lslpub_OTB

Generated by Doxygen 1.8.13

Contents

Chapter 1

OTBconfigGUI

Chapter 2

lslpub_OTB

C++ programs that gets the data from the OTBiolab quattrocento device and publishes them in a LSL stream.

1 Architecture

1.1 INPUTS:

- from OTBiolab quattrocento ethernet packets.

1.2 OUTPUTS:

- LSL stream

2 Installation

2.1 Ubuntu 18

2.1.1 Requirements

None.

2.1.2 Steps

- Clone the repository and go inside.

```
git clone https://github.com/Aightech/lslpub_OTB.git && cd lslpub_OTB
```

- Create a build directory and go inside.
- Configure the project.
- Build the project.

```
mkdir build && cd build && cmake .. && cmake --build .
```

create wired connection

Sttings>Network> add a new wired connection IPv4> Manual: address: 169.254.1.0 | netmask: 255.255.255.0 | gateway: 169.254.1.0

Rq: *The exe file is called lslpub_OTB. This file has also been copied in the bin floder of the git repository root.*

2.2 Windows 10

2.2.1 Requirements

- CMake (download and execute the installer for windows , add to the PATH variable)
- MinGW32 (download and execute the installer for windows, chose i686_64 architecture, add the the bin folder address of minGW to the PATH environement variable).
- **Git** Download and install git.

Steps

- Clone the repository and go inside.

```
git clone https://github.com/Aightech/lslpub_OTB.git && cd lslpub_OTB
```

- Create a build directory.
- Configure and generate the CMake project.
- Build the project.

```
mkdir build && cd build && cmake .. -G "MinGW Makefiles" && mingw32-make
```

Rq: *The exe file is called lslpub_OTB. This file has also been copied in the bin floder of the git repository root.*

2.2.3 Build LSL library on windows

You can also follow this guide https://github.com/sccn/labstreaminglayer/blob/master/doc/↩BUILD.md.

2.2.3.1 Requirements

- **Cmake** Download and install cmake.
- **Qt** Download qt installer (open source version).
- **Boost libraries** Download boost lib (last binaries version).
- **Git** Download and install git.
- **MinGW32** (download and execute the installer for windows, chose i686_64 architecture, add the the bin folder address of minGW to the PATH environement variable)

2.2.3.2 Steps

- Clone the repository and go inside.

```
git clone --recurse-submodules https://github.com/labstreaminglayer/labstreaminglayer.git &&cd
       labstreaminglayer
```

- Create a build repository and go inside.

```
mkdir build && cd build
```

Configure and generate the project with the GUI.

```
cmake-gui
```

Click on "configure". Select the MinGW MakeFile. Check if the boost libraries and Qt are correct. Then click on "generate". In the command prompt, build the project.

```
cmake --build . --config Release --target install
```

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 lsl Namespace Reference

Classes

- class continuous_resolver
- class lost_error

  Exception class that indicates that a stream inlet's source has been irrecoverably lost.
- class stream_info
- class stream_inlet
- class stream_outlet
- class timeout_error

  Exception class that indicates that an operation failed due to a timeout.
- class xml_element

Typedefs

- typedef struct lsl_streaminfo_struct_ ∗ lsl_streaminfo
- typedef struct lsl_outlet_struct_ ∗ lsl_outlet
- typedef struct lsl_inlet_struct_ ∗ lsl_inlet
- typedef struct lsl_xml_ptr_struct_ ∗ lsl_xml_ptr
- typedef struct lsl_continuous_resolver_ ∗ lsl_continuous_resolver

Enumerations

- enum lsl_channel_format_t {
  cft_float32 = 1, cft_double64 = 2, cft_string = 3, cft_int32 = 4,
  cft_int16 = 5, cft_int8 = 6, cft_int64 = 7, cft_undefined = 0 }
- enum lsl_processing_options_t {
  proc_none = 0, proc_clocksync = 1, proc_dejitter = 2, proc_monotonize = 4,
  proc_threadsafe = 8, proc_ALL = 1|2|4|8 }
- enum lsl_error_code_t {
  lsl_no_error = 0, lsl_timeout_error = -1, lsl_lost_error = -2, lsl_argument_error = -3,
  lsl_internal_error = -4 }
- enum channel_format_t {
  cf_float32 = 1, cf_double64 = 2, cf_string = 3, cf_int32 = 4,
  cf_int16 = 5, cf_int8 = 6, cf_int64 = 7, cf_undefined = 0 }
- enum processing_options_t {
  post_none = 0, post_clocksync = 1, post_dejitter = 2, post_monotonize = 4,
  post_threadsafe = 8, post_ALL = 1|2|4|8 }

