in time (past or future)
- bool setResult (TimerState)

  Function to assing the result of the race to the timer.
- virtual bool reset ()

  Function to reset the timer.
- TimerState getState ()

  Function to get the current state of the timer.
- double getCurrentTime ()

  Function to get the current time of the timer.
- double getReactionTime ()

  Function to get the reaction time of the climber.
- QString getText ()

  Function to get the text, a timer display is supposed to show.
- QString getLetter ()

  Function to get the letter of the timer.
- void setDisabled (bool disabled)

  Function to set if the timer is supposed to be disabled.
- bool getWantsToBeDisabled ()

  Function to check if the timer currently wants to be disabled.
- virtual ScStwTimer::ReadyState getReadyState ()

  Function to get the current ready status of a timer.
- bool **isRunning** ()
- bool **isDisabled** ()

Signals

- void **stateChanged** (TimerState state)

  Emitted when the state of the timer changed.
- void **reactionTimeChanged** ()

  Emitted when the reaction time changed.
- void wantsToBeDisabledChanged (ScStwTimer ∗timer, bool wantsToBeDisabled)

  Emitted when the timer wants its state to be changed by the external handler.
- void readyStateChanged (ReadyState readyState)

  Emitted when the ready state of the timer changes.

Public Member Functions

- **ScStwTimer** (QObject ∗parent=nullptr)
- ScStwTimer (QString letter, QObject ∗parent=nullptr)

  ScStwTimer constructor.
- **Q_ENUM** (TimerState)

Protected Slots

- void handleClimberStart (double timeOfStart)

    slot to call when the climber has started
- void setState (TimerState newState)

    Function to change the state of the timer.
- void setWantsToBeDisabled (bool wantsToBeDisabled)

    Function to set whether the timer currently wants to be disabled.
- void technicalIncident ()

    Function to set the timer into INCIDENT state immidieately.
- bool wildcard ()

    Function to set the timer into WILDCARD state.

Protected Attributes

- TimerState **state**

    The current state of the timer.
- double **startTime**

    The time the timer was started at.
- double **stopTime**

    The time the timer was stopped at.
- double **reactionTime**

    the reaction time of the climber
- QString **letter**

    The letter (eg. "A" or "B") of the Timer (only one char)
- bool **wantsToBeDisabled**

    Defines if the timer currently wants to be disabled or not.

Friends

- class **ScStwRace**

5.8.1 Detailed Description

The ScStwTimer class is used for advanced time measurement.

It does not work on its own though. It is recommended to use it in combination with the ScStwRace class.

When using standalone:

```
ScStwTimer timer;
// start the timer
timer.start();
// stop the timer
timer.stop();
```

The timer will now go into ScStw::WAITING state. That indicates that the timer has stopped and the final result has to be assigned by an external handler.

```
// assign result 'won'
timer.setResult(ScStwTimer::WON);
```

The timer is now in ScStwTimer::WON state.

```
// reset the timer
timer.reset();
```

The timer is not in ScStwTimer::IDLE state again.

5.8.2 Member Enumeration Documentation

5.8.2.1 ReadyState

```
enum ScStwTimer::ReadyState
```

The ReadyStatus enum contains all possible reasons for a timer not to be ready (>0) and the case that it is ready (0)

Enumerator

IsReady	Timer is ready for start
NotInIdleState	Timer is not in IDLE state
IsDisabled	Timer is disabled
ExtensionIsNotConnected	Not all extension of the timer are conneted
ExtensionBatteryIsCritical	The battery level of one or more extension is critical or unknown
ClimberIsNotReady	The startpad of the timer is not triggered

5.8.2.2 TimerState

```
enum ScStwTimer::TimerState
```

The TimerState enum contains all state the timer can be in.

Enumerator

IDLE	Timer is waiting to be started
STARTING	Timer is starting and will react with a false start if the climber starts
RUNNING	Timer is running
WAITING	Timer was stopped and is waiting to be set to either WON or LOST
WON	Timer has won
LOST	Timer has lost
FAILING	Timer encountered a false start and is waiting to be set to either FAILED or WILDCARD
WILDCARD	The opponent has done a false start
FAILED	A false start occured
CANCELLED	Timer was cancelled
INCIDENT	There was a technical incident (most likely a disconnected extension)
DISABLED	Timer is disabled

5.8.3 Constructor & Destructor Documentation

5.8.3.1 ScStwTimer()

```
ScStwTimer::ScStwTimer (
            QString letter,
            QObject * parent = nullptr )  [explicit]
```

ScStwTimer constructor.

