QSense Sidecar Library

1.0.0

Generated by Doxygen 1.8.9.1

Thu Dec 10 2015 08:21:15

Contents

Chapter 1

QSense Sidecar Library

1.1 Contents

1.2 Overview

A standard C++ library for interacting with the Sidecar Event API. Developed for use on Arduino platform, but was developed and tested on Mac OS X and Windows.

The library exposes two interfaces. A simple (SimpleSidecarClient) Arduino specific class that abstracts the raw C++ API, and the raw C++ API that can be used from all platforms (including Arduino).

1.3 Workflow

The general workflow for publishing an event to the Sidecar Event API is as follows:

- Register with Sidecar and provision your application with an access key/secret pair.
- While bootstrapping your application, provision a user account (email address/password) and retrieve the user access key/secret pair. Use the user access key/secret to initialise the library.
- Create an instance of qsense::Event.
- Add instances of qsense::Reading to the event.
- Add any tag values to the event (tags are instances of qsense::QString).
- Create an instance of qsense::net::SidecarClient
- Publish the event using qsense::net::SidecarClient::publish

The simplified Arduino client encapsulates all these steps and variety of classes into a single class. There is no need to create instances of qsense::Event or qsense::Reading. You can add reading data directly to the client and then publish the accumulated readings as one event.

1.4 Initialisation

The QSense Sidecar Library needs some initialisation (from the Arduino sketch for instance). The accompanying example sketches illustrate the recommended way of initialising the library.

1.4.1 Simple Arduino API

- Initialise the library with the type of networking used by the device using the SimpleSidecarClient::initNetworkType static method.
- Initialise the UUID engine with a proper MAC address using the SimpleSidecarClient::initUUID methods. Note, that there is an overloaded version of the method that will generate a random MAC address to use to initialise the UUID engine.

1.4.1.1 Provisioning API

- Initialise the API with the application access key/secret pair via the SimpleSidecarClient::initAPIKey method.
- Authenticate (or create a user account) with the Sidecar service using user's email address and password. This will retrieve the user access key/secret pair that will be used to submit sensor data to Sidecar. Use the SimpleSidecarClient::authenticate and SimpleSidecarClient::createUser methods.

1.4.1.2 Event API

- Initialise the Sidecar library with the user access key and secret used for authentication/authorisation. Use the SimpleSidecarClient::initUserKey method.
- Initialise the Event API with the stream identifier, device UUID, and a default geographic location. Create an instance of the SimpleSidecarClient::EventAPIData structure, populate the fields and use the SimpleSidecarClient::initEventAPI method.

1.4.2 Core API

- Initialise the library with the type of networking used by the device. Use the qsense::net::initNetworkType(qsense::net::NetworkType) function to indicate the type of network being used.
- Initialise the UUID engine with a proper MAC address (qsense::UUID::init). Official Arduino ethernet shields come with a MAC address labelled on the board (as per Arduino documentation). If using WiFi, the Arduino WiFi API allows lookup of the current MAC address. A stable and unique MAC address is essential for ensuring that UUID values generated (generates time based values) are truly universally unique.
- Initialise the API with the application access key/secret pair. Use the qsense::net::SidecarClient::initAPIKey method.
- Authenticate (or create a user account) with the Sidecar service using user's email address and password. This will retrieve the user access key/secret pair that will be used to submit sensor data to Sidecar. Use the qsense::net::SidecarClient::authenticate or qsense::net::SidecarClient::createUser methods to authenticate or create and provision the user account.
- Initialise the Sidecar library with the user access key and secret used for authentication/authorisation (qsense::net::SidecarClient::initUserKey).
- Initialise the Event API with the stream identifier, device UUID, and a default geographic qsense::Location (qsense::Event::init).

1.4.3 Seed Studio Ethernet Shield

The standard Arduino Ethernet Library seems to be incompatible with Arduino Mega boards (the board the library supports). For users of the common `Seed Studio Ethernet Shield`, Seed Studio provides their own `Ethernet Shield V2.0` library (which is also included in the source distribution for convenience). Separate example sketches (ending with -SeedStudio) are provided for use with Seed Studio Ethernet Shields. A minor modification to the QSense.h file is necessary to be able to use the Seed Studio library (documented in the example sketches as well). Set the `USE_Ethernet_Shield_V2` to 1 in QSense.h to use the Seed Studio Ethernet library instead of the standard Arduino Ethernet library.

1.4.4 Debug Output

To enable debug information (print request/response data to the Serial interface), edit the QSense.h file and set the value of `DEBUG` to 1. The default value is 0, and no debug information will be output by the API. Users building applications using tools other than the official Arduino IDE may be able to specify the DEBUG pre-processor definition as part of the build process.

1.5 Publish Sensor Data

Once the Sidecar library has been properly initialised with the user access key and secret combination, sensor data may be published to Sidecar as events at any time.

1.5.1 Simple Arduino API

- Add readings to be published as one event using the SimpleSidecarClient::addReading method.
- Optionally add tags to the event using the SimpleSidecarClient::addTag method.
- Publish the accumulated readings as an event to Sidecar. Use the SimpleSidecarClient::publish method to publish the event.

1.5.2 Core API

- Create an instance of qsense::Event.
- Create instances of qsense::Reading and assign it sensor data and any other relevant data that in combination constitute an event to be published to Sidecar. Use the qsense::Event::add(const qsense::Reading&) method.
- Optionally add tags to the event. Tags may be used when querying data stored in Sidecar. Use the qsense::Event::add(const qsense::QString&)} method.
- Create an instance of the qsense::net::SidecarClient and publish the event using qsense::net::SidecarClient::publish.

1.6 Libraries

The only third-party library necessary is the Standard C++ Library.

- `Standard C++`
- `Ethernet Shield V2.0` for Seed Studio Ethernet boards
- `Poco` - needed for using on non-Arduino platforms (for network interactions). We have tested only on Mac OS X and Windows, however Poco runs on most platforms including iOS and Android.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 qsense Namespace Reference

Namespaces

- hash
- net

Classes

- class AutoPtr

  AutoPtr is a "smart" pointer for classes implementing reference counting based garbage collection.
- class ByteOrder

  This class contains a number of static methods to convert between big-endian and little-endian integers of various sizes.
- class Event

  A simple class that encapsulates an event sent to Sidecar. Events are holders for readings. Events can be serialised to JSON using the toString method.
- class Location

  A simple representation of geographical location.
- class Reading

  A class that represents a single reading. Readings are added to an Event.
- class RefCountedObject

  A base class for objects that employ reference counting based garbage collection.
- class UUID

  A class that represents a UUID/GUID.

Typedefs

- typedef std::string QString
- typedef unsigned char Byte

Functions

- template<class C >
  void swap (AutoPtr< C > &p1, AutoPtr< C > &p2)
- std::ostream & operator<< (std::ostream &os, const Event &event)

Serialise the specified event as JSON to the output stream.

- std::ostream & operator<< (std::ostream &os, const qsense::Location &location)

  Serialise the specified location to the output stream.
- std::ostream & operator<< (std::ostream &os, const qsense::Reading &reading)

  Serialise the reading as JSON to the output stream.
- std::ostream & operator<< (std::ostream &os, const UUID &uuid)

  Serialise the string representation of the UUID to the output stream.

6.1.1 Detailed Description

The namespace for the QSense Sidecar Library.

6.1.2 Typedef Documentation

6.1.2.1 typedef unsigned char qsense::Byte

8-bit unsigned char

6.1.2.2 typedef std::string qsense::QString

Not really necessary any more. Initially had it to use String class from Arduino library, but that never seems to work when used for hashing etc.

6.1.3 Function Documentation

6.1.3.1 std::ostream& qsense::operator<< (std::ostream & *os,* const qsense::Location & *location*)

Serialise the specified location to the output stream.

6.1.3.2 std::ostream& qsense::operator<< (std::ostream & *os,* const qsense::Reading & *reading*)

Serialise the reading as JSON to the output stream.

6.1.3.3 std::ostream& qsense::operator<< (std::ostream & *os,* const Event & *event*)

Serialise the specified event as JSON to the output stream.

6.1.3.4 std::ostream& qsense::operator<< (std::ostream & *os,* const UUID & *uuid*) [inline]

Serialise the string representation of the UUID to the output stream.

6.1.3.5 template<class C > void qsense::swap (AutoPtr< C > & *p1,* AutoPtr< C > & *p2*) [inline]

6.2 qsense::hash Namespace Reference

Namespaces

- base64

Classes

- class MD5

  Class for generating MD5 hashes.
- class Sha1

  Class for hashing using SHA1 algorithm.

6.2.1 Detailed Description

Namespace for classes and functions that provide hashing support.

6.3 qsense::hash::base64 Namespace Reference

Functions

- int32_t encodeLength (int32_t len)
- int32_t encode (char ∗output, const char ∗input, int32_t inputLength)
- int32_t decodeLength (const char ∗code)
- int32_t decode (char ∗outputPlainText, const char ∗encoded)

6.3.1 Detailed Description

Namespace for functions that provide Base64 encoding/decoding support.

6.3.2 Function Documentation

6.3.2.1 int32_t qsense::hash::base64::decode (char ∗ *outputPlainText,* const char ∗ *encoded*)

Decode into outputPlainText the encoded contents

6.3.2.2 int32_t qsense::hash::base64::decodeLength (const char ∗ *code*)

Use to specify size of output array to decode into

6.3.2.3 int32_t qsense::hash::base64::encode (char ∗ *output,* const char ∗ *input,* int32_t *inputLength*)

Encode into output contents of plain_src of specified length

6.3.2.4 int32_t qsense::hash::base64::encodeLength (int32_t *len*)

Use to specify size of output array to encode into

6.4 qsense::net Namespace Reference

Classes

- class DateTime

  Represents current date/time. Seeds initially (and daily) from a network time service, and uses internal timer to represent a real-time clock.

- class HttpClient

  A HTTP Client class for use with either ethernet or wifi. Before use, the client should be initialised with the network type used by the device (qsense::net::initNetworkType.
- class HttpRequest

  A simple class that represents a HTTP request. Request encapsulates the URI path, any request parameters, header attributes, body etc. as appropriate.
- class SidecarClient

  Class that encapsulates interactions with the Sidecar REST API.

Enumerations

- enum NetworkType { Ethernet = 0, WiFi = 1 }

  Enumeration of network connection types for device.

Functions

- uint32_t millis ()
- void initNetworkType (NetworkType type)

6.4.1 Detailed Description

Namespace for classes that provide network services and require a network connection to work.

6.4.2 Enumeration Type Documentation

6.4.2.1 enum qsense::net::NetworkType

Enumeration of network connection types for device.

Enumerator

Ethernet

WiFi

6.4.3 Function Documentation

6.4.3.1 void qsense::net::initNetworkType (NetworkType *type*)

Initialise the API to use the specified type. HttpClient::create uses this type to create appropriate implementation.

6.4.3.2 uint32_t qsense::net::millis ( )

Chapter 7

Class Documentation

7.1 qsense::AutoPtr< C > Class Template Reference

AutoPtr is a "smart" pointer for classes implementing reference counting based garbage collection.

`#include <AutoPtr.h>`

Public Member Functions

- AutoPtr ()

  Default constructor. Creates a new instance that points to nothing.
- AutoPtr (C *ptr)

  Create an auto pointer that takes ownership of the specified pointer.
- AutoPtr (C *ptr, bool shared)

  AutoPtr Create an auto pointer that takes ownership of the specified pointer.
- AutoPtr (const AutoPtr &ptr)

  Copy constructor. Increases the reference count for the owned object.
- template<class Other >
  AutoPtr (const AutoPtr< Other > &ptr)

  Copy constructor for taking ownership of another type of object.
- ~AutoPtr ()

  Destructor. Invokes `release` on the owned object.
- AutoPtr & assign (C *ptr)

  Use to (re)set the owned object. If current owned object is not `null`, invokes `release` on the object.
- AutoPtr & assign (C *ptr, bool shared)

  Reset the owned object. If current owned object is not `null`, invokes `release` on the object. Will invoke `duplicate` on the new instance if `shared` is specified.
- AutoPtr & assign (const AutoPtr &ptr)

  Share the pointer owned by the specified auto pointer instance.
- template<class Other >
  AutoPtr & assign (const AutoPtr< Other > &ptr)

  Share the pointer owned by the specified auto pointer of different type.
- AutoPtr & operator= (C *ptr)

  Copy assignment operator. Delegates to assign.
- AutoPtr & operator= (const AutoPtr &ptr)

  Copy assignment operator. Delegates to assign.
- template<class Other >
  AutoPtr & operator= (const AutoPtr< Other > &ptr)

Copy assignment operator. Delegates to assign.

- void swap (AutoPtr &ptr)

  Swap the pointers of this instance with the specified instance.

- template<class Other >
  AutoPtr< Other > cast () const

  Casts the AutoPtr via a dynamic cast to the given type. Returns an AutoPtr containing NULL if the cast fails. Example: (assume class Sub: public Super) AutoPtr<Super> super(new Sub()); AutoPtr<Sub> sub = super.↩ cast<Sub>(); poco_assert (sub.get());.

- template<class Other >
  AutoPtr< Other > unsafeCast () const

  Casts the AutoPtr via a static cast to the given type. Example: (assume class Sub: public Super) AutoPtr<Super> super(new Sub()); AutoPtr<Sub> sub = super.unsafeCast<Sub>(); poco_assert (sub.get());.