Functions

- LIBLSL_C_API int32_t lsl_protocol_version ()
- LIBLSL_C_API int32_t lsl_library_version ()
- LIBLSL_C_API const char ∗ lsl_library_info ()
- LIBLSL_C_API double lsl_local_clock ()
- LIBLSL_C_API int32_t lsl_resolve_all (lsl_streaminfo ∗buffer, uint32_t buffer_elements, double wait_time)
- LIBLSL_C_API int32_t lsl_resolve_byprop (lsl_streaminfo ∗buffer, uint32_t buffer_elements, const char ∗prop, const char ∗value, int32_t minimum, double timeout)
- LIBLSL_C_API int32_t lsl_resolve_bypred (lsl_streaminfo ∗buffer, uint32_t buffer_elements, const char ∗pred, int32_t minimum, double timeout)
- LIBLSL_C_API void lsl_destroy_string (char ∗s)
- LIBLSL_C_API lsl_streaminfo lsl_create_streaminfo (const char ∗name, const char ∗type, int32_t channel↩ _count, double nominal_srate, lsl_channel_format_t channel_format, const char ∗source_id)
- LIBLSL_C_API void lsl_destroy_streaminfo (lsl_streaminfo info)
- LIBLSL_C_API lsl_streaminfo lsl_copy_streaminfo (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_name (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_type (lsl_streaminfo info)
- LIBLSL_C_API int32_t lsl_get_channel_count (lsl_streaminfo info)
- LIBLSL_C_API double lsl_get_nominal_srate (lsl_streaminfo info)
- LIBLSL_C_API lsl_channel_format_t lsl_get_channel_format (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_source_id (lsl_streaminfo info)
- LIBLSL_C_API int32_t lsl_get_version (lsl_streaminfo info)
- LIBLSL_C_API double lsl_get_created_at (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_uid (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_session_id (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_hostname (lsl_streaminfo info)
- LIBLSL_C_API lsl_xml_ptr lsl_get_desc (lsl_streaminfo info)
- LIBLSL_C_API char ∗ lsl_get_xml (lsl_streaminfo info)
- LIBLSL_C_API int32_t lsl_get_channel_bytes (lsl_streaminfo info)

  Number of bytes occupied by a channel (0 for string-typed channels).
- LIBLSL_C_API int32_t lsl_get_sample_bytes (lsl_streaminfo info)

  Number of bytes occupied by a sample (0 for string-typed channels).
- LIBLSL_C_API int lsl_stream_info_matches_query (lsl_streaminfo info, const char ∗query)
- LIBLSL_C_API lsl_streaminfo lsl_streaminfo_from_xml (const char ∗xml)

  Create a streaminfo object from an XML representation.
- LIBLSL_C_API lsl_outlet lsl_create_outlet (lsl_streaminfo info, int32_t chunk_size, int32_t max_buffered)
- LIBLSL_C_API void lsl_destroy_outlet (lsl_outlet out)
- LIBLSL_C_API int32_t lsl_push_sample_f (lsl_outlet out, const float ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_ft (lsl_outlet out, const float ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_ftp (lsl_outlet out, const float ∗data, double timestamp, int32_↩ t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_d (lsl_outlet out, const double ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_dt (lsl_outlet out, const double ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_dtp (lsl_outlet out, const double ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_l (lsl_outlet out, const long ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_lt (lsl_outlet out, const long ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_ltp (lsl_outlet out, const long ∗data, double timestamp, int32_↩ t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_i (lsl_outlet out, const int32_t ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_it (lsl_outlet out, const int32_t ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_itp (lsl_outlet out, const int32_t ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_s (lsl_outlet out, const int16_t ∗data)