Parameters

parent	the parent object
directControlEnabled	Defines if protected properties (startTimer, stopTime, reactionTime and state) can be directly set from outside.
letter	the letter of the timer (only first char will be used!)

5.8.4 Member Function Documentation

5.8.4.1 cancel

```
bool ScStwTimer::cancel ( )  [slot]
```

Function to cancel the timer.

To do this, the timer has to be in ScStwTimer::IDLE, ScStwTimer::STARTING or ScStwTimer::RUNNING state!

Returns

false if the timer was not in the required state and therefore not cancelled, true otherwise

5.8.4.2 getCurrentTime

```
double ScStwTimer::getCurrentTime ( )  [slot]
```

Function to get the current time of the timer.

To do this, the timer has to be in ScStwTimer::RUNNING, ScStwTimer::WAITING, ScStwTimer::WON or ScSTw::↩
LOST state!

Returns

The current / final time of the timer or -1 if it is not in the required state

5.8.4.3 getLetter

```
QString ScStwTimer::getLetter ( )  [slot]
```

Function to get the letter of the timer.

Returns

The letter of the timer or ""

5.8.4.4 getReactionTime

```
double ScStwTimer::getReactionTime ( )  [slot]
```

Function to get the reaction time of the climber.

Returns

The climbers reaction time

5.8.4.5 getReadyState

```
ScStwTimer::ReadyState ScStwTimer::getReadyState ( )  [virtual], [slot]
```

Function to get the current ready status of a timer.

Returns

The current ready status

5.8.4.6 getState

```
ScStwTimer::TimerState ScStwTimer::getState ( )  [slot]
```

Function to get the current state of the timer.

Returns

current state of the timer

See also

ScStwTimer::TimerState

5.8.4.7 getText

```
QString ScStwTimer::getText ( )  [slot]
```

Function to get the text, a timer display is supposed to show.

Returns

The text to show

5.8.4.8 getWantsToBeDisabled

```
bool ScStwTimer::getWantsToBeDisabled ( )  [slot]
```

Function to check if the timer currently wants to be disabled.

Returns

true or false

5.8.4.9 handleClimberStart

```
void ScStwTimer::handleClimberStart (
            double timeOfStart )  [protected], [slot]
```

slot to call when the climber has started

Parameters

timeOfStart	time (msecs since epoch) when the climber started

5.8.4.10 readyStateChanged

```
void ScStwTimer::readyStateChanged (
            ReadyState readyState )  [signal]
```

Emitted when the ready state of the timer changes.

Parameters

readyState	the new ReadyState

5.8.4.11 reset

```
bool ScStwTimer::reset ( )  [virtual], [slot]
```

Function to reset the timer.

To do this, the timer has to be in ScStwTimer::WON or ScSTw::LOST state!

Returns

false if the timer was not in the required state and therefore not reset, true otherwise

5.8.4.12 setDisabled

```
void ScStwTimer::setDisabled (
            bool disabled )  [slot]
```

Function to set if the timer is supposed to be disabled.

!!! CAUTION use this function with care, it immidiately changes the state of the timer !!! It is recommended to only use this function to change the timers state after the ScStwTimer::requestTimerEnableChange() signal was called, during the race, the timer is used in, is in IDLE state.

Parameters

disabled	if the timer is supposed to be diabled

5.8.4.13 setResult

```
bool ScStwTimer::setResult (
            TimerState result )  [slot]
```

Function to assing the result of the race to the timer.

To do this, the timer has to be in ScStwTimer::WAITING state!

Returns

false if the timer was not in the required state and the result therefore not set, true otherwise

5.8.4.14 setState

```
void ScStwTimer::setState (
            TimerState newState )  [protected], [slot]
```

Function to change the state of the timer.

Doing this will emit the ScStwTimer::stateChanged() signal (only if the new state differs from the current one)

Parameters

newState	The new state

5.8.4.15 setWantsToBeDisabled

```
void ScStwTimer::setWantsToBeDisabled (
            bool wantsToBeDisabled )  [protected], [slot]
```

Function to set whether the timer currently wants to be disabled.

Parameters

wantsToBeDisabled	true or false

5.8.4.16 start `[1/2]`

```
bool ScStwTimer::start ( )  [slot]
```

Function to start the timer.

To do this, the timer has to be in ScStwTimer::STARTING state!