- C * operator-> ()

  Pointer acess operator for the owned object. Returns `NULL` if invalid.

- const C * operator-> () const

  Pointer acess operator for the owned object. Returns `NULL` if invalid.

- C & operator* ()

  Dereference operator for the owned object. Will lead to program termination if the owned object is not valid.

- const C & operator* () const

  Dereference operator for the owned object. Will lead to program termination if the owned object is not valid.

- C * get ()

  Return the owned pointer. Callers must not `delete`.

- const C * get () const

  Return the owned pointer. Callers must not `delete`.

- operator C * ()

  Function operator. Return the owned pointer.

- operator const C * () const

  Function operator. Return the owned pointer.

- bool operator! () const

  Negative check of make sure the owned pointer is not valid.

- bool isNull () const

  Negative check of make sure the owned pointer is not valid.

- C * duplicate ()

  Invokes `duplicate()` on the owned pointer if valid.

- bool operator== (const AutoPtr &ptr) const

  Compare the owned objects for equality.

- bool operator== (const C *ptr) const

  Compare the owned object against the specified object for equality.

- bool operator== (C *ptr) const

  Compare the owned object against the specified object for equality.

- bool operator!= (const AutoPtr &ptr) const

  Compare the owned objects for inequality.

- bool operator!= (const C *ptr) const

  Compare the owned object against the specified object for inequality.

- bool operator!= (C *ptr) const

  Compare the owned object against the specified object for inequality.

- bool operator< (const AutoPtr &ptr) const

  Compare the owned objects for ordering.

- bool operator< (const C *ptr) const

  Compare the owned object against the specified object for ordering.

- bool operator< (C *ptr) const

Compare the owned object against the specified object for ordering.

- bool operator<= (const AutoPtr &ptr) const
- bool operator<= (const C *ptr) const
- bool operator<= (C *ptr) const
- bool operator> (const AutoPtr &ptr) const
- bool operator> (const C *ptr) const
- bool operator> (C *ptr) const
- bool operator>= (const AutoPtr &ptr) const
- bool operator>= (const C *ptr) const
- bool operator>= (C *ptr) const

7.1.1 Detailed Description

template<class C>class qsense::AutoPtr< C >

AutoPtr is a "smart" pointer for classes implementing reference counting based garbage collection.

To be usable with the AutoPtr template, a class must implement the following behaviour:

- A class must maintain a reference count.
- The constructors of the object initialize the reference count to one.
- The class must implement a public duplicate() method: void duplicate(); that increments the reference count by one.
- The class must implement a public release() method: void release() that decrements the reference count by one, and, if the reference count reaches zero, deletes the object.

AutoPtr works in the following way:

- If an AutoPtr is assigned an ordinary pointer to an object (via the constructor or the assignment operator), it takes ownership of the object and the object's reference count remains unchanged.
- If the AutoPtr is assigned another AutoPtr, the object's reference count is incremented by one by calling duplicate() on its object.
- The destructor of AutoPtr calls release() on its object.
- AutoPtr supports dereferencing with both the -> and the * operator. An attempt to dereference a null AutoPtr results in a error that will cause application termination. AutoPtr also implements all relational operators. Note that AutoPtr allows casting of its encapsulated data types.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 template<class C > qsense::AutoPtr< C >::AutoPtr ( ) [inline]

Default constructor. Creates a new instance that points to nothing.

7.1.2.2 template<class C > qsense::AutoPtr< C >::AutoPtr (C * *ptr*) [inline]

Create an auto pointer that takes ownership of the specified pointer.

7.1.2.3 template<class C > qsense::AutoPtr< C >::AutoPtr (C * *ptr*, bool *shared*) [inline]

AutoPtr Create an auto pointer that takes ownership of the specified pointer.

Parameters

ptr	The pointer to take ownership of
shared	If `true` then increment the reference count for the specified pointer.

7.1.2.4 template<class C > qsense::AutoPtr< C >::AutoPtr (const AutoPtr< C > & *ptr*) [inline]

Copy constructor. Increases the reference count for the owned object.

7.1.2.5 template<class C > template<class Other > qsense::AutoPtr< C >::AutoPtr (const AutoPtr< Other > & *ptr*) [inline]

Copy constructor for taking ownership of another type of object.

7.1.2.6 template<class C > qsense::AutoPtr< C >::~AutoPtr ( ) [inline]

Destructor. Invokes `release` on the owned object.

7.1.3 Member Function Documentation

7.1.3.1 template<class C > AutoPtr& qsense::AutoPtr< C >::assign (C * *ptr*) [inline]

Use to (re)set the owned object. If current owned object is not `null`, invokes `release` on the object.

7.1.3.2 template<class C > AutoPtr& qsense::AutoPtr< C >::assign (C * *ptr*, bool *shared*) [inline]

Reset the owned object. If current owned object is not `null`, invokes `release` on the object. Will invoke `duplicate` on the new instance if `shared` is specified.

Parameters

ptr	The new object to take ownership of.
shared	If `true`, `duplicate()` the owned object.

Returns

Reference to this instance for convenience.

7.1.3.3 template<class C > AutoPtr& qsense::AutoPtr< C >::assign (const AutoPtr< C > & *ptr*) [inline]

Share the pointer owned by the specified auto pointer instance.

7.1.3.4 template<class C > template<class Other > AutoPtr& qsense::AutoPtr< C >::assign (const AutoPtr< Other > & *ptr*) [inline]

Share the pointer owned by the specified auto pointer of different type.

7.1.3.5 template<class C > template<class Other > AutoPtr<Other> qsense::AutoPtr< C >::cast ( ) const [inline]

Casts the AutoPtr via a dynamic cast to the given type. Returns an AutoPtr containing NULL if the cast fails. Example: (assume class Sub: public Super) AutoPtr<Super> super(new Sub()); AutoPtr<Sub> sub = super.↩ cast<Sub>(); poco_assert (sub.get());.

7.1.3.6 template<class C > C* qsense::AutoPtr< C >::duplicate ( ) [inline]

Invokes `duplicate()` on the owned pointer if valid.

7.1.3.7 template<class C > C* qsense::AutoPtr< C >::get ( ) [inline]

Return the owned pointer. Callers must not `delete`.

7.1.3.8 template<class C > const C* qsense::AutoPtr< C >::get ( ) const [inline]

Return the owned pointer. Callers must not `delete`.

7.1.3.9 template<class C > bool qsense::AutoPtr< C >::isNull ( ) const [inline]

Negative check of make sure the owned pointer is not valid.

7.1.3.10 template<class C > qsense::AutoPtr< C >::operator C * ( ) [inline]

Function operator. Return the owned pointer.

7.1.3.11 template<class C > qsense::AutoPtr< C >::operator const C * ( ) const [inline]

Function operator. Return the owned pointer.

7.1.3.12 template<class C > bool qsense::AutoPtr< C >::operator! ( ) const [inline]

Negative check of make sure the owned pointer is not valid.

7.1.3.13 template<class C > bool qsense::AutoPtr< C >::operator!= (const AutoPtr< C > & *ptr*) const [inline]

Compare the owned objects for inequality.

7.1.3.14 template<class C > bool qsense::AutoPtr< C >::operator!= (const C * *ptr*) const [inline]

Compare the owned object against the specified object for inequality.

7.1.3.15 template<class C > bool qsense::AutoPtr< C >::operator!= (C * *ptr*) const [inline]

Compare the owned object against the specified object for inequality.

7.1.3.16 template<class C > C& qsense::AutoPtr< C >::operator* ( ) [inline]

Dereference operator for the owned object. Will lead to program termination if the owned object is not valid.

7.1.3.17 template<class C > const C& qsense::AutoPtr< C >::operator* ( ) const [inline]

Dereference operator for the owned object. Will lead to program termination if the owned object is not valid.

7.1.3.18 template<class C > C* qsense::AutoPtr< C >::operator->( ) [inline]

Pointer acess operator for the owned object. Returns NULL if invalid.

7.1.3.19 template<class C > const C* qsense::AutoPtr< C >::operator->( ) const [inline]

Pointer acess operator for the owned object. Returns NULL if invalid.

7.1.3.20 template<class C > bool qsense::AutoPtr< C >::operator<(const AutoPtr< C > & *ptr*) const [inline]

Compare the owned objects for ordering.

7.1.3.21 template<class C > bool qsense::AutoPtr< C >::operator<(const C * *ptr*) const [inline]

Compare the owned object against the specified object for ordering.

7.1.3.22 template<class C > bool qsense::AutoPtr< C >::operator<(C * *ptr*) const [inline]

Compare the owned object against the specified object for ordering.

7.1.3.23 template<class C > bool qsense::AutoPtr< C >::operator<=(const AutoPtr< C > & *ptr*) const [inline]

7.1.3.24 template<class C > bool qsense::AutoPtr< C >::operator<=(const C * *ptr*) const [inline]

7.1.3.25 template<class C > bool qsense::AutoPtr< C >::operator<=(C * *ptr*) const [inline]

7.1.3.26 template<class C > AutoPtr& qsense::AutoPtr< C >::operator=(C * *ptr*) [inline]

Copy assignment operator. Delegates to assign.

7.1.3.27 template<class C > AutoPtr& qsense::AutoPtr< C >::operator=(const AutoPtr< C > & *ptr*) [inline]

Copy assignment operator. Delegates to assign.

7.1.3.28 template<class C > template<class Other > AutoPtr& qsense::AutoPtr< C >::operator=(const AutoPtr< Other > & *ptr*) [inline]

Copy assignment operator. Delegates to assign.

7.1.3.29 template<class C > bool qsense::AutoPtr< C >::operator==(const AutoPtr< C > & *ptr*) const [inline]

Compare the owned objects for equality.

7.1.3.30 template<class C > bool qsense::AutoPtr< C >::operator==(const C * *ptr*) const [inline]

Compare the owned object against the specified object for equality.

7.1.3.31 template<class C > bool qsense::AutoPtr< C >::operator== (C * *ptr*) const [inline]

Compare the owned object against the specified object for equality.

7.1.3.32 template<class C > bool qsense::AutoPtr< C >::operator> (const AutoPtr< C > & *ptr*) const [inline]

7.1.3.33 template<class C > bool qsense::AutoPtr< C >::operator> (const C * *ptr*) const [inline]

7.1.3.34 template<class C > bool qsense::AutoPtr< C >::operator> (C * *ptr*) const [inline]

7.1.3.35 template<class C > bool qsense::AutoPtr< C >::operator>= (const AutoPtr< C > & *ptr*) const [inline]

7.1.3.36 template<class C > bool qsense::AutoPtr< C >::operator>= (const C * *ptr*) const [inline]

7.1.3.37 template<class C > bool qsense::AutoPtr< C >::operator>= (C * *ptr*) const [inline]

7.1.3.38 template<class C > void qsense::AutoPtr< C >::swap (AutoPtr< C > & *ptr*) [inline]

Swap the pointers of this instance with the specified instance.

7.1.3.39 template<class C > template<class Other > AutoPtr<Other> qsense::AutoPtr< C >::unsafeCast ( ) const [inline]

Casts the AutoPtr via a static cast to the given type. Example: (assume class Sub: public Super) AutoPtr<Super> super(new Sub()); AutoPtr<Sub> sub = super.unsafeCast<Sub>(); poco_assert (sub.get());.

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

- AutoPtr.h

7.2 qsense::ByteOrder Class Reference

This class contains a number of static methods to convert between big-endian and little-endian integers of various sizes.

`#include <ByteOrder.h>`

Static Public Member Functions

- static int16_t flipBytes (int16_t value)
- static uint16_t flipBytes (uint16_t value)
- static int32_t flipBytes (int32_t value)
- static uint32_t flipBytes (uint32_t value)
- static int64_t flipBytes (int64_t value)
- static uint64_t flipBytes (uint64_t value)
- static int16_t toBigEndian (int16_t value)
- static uint16_t toBigEndian (uint16_t value)
- static int32_t toBigEndian (int32_t value)
- static uint32_t toBigEndian (uint32_t value)
- static int64_t toBigEndian (int64_t value)
- static uint64_t toBigEndian (uint64_t value)
- static int16_t fromBigEndian (int16_t value)
- static uint16_t fromBigEndian (uint16_t value)

- static int32_t fromBigEndian (int32_t value)
- static uint32_t fromBigEndian (uint32_t value)
- static int64_t fromBigEndian (int64_t value)
- static uint64_t fromBigEndian (uint64_t value)
- static int16_t toLittleEndian (int16_t value)
- static uint16_t toLittleEndian (uint16_t value)
- static int32_t toLittleEndian (int32_t value)
- static uint32_t toLittleEndian (uint32_t value)
- static int64_t toLittleEndian (int64_t value)
- static uint64_t toLittleEndian (uint64_t value)
- static int16_t fromLittleEndian (int16_t value)
- static uint16_t fromLittleEndian (uint16_t value)
- static int32_t fromLittleEndian (int32_t value)
- static uint32_t fromLittleEndian (uint32_t value)
- static int64_t fromLittleEndian (int64_t value)
- static uint64_t fromLittleEndian (uint64_t value)
- static int16_t toNetwork (int16_t value)
- static uint16_t toNetwork (uint16_t value)
- static int32_t toNetwork (int32_t value)
- static uint32_t toNetwork (uint32_t value)
- static int64_t toNetwork (int64_t value)
- static uint64_t toNetwork (uint64_t value)
- static int16_t fromNetwork (int16_t value)
- static uint16_t fromNetwork (uint16_t value)
- static int32_t fromNetwork (int32_t value)
- static uint32_t fromNetwork (uint32_t value)
- static int64_t fromNetwork (int64_t value)
- static uint64_t fromNetwork (uint64_t value)

7.2.1 Detailed Description

This class contains a number of static methods to convert between big-endian and little-endian integers of various sizes.