- LIBLSL_C_API int32_t lsl_push_sample_st (lsl_outlet out, const int16_t ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_stp (lsl_outlet out, const int16_t ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_c (lsl_outlet out, const char ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_ct (lsl_outlet out, const char ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_ctp (lsl_outlet out, const char ∗data, double timestamp, int32_↩ t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_str (lsl_outlet out, const char ∗∗data)
- LIBLSL_C_API int32_t lsl_push_sample_strt (lsl_outlet out, const char ∗∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_strtp (lsl_outlet out, const char ∗∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_buf (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths)
- LIBLSL_C_API int32_t lsl_push_sample_buft (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_buftp (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_v (lsl_outlet out, const void ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_vt (lsl_outlet out, const void ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_vtp (lsl_outlet out, const void ∗data, double timestamp, int32_↩ t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_f (lsl_outlet out, const float ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_ft (lsl_outlet out, const float ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_ftp (lsl_outlet out, const float ∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_ftn (lsl_outlet out, const float ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_ftnp (lsl_outlet out, const float ∗data, unsigned long data_elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_d (lsl_outlet out, const double ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_dt (lsl_outlet out, const double ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_dtp (lsl_outlet out, const double ∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_dtn (lsl_outlet out, const double ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_dtnp (lsl_outlet out, const double ∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int lsl_push_chunk_l (lsl_outlet out, const long ∗data, unsigned long data_elements)
- LIBLSL_C_API int lsl_push_chunk_lt (lsl_outlet out, const long ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int lsl_push_chunk_ltp (lsl_outlet out, const long ∗data, unsigned long data_elements, double timestamp, int pushthrough)
- LIBLSL_C_API int lsl_push_chunk_ltn (lsl_outlet out, const long ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int lsl_push_chunk_ltnp (lsl_outlet out, const long ∗data, unsigned long data_elements, const double ∗timestamps, int pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_i (lsl_outlet out, const int32_t ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_it (lsl_outlet out, const int32_t ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_itp (lsl_outlet out, const int32_t ∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_itn (lsl_outlet out, const int32_t ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_itnp (lsl_outlet out, const int32_t ∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)

- LIBLSL_C_API int32_t lsl_push_chunk_s (lsl_outlet out, const int16_t ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_st (lsl_outlet out, const int16_t ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_stp (lsl_outlet out, const int16_t ∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_stn (lsl_outlet out, const int16_t ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_stnp (lsl_outlet out, const int16_t ∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_c (lsl_outlet out, const char ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_ct (lsl_outlet out, const char ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_ctp (lsl_outlet out, const char ∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_ctn (lsl_outlet out, const char ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_ctnp (lsl_outlet out, const char ∗data, unsigned long data_elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_str (lsl_outlet out, const char ∗∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_strt (lsl_outlet out, const char ∗∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_strtp (lsl_outlet out, const char ∗∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_strtn (lsl_outlet out, const char ∗∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_strtnp (lsl_outlet out, const char ∗∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_buf (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_buft (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_buftp (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_buftn (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_buftnp (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_have_consumers (lsl_outlet out)
- LIBLSL_C_API int32_t lsl_wait_for_consumers (lsl_outlet out, double timeout)
- LIBLSL_C_API lsl_streaminfo lsl_get_info (lsl_outlet out)
- LIBLSL_C_API lsl_inlet lsl_create_inlet (lsl_streaminfo info, int32_t max_buflen, int32_t max_chunklen, int32_t recover)
- LIBLSL_C_API void lsl_destroy_inlet (lsl_inlet in)
- LIBLSL_C_API lsl_streaminfo lsl_get_fullinfo (lsl_inlet in, double timeout, int32_t ∗ec)
- LIBLSL_C_API void lsl_open_stream (lsl_inlet in, double timeout, int32_t ∗ec)
- LIBLSL_C_API void lsl_close_stream (lsl_inlet in)
- LIBLSL_C_API double lsl_time_correction (lsl_inlet in, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_time_correction_ex (lsl_inlet in, double ∗remote_time, double ∗uncertainty, double timeout, int32_t ∗ec)
- LIBLSL_C_API int32_t lsl_set_postprocessing (lsl_inlet in, uint32_t flags)
- LIBLSL_C_API double lsl_pull_sample_f (lsl_inlet in, float ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_d (lsl_inlet in, double ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_l (lsl_inlet in, long ∗buffer, int buffer_elements, double timeout, int ∗ec)