Returns

false if the timer was not in the required state and therefore not started, true otherwise

5.8.4.17 start `[2/2]`

```
bool ScStwTimer::start (
            double timeOfStart )  [virtual], [slot]
```

Function to start the timer at a given point in time (present or future)

To do this, the timer has to be in ScStwTimer::STARTING state!

Parameters

timeOfStart	the point in time (msecs since epoch) when the timer is supposted to be started

Returns

false if the timer was not in the required state and therefore not started, true otherwise

5.8.4.18 stop `[1/2]`

```
bool ScStwTimer::stop ( )  [slot]
```

Function to stop the timer.

To do this, the timer has to be in ScStwTimer::RUNNING state!

Returns

false if the timer was not in the required state and therefore not stopped, true otherwise

5.8.4.19 stop `[2/2]`

```
bool ScStwTimer::stop (
             double timeOfStop )  [slot]
```

Function to stop the timer at a given point in time (past or future)

To do this, the timer has to be in ScStwTimer::RUNNING state!

Parameters

timeOfStop	the point in time (msecs since epoch) when the timer is supposted to be stopped

Returns

false if the timer was not in the required state and therefore not stopped, true otherwise

5.8.4.20 technicalIncident

```
void ScStwTimer::technicalIncident ( )  [protected], [slot]
```

Function to set the timer into INCIDENT state immidieately.

The current state of the timer will be ignored! It can only get out of this state by calling ScStwTimer::reset

See also

reset

5.8.4.21 wantsToBeDisabledChanged

```
void ScStwTimer::wantsToBeDisabledChanged (
            ScStwTimer * timer,
            bool wantsToBeDisabled )  [signal]
```

Emitted when the timer wants its state to be changed by the external handler.

Parameters

timer	the timer object

5.8.4.22 wildcard

```
bool ScStwTimer::wildcard ( )  [protected], [slot]
```

Function to set the timer into WILDCARD state.

Only works when the timer is in STARTING state.

Returns

false if not in STARTING state

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

- /drone/src/ScStwLibraries/headers/scstwtimer.h
- /drone/src/ScStwLibraries/sources/scstwtimer.cpp

Chapter 6

File Documentation

6.1 ScStw.hpp

```
/****************************************************************************
 ** ScStw Libraries
 ** Copyright (C) 2020  Itsblue development
 **
 ** This program is free software: you can redistribute it and/or modify
 ** it under the terms of the GNU General Public License as published by
 ** the Free Software Foundation, either version 3 of the License, or
 ** (at your option) any later version.
 **
 ** This program is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 ** GNU General Public License for more details.
 **
 ** You should have received a copy of the GNU General Public License
 ** along with this program.  If not, see <http://www.gnu.org/licenses/>.
 ****************************************************************************/

#ifndef SCSTW_HPP
#define SCSTW_HPP

#include <QObject>
#include <QMap>
#include <QMetaEnum>

class ScStw : public QObject {
    Q_OBJECT
public:

    enum SignalKey {
        InvalidSignal = -1,
        RaceStateChanged = 9000,
        TimersChanged = 9001,
        ExtensionsChanged = 9002,
        CurrentStartDelayChanged = 9003, /*, ProfilesChanged*/
        SettingChanged = 9004,
        RaceDetailsChanged = 9005
    };
    Q_ENUM(SignalKey)


    enum SocketCommand {
        InvalidCommand = -1,

        InitializeSessionCommand = 1,

        StartRaceCommand = 1000,
        StopRaceCommand = 1001,
        ResetRaceCommand = 1002,
        CancelRaceCommand = 1003,
        SetTimerDisabledCommand = 1004,

        GetRaceStateCommand = 2000,
        GetRaceDetailsCommand = 2001,
        GetExtensionsCommand = 2006,
        GetTimersCommand = 2007,
        GetCurrentStartDelayCommand = 2009,

       WriteSettingCommand = 3000,
       ReadSettingCommand = 3001,

       LoginAthleteCommand = 4000,
       CreateAthleteCommand = 4001,
       DeleteAthleteCommand = 4002,
       GetAtheletesCommand = 4003,
       GetAthleteResultsCommand = 4004,

        UpdateFirmwareCommand = 5000,
        UpdateSystemTimeCommand = 5001,
        PairExtensionsCommand = 5002
    };
    Q_ENUM(SocketCommand);

    enum StatusCode {
        Success = 200,

        FirmwareAlreadyUpToDateInfo = 304,

        AccessDeniedError = 401,
        UpdateSignatureInvalidError = 402,
        CurrentStateNotVaildForOperationError = 403,
        CommandNotFoundError = 404,
        RequiredParameterNotGivenError = 405,
        TimestampTooSmallError = 406,
        ClientSessionAlreadyActiveError = 407,
        NoSessionActiveError = 408,
        ItemNotFoundError = 409,
        LastTimerCannotBeDisabledError = 410,

        UpdateFailedError = 500,

        Error = 900,
        NotConnectedError = 910,
        TimeoutError = 911,
        SettingNotAccessibleError = 901,
        InternalError = 950,
        InternalErrorTimerOperationFailed = 951,
        ApiVersionNotSupportedError = 952,
        CompetitionModeProhibitsThisError = 953,
        FirmwareUpdateFormatInvalidError = 954,
        TimersNotReadyError = 501
    };
    Q_ENUM(ScStw::StatusCode)


    enum ExtensionType {
        StartPad,
        TopPad
    };
    Q_ENUM(ExtensionType);


    enum ExtensionState {
        ExtensionDisconnected = 0,
        ExtensionConnecting = 1,
        ExtensionInitialising = 2,
        ExtensionConnected = 3
    };
    Q_ENUM(ExtensionState);


    enum ExtensionBatteryState {
        BatteryUnknown = -1,
        BatteryCritical = 0,
        BatteryWarning = 1,
        BatteryFine = 2,
        BatteryCharging = 3,
        BatteryNotCharging = 4
    };
    Q_ENUM(ExtensionBatteryState);

    enum PadState {
        PadNotPressed = 0,
        PadPressed = 1
    };
    Q_ENUM(PadState);


    static const char* SOCKET_MESSAGE_START_KEY;

    static const char* SOCKET_MESSAGE_END_KEY;

    static SignalKey signalKeyFromInt(int i);

    static SocketCommand socketCommandFromInt(int i);

    static QString extensionTypeToString(ExtensionType t);

    static int firmwareCompare(QString a, QString b);

    template <typename Enum>
    static Enum toEnumValue(const int &value, bool *ok)
    {
        QMetaEnum enumeration = QMetaEnum::fromType<Enum>();
        return static_cast<Enum>(enumeration.keyToValue(enumeration.valueToKey(value), ok));
    }

    ScStw() : QObject(nullptr) {};
private:
};

#endif // SCSTW_HPP
```