7.2.2 Member Function Documentation

7.2.2.1 int16_t qsense::ByteOrder::flipBytes (int16_t *value*) [inline], [static]

7.2.2.2 uint16_t qsense::ByteOrder::flipBytes (uint16_t *value*) [inline], [static]

7.2.2.3 int32_t qsense::ByteOrder::flipBytes (int32_t *value*) [inline], [static]

7.2.2.4 uint32_t qsense::ByteOrder::flipBytes (uint32_t *value*) [inline], [static]

7.2.2.5 int64_t qsense::ByteOrder::flipBytes (int64_t *value*) [inline], [static]

7.2.2.6 uint64_t qsense::ByteOrder::flipBytes (uint64_t *value*) [inline], [static]

7.2.2.7 static int16_t qsense::ByteOrder::fromBigEndian (int16_t *value*) [static]

7.2.2.8 static uint16_t qsense::ByteOrder::fromBigEndian (uint16_t *value*) [static]

7.2.2.9 static int32_t qsense::ByteOrder::fromBigEndian (int32_t *value*) [static]

7.2.2.10 static uint32_t qsense::ByteOrder::fromBigEndian (uint32_t *value*) [static]

7.2.2.11 static int64_t qsense::ByteOrder::fromBigEndian (int64_t *value*) [static]

7.2.2.12 static uint64_t qsense::ByteOrder::fromBigEndian (uint64_t *value*) [static]

7.2.2.13 static int16_t qsense::ByteOrder::fromLittleEndian (int16_t *value*) [static]

7.2.2.14 static uint16_t qsense::ByteOrder::fromLittleEndian (uint16_t *value*) [static]

7.2.2.15 static int32_t qsense::ByteOrder::fromLittleEndian (int32_t *value*) [static]

7.2.2.16 static uint32_t qsense::ByteOrder::fromLittleEndian (uint32_t *value*) [static]

7.2.2.17 static int64_t qsense::ByteOrder::fromLittleEndian (int64_t *value*) [static]

7.2.2.18 static uint64_t qsense::ByteOrder::fromLittleEndian (uint64_t *value*) [static]

7.2.2.19 static int16_t qsense::ByteOrder::fromNetwork (int16_t *value*) [static]

7.2.2.20 static uint16_t qsense::ByteOrder::fromNetwork (uint16_t *value*) [static]

7.2.2.21 static int32_t qsense::ByteOrder::fromNetwork (int32_t *value*) [static]

7.2.2.22 static uint32_t qsense::ByteOrder::fromNetwork (uint32_t *value*) [static]

7.2.2.23 static int64_t qsense::ByteOrder::fromNetwork (int64_t *value*) [static]

7.2.2.24 static uint64_t qsense::ByteOrder::fromNetwork (uint64_t *value*) [static]

7.2.2.25 static int16_t qsense::ByteOrder::toBigEndian (int16_t *value*) [static]

7.2.2.26 static uint16_t qsense::ByteOrder::toBigEndian (uint16_t *value*) [static]

7.2.2.27 static int32_t qsense::ByteOrder::toBigEndian (int32_t *value*) [static]

7.2.2.28 static uint32_t qsense::ByteOrder::toBigEndian (uint32_t *value*) [static]

7.2.2.29 static int64_t qsense::ByteOrder::toBigEndian (int64_t *value*) [static]

7.2.2.30 static uint64_t qsense::ByteOrder::toBigEndian (uint64_t *value*) [static]

7.2.2.31 static int16_t qsense::ByteOrder::toLittleEndian (int16_t *value*) [static]

7.2.2.32 static uint16_t qsense::ByteOrder::toLittleEndian (uint16_t *value*) [static]

7.2.2.33 static int32_t qsense::ByteOrder::toLittleEndian (int32_t *value*) [static]

7.2.2.34 static uint32_t qsense::ByteOrder::toLittleEndian (uint32_t *value*) [static]

7.2.2.35 static int64_t qsense::ByteOrder::toLittleEndian (int64_t *value*) [static]

7.2.2.36 static uint64_t qsense::ByteOrder::toLittleEndian (uint64_t *value*) [static]

7.2.2.37 static int16_t qsense::ByteOrder::toNetwork (int16_t *value*) [static]

7.2.2.38 static uint16_t qsense::ByteOrder::toNetwork (uint16_t *value*) [static]

7.2.2.39 static int32_t qsense::ByteOrder::toNetwork (int32_t *value*) [static]

7.2.2.40 static uint32_t qsense::ByteOrder::toNetwork (uint32_t *value*) [static]

7.2.2.41 static int64_t qsense::ByteOrder::toNetwork (int64_t *value*) [static]

7.2.2.42 static uint64_t qsense::ByteOrder::toNetwork (uint64_t *value*) [static]

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

- ByteOrder.h

7.3 qsense::hash::Sha1::Context Struct Reference

SHA1 context representation.

`#include <Sha1.h>`

Public Attributes

- unsigned long total [2]
- unsigned long state [5]
- unsigned char buffer [64]
- unsigned char ipad [64]
- unsigned char opad [64]

7.3.1 Detailed Description

SHA1 context representation.

7.3.2 Member Data Documentation

7.3.2.1 unsigned char qsense::hash::Sha1::Context::buffer[64]

data block being processed

7.3.2.2 unsigned char qsense::hash::Sha1::Context::ipad[64]

HMAC: inner padding

7.3.2.3 unsigned char qsense::hash::Sha1::Context::opad[64]

HMAC: outer padding

7.3.2.4 unsigned long qsense::hash::Sha1::Context::state[5]

intermediate digest state

7.3.2.5 unsigned long qsense::hash::Sha1::Context::total[2]

number of bytes processed

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

- Sha1.h

7.4 qsense::net::DateTime Class Reference

Represents current date/time. Seeds initially (and daily) from a network time service, and uses internal timer to represent a real-time clock.

`#include <DateTime.h>`

Public Member Functions

- DateTime ()

  Default constructor. Use singleton in general.
- const qsense::QString currentTime ()

  Return the current date/time in ISO 8601 format.
- const qsense::QString date ()

  Return the current date in ISO 8601 format.
- int64_t currentTimeMillis ()

  Return the milli seconds since UNIX epoch.

Static Public Member Functions

- static DateTime & singleton ()

  Return a singleton instance to use. This is the preferred way of using this class.

7.4.1 Detailed Description

Represents current date/time. Seeds initially (and daily) from a network time service, and uses internal timer to represent a real-time clock.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 qsense::net::DateTime::DateTime ( )

Default constructor. Use singleton in general.

7.4.3 Member Function Documentation

7.4.3.1 const qsense::QString qsense::net::DateTime::currentTime ( )

Return the current date/time in ISO 8601 format.

7.4.3.2 int64_t qsense::net::DateTime::currentTimeMillis ( )

Return the milli seconds since UNIX epoch.

7.4.3.3 const qsense::QString qsense::net::DateTime::date ( )

Return the current date in ISO 8601 format.

7.4.3.4 static DateTime& qsense::net::DateTime::singleton ( ) [inline], [static]

Return a singleton instance to use. This is the preferred way of using this class.

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

- DateTime.h

7.5 qsense::Event Class Reference

A simple class that encapsulates an event sent to Sidecar. Events are holders for readings. Events can be serialised to JSON using the toString method.

`#include <Event.h>`

Public Types

- typedef std::vector< Reading > Readings

  The vector of readings encapsulated in this event.
- typedef std::vector< QString > Tags

  The vector of tags associated with this event.
- typedef std::map< QString, Tags > KeyTags

  The map of key tags associated with this event.
- typedef Readings::const_iterator ReadingsIterator

  Iterator for the readings encapsulated in this event.
- typedef Tags::const_iterator TagsIterator

  Iterator for the tags associated with this event.
- typedef KeyTags::const_iterator KeyTagsIterator

  Iterator for the key=tags associated with this event.

Public Member Functions

- Event ()

  Default constructor. Uses default location set through init.
- Event (const Location &location)

  Create a new event with the specified location.
- ~Event ()

  Destructor. No actions required.
- Event & add (const Reading &reading)

  Add the specified reading to this event.
- Event & add (const QString &tag)

  Add the specified tag to this event. ***NOTE:*** *Tags should be single words without spaces.*
- Event & add (const QString &key, const QString &tag)

  Add the specified key-tag to this event. To specify multiple tags for the same key, call this method with the same key. ***NOTE:*** *Tags should be single words without spaces.*
- Event & operator+= (const Reading &reading)

  Operator for adding a reading to the event.

- Event & operator+= (const QString &tag)

  *Operator for adding a tag to the event. **NOTE:** Tags should be single words without spaces.*
- std::size_t numberOfReadings () const

  Return the number of readings in this event.
- std::size_t numberOfTags () const

  Return the number of tags associated with this event.
- std::size_t numberOfKeyTags () const

  Return the number of key-tags associated with this event.
- ReadingsIterator beginReadings () const

  Return a constant iterator to the beginning of the readings vector.
- ReadingsIterator endReadings () const

  The end of the readings vector to check in loops.
- TagsIterator beginTags () const

  Return a constant iterator to the beginning of the tags vector.
- TagsIterator endTags () const

  The end of the tags vector to check in loops.
- KeyTagsIterator beginKeyTags () const

  Return a constant iterator to the beginning of the key-tags vector.
- KeyTagsIterator endKeyTags () const

  The end of the key-tags map to check in loops.
- const qsense::Reading & operator[ ] (std::size_t index) const

  operator [] Retrieve the reading at specified index. Will throw exception if index is out of bounds. Check the size of the container before using this operator.
- const qsense::Location & getLocation () const

  Return the location used by this event.
- const qsense::QString toString () const

  Serialise the event to JSON.

Static Public Member Functions

- static void init (const qsense::UUID &deviceId, const qsense::QString &stream, const qsense::Location &location)

  Initialise the Event API.

7.5.1 Detailed Description

A simple class that encapsulates an event sent to Sidecar. Events are holders for readings. Events can be serialised to JSON using the toString method.

7.5.2 Member Typedef Documentation

7.5.2.1 typedef std::map<QString,Tags> qsense::Event::KeyTags

The map of key tags associated with this event.

7.5.2.2 typedef KeyTags::const_iterator qsense::Event::KeyTagsIterator

Iterator for the key=tags associated with this event.

7.5.2.3 typedef std::vector<Reading> qsense::Event::Readings

The vector of readings encapsulated in this event.

7.5.2.4 typedef Readings::const_iterator qsense::Event::ReadingsIterator

Iterator for the readings encapsulated in this event.

7.5.2.5 typedef std::vector<QString> qsense::Event::Tags

The vector of tags associated with this event.

7.5.2.6 typedef Tags::const_iterator qsense::Event::TagsIterator

Iterator for the tags associated with this event.

7.5.3 Constructor & Destructor Documentation

7.5.3.1 qsense::Event::Event ( )

Default constructor. Uses default location set through init.

7.5.3.2 qsense::Event::Event (const Location & *location*)

Create a new event with the specified location.

7.5.3.3 qsense::Event::~Event ( ) [inline]

Destructor. No actions required.

7.5.4 Member Function Documentation

7.5.4.1 Event& qsense::Event::add (const Reading & *reading*)

Add the specified reading to this event.

7.5.4.2 Event& qsense::Event::add (const QString & *tag*)

Add the specified tag to this event. **NOTE:** Tags should be single words without spaces.

7.5.4.3 Event& qsense::Event::add (const QString & *key,* const QString & *tag*)

Add the specified key-tag to this event. To specify multiple tags for the same key, call this method with the same key. **NOTE:** Tags should be single words without spaces.

7.5.4.4 KeyTagsIterator qsense::Event::beginKeyTags ( ) const [inline]

Return a constant iterator to the beginning of the key-tags vector.

7.5.4.5 ReadingsIterator qsense::Event::beginReadings ( ) const [inline]

Return a constant iterator to the beginning of the readings vector.

7.5.4.6 TagsIterator qsense::Event::beginTags ( ) const [inline]

Return a constant iterator to the beginning of the tags vector.

7.5.4.7 KeyTagsIterator qsense::Event::endKeyTags ( ) const [inline]

The end of the key-tags map to check in loops.

7.5.4.8 ReadingsIterator qsense::Event::endReadings ( ) const [inline]

The end of the readings vector to check in loops.

7.5.4.9 TagsIterator qsense::Event::endTags ( ) const [inline]

The end of the tags vector to check in loops.

7.5.4.10 const qsense::Location& qsense::Event::getLocation ( ) const [inline]

Return the location used by this event.

7.5.4.11 static void qsense::Event::init (const qsense::UUID & *deviceId,* const qsense::QString & *stream,* const qsense::Location & *location*) [static]

Initialise the Event API.