- LIBLSL_C_API double lsl_pull_sample_i (lsl_inlet in, int32_t ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_s (lsl_inlet in, int16_t ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_c (lsl_inlet in, char ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_str (lsl_inlet in, char ∗∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_buf (lsl_inlet in, char ∗∗buffer, uint32_t ∗buffer_lengths, int32_↩ t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_v (lsl_inlet in, void ∗buffer, int32_t buffer_bytes, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_f (lsl_inlet in, float ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_d (lsl_inlet in, double ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_l (lsl_inlet in, long ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_i (lsl_inlet in, int32_t ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_s (lsl_inlet in, int16_t ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_c (lsl_inlet in, char ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_↩ t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_str (lsl_inlet in, char ∗∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_buf (lsl_inlet in, char ∗∗data_buffer, uint32_t ∗lengths_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API uint32_t lsl_samples_available (lsl_inlet in)
- LIBLSL_C_API uint32_t lsl_was_clock_reset (lsl_inlet in)
- LIBLSL_C_API int32_t lsl_smoothing_halftime (lsl_inlet in, float value)
- LIBLSL_C_API lsl_xml_ptr lsl_first_child (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl_last_child (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl_next_sibling (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl_previous_sibling (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl_parent (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl_child (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl_next_sibling_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl_previous_sibling_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API int32_t lsl_empty (lsl_xml_ptr e)
- LIBLSL_C_API int32_t lsl_is_text (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl_name (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl_value (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl_child_value (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl_child_value_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl_append_child_value (lsl_xml_ptr e, const char ∗name, const char ∗value)
- LIBLSL_C_API lsl_xml_ptr lsl_prepend_child_value (lsl_xml_ptr e, const char ∗name, const char ∗value)
- LIBLSL_C_API int32_t lsl_set_child_value (lsl_xml_ptr e, const char ∗name, const char ∗value)
- LIBLSL_C_API int32_t lsl_set_name (lsl_xml_ptr e, const char ∗rhs)
- LIBLSL_C_API int32_t lsl_set_value (lsl_xml_ptr e, const char ∗rhs)
- LIBLSL_C_API lsl_xml_ptr lsl_append_child (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl_prepend_child (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl_append_copy (lsl_xml_ptr e, lsl_xml_ptr e2)
- LIBLSL_C_API lsl_xml_ptr lsl_prepend_copy (lsl_xml_ptr e, lsl_xml_ptr e2)

- LIBLSL_C_API void lsl_remove_child_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API void lsl_remove_child (lsl_xml_ptr e, lsl_xml_ptr e2)
- LIBLSL_C_API lsl_continuous_resolver lsl_create_continuous_resolver (double forget_after)
- LIBLSL_C_API lsl_continuous_resolver lsl_create_continuous_resolver_byprop (const char ∗prop, const char ∗value, double forget_after)
- LIBLSL_C_API lsl_continuous_resolver lsl_create_continuous_resolver_bypred (const char ∗pred, double forget_after)
- LIBLSL_C_API int32_t lsl_resolver_results (lsl_continuous_resolver res, lsl_streaminfo ∗buffer, uint32_↩ t buffer_elements)
- LIBLSL_C_API void lsl_destroy_continuous_resolver (lsl_continuous_resolver res)
- int32_t protocol_version ()
- int32_t library_version ()
- const char ∗ library_info ()
- double local_clock ()
- std::vector< stream_info > resolve_streams (double wait_time=1.0)
- std::vector< stream_info > resolve_stream (const std::string &prop, const std::string &value, int32_t minimum=1, double timeout=FOREVER)
- std::vector< stream_info > resolve_stream (const std::string &pred, int32_t minimum=1, double timeout=F↩ OREVER)
- void check_error (int32_t ec)

Variables

- const double IRREGULAR_RATE = 0.0
- const double DEDUCED_TIMESTAMP = -1.0
- const double FOREVER = 32000000.0

7.1.1 Detailed Description

C++ API for the lab streaming layer.