6.2 scstwlibraries.h

```
/****************************************************************************
 ** ScStw Libraries
 ** Copyright (C) 2020  Itsblue development
 **
 ** This program is free software: you can redistribute it and/or modify
 ** it under the terms of the GNU General Public License as published by
 ** the Free Software Foundation, either version 3 of the License, or
 ** (at your option) any later version.
 **
 ** This program is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 ** GNU General Public License for more details.
 **
 ** You should have received a copy of the GNU General Public License
 ** along with this program.  If not, see <http://www.gnu.org/licenses/>.
 ****************************************************************************/

#ifndef SCSTWLIBRARIES_H
#define SCSTWLIBRARIES_H

#include <QObject>
#ifdef ScStwLibraries_QML
#include <QQmlApplicationEngine>

#ifdef ScStwLibraries_Styling
#include "scstwappthememanager.h"
#include "scstwapptheme.h"
#endif

#endif
#include "scstwtimer.h"
#include "scstwrace.h"
#include "scstwsettings.h"
#ifdef ScStwLibraries_ClientLibs
#include "scstwsetting.h"
#include "scstwremoterace.h"
#include "scstwclient.h"
#include "scstwremotesettings.h"
#endif

class ScStwLibraries : public QObject
{
    Q_OBJECT

public:
    static void init();

#ifdef ScStwLibraries_QML
#ifdef ScStwLibraries_Styling
    static void initStyling(QQmlApplicationEngine *engine);
#endif
#endif

private:
    explicit ScStwLibraries(QObject *parent = nullptr);

signals:

};

#endif // SCSTWLIBRARIES_H
```