Parameters

deviceId	The deviceId to use. No way at present to retrieve using API
stream	The stream identifier to use with Sidecar
location	A default location to use. No location tracking at present

7.5.4.12 std::size_t qsense::Event::numberOfKeyTags ( ) const [inline]

Return the number of key-tags associated with this event.

7.5.4.13 std::size_t qsense::Event::numberOfReadings ( ) const [inline]

Return the number of readings in this event.

7.5.4.14 std::size_t qsense::Event::numberOfTags ( ) const [inline]

Return the number of tags associated with this event.

7.5.4.15 Event& qsense::Event::operator+= (const Reading & *reading*) [inline]

Operator for adding a reading to the event.

7.5.4.16 Event& qsense::Event::operator+= (const QString & *tag*) [inline]

Operator for adding a tag to the event. **NOTE:** Tags should be single words without spaces.

7.5.4.17 const qsense::Reading& qsense::Event::operator[ ] (std::size_t *index*) const [inline]

operator [] Retrieve the reading at specified index. Will throw exception if index is out of bounds. Check the size of the container before using this operator.

Parameters

index	The index into the vector of readings.

Returns

The reading at the specified index.

7.5.4.18 const qsense::QString qsense::Event::toString ( ) const

Serialise the event to JSON.

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

- Event.h

7.6 SimpleSidecarClient::EventAPIData Struct Reference

A simple data structure that encapsulates the data required to initialise the Event API.

`#include <SimpleSidecarClient.h>`

Public Attributes

- String deviceUUID

  The device identifier for the hardware. This value should be considered static (e.g., a MAC Address).
- String stream

  The user defined stream name.
- float latitude

  Latitude for current device location.
- float longitude

  Longitude for current device location.

7.6.1 Detailed Description

A simple data structure that encapsulates the data required to initialise the Event API.

7.6.2 Member Data Documentation

7.6.2.1 String SimpleSidecarClient::EventAPIData::deviceUUID

The device identifier for the hardware. This value should be considered static (e.g., a MAC Address).

Note: Sidecar expects this value to be a 36 character UUID.

7.6.2.2 float SimpleSidecarClient::EventAPIData::latitude

Latitude for current device location.

7.6.2.3 float SimpleSidecarClient::EventAPIData::longitude

Longitude for current device location.

7.6.2.4 String SimpleSidecarClient::EventAPIData::stream

The user defined stream name.

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

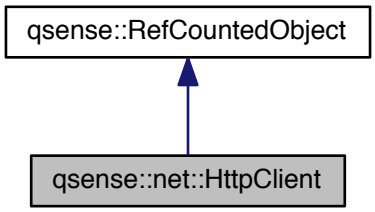
- SimpleSidecarClient.h

7.7 qsense::net::HttpClient Class Reference

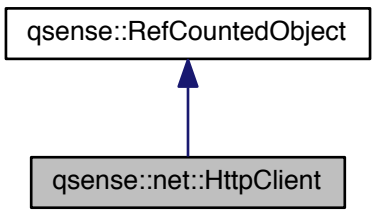
A HTTP Client class for use with either ethernet or wifi. Before use, the client should be initialised with the network type used by the device (qsense::net::initNetworkType.

`#include <QHttpClient.h>`

Inheritance diagram for qsense::net::HttpClient:



Collaboration diagram for qsense::net::HttpClient:



Public Types

- typedef AutoPtr< HttpClient > Ptr

  Type for auto pointer to a http client instance.

Public Member Functions

- HttpClient ()

  Default constructor.
- virtual ~HttpClient ()

  Destructor for sub-classes.
- virtual int16_t connect (const qsense::QString &server, uint16_t port=80)=0

  Make a socket connection to the specified server on specified port (default 80)
- virtual bool connected ()=0

  Check to see if the client is connected to the server.
- virtual uint16_t get (const HttpRequest &request)=0

  Perform a GET request using information in the request object.
- virtual uint16_t post (const HttpRequest &request)=0

  Perform a POST request using information in the request object.
- virtual uint16_t remove (const HttpRequest &request)=0

  Perform a DELETE request using information in the request object. Named remove to get around delete being keyword.
- virtual const qsense::QString readLine ()=0

  Read a line from the HTTP response.
- virtual HttpRequest::Map readHeaders ()=0

  Return a map of the HTTP response headers.
- virtual const qsense::QString readBody ()=0

  Read the entire contents of the server response body. Note: This method also reads headers. If headers have already been read, it may end up losing some of the response body.

Static Public Member Functions

- static Ptr create ()

  Factory method for creating concrete instances based on initialisation.

Protected Member Functions

- virtual void writeHeaders (const HttpRequest &request, bool close=true)=0

  Send the specified request headers to the HTTP server.

7.7.1 Detailed Description

A HTTP Client class for use with either ethernet or wifi. Before use, the client should be initialised with the network type used by the device (qsense::net::initNetworkType.

7.7.2 Member Typedef Documentation

7.7.2.1 typedef AutoPtr<HttpClient> qsense::net::HttpClient::Ptr

Type for auto pointer to a http client instance.

7.7.3 Constructor & Destructor Documentation

7.7.3.1 qsense::net::HttpClient::HttpClient ( ) [inline]

Default constructor.

7.7.3.2 virtual qsense::net::HttpClient::~HttpClient ( ) [inline], [virtual]

Destructor for sub-classes.

7.7.4 Member Function Documentation

7.7.4.1 virtual int16_t qsense::net::HttpClient::connect (const qsense::QString & *server,* uint16_t *port* = 80) [pure virtual]

Make a socket connection to the specified server on specified port (default 80)

7.7.4.2 virtual bool qsense::net::HttpClient::connected ( ) [pure virtual]

Check to see if the client is connected to the server.

7.7.4.3 static Ptr qsense::net::HttpClient::create ( ) [static]

Factory method for creating concrete instances based on initialisation.

Returns

An instance that uses either ethernet or wifi to connect to the network. Callers must delete the returned instance.

7.7.4.4 virtual uint16_t qsense::net::HttpClient::get (const HttpRequest & *request*) [pure virtual]

Perform a GET request using information in the request object.

Parameters

request	The request object that encapsulates the uri and other relevant information

Returns

The HTTP response code from server.

7.7.4.5 virtual uint16_t qsense::net::HttpClient::post (const HttpRequest & *request*) [pure virtual]

Perform a POST request using information in the request object.

Parameters

request	The request object that encapsulates the uri and other relevant information

Returns

The HTTP response code from server.

7.7.4.6 virtual const qsense::QString qsense::net::HttpClient::readBody ( ) [pure virtual]

Read the entire contents of the server response body. Note: This method also reads headers. If headers have already been read, it may end up losing some of the response body.

WARNING: Use with caution. Can run embedded devices out of memory very easily.

Returns

The entire http response body content.

7.7.4.7 virtual HttpRequest::Map qsense::net::HttpClient::readHeaders ( ) [pure virtual]

Return a map of the HTTP response headers.

7.7.4.8 virtual const qsense::QString qsense::net::HttpClient::readLine ( ) [pure virtual]

Read a line from the HTTP response.

A line can be either a header or content. Use to process raw HTTP response line by line.

Returns

A line (content until newline character) of text from raw response.

7.7.4.9 virtual uint16_t qsense::net::HttpClient::remove (const HttpRequest & *request*) [pure virtual]

Perform a DELETE request using information in the request object. Named remove to get around delete being keyword.

Parameters

request	The request object that encapsulates the uri and other relevant information

Returns

The HTTP response code from server.

7.7.4.10 virtual void qsense::net::HttpClient::writeHeaders (const HttpRequest & *request,* bool *close* = true) [protected], [pure virtual]

Send the specified request headers to the HTTP server.

Parameters

headers	The map of headers to send to the server.
close	Flag indicating whether HTTP keep-alive is NOT to be used.

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

- QHttpClient.h

7.8 qsense::net::HttpRequest Class Reference

A simple class that represents a HTTP request. Request encapsulates the URI path, any request parameters, header attributes, body etc. as appropriate.

`#include <HttpRequest.h>`

Public Types

- typedef std::map< QString, QString > Map

  Map used to represent request parameters and headers.
- typedef Map::const_iterator Iterator

  Constant iterator to access contents of the parameters and headers.

Public Member Functions

- HttpRequest (const QString &uri)

  Constructor. Create a request for the specified server resource.
- ~HttpRequest ()

  Destructor. No actions required.
- const QString & getUri () const

  Return the uri for which this request was created.
- const QString & getBody () const

  Return the body to send as part of the request. For GET requests, this will be empty.
- HttpRequest & setBody (const QString &body)

  Set the body to send as part of the request. This is meant for use primarily with POST/PUT type requests.
- HttpRequest & setParameter (const QString &key, const QString &value)

  Add the specified key/value combination as a request parameter to this request. If a mapping already exists with the specified key, it will be replaced with the specified value.
- HttpRequest & setHeader (const QString &key, const QString &value)

  Add the specified key/value combination as a request attribute to this request. If a mapping already exists with the specified key, it will be replaced with the specified value.
- const QString getParamters () const

  Return the requests parameters as a string. ***Note:*** *A leading ? symbol will not be added, which is required for GET requests. Calls must add if making a GET request.*
- Iterator beginHeaders () const

  Return a constant iterator to the beginning of the headers map.
- Iterator endHeaders () const

  Return a constant iterator to the beginning of the headers map.

7.8.1 Detailed Description

A simple class that represents a HTTP request. Request encapsulates the URI path, any request parameters, header attributes, body etc. as appropriate.

7.8.2 Member Typedef Documentation

7.8.2.1 typedef Map::const_iterator qsense::net::HttpRequest::Iterator

Constant iterator to access contents of the parameters and headers.

7.8.2.2 typedef std::map<QString,QString> qsense::net::HttpRequest::Map

Map used to represent request parameters and headers.

7.8.3 Constructor & Destructor Documentation

7.8.3.1 qsense::net::HttpRequest::HttpRequest (const QString & *uri*)

Constructor. Create a request for the specified server resource.

7.8.3.2 qsense::net::HttpRequest::~HttpRequest ( ) [inline]

Destructor. No actions required.

7.8.4 Member Function Documentation

7.8.4.1 Iterator qsense::net::HttpRequest::beginHeaders ( ) const [inline]

Return a constant iterator to the beginning of the headers map.

7.8.4.2 Iterator qsense::net::HttpRequest::endHeaders ( ) const [inline]

Return a constant iterator to the beginning of the headers map.

7.8.4.3 const QString& qsense::net::HttpRequest::getBody ( ) const [inline]

Return the body to send as part of the request. For GET requests, this will be empty.

7.8.4.4 const QString qsense::net::HttpRequest::getParamters ( ) const

Return the requests parameters as a string. **Note:** A leading ? symbol will not be added, which is required for GET requests. Calls must add if making a GET request.

7.8.4.5 const QString& qsense::net::HttpRequest::getUri ( ) const [inline]

Return the uri for which this request was created.

7.8.4.6 HttpRequest& qsense::net::HttpRequest::setBody (const QString & *body*)

Set the body to send as part of the request. This is meant for use primarily with POST/PUT type requests.

7.8.4.7 HttpRequest& qsense::net::HttpRequest::setHeader (const QString & *key,* const QString & *value*)

Add the specified key/value combination as a request attribute to this request. If a mapping already exists with the specified key, it will be replaced with the specified value.

7.8.4.8 HttpRequest& qsense::net::HttpRequest::setParameter (const QString & *key,* const QString & *value*)

Add the specified key/value combination as a request parameter to this request. If a mapping already exists with the specified key, it will be replaced with the specified value.

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

- HttpRequest.h

7.9 qsense::Location Class Reference

A simple representation of geographical location.

`#include <Location.h>`

Public Member Functions

- Location ()

  Default constructor.
- Location (float lat, float lon)

  Create a new instance with the specified co-ordinates.
- Location (const Location &location)

  Copy constructor.
- ~Location ()

  Destructor. No action required.
- Location & operator= (const Location &location)

  Copy assignment operator.
- const qsense::QString toString () const

  Serialise this instance to a JSON representation.
- float getLatitude () const
- float getLongitude () const

7.9.1 Detailed Description

A simple representation of geographical location.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 qsense::Location::Location ( ) [inline]

Default constructor.

7.9.2.2 qsense::Location::Location (float *lat,* float *lon*) [inline]

Create a new instance with the specified co-ordinates.

Parameters

lat	The latitude
lon	The longitude

7.9.2.3 qsense::Location::Location (const Location & *location*) [inline]

Copy constructor.

7.9.2.4 qsense::Location::~Location ( ) [inline]

Destructor. No action required.

7.9.3 Member Function Documentation

7.9.3.1 float qsense::Location::getLatitude ( ) const [inline]

Returns

Return the latitude value

7.9.3.2 float qsense::Location::getLongitude ( ) const [inline]

Returns

Return the longitude value

7.9.3.3 Location& qsense::Location::operator= (const Location & *location*)

Copy assignment operator.

7.9.3.4 const qsense::QString qsense::Location::toString ( ) const

Serialise this instance to a JSON representation.

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

- Location.h

7.10 qsense::hash::MD5 Class Reference

Class for generating MD5 hashes.