The lab streaming layer provides a set of functions to make instrument data accessible in real time within a lab network. From there, streams can be picked up by recording programs, viewing programs or custom experiment applications that access data streams in real time.

The API covers two areas:

- The "push API" allows to create stream outlets and to push data (regular or irregular measurement time series, event data, coded audio/video frames, etc.) into them.
- The "pull API" allows to create stream inlets and read time-synched experiment data from them (for recording, viewing or experiment control).

To use this library you need to link to either the liblsl32 or liblsl64 shared library that comes with this header. Under Visual Studio the library is linked in automatically.

7.1.2 Typedef Documentation

7.1.2.1 lsl_continuous_resolver

`typedef struct lsl_continuous_resolver_* lsl::lsl_continuous_resolver`

Handle to a convenience object that resolves streams continuously in the background throughout its lifetime and which can be queried at any time for the set of streams that are currently visible on the network.

7.1.2.2 lsl_inlet

`typedef struct lsl_inlet_struct_* lsl::lsl_inlet`

A stream inlet handle. Inlets are used to receive streaming data (and meta-data) from the lab network.

7.1.2.3 lsl_outlet

`typedef struct lsl_outlet_struct_* lsl::lsl_outlet`

A stream outlet handle. Outlets are used to make streaming data (and the meta-data) available on the lab network.

7.1.2.4 lsl_streaminfo

`typedef struct lsl_streaminfo_struct_* lsl::lsl_streaminfo`

Handle to a stream info object. Stores the declaration of a data stream. Represents the following information: a) stream data format (#channels, channel format) b) core information (stream name, content type, sampling rate) c) optional meta-data about the stream content (channel labels, measurement units, etc.)

Whenever a program wants to provide a new stream on the lab network it will typically first create an lsl_streaminfo to describe its properties and then construct an lsl_outlet with it to create the stream on the network. Other parties who discover/resolve the outlet on the network can query the stream info; it is also written to disk when recording the stream (playing a similar role as a file header).

7.1.2.5 lsl_xml_ptr

`typedef struct lsl_xml_ptr_struct_* lsl::lsl_xml_ptr`

A lightweight XML element tree handle; models the description of a streaminfo object. XML elements behave like advanced pointers into memory that is owned by some respective streaminfo. Has a name and can have multiple named children or have text content as value; attributes are omitted. Insider note: The interface is modeled after a subset of pugixml's node type and is compatible with it. Type-casts between pugi::xml_node_struct* and lsl_↩
xml_ptr are permitted (in both directions) since the types are binary compatible. See also pugixml.googlecode.↩
com/svn/tags/latest/docs/manual/access.html for additional documentation.

7.1.3 Enumeration Type Documentation

7.1.3.1 channel_format_t

`enum lsl::channel_format_t`

Data format of a channel (each transmitted sample holds an array of channels).

Enumerator

cf_float32	
cf_double64	
cf_string	
cf_int32	
cf_int16	
cf_int8	
cf_int64	
cf_undefined	

7.1.3.2 lsl_channel_format_t

```
enum lsl::lsl_channel_format_t
```

Data format of a channel (each transmitted sample holds an array of channels).

Enumerator

cft_float32	
cft_double64	
cft_string	
cft_int32	
cft_int16	
cft_int8	
cft_int64	
cft_undefined	

7.1.3.3 lsl_error_code_t

```
enum lsl::lsl_error_code_t
```

Possible error codes.

Enumerator

lsl_no_error	
lsl_timeout_error	
lsl_lost_error	
lsl_argument_error	
lsl_internal_error	

7.1.3.4 lsl_processing_options_t

```
enum lsl::lsl_processing_options_t
```

Post-processing options for stream inlets.

Enumerator

proc_none	
proc_clocksync	
proc_dejitter	
proc_monotonize	
proc_threadsafe	
proc_ALL	

7.1.3.5 processing_options_t

```
enum lsl::processing_options_t
```

Post-processing options for stream inlets.