6.3 ScStwLibraries_global.h

```
/****************************************************************************
 ** ScStw Libraries
 ** Copyright (C) 2020  Itsblue development
 **
 ** This program is free software: you can redistribute it and/or modify
 ** it under the terms of the GNU General Public License as published by
 ** the Free Software Foundation, either version 3 of the License, or
 ** (at your option) any later version.
 **
  ** This program is distributed in the hope that it will be useful,
  ** but WITHOUT ANY WARRANTY; without even the implied warranty of
  ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  ** GNU General Public License for more details.
  **
  ** You should have received a copy of the GNU General Public License
  ** along with this program.  If not, see <http://www.gnu.org/licenses/>.
  ****************************************************************************/

#ifndef SCSTWLIBRARIES_GLOBAL_H
#define SCSTWLIBRARIES_GLOBAL_H

#include <QtCore/qglobal.h>

#if defined(SCSTWLIBRARIES_LIBRARY)
#  define SCSTWLIBRARIES_EXPORT Q_DECL_EXPORT
#else
#  define SCSTWLIBRARIES_EXPORT Q_DECL_IMPORT
#endif

#endif // SCSTWCLIENT_GLOBAL_H
```

6.4 scstwrace.h

```
/****************************************************************************
 ** ScStw Libraries
 ** Copyright (C) 2020  Itsblue development
 **
 ** This program is free software: you can redistribute it and/or modify
 ** it under the terms of the GNU General Public License as published by
 ** the Free Software Foundation, either version 3 of the License, or
 ** (at your option) any later version.
 **
  ** This program is distributed in the hope that it will be useful,
  ** but WITHOUT ANY WARRANTY; without even the implied warranty of
  ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  ** GNU General Public License for more details.
  **
  ** You should have received a copy of the GNU General Public License
  ** along with this program.  If not, see <http://www.gnu.org/licenses/>.
  ****************************************************************************/

#ifndef SCSTWRACE_H
#define SCSTWRACE_H

#include <QObject>
#include <QDebug>
#include <QTimer>
#include <QEventLoop>
#include "scstwtimer.h"
#include "scstwsoundplayer.h"
#include "scstwsettings.h"

class ScStwRemoteRace;

class ScStwRace : public QObject
{
    Q_OBJECT
    Q_PROPERTY(RaceState state READ getState NOTIFY stateChanged)
    Q_PROPERTY(QVariantList timers READ getTimerDetailList NOTIFY timersChanged)
    Q_PROPERTY(QVariantMap currentStartDelay READ getCurrentStartDelay NOTIFY currentStartDelayChanged)
    Q_PROPERTY(bool isReadyForNextState READ getIsReadyForNextState NOTIFY isReadyForNextStateChanged)
    Q_PROPERTY(bool competitionMode READ getCompetitionMode NOTIFY competitionModeChanged)
    Q_PROPERTY(bool readySoundEnabled READ getReadySoundEnabled NOTIFY readySoundEnabledChanged)
    Q_PROPERTY(QVariantMap details READ getDetails NOTIFY detailsChanged)
    Q_PROPERTY(ScStwSettings* settings READ getSettings WRITE setSettings NOTIFY settingsChanged)
    Q_PROPERTY(bool autoRefreshTimerText READ getAutoRefreshTimerText WRITE setAutoRefreshTimerText
    NOTIFY autoRefreshTimerTextChanged)

public:
    explicit ScStwRace(QObject *parent = nullptr);
    explicit ScStwRace(ScStwSettings *settings, QObject *parent = nullptr);

    friend class ScStwRemoteRace;

    enum RaceState { IDLE, PREPAIRING, WAITING, STARTING, RUNNING, STOPPED, INCIDENT };
    Q_ENUM(RaceState)

protected:
    QList<ScStwTimer *> timers;
    void setState(RaceState newState);

private:
    RaceState state;

    QTimer *startDelayTimer;
    QTimer *timerTextRefreshTimer;
    QEventLoop *startWaitLoop;

    // sounds
    ScStwSoundPlayer * soundPlayer;

    // settings
    ScStwSettings *settings;
    bool competitionMode;
    bool autoRefreshTimerText;

    enum LoopExitTypes {
        LoopAutomaticExit = 0,
        LoopReadyStateChangeExit = 1,
        LoopManualExit = 2,
        LoopCancelExit = 3
    };


public slots:
     virtual ScStw::StatusCode start(bool asyncronous = true);

     virtual ScStw::StatusCode stop();

     virtual ScStw::StatusCode reset();
     virtual ScStw::StatusCode cancel();

     virtual ScStw::StatusCode setTimerDisabled(int id, bool disabled);

     Q_INVOKABLE virtual bool addTimer(ScStwTimer *timer);

     // getters
     RaceState getState();
     virtual QVariantMap getCurrentStartDelay();
     QList<ScStwTimer*> getTimers();
     QVariantList getTimerDetailList();
     QVariantMap getDetails();
     bool getCompetitionMode();
     virtual bool getReadySoundEnabled();

     ScStwSettings* getSettings();
     void setSettings(ScStwSettings* settings);

     bool getAutoRefreshTimerText();
     void setAutoRefreshTimerText(bool autoRefresh);

protected slots:

private slots:
     void handleTimerStateChange(ScStwTimer::TimerState newState);

     void handleTimerStop();
     void handleFalseStart();


     void handleTimerWantsToBeDisabledChange(ScStwTimer* timer, bool wantsToBeDisabled);
     bool playSoundsAndStartTimers();
     ScStwSoundPlayer::PlayResult doDelayAndSoundOfCurrentStartState(double *timeOfSoundPlaybackStart =
    nullptr);
     void technicalIncident();
     ScStw::StatusCode setTimerDisabled(ScStwTimer* timer, bool disabled);

     virtual void refreshCompetitionMode();

     double getSoundVolume();
     ScStwSoundPlayer::StartSound getSoundForState(ScStwRace::RaceState state);
     bool getSoundEnabledSetting(ScStwSoundPlayer::StartSound sound);
     int getSoundDelaySetting(ScStwSoundPlayer::StartSound sound);


     bool isStarting();
     virtual bool getIsReadyForNextState();
     void handleTimerReadyStateChange(ScStwTimer::ReadyState readyState);
```

```
signals:
    void startTimers();
    void stopTimers(int type);
    void resetTimers();
    void stateChanged(RaceState state);
    void currentStartDelayChanged();
    void timersChanged();
    void isReadyForNextStateChanged();
    void detailsChanged();
    void competitionModeChanged();
    void readySoundEnabledChanged();
    void settingsChanged();
    void autoRefreshTimerTextChanged();

};

#endif // SCSTWRACE_H
```