`#include <MD5.h>`

Public Types

- typedef qsense::Byte Byte
  8-bit byte
- typedef uint32_t Word
  32-bit word

Public Member Functions

- MD5 ()
  Default constructor.
- ~MD5 ()
  Destructor. Destroys the context.
- void compute (const Byte *data, Word nbytes, Byte digest[MD5_HASH_LENGTH])
- qsense::QString compute (const qsense::QString &input)

7.10.1 Detailed Description

Class for generating MD5 hashes.

7.10.2 Member Typedef Documentation

7.10.2.1 typedef qsense::Byte qsense::hash::MD5::Byte

8-bit byte

7.10.2.2 typedef uint32_t qsense::hash::MD5::Word

32-bit word

7.10.3 Constructor & Destructor Documentation

7.10.3.1 qsense::hash::MD5::MD5 ( ) [inline]

Default constructor.

7.10.3.2 qsense::hash::MD5::~MD5 ( ) [inline]

Destructor. Destroys the context.

7.10.4 Member Function Documentation

7.10.4.1 void qsense::hash::MD5::compute (const Byte * *data,* Word *nbytes,* Byte *digest[MD5_HASH_LENGTH]*)

Compute the MD5 digest for the specified data of length nbytes into digest

7.10.4.2 qsense::QString qsense::hash::MD5::compute (const qsense::QString & *input*)

Compute MD5 digest for specified data and return base64 encoded string

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

- MD5.h

7.11 qsense::Reading Class Reference

A class that represents a single reading. Readings are added to an Event.

`#include <Reading.h>`

Public Member Functions

- Reading (const qsense::QString &k, const qsense::QString &v, const qsense::QString &ts=qsense::net::DateTime::singleton().currentTime())

  Create a new reading with specified values.
- Reading (const qsense::QString &k, float v, const qsense::QString &ts=qsense::net::DateTime::singleton().currentTime())

  Create a new reading with specified values.
- ~Reading ()

  Destructor. No actions required.
- const qsense::QString & getKey () const

Return the key for the reading.

- const qsense::QString & getValue () const

  Return the value of the reading.

- const qsense::QString & getTimestamp () const

  Return the time at which the reading was taken.

- const qsense::QString toString () const

  Return a JSON representation of the reading.

7.11.1 Detailed Description

A class that represents a single reading. Readings are added to an Event.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 qsense::Reading::Reading (const qsense::QString & *k*, const qsense::QString & *v*, const qsense::QString & *ts* = qsense::net::DateTime::singleton().currentTime()) [inline]

Create a new reading with specified values.

Parameters

k	The key to associate with the reading
v	The value of the reading
ts	The timestamp (optional) at which reading was taken.

7.11.2.2 qsense::Reading::Reading (const qsense::QString & *k*, float *v*, const qsense::QString & *ts* = qsense::net::DateTime::singleton().currentTime())

Create a new reading with specified values.

Parameters

k	The key to associate with the reading
v	The float value of the reading
ts	The timestamp (optional) at which reading was taken.

7.11.2.3 qsense::Reading::~Reading ( ) [inline]

Destructor. No actions required.

7.11.3 Member Function Documentation

7.11.3.1 const qsense::QString& qsense::Reading::getKey ( ) const [inline]

Return the key for the reading.

7.11.3.2 const qsense::QString& qsense::Reading::getTimestamp ( ) const [inline]

Return the time at which the reading was taken.

7.11.3.3 const qsense::QString& qsense::Reading::getValue ( ) const [inline]

Return the value of the reading.

7.11.3.4 const qsense::QString qsense::Reading::toString ( ) const

Return a JSON representation of the reading.

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

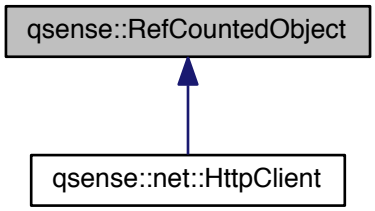
- Reading.h

7.12 qsense::RefCountedObject Class Reference

A base class for objects that employ reference counting based garbage collection.

`#include <RefCountedObject.h>`

Inheritance diagram for qsense::RefCountedObject:



Public Member Functions

- RefCountedObject ()

  Creates the RefCountedObject. The initial reference count is one.
- void duplicate () const

  Increments the object's reference count.
- void release () const

  Decrements the object's reference count and deletes the object if the count reaches zero.
- int referenceCount () const

  Returns the reference count.

Protected Member Functions

- virtual ~RefCountedObject ()

  Destroys the RefCountedObject.

7.12.1 Detailed Description

A base class for objects that employ reference counting based garbage collection.

Reference-counted objects inhibit construction by copying and assignment.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 qsense::RefCountedObject::RefCountedObject ( ) [inline]

Creates the RefCountedObject. The initial reference count is one.

7.12.2.2 virtual qsense::RefCountedObject::~RefCountedObject ( ) [inline], [protected], [virtual]

Destroys the RefCountedObject.

7.12.3 Member Function Documentation

7.12.3.1 void qsense::RefCountedObject::duplicate ( ) const [inline]

Increments the object's reference count.

7.12.3.2 int qsense::RefCountedObject::referenceCount ( ) const [inline]

Returns the reference count.

7.12.3.3 void qsense::RefCountedObject::release ( ) const [inline]

Decrements the object's reference count and deletes the object if the count reaches zero.

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

- RefCountedObject.h

7.13 qsense::hash::Sha1 Class Reference

Class for hashing using SHA1 algorithm.

`#include <Sha1.h>`

Classes

- struct Context

  SHA1 context representation.

Public Member Functions

- Sha1 ()

  Default constructor.
- ~Sha1 ()

  Destructor. No actions required.

- void hash (unsigned char *input, int ilen, unsigned char output[20])

  Generate SHA1 hash for specified input into output array.
- qsense::QString hash (const qsense::QString &input)

  Generate SHA1 hash for the specified input string.
- void hmac (unsigned char *key, int keylen, unsigned char *input, int ilen, unsigned char output[20])

  Generate SHA1 HMAC for the specified input using specified key.
- qsense::QString hmac (const qsense::QString &key, const qsense::QString &input)

  Generate SHA1 HMAC using the specified key for the input string.
- qsense::QString sign (const qsense::QString &privateKey, const qsense::QString &httpMethod, const qsense::QString &uriPath, const qsense::QString &date, const qsense::QString &contentMd5, const qsense::QString &signatureVersion=qsense::QString("1"))

  Generate the signature for the Sidecar Authorization header.

7.13.1 Detailed Description

Class for hashing using SHA1 algorithm.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 qsense::hash::Sha1::Sha1 ( )

Default constructor.

7.13.2.2 qsense::hash::Sha1::~Sha1 ( ) [inline]

Destructor. No actions required.

7.13.3 Member Function Documentation

7.13.3.1 void qsense::hash::Sha1::hash (unsigned char * *input*, int *ilen*, unsigned char *output[20]*)

Generate SHA1 hash for specified input into output array.

Parameters

input	The input char array that is to be hashed.
ilen	The length of the input char array.
output	The output char array into which hash value is written.

7.13.3.2 qsense::QString qsense::hash::Sha1::hash (const qsense::QString & *input*)

Generate SHA1 hash for the specified input string.

7.13.3.3 void qsense::hash::Sha1::hmac (unsigned char * *key*, int *keylen*, unsigned char * *input*, int *ilen*, unsigned char *output[20]*)

Generate SHA1 HMAC for the specified input using specified key.

Parameters

key	The key to use to generate the HMAC
keylen	The length of the key
input	The input char array to hash
ilen	The length of the input char array
output	The output char array into which hash value is written.

7.13.3.4 qsense::QString qsense::hash::Sha1::hmac (const qsense::QString & *key*, const qsense::QString & *input*)

Generate SHA1 HMAC using the specified key for the input string.

7.13.3.5 qsense::QString qsense::hash::Sha1::sign (const qsense::QString & *privateKey*, const qsense::QString & *httpMethod*, const qsense::QString & *uriPath*, const qsense::QString & *date*, const qsense::QString & *contentMd5*, const qsense::QString & *signatureVersion* = qsense::QString("1"))

Generate the signature for the Sidecar Authorization header.

Parameters

privateKey	The api secret to use to sign
httpMethod	The HTTP method used for the Sidecar API interaction
uriPath	The URI path with which to interact
date	The current timestamp
contentMd5	The MD5 hash for the content to be submitted to Sidecar
signatureVersion	The signature version to specify in header

Returns

The Base64 encoded authorisation signature

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

- Sha1.h

7.14 qsense::net::SidecarClient Class Reference

Class that encapsulates interactions with the Sidecar REST API.

`#include <SidecarClient.h>`

Classes

- struct UserResponse

  A simple structure that represents the result of a user provisioning request.

Public Member Functions

- UserResponse createUser (const QString &username, const QString &password)

  Create a new user account with Sidecar. This is usually the first interaction with the Sidecar service. Needed only once per application.
- UserResponse createOrRetrieveAccessKeys (const QString &username, const QString &password)

  Create or retrieve application access keys for the specified user.

- UserResponse authenticate (const QString &username, const QString &password)

  Authenticate the user against Sidecar. Return the existing user key/secret pair to use with the events API.
- int16_t deleteUser (const QString &username, const QString &password)

  deleteUser Deprovision a user from the system. Removes the user account, access key/secret and devices associated with the user.
- bool publish (const Event &event) const

  Publish the specified event to the Sidecar Event API.

Static Public Member Functions

- static void initAPIKey (const QString &apiKey, const QString &apiSecret)

  Initialise the API with the API key and secret used to sign provisioning requests.
- static void initUserKey (const QString &userKey, const QString &userSecret)

  Initialise the API with the user key and secret used to sign event requests.

7.14.1 Detailed Description

Class that encapsulates interactions with the Sidecar REST API.

7.14.2 Member Function Documentation

7.14.2.1 UserResponse qsense::net::SidecarClient::authenticate (const QString & *username*, const QString & *password*)

Authenticate the user against Sidecar. Return the existing user key/secret pair to use with the events API.

Parameters

username	The username of the user to authenticate as
password	The password to use to authenticate

Returns

The response which on success will contain the key/secret pair

7.14.2.2 UserResponse qsense::net::SidecarClient::createOrRetrieveAccessKeys (const QString & *username*, const QString & *password*)

Create or retrieve application access keys for the specified user.

Parameters

username	The username preferred by user (email format)
password	The password to associate with user account (8-20 char length)

Returns

Return a UserResponse struct with key/secret pair populated on success.

7.14.2.3 UserResponse qsense::net::SidecarClient::createUser (const QString & *username,* const QString & *password*)

Create a new user account with Sidecar. This is usually the first interaction with the Sidecar service. Needed only once per application.

If the method returns with empty key/secret pair, it generally indicates that the username is taken or username/password failed format/length rules.

Parameters

username	The username preferred by user (email format)
password	The password to associate with user account (8-20 char length)

Returns

Return a UserResponse struct with key/secret pair populated on success.

7.14.2.4 int16_t qsense::net::SidecarClient::deleteUser (const QString & *username*, const QString & *password*)

deleteUser Deprovision a user from the system. Removes the user account, access key/secret and devices associated with the user.

Parameters

username	The username of the user account
password	The password for the user account.

Returns

The HTTP response code returned by Sidecar. Response code `204` indicates success.

7.14.2.5 static void qsense::net::SidecarClient::initAPIKey (const QString & *apiKey*, const QString & *apiSecret*) [static]

Initialise the API with the API key and secret used to sign provisioning requests.

7.14.2.6 static void qsense::net::SidecarClient::initUserKey (const QString & *userKey*, const QString & *userSecret*) [static]

Initialise the API with the user key and secret used to sign event requests.

7.14.2.7 bool qsense::net::SidecarClient::publish (const Event & *event*) const

Publish the specified event to the Sidecar Event API.

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

- SidecarClient.h

7.15 SimpleSidecarClient Class Reference

A simple client implementation that hides the low-level API.

`#include <SimpleSidecarClient.h>`

Classes

- struct EventAPIData

  A simple data structure that encapsulates the data required to initialise the Event API.
- struct UserResponse

  A simple data structure that represents the response from Sidecar Provisioning API.

Public Types

- enum NetworkType { Ethernet = 0, WiFi = 1 }

  Enumeration of network connection types for device.

Public Member Functions

- SimpleSidecarClient ()

  Default constructor.
- UserResponse authenticate (const String &username, const String &password)

  Authenticate the user against Sidecar. Return the existing user key/secret pair to use with the events API.
- UserResponse createUser (const String &username, const String &password)

  Create a new user account with Sidecar. Use this on first run of application if the authenticate method returned an invalid (not 200) `responseCode`.
- UserResponse createOrRetrieveAccessKeys (const String &username, const String &password)

  Create or retrieve application access keys for the specified user.
- int16_t deleteUser (const String &username, const String &password)

  deleteUser Deprovision a user from the system. Removes the user account, access key/secret and devices associated with the user.
- void addReading (const String &key, const float value)

  Add the specified reading key-value pair to an event. Add as many readings as desired before publishing the event to Sidecar (publish). On publish, the event is re-initialised for publishing subsequent readings.
- void addTag (const String &value)

  Add optional tag values to help analyse the event after publishing to Sidecar.
- void addKeyTag (const String &key, const String &tag)

  Add option key-tag values to help identify/analyse the event after publishing to Sidecar. Use the same key to assign multiple values to a key.
- bool publish ()

  publish Publish the built up event to the Sidecar Event API. Invoke addReading with the individual readings that are part of the current event, and addTag as needed to build up a complete event before publishing to Sidecar.
- const String currentTime ()

  Return the current date/time in ISO 8601 format.
- const String date ()

  Return the current date in ISO 8601 format.
- int64_t currentTimeMillis ()

  Return the milli seconds since UNIX epoch.