Enumerator

post_none	
post_clocksync	
post_dejitter	
post_monotonize	
post_threadsafe	
post_ALL	

7.1.4 Function Documentation

7.1.4.1 check_error()

```
void lsl::check_error (
             int32_t ec )  [inline]
```

A stream inlet. Inlets are used to receive streaming data (and meta-data) from the lab network.

Check error codes returned from the C interface and translate into appropriate exceptions.

7.1.4.2 library_info()

```
const char* lsl::library_info ( )  [inline]
```

Get a string containing library information. The format of the string shouldn't be used for anything important except giving a a debugging person a good idea which exact library version is used.

7.1.4.3 library_version()

```
int32_t lsl::library_version ( )  [inline]
```

Version of the liblsl library. The major version is library_version() / 100; The minor version is library_version() % 100;

7.1.4.4 local_clock()

```
double lsl::local_clock ( )  [inline]
```

Obtain a local system time stamp in seconds. The resolution is better than a millisecond. This reading can be used to assign time stamps to samples as they are being acquired. If the "age" of a sample is known at a particular time (e.g., from USB transmission delays), it can be used as an offset to local_clock() to obtain a better estimate of when a sample was actually captured. See stream_outlet::push_sample() for a use case.

7.1.4.5 lsl_append_child()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_append_child (
             lsl_xml_ptr e,
             const char * name )
```

Append a child element with the specified name.

7.1.4.6 lsl_append_child_value()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_append_child_value (
             lsl_xml_ptr e,
             const char * name,
             const char * value )
```

Append a child node with a given name, which has a (nameless) plain-text child with the given text value.

7.1.4.7 lsl_append_copy()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_append_copy (
             lsl_xml_ptr e,
             lsl_xml_ptr e2 )
```

Append a copy of the specified element as a child.

7.1.4.8 lsl_child()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_child (
             lsl_xml_ptr e,
             const char * name )
```

Get a child with a specified name.

7.1.4.9 lsl_child_value()

```
LIBLSL_C_API const char* lsl::lsl_child_value (
             lsl_xml_ptr e )
```

Get child value (value of the first child that is text).

7.1.4.10 lsl_child_value_n()

```
LIBLSL_C_API const char* lsl::lsl_child_value_n (
             lsl_xml_ptr e,
             const char * name )
```

Get child value of a child with a specified name.

7.1.4.11 lsl_close_stream()

```
LIBLSL_C_API void lsl::lsl_close_stream (
             lsl_inlet in )
```

Drop the current data stream. All samples that are still buffered or in flight will be dropped and transmission and buffering of data for this inlet will be stopped. If an application stops being interested in data from a source (temporarily or not) but keeps the outlet alive, it should call lsl_close_stream() to not waste unnecessary system and network resources.

7.1.4.12 lsl_copy_streaminfo()

```
LIBLSL_C_API lsl_streaminfo lsl::lsl_copy_streaminfo (
             lsl_streaminfo info )
```

Copy an existing streaminfo object (rarely used).

7.1.4.13 lsl_create_continuous_resolver()

```
LIBLSL_C_API lsl_continuous_resolver lsl::lsl_create_continuous_resolver (
             double forget_after )
```

Construct a new continuous_resolver that resolves all streams on the network. This is analogous to the functionality offered by the free function resolve_streams().

Parameters

forget_after	When a stream is no longer visible on the network (e.g., because it was shut down), this is the time in seconds after which it is no longer reported by the resolver. The recommended default value is 5.0.

7.1.4.14 lsl_create_continuous_resolver_bypred()

```
LIBLSL_C_API lsl_continuous_resolver lsl::lsl_create_continuous_resolver_bypred (
            const char * pred,
            double forget_after )
```

Construct a new continuous_resolver that resolves all streams that match a given XPath 1.0 predicate. This is analogous to the functionality provided by the free function resolve_stream(pred).

Parameters

pred	The predicate string, e.g. "name='BioSemi'" or "type='EEG' and starts-with(name,'BioSemi') and count(info/desc/channel)=32"
forget_after	When a stream is no longer visible on the network (e.g., because it was shut down), this is the time in seconds after which it is no longer reported by the resolver. The recommended default value is 5.0.