6.5 scstwsetting.h

```
/****************************************************************************
** ScStw Libraries
** Copyright (C) 2020  Itsblue development
**
** This program is free software: you can redistribute it and/or modify
** it under the terms of the GNU General Public License as published by
** the Free Software Foundation, either version 3 of the License, or
** (at your option) any later version.
**
** This program is distributed in the hope that it will be useful,
** but WITHOUT ANY WARRANTY; without even the implied warranty of
** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
** GNU General Public License for more details.
**
** You should have received a copy of the GNU General Public License
** along with this program.  If not, see <http://www.gnu.org/licenses/>.
****************************************************************************/

#ifndef SCSTWQMLSETTING_H
#define SCSTWQMLSETTING_H

#include <QObject>
#include <QVariant>

class ScStwSettings;

class ScStwSetting : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QVariant value READ getValue WRITE setValue NOTIFY valueChanged)
    Q_PROPERTY(QVariant readonlyValue READ getValue NOTIFY valueChanged)

protected:
    explicit ScStwSetting(int key, int keyLevel, ScStwSettings*scStwSettings, QObject *parent);

    friend class ScStwSettings;

    int key;
    int keyLevel;
    bool hasToReload;

private:
    QVariant valueCache;
    ScStwSettings *scStwSettings;

public slots:
    QVariant getValue();
    void setValue(QVariant value);

protected slots:
    void handleSettingChange(int key, int keyLevel, QVariant value);

signals:
    void valueChanged();

};

#endif // SCSTWQMLSETTING_H
```