Static Public Member Functions

- static void initNetworkType (NetworkType type)

  Initialise the API to use the specified type.
- static void initUUID (byte mac[6])
- static void initUUID ()

  Initialise UUID engine using a random seed.
- static void initAPIKey (const String &apiKey, const String &apiSecret)
- static void initUserKey (const String &userKey, const String &userSecret)
- static void initEventAPI (const EventAPIData &data)

  Initialise the Sidecar Event API.

7.15.1 Detailed Description

A simple client implementation that hides the low-level API.

The low-level API is a cross-platform standard C++ API that has been tested on Mac OS X and Windows in addition to Arduino Mega. The simple client is specific to Arduino and is intended to provide a single class that Arduino applications may use to provision devices and publish event data.

The methods in this class are declared in the same order that a typical calling application will need to make to interact with Sidecar.

7.15.2 Member Enumeration Documentation

7.15.2.1 enum SimpleSidecarClient::NetworkType

Enumeration of network connection types for device.

Enumerator

Ethernet

WiFi

7.15.3 Constructor & Destructor Documentation

7.15.3.1 SimpleSidecarClient::SimpleSidecarClient ( ) [inline]

Default constructor.

7.15.4 Member Function Documentation

7.15.4.1 void SimpleSidecarClient::addKeyTag (const String & *key*, const String & *tag*)

Add option key-tag values to help identify/analyse the event after publishing to Sidecar. Use the same key to assign multiple values to a key.

NOTE: Tags should be single words without spaces.

Parameters

key	The key for the key-tag pair.
tag	A tag value to associate with the key.

7.15.4.2 void SimpleSidecarClient::addReading (const String & *key*, const float *value*)

Add the specified reading key-value pair to an event. Add as many readings as desired before publishing the event to Sidecar (publish). On publish, the event is re-initialised for publishing subsequent readings.

Parameters

key	A user defined key for the reading
value	The value for the reading.

7.15.4.3 void SimpleSidecarClient::addTag (const String & *value*)

Add optional tag values to help analyse the event after publishing to Sidecar.

NOTE: Tags should be single words without spaces.

Parameters

value	A tag value.

7.15.4.4 UserResponse SimpleSidecarClient::authenticate (const String & *username*, const String & *password*)

Authenticate the user against Sidecar. Return the existing user key/secret pair to use with the events API.

This is generally the first call to use on first run of the application. Use `authenticate` before createUser to ensure that the account name (email id) and password are not already in use.

If the response contains a valid (non-empty) keyId and secret, calling applications may chose to save it to flash storage using `PROGMEM`. Once stored the provisioning process is complete, and the application may interact directly with the Event API methods. On each run of the application invoke initUserKey with the stored user key and secret.

Parameters

username	The username of the user to authenticate as
password	The password to use to authenticate

Returns

The response which on success will contain the key/secret pair. In case of failure the `responseCode` value will be other than `200`.

7.15.4.5 UserResponse SimpleSidecarClient::createOrRetrieveAccessKeys (const String & *username*, const String & *password*)

Create or retrieve application access keys for the specified user.

Parameters

username	The username preferred by user (email format)
password	The password to associate with user account (8-20 char length)

Returns

Return a UserResponse struct with key/secret pair populated on success. The `responseCode` value will be `200` on success.

7.15.4.6 UserResponse SimpleSidecarClient::createUser (const String & *username*, const String & *password*)

Create a new user account with Sidecar. Use this on first run of application if the authenticate method returned an invalid (not 200) `responseCode`.

If the method returns with empty key/secret pair, it generally indicates that the username is taken or username/password failed format/length rules.

Parameters

username	The username preferred by user (email format)
password	The password to associate with user account (8-20 char length)

Returns

Return a UserResponse struct with key/secret pair populated on success.

7.15.4.7 const String SimpleSidecarClient::currentTime ( )

Return the current date/time in ISO 8601 format.

7.15.4.8 int64_t SimpleSidecarClient::currentTimeMillis ( )

Return the milli seconds since UNIX epoch.

7.15.4.9 const String SimpleSidecarClient::date ( )

Return the current date in ISO 8601 format.

7.15.4.10 int16_t SimpleSidecarClient::deleteUser (const String & *username,* const String & *password*)

deleteUser Deprovision a user from the system. Removes the user account, access key/secret and devices associated with the user.

Parameters

username	The username of the user account
password	The password for the user account.

Returns

The HTTP response code returned by Sidecar. Response code `204` indicates success.

7.15.4.11 static void SimpleSidecarClient::initAPIKey (const String & *apiKey,* const String & *apiSecret*) [static]

Initialise the API with the API key and secret used to sign provisioning requests. This is usually the first step in initialising the API for a device. Users must register their application with Sidecar and specify the generated API access key and secret.

Calling applications will generally store the API key and secret in flash storage using `PROGMEM` and use that to initialise the API on each run of the application.

7.15.4.12 static void SimpleSidecarClient::initEventAPI (const **EventAPIData** & *data*) [static]

Initialise the Sidecar Event API.

Parameters

data	The structure with initialisation data.

7.15.4.13 static void SimpleSidecarClient::initNetworkType (**NetworkType** *type*) [static]

Initialise the API to use the specified type.

7.15.4.14 static void SimpleSidecarClient::initUserKey (const String & *userKey,* const String & *userSecret*) [static]

Initialise the API with the user key and secret used to sign event requests. This step can be performed after provisioning a user account for a device. Use the provisioning API

Calling applications will generally store the user access key and secret in flash storage using `PROGMEM` and use that to initialise the API on each run of the application.

7.15.4.15 static void SimpleSidecarClient::initUUID (byte *mac[6]*) [static]

Initialise UUID engine. If using WiFi, the WiFi api provides a way to look up current MAC address. That would be better to get proper UUID values.

7.15.4.16 static void SimpleSidecarClient::initUUID ( ) [static]

Initialise UUID engine using a random seed.

7.15.4.17 bool SimpleSidecarClient::publish ( )

publish Publish the built up event to the Sidecar Event API. Invoke addReading with the individual readings that are part of the current event, and addTag as needed to build up a complete event before publishing to Sidecar.

Returns

Returns `true` if publish succeeded.

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

- SimpleSidecarClient.h

7.16 SimpleSidecarClient::UserResponse Struct Reference

A simple data structure that represents the response from Sidecar Provisioning API.

`#include <SimpleSidecarClient.h>`

Public Member Functions

- UserResponse (uint16_t response, const String &key, const String &sec)
  Create a new instance of the data structure.

Public Attributes

- const uint16_t responseCode
  The HTTP response code returned by the Sidecar Provisioning API.
- const String keyId
  The user access key value.
- const String secret
  The user access secret value.

7.16.1 Detailed Description

A simple data structure that represents the response from Sidecar Provisioning API.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 SimpleSidecarClient::UserResponse::UserResponse (uint16_t *response,* const String & *key,* const String & *sec*) [inline]

Create a new instance of the data structure.

7.16.3 Member Data Documentation

7.16.3.1 const String SimpleSidecarClient::UserResponse::keyId

The user access key value.

7.16.3.2 const uint16_t SimpleSidecarClient::UserResponse::responseCode

The HTTP response code returned by the Sidecar Provisioning API.

7.16.3.3 const String SimpleSidecarClient::UserResponse::secret

The user access secret value.

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

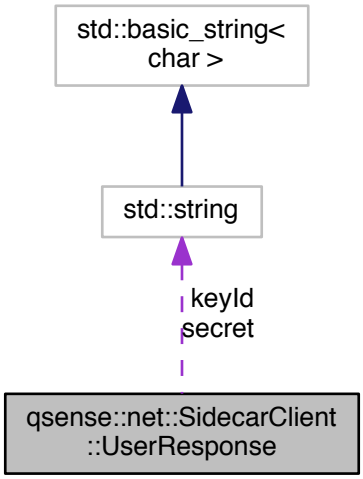
- SimpleSidecarClient.h

7.17 qsense::net::SidecarClient::UserResponse Struct Reference

A simple structure that represents the result of a user provisioning request.

`#include <SidecarClient.h>`

Collaboration diagram for qsense::net::SidecarClient::UserResponse:



Public Member Functions

- UserResponse (uint16_t response, const QString &key=QString(), const QString &sec=QString())

  Create a new UserResponse instance.

Static Public Member Functions

- static const UserResponse create (uint16_t response, const QString &body)

  Create a new UserResponse instance and populate the key and secret by parsing the response body.

Public Attributes

- uint16_t responseCode
- const QString keyId
- const QString secret

7.17.1 Detailed Description

A simple structure that represents the result of a user provisioning request.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 qsense::net::SidecarClient::UserResponse::UserResponse (uint16_t *response,* const QString & *key* = QString (), const QString & *sec* = QString ()) [inline]

Create a new UserResponse instance.

7.17.3 Member Function Documentation

7.17.3.1 static const UserResponse qsense::net::SidecarClient::UserResponse::create (uint16_t *response,* const QString & *body*) [static]

Create a new UserResponse instance and populate the key and secret by parsing the response body.

7.17.4 Member Data Documentation

7.17.4.1 const QString qsense::net::SidecarClient::UserResponse::keyId

7.17.4.2 uint16_t qsense::net::SidecarClient::UserResponse::responseCode

7.17.4.3 const QString qsense::net::SidecarClient::UserResponse::secret

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

- SidecarClient.h

7.18 qsense::UUID Class Reference

A class that represents a UUID/GUID.

`#include <UUID.h>`

Public Types

- enum Version { UUID_TIME_BASED = 0x01, UUID_DCE_UID = 0x02, UUID_NAME_BASED = 0x03, UUID_RANDOM = 0x04 }

Public Member Functions

- UUID ()

  Creates a nil (all zero) UUID.
- UUID (const UUID &uuid)

  Copy constructor.
- UUID (const QString &uuid)

  Parses the UUID from a string.
- UUID (const char *uuid)

  Parses the UUID from a char array.
- ~UUID ()

  Destroys the UUID.
- UUID & operator= (const UUID &uuid)

  Assignment operator.
- bool parse (const QString &uuid)

  Tries to interpret the given string as an UUID.
- QString toString () const

  Returns a string representation of the UUID consisting of groups of hexadecimal digits separated by hyphens.
- void copyFrom (const char *buffer)

  Copies the UUID (16 bytes) from a buffer or byte array. The UUID fields are expected to be stored in network byte order.
- void copyTo (char *buffer) const

  Copies the UUID to the buffer. The fields are in network byte order. The buffer need not be aligned.
- Version version () const

  Returns the version of the UUID.
- int variant () const

  Returns the variant number of the UUID:
- bool operator== (const UUID &uuid) const
- bool operator!= (const UUID &uuid) const
- bool operator< (const UUID &uuid) const
- bool operator<= (const UUID &uuid) const
- bool operator> (const UUID &uuid) const
- bool operator>= (const UUID &uuid) const
- bool isNull () const

Static Public Member Functions

- static const UUID & null ()

  Returns a null/nil UUID.
- static const UUID & dns ()

  Returns the namespace identifier for the DNS namespace.
- static const UUID & uri ()

  Returns the namespace identifier for the URI (former URL) namespace.
- static const UUID & oid ()

  Returns the namespace identifier for the OID namespace.
- static const UUID & x500 ()

  Returns the namespace identifier for the X500 namespace.
- static const UUID create ()

  Generate a time based UUID instance.
- static void init (uint8_t node[6])

  Initialise the UUID engine. On application start, invoke with the current MAC address.

Protected Member Functions

- UUID (uint32_t timeLow, uint32_t timeMid, uint32_t timeHiAndVersion, uint16_t clockSeq, uint8_t node[6])
- UUID (const char *bytes, Version version)
- int compare (const UUID &uuid) const
- void fromNetwork ()
- void toNetwork ()

Static Protected Member Functions

- static void appendHex (QString &str, uint8_t n)
- static void appendHex (QString &str, uint16_t n)
- static void appendHex (QString &str, uint32_t n)
- static uint8_t nibble (char hex)
- static uint32_t randomNumber (int32_t input)

7.18.1 Detailed Description

A class that represents a UUID/GUID.

A UUID is an identifier that is unique across both space and time, with respect to the space of all UUIDs. Since a UUID is a fixed size and contains a time field, it is possible for values to rollover (around A.D. 3400, depending on the specific algorithm used). A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network.

This class implements a Universal Unique Identifier, as specified in Appendix A of the DCE 1.1 Remote Procedure Call Specification (http://www.opengroup.org/onlinepubs/9629399/), RFC 2518 (WebDAV), section 6.4.1 and the UUIDs and GUIDs internet draft by Leach/Salz from February, 1998 (http://www.ics.uci.edu/~ejw/authoring/uuid-guid/draft-leach-uuids-guids-01.txt) and also http://tools.ietf.org/html/draft-mealling-uuid-urn-05

7.18.2 Member Enumeration Documentation

7.18.2.1 enum qsense::UUID::Version

Enumerator

UUID_TIME_BASED

UUID_DCE_UID

UUID_NAME_BASED

UUID_RANDOM

7.18.3 Constructor & Destructor Documentation

7.18.3.1 qsense::UUID::UUID ( )

Creates a nil (all zero) UUID.