7.1.4.15 lsl_create_continuous_resolver_byprop()

```
LIBLSL_C_API lsl_continuous_resolver lsl::lsl_create_continuous_resolver_byprop (
            const char * prop,
            const char * value,
            double forget_after )
```

Construct a new continuous_resolver that resolves all streams with a specific value for a given property. This is analogous to the functionality provided by the free function resolve_stream(prop,value).

Parameters

prop	The stream_info property that should have a specific value (e.g., "name", "type", "source_id", or "desc/manufaturer").
value	The string value that the property should have (e.g., "EEG" as the type property).
forget_after	When a stream is no longer visible on the network (e.g., because it was shut down), this is the time in seconds after which it is no longer reported by the resolver. The recommended default value is 5.0.

7.1.4.16 lsl_create_inlet()

```
LIBLSL_C_API lsl_inlet lsl::lsl_create_inlet (
             lsl_streaminfo info,
             int32_t max_buflen,
             int32_t max_chunklen,
             int32_t recover )
```

Construct a new stream inlet from a resolved stream info.

Parameters

info	A resolved stream info object (as coming from one of the resolver functions). Note: the inlet makes a copy of the info object at its construction. Note: the stream_inlet may also be constructed with a fully-specified stream_info, if the desired channel format and count is already known up-front, but this is strongly discouraged and should only ever be done if there is no time to resolve the stream up-front (e.g., due to limitations in the client program).
max_buflen	Optionally the maximum amount of data to buffer (in seconds if there is a nominal sampling rate, otherwise x100 in samples). Recording applications want to use a fairly large buffer size here, while real-time applications would only buffer as much as they need to perform their next calculation. A good default is 360, which corresponds to 6 minutes of data.
max_chunklen	Optionally the maximum size, in samples, at which chunks are transmitted. If specified as 0, the chunk sizes preferred by the sender are used. Recording applications can use a generous size here (leaving it to the network how to pack things), while real-time applications may want a finer (perhaps 1-sample) granularity.
recover	Try to silently recover lost streams that are recoverable (=those that that have a source_id set). It is generally a good idea to enable this, unless the application wants to act in a special way when a data provider has temporarily crashed. If recover is 0 or the stream is not recoverable, most outlet functions will return an lsl_lost_error if the stream's source is lost.

Returns

A newly created lsl_inlet handle or NULL in the event that an error occurred.

7.1.4.17 lsl_create_outlet()

```
LIBLSL_C_API lsl_outlet lsl::lsl_create_outlet (
             lsl_streaminfo info,
             int32_t chunk_size,
             int32_t max_buffered )
```

Establish a new stream outlet. This makes the stream discoverable.

Parameters

info	The stream information to use for creating this stream. Stays constant over the lifetime of the outlet. Note: the outlet makes a copy of the streaminfo object upon construction (so the old info should still be destroyed.)
chunk_size	Optionally the desired chunk granularity (in samples) for transmission. If specified as 0, each push operation yields one chunk. Stream recipients can have this setting bypassed.
max_buffered	Optionally the maximum amount of data to buffer (in seconds if there is a nominal sampling rate, otherwise x100 in samples). A good default is 360, which corresponds to 6 minutes of data. Note that, for high-bandwidth data you will almost certainly want to use a lower value here to avoid running out of RAM.

Returns

A newly created lsl_outlet handle or NULL in the event that an error occurred.

7.1.4.18 lsl_create_streaminfo()

```
LIBLSL_C_API lsl_streaminfo lsl::lsl_create_streaminfo (
            const char * name,
            const char * type,
            int32_t channel_count,
            double nominal_srate,
            lsl_channel_format_t channel_format,
            const char * source_id )
```

Construct a new streaminfo object. Core stream information is specified here. Any remaining meta-data can be added later.

Parameters

name	Name of the stream. Describes the device (or product series) that this stream makes available (for use by programs, experimenters or data analysts). Cannot be empty.
type	Content type of the stream. Please see https://github.com/sccn/xdf/wiki/Meta-Data (or web search for: XDF