6.6 scstwsettings.h

```
/****************************************************************************
 ** ScStw Libraries
 ** Copyright (C) 2020  Itsblue development
 **
 ** This program is free software: you can redistribute it and/or modify
 ** it under the terms of the GNU General Public License as published by
 ** the Free Software Foundation, either version 3 of the License, or
 ** (at your option) any later version.
 **
 ** This program is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 ** GNU General Public License for more details.
 **
 ** You should have received a copy of the GNU General Public License
 ** along with this program.  If not, see <http://www.gnu.org/licenses/>.
 ****************************************************************************/

#ifndef SCSTWSETTINGS_H
#define SCSTWSETTINGS_H

#include <QObject>
#include <QVariant>
#include <QMetaEnum>
#include <QtDebug>
#include <QFile>
#include <QStandardPaths>
#include <QJsonDocument>
#include <ScStw.hpp>
#include <QDir>
#include <scstwsetting.h>

class ScStwSettings : public QObject
{
    Q_OBJECT
public:
    explicit ScStwSettings(QObject *parent = nullptr, bool overwriteFileOnErrors = true);

    typedef QString(*keyToStringConverter)(int);
    typedef QVariant::Type(*keyToTypeConverter)(int);

    enum BaseStationSetting {
        InvalidSetting = -1,
        ReadySoundEnableSetting,
        ReadySoundDelaySetting,
        AtYourMarksSoundEnableSetting,
        AtYourMarksSoundDelaySetting,
        SoundVolumeSetting,
        CompetitionModeSetting
    };
    Q_ENUM(BaseStationSetting)

    enum KeyLevelEnum {
        KeyLevel = 0
    };
    Q_ENUM(KeyLevelEnum)

    virtual QVariant readSetting(BaseStationSetting key);
    Q_INVOKABLE virtual QVariant readSetting(int key, int keyLevel);
    virtual bool writeSetting(BaseStationSetting key, QVariant value);
    Q_INVOKABLE virtual bool writeSetting(int key, int keyLevel, QVariant value);
    virtual bool setDefaultSetting(BaseStationSetting key, QVariant defaultValue);
    Q_INVOKABLE virtual bool setDefaultSetting(int key, int keyLevel, QVariant defaultValue);

    Q_INVOKABLE ScStwSetting * getSetting(int key, int keyLevel);

    static BaseStationSetting keyFromInt(int i) {
        QMetaEnum enumeration = QMetaEnum::fromType<BaseStationSetting>();
        return static_cast<BaseStationSetting>(enumeration.keyToValue(enumeration.valueToKey(i)));
    }

    static QString keyToString(int key) {
        return QMetaEnum::fromType<BaseStationSetting>().valueToKey(key);
    }

    static QVariant::Type keyToType(int key) {
        QMap<BaseStationSetting, QVariant::Type> types = {
            {ReadySoundEnableSetting, QVariant::Bool},
            {ReadySoundDelaySetting, QVariant::Double},
            {AtYourMarksSoundEnableSetting, QVariant::Bool},
            {AtYourMarksSoundDelaySetting, QVariant::Double},
            {SoundVolumeSetting, QVariant::Double},
            {CompetitionModeSetting, QVariant::Bool}
        };

        if(types.contains(BaseStationSetting(key)))
            return types[BaseStationSetting(key)];

        return QVariant::Invalid;
    }

protected:
    virtual QVariant readSetting(QString key, int keyInt = -1, int keyLevel = -1);
    virtual bool writeSetting(QString key, QVariant value, int keyInt = -1,int keyLevel = -1);
    virtual bool setDefaultSetting(QString key, QVariant defaultValue, int keyInt,int keyLevel = -1);
    bool registerKeyLevelConverters(int keyLevel, keyToStringConverter, keyToTypeConverter);

private:
    bool fileIsReadonly;

    QFile * settingsFile;
    QVariantMap settingsCache;

    bool loadSettingsFromFile();

    QMap<int, keyToStringConverter> keyToStringConverters;
    QMap<int, keyToTypeConverter> keyToTypeConverters;
    QMap<int, QMap<int, ScStwSetting*» internalSettingHandlers;

private slots:
    bool writeSettingsToFile();

signals:
    void settingChanged(int key, int keyLevel, QVariant value);

};

#endif // SCSTWSETTINGS_H
```

6.7 scstwsoundplayer.h

```
/****************************************************************************
 ** ScStw Libraries
 ** Copyright (C) 2020  Itsblue development
 **
 ** This program is free software: you can redistribute it and/or modify
 ** it under the terms of the GNU General Public License as published by
 ** the Free Software Foundation, either version 3 of the License, or
 ** (at your option) any later version.
 **
 ** This program is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 ** GNU General Public License for more details.
 **
 ** You should have received a copy of the GNU General Public License
 ** along with this program.  If not, see <http://www.gnu.org/licenses/>.
 ****************************************************************************/

#ifndef SCSTWSTARTSOUNDPLAYER_H
#define SCSTWSTARTSOUNDPLAYER_H

#include <QObject>
#include <QFile>
#include <QAudioOutput>
#include <QDebug>
#include <QEventLoop>
#include <QTimer>
#include <QDateTime>
#include <QSoundEffect>
#include <QAudioDeviceInfo>

#ifdef ScStwLibraries_Raspi
#include <alsa/asoundlib.h>
#endif

class ScStwSoundPlayer : public QObject
{
    Q_OBJECT
public:
    explicit ScStwSoundPlayer(QObject *parent = nullptr);

    enum StartSound {
        None = -1,
        AtYourMarks = 0,
        Ready = 1,
        Start = 2,
        FalseStart = 3
    };

    enum PlayResult {
        Success = 0,
        Cancelled = -1,
        Error = -2
    };

private:

    bool _setSoundVolume(double volume);

    void _initializeSondEffect();

    QMap<StartSound, QVariantMap> soundFiles;

    QSoundEffect *soundEffect;

    QAudioDeviceInfo *_audioOutputDevice;

    QEventLoop *waitLoop;

    QTimer *waitTimer;

    StartSound currentlyPlayingSound;

    double playingStartedAt;

public slots:

    ScStwSoundPlayer::PlayResult play(StartSound sound, double volume, double *timeOfStart = nullptr);

    ScStwSoundPlayer::PlayResult waitForSoundFinish(double *timeOfStop = nullptr);

    bool cancel();

    bool isPlaying();

private slots:

signals:

    void playbackStarted();

};

#endif // SCSTWSTARTSOUNDPLAYER_H
```