7.18.3.2 qsense::UUID::UUID (const UUID & *uuid*)

Copy constructor.

7.18.3.3 qsense::UUID::UUID (const QString & *uuid*) [explicit]

Parses the UUID from a string.

7.18.3.4 qsense::UUID::UUID (const char * *uuid*) [explicit]

Parses the UUID from a char array.

7.18.3.5 qsense::UUID::~UUID ( )

Destroys the UUID.

7.18.3.6 qsense::UUID::UUID (uint32_t *timeLow,* uint32_t *timeMid,* uint32_t *timeHiAndVersion,* uint16_t *clockSeq,* uint8_t *node[6]*) [protected]

7.18.3.7 qsense::UUID::UUID (const char * *bytes,* Version *version*) [protected]

7.18.4 Member Function Documentation

7.18.4.1 static void qsense::UUID::appendHex (QString & *str,* uint8_t *n*) [static], [protected]

7.18.4.2 static void qsense::UUID::appendHex (QString & *str,* uint16_t *n*) [static], [protected]

7.18.4.3 static void qsense::UUID::appendHex (QString & *str,* uint32_t *n*) [static], [protected]

7.18.4.4 int qsense::UUID::compare (const UUID & *uuid*) const [protected]

7.18.4.5 void qsense::UUID::copyFrom (const char * *buffer*)

Copies the UUID (16 bytes) from a buffer or byte array. The UUID fields are expected to be stored in network byte order.

Parameters

buffer	The buffer need not be aligned.

7.18.4.6 void qsense::UUID::copyTo (char * *buffer*) const

Copies the UUID to the buffer. The fields are in network byte order. The buffer need not be aligned.

Parameters

buffer	There must be room for at least 16 bytes.

7.18.4.7 static const UUID qsense::UUID::create ( ) [static]

Generate a time based UUID instance.

7.18.4.8 static const UUID& qsense::UUID::dns ( ) [static]

Returns the namespace identifier for the DNS namespace.

7.18.4.9 void qsense::UUID::fromNetwork ( ) [protected]

7.18.4.10 static void qsense::UUID::init (uint8_t *node[6]*) [static]

Initialise the UUID engine. On application start, invoke with the current MAC address.

Parameters

node	The MAC address.

7.18.4.11 bool qsense::UUID::isNull ( ) const [inline]

Returns

Returns true if the UUID is nil (in other words, consists of all zeros).

7.18.4.12 static uint8_t qsense::UUID::nibble (char *hex*) [static], [protected]

7.18.4.13 static const UUID& qsense::UUID::null ( ) [static]

Returns a null/nil UUID.

7.18.4.14 static const UUID& qsense::UUID::oid ( ) [static]

Returns the namespace identifier for the OID namespace.

7.18.4.15 bool qsense::UUID::operator!= (const UUID & *uuid*) const [inline]

7.18.4.16 bool qsense::UUID::operator< (const UUID & *uuid*) const [inline]

7.18.4.17 bool qsense::UUID::operator<= (const UUID & *uuid*) const [inline]

7.18.4.18 UUID& qsense::UUID::operator= (const UUID & *uuid*)

Assignment operator.

7.18.4.19 bool qsense::UUID::operator== (const UUID & *uuid*) const [inline]

7.18.4.20 bool qsense::UUID::operator> (const UUID & *uuid*) const [inline]

7.18.4.21 bool qsense::UUID::operator>= (const UUID & *uuid*) const [inline]

7.18.4.22 bool qsense::UUID::parse (const QString & *uuid*)

Tries to interpret the given string as an UUID.

Parameters

uuid	The value to parse

Returns

If the UUID is syntactically valid, assigns the members and returns true. Otherwise leaves the object unchanged and returns false.

7.18.4.23 static uint32_t qsense::UUID::randomNumber (int32_t *input*) [static], [protected]

7.18.4.24 void qsense::UUID::toNetwork ( ) [protected]

7.18.4.25 QString qsense::UUID::toString ( ) const

Returns a string representation of the UUID consisting of groups of hexadecimal digits separated by hyphens.

7.18.4.26 static const UUID& qsense::UUID::uri ( ) [static]

Returns the namespace identifier for the URI (former URL) namespace.

7.18.4.27 int qsense::UUID::variant ( ) const

Returns the variant number of the UUID:

Returns

- 0 reserved for NCS backward compatibility
- 2 the Leach-Salz variant (used by this class)
- 6 reserved, Microsoft Corporation backward compatibility
- 7 reserved for future definition

7.18.4.28 UUID::Version qsense::UUID::version ( ) const [inline]

Returns the version of the UUID.

7.18.4.29 static const UUID& qsense::UUID::x500 ( ) [static]

Returns the namespace identifier for the X500 namespace.

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

- UUID.h

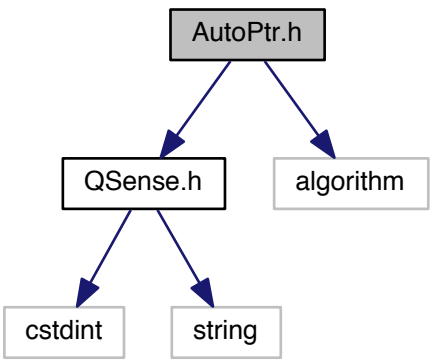
Chapter 8

File Documentation

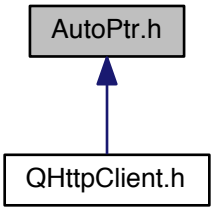
8.1 AutoPtr.h File Reference

```
#include <QSense.h>
#include <algorithm>
```

Include dependency graph for AutoPtr.h:



This graph shows which files directly or indirectly include this file:



Classes

- class qsense::AutoPtr< C >
  AutoPtr is a "smart" pointer for classes implementing reference counting based garbage collection.

Namespaces

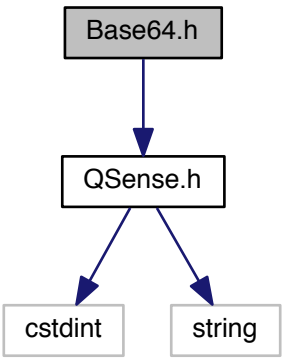
- qsense

Functions

- template<class C >
  void qsense::swap (AutoPtr< C > &p1, AutoPtr< C > &p2)

8.2 Base64.h File Reference

`#include <QSense.h>`
Include dependency graph for Base64.h:



Namespaces

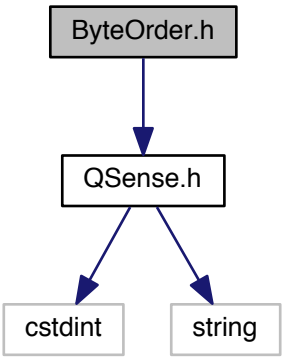
- qsense
- qsense::hash
- qsense::hash::base64

Functions

- int32_t qsense::hash::base64::encodeLength (int32_t len)
- int32_t qsense::hash::base64::encode (char ∗output, const char ∗input, int32_t inputLength)
- int32_t qsense::hash::base64::decodeLength (const char ∗code)
- int32_t qsense::hash::base64::decode (char ∗outputPlainText, const char ∗encoded)

8.3 ByteOrder.h File Reference

`#include <QSense.h>`
Include dependency graph for ByteOrder.h:



Classes

- class qsense::ByteOrder

  This class contains a number of static methods to convert between big-endian and little-endian integers of various sizes.

Namespaces

- qsense

Macros

- #define IMPLEMENT_BYTEORDER_NOOP_(op, type)
- #define IMPLEMENT_BYTEORDER_FLIP_(op, type)
- #define IMPLEMENT_BYTEORDER_NOOP(op)

- #define IMPLEMENT_BYTEORDER_FLIP(op)
- #define IMPLEMENT_BYTEORDER_BIG IMPLEMENT_BYTEORDER_FLIP
- #define IMPLEMENT_BYTEORDER_LIT IMPLEMENT_BYTEORDER_NOOP

8.3.1 Macro Definition Documentation

8.3.1.1 #define IMPLEMENT_BYTEORDER_BIG IMPLEMENT_BYTEORDER_FLIP

8.3.1.2 #define IMPLEMENT_BYTEORDER_FLIP(*op*)

Value:

```
IMPLEMENT_BYTEORDER_FLIP_(op, int16_t) \
    IMPLEMENT_BYTEORDER_FLIP_(op, uint16_t)    \
    IMPLEMENT_BYTEORDER_FLIP_(op, int32_t) \
    IMPLEMENT_BYTEORDER_FLIP_(op, uint32_t)    \
    IMPLEMENT_BYTEORDER_FLIP_(op, int64_t) \
    IMPLEMENT_BYTEORDER_FLIP_(op, uint64_t)
```

8.3.1.3 #define IMPLEMENT_BYTEORDER_FLIP_(*op*, *type*)

Value:

```
inline type ByteOrder::op(type value)       \
    {                                           \
        return flipBytes(value);                \
    }
```

8.3.1.4 #define IMPLEMENT_BYTEORDER_LIT IMPLEMENT_BYTEORDER_NOOP

8.3.1.5 #define IMPLEMENT_BYTEORDER_NOOP(*op*)

Value:

```
IMPLEMENT_BYTEORDER_NOOP_(op, int16_t) \
    IMPLEMENT_BYTEORDER_NOOP_(op, uint16_t)    \
    IMPLEMENT_BYTEORDER_NOOP_(op, int32_t) \
    IMPLEMENT_BYTEORDER_NOOP_(op, uint32_t)    \
    IMPLEMENT_BYTEORDER_NOOP_(op, int64_t) \
    IMPLEMENT_BYTEORDER_NOOP_(op, uint64_t)
```

8.3.1.6 #define IMPLEMENT_BYTEORDER_NOOP_(*op*, *type*)

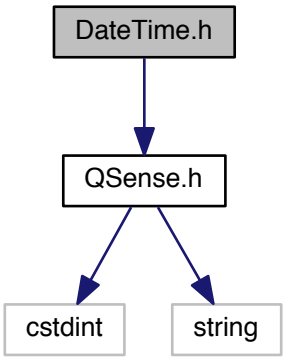
Value:

```
inline type ByteOrder::op(type value)       \
    {                                           \
        return value;                           \
    }
```

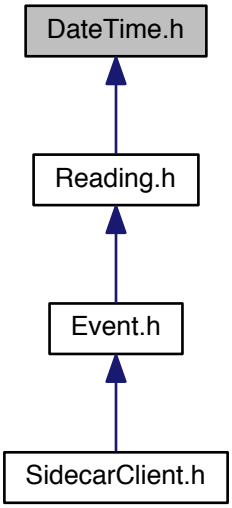
8.4 DateTime.h File Reference

```
#include <QSense.h>
```

Include dependency graph for DateTime.h:



This graph shows which files directly or indirectly include this file:



Classes

- class qsense::net::DateTime

  Represents current date/time. Seeds initially (and daily) from a network time service, and uses internal timer to represent a real-time clock.

Namespaces

- qsense

- qsense::net

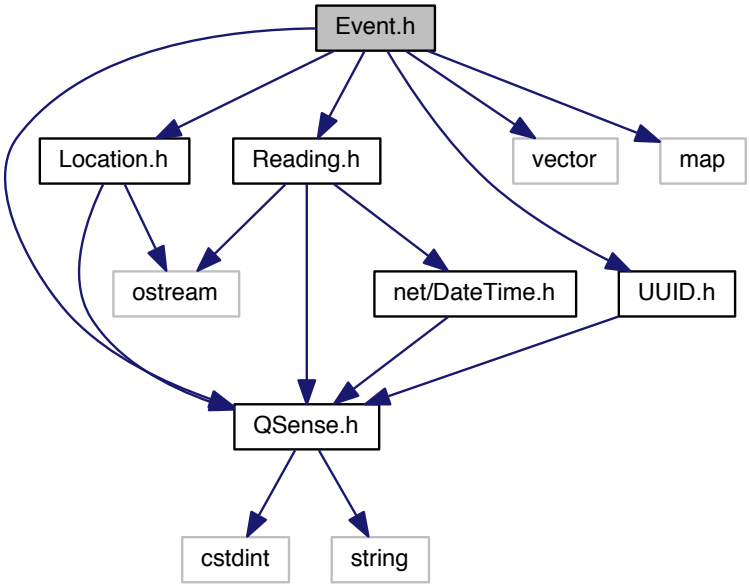
Functions

- uint32_t qsense::net::millis ()

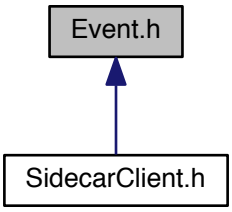
8.5 Event.h File Reference

```
#include <QSense.h>
#include <Location.h>
#include <Reading.h>
#include <UUID.h>
#include <vector>
#include <map>
```

Include dependency graph for Event.h:



This graph shows which files directly or indirectly include this file:



Classes

- class qsense::Event

  A simple class that encapsulates an event sent to Sidecar. Events are holders for readings. Events can be serialised to JSON using the toString method.