6.8 scstwstartsoundplayer.h

```
#ifndef SCSTWSTARTSOUNDPLAYER_H
#define SCSTWSTARTSOUNDPLAYER_H

#include <QObject>
#include <QFile>
#include <QAudioOutput>
#include <QDebug>
#include <QEventLoop>
#include <QTimer>
#include <QDateTime>

class ScStwStartSoundPlayer : public QObject
{
    Q_OBJECT
public:
    explicit ScStwStartSoundPlayer(QObject *parent = nullptr);

private:
    QFile *startSoundFile;
    QAudioOutput *audioOutput;
    QEventLoop *waitLoop;

public slots:
    bool play(double volume, double *timeOfStop = nullptr);
    //int interrupt();

private slots:
    void handleStateChanged(QAudio::State newState);

signals:

};

#endif // SCSTWSTARTSOUNDPLAYER_H
```

6.9 scstwtimer.h

```
/****************************************************************************
 ** ScStw Libraries
 ** Copyright (C) 2020  Itsblue development
 **
 ** This program is free software: you can redistribute it and/or modify
 ** it under the terms of the GNU General Public License as published by
 ** the Free Software Foundation, either version 3 of the License, or
 ** (at your option) any later version.
 **
 ** This program is distributed in the hope that it will be useful,
 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 ** GNU General Public License for more details.
 **
 ** You should have received a copy of the GNU General Public License
 ** along with this program.  If not, see <http://www.gnu.org/licenses/>.
 ****************************************************************************/

#ifndef SCSTWTIMER_H
#define SCSTWTIMER_H

#include <QObject>
#include <QDateTime>
#include <QDebug>
#include <QTimer>
#include "ScStw.hpp"

class ScStwRace;
class ScStwRemoteRace;

class ScStwTimer : public QObject
{
    Q_OBJECT
public:

    explicit ScStwTimer(QObject *parent = nullptr);

    explicit ScStwTimer(QString letter, QObject *parent = nullptr);

    friend class ScStwRace;

    enum TimerState {
        IDLE,
        STARTING,
        RUNNING,
        WAITING,
        WON,
        LOST,
        FAILING,
        WILDCARD,
        FAILED,
        CANCELLED,
        INCIDENT,
        DISABLED
    };
    Q_ENUM(TimerState);

    enum ReadyState {
        IsReady = 0,
        NotInIdleState,
        IsDisabled,
        ExtensionIsNotConnected,
        ExtensionBatteryIsCritical,
        ClimberIsNotReady
    };
    Q_ENUM(ReadyState)

protected:
    TimerState state;

    double startTime;

    double stopTime;

    double reactionTime;

    QString letter;

    bool wantsToBeDisabled;

public slots:

    bool start();

    virtual bool start(double timeOfStart);

    bool cancel();

    bool stop();

    bool stop(double timeOfStop);

    bool setResult(TimerState);

    virtual bool reset();

    // -- helper functions --
    TimerState getState();

    double getCurrentTime();

    double getReactionTime();

    QString getText();

    QString getLetter();

    void setDisabled(bool disabled);

    bool getWantsToBeDisabled();

    virtual ScStwTimer::ReadyState getReadyState();

    bool isRunning();

    bool isDisabled();

protected slots:

    void handleClimberStart(double timeOfStart);

    void setState(TimerState newState);

    void setWantsToBeDisabled(bool wantsToBeDisabled);

    void technicalIncident();

    bool wildcard();


signals:
    void stateChanged(TimerState state);

    void reactionTimeChanged();

    void wantsToBeDisabledChanged(ScStwTimer* timer, bool wantsToBeDisabled);

    void readyStateChanged(ReadyState readyState);

};

#endif // SCSTWTIMER_H
```

Index