Namespaces

- qsense

Functions

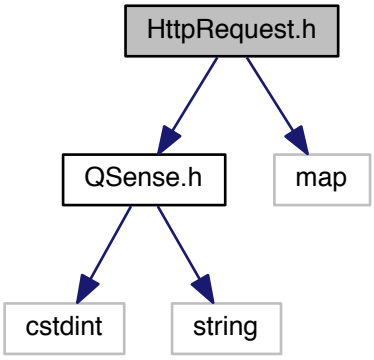
- std::ostream & qsense::operator<< (std::ostream &os, const Event &event)

  Serialise the specified event as JSON to the output stream.

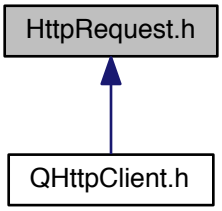
8.6 HttpRequest.h File Reference

```
#include <QSense.h>
#include <map>
```

Include dependency graph for HttpRequest.h:



This graph shows which files directly or indirectly include this file:



Classes

- class qsense::net::HttpRequest

  A simple class that represents a HTTP request. Request encapsulates the URI path, any request parameters, header attributes, body etc. as appropriate.

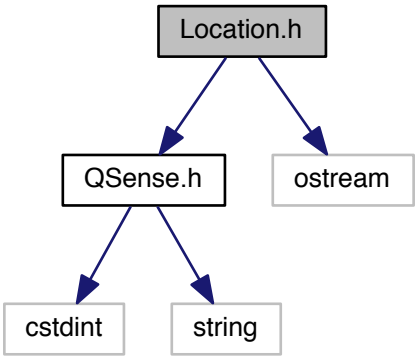
Namespaces

- qsense
- qsense::net

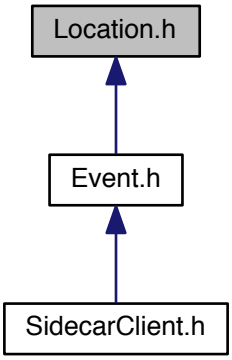
8.7 Location.h File Reference

```
#include <QSense.h>
#include <ostream>
```

Include dependency graph for Location.h:



This graph shows which files directly or indirectly include this file:



Classes

- class qsense::Location

  A simple representation of geographical location.

Namespaces

- qsense

Functions

- std::ostream & qsense::operator<< (std::ostream &os, const qsense::Location &location)

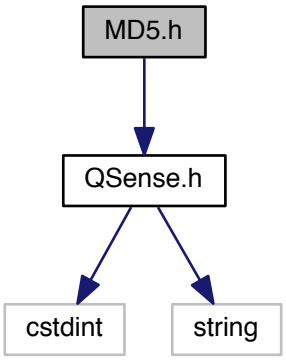
Serialise the specified location to the output stream.

8.8 mainpage.dox File Reference

8.9 MD5.h File Reference

```
#include <QSense.h>
```

Include dependency graph for MD5.h:



Classes

- class qsense::hash::MD5

  Class for generating MD5 hashes.

Namespaces

- qsense
- qsense::hash

Macros

- #define MD5_HASH_LENGTH 16

8.9.1 Macro Definition Documentation

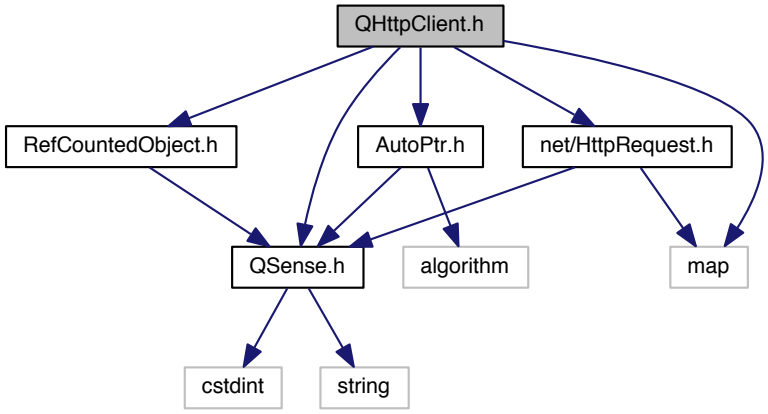
8.9.1.1 #define MD5_HASH_LENGTH 16

8.10 QHttpClient.h File Reference

```
#include "QSense.h"
```

```
#include <AutoPtr.h>
#include <RefCountedObject.h>
#include <net/HttpRequest.h>
#include <map>
```

Include dependency graph for QHttpClient.h:



Classes

- class qsense::net::HttpClient

  A HTTP Client class for use with either ethernet or wifi. Before use, the client should be initialised with the network type used by the device (qsense::net::initNetworkType.

Namespaces

- qsense
- qsense::net

Enumerations

- enum qsense::net::NetworkType { qsense::net::Ethernet = 0, qsense::net::WiFi = 1 }

  Enumeration of network connection types for device.

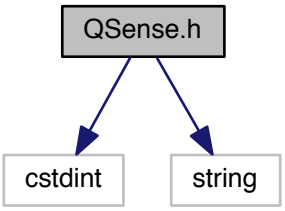
Functions

- void qsense::net::initNetworkType (NetworkType type)

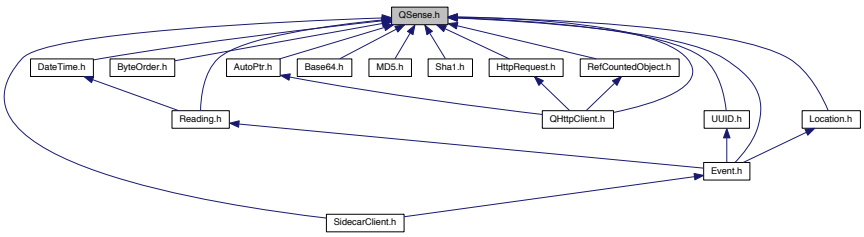
8.11 QSense.h File Reference

```
#include <cstdint>
#include <string>
```

Include dependency graph for QSense.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- qsense

Macros

- #define F(x) x
- #define DEBUG 0

Typedefs

- typedef std::string qsense::QString
- typedef unsigned char qsense::Byte

8.11.1 Macro Definition Documentation

8.11.1.1 #define DEBUG 0

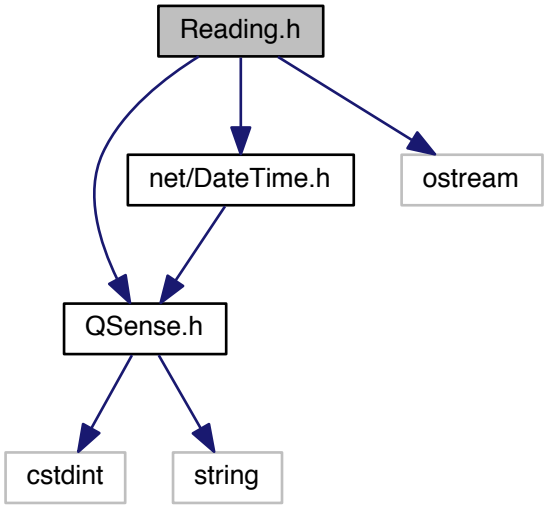
8.11.1.2 #define F(*x*) x

8.12 Reading.h File Reference

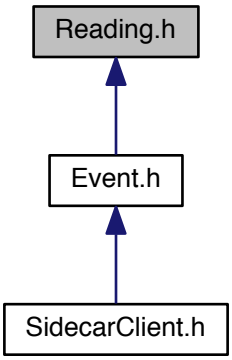
```
#include <QSense.h>
```

```
#include <net/DateTime.h>
#include <ostream>
```

Include dependency graph for Reading.h:



This graph shows which files directly or indirectly include this file:



Classes

- class qsense::Reading

  A class that represents a single reading. Readings are added to an Event.

Namespaces

- qsense

Functions

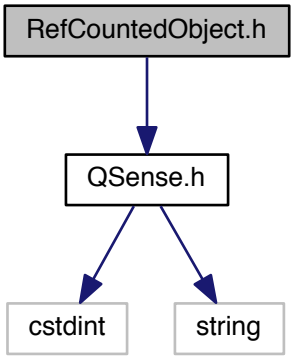
- std::ostream & qsense::operator<< (std::ostream &os, const qsense::Reading &reading)

  Serialise the reading as JSON to the output stream.

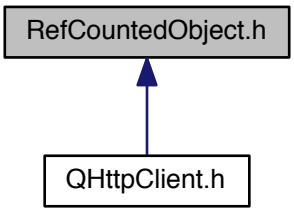
8.13 RefCountedObject.h File Reference

```
#include <QSense.h>
```

Include dependency graph for RefCountedObject.h:



This graph shows which files directly or indirectly include this file:



Classes

- class qsense::RefCountedObject

  A base class for objects that employ reference counting based garbage collection.

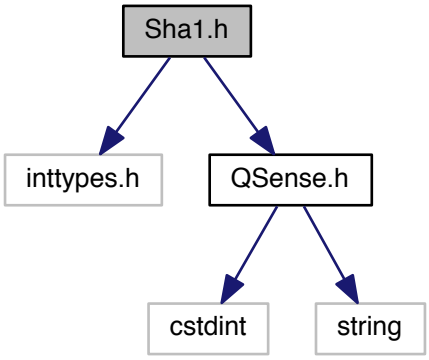
Namespaces

- qsense

8.14 Sha1.h File Reference

```
#include <inttypes.h>
#include <QSense.h>
```

Include dependency graph for Sha1.h:



Classes

- class qsense::hash::Sha1

  Class for hashing using SHA1 algorithm.

- struct qsense::hash::Sha1::Context

  SHA1 context representation.

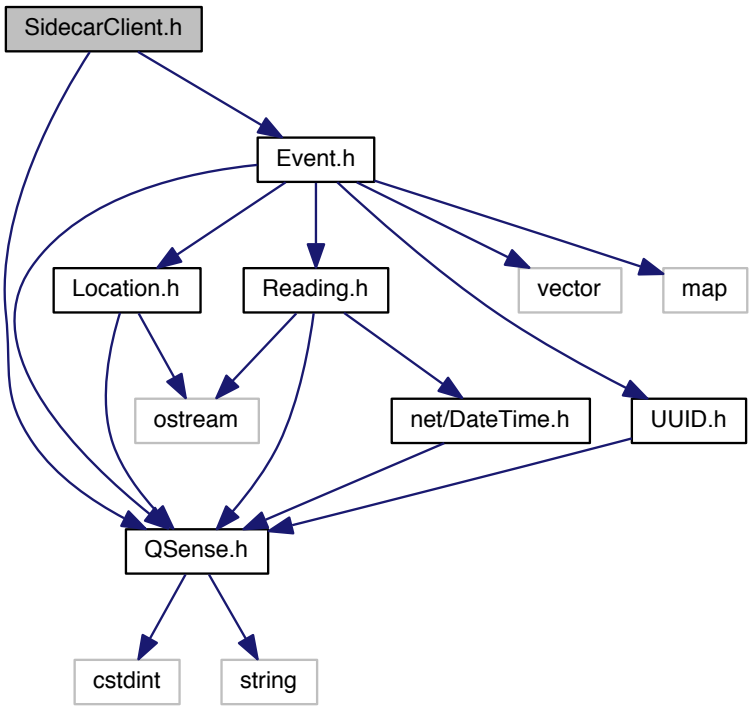
Namespaces

- qsense
- qsense::hash

8.15 SidecarClient.h File Reference

```
#include <QSense.h>
#include <Event.h>
```

Include dependency graph for SidecarClient.h:



Classes

- class qsense::net::SidecarClient

  Class that encapsulates interactions with the Sidecar REST API.

- struct qsense::net::SidecarClient::UserResponse

  A simple structure that represents the result of a user provisioning request.

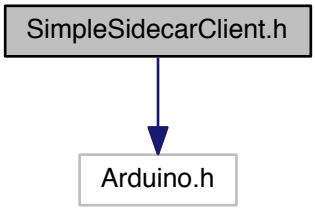
Namespaces

- qsense
- qsense::net

8.16 SimpleSidecarClient.h File Reference

```
#include <Arduino.h>
```

Include dependency graph for SimpleSidecarClient.h:



Classes

- class SimpleSidecarClient

  A simple client implementation that hides the low-level API.

- struct SimpleSidecarClient::UserResponse

  A simple data structure that represents the response from Sidecar Provisioning API.

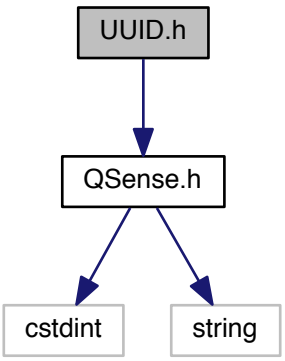
- struct SimpleSidecarClient::EventAPIData

  A simple data structure that encapsulates the data required to initialise the Event API.

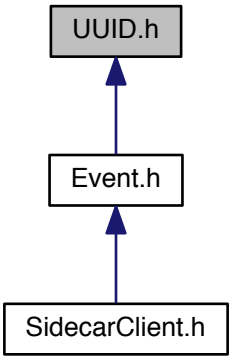
8.17 UUID.h File Reference

```
#include <QSense.h>
```

Include dependency graph for UUID.h:



This graph shows which files directly or indirectly include this file:



Classes

- class qsense::UUID

  A class that represents a UUID/GUID.

Namespaces

- qsense

Functions

- std::ostream & qsense::operator<< (std::ostream &os, const UUID &uuid)

  Serialise the string representation of the UUID to the output stream.

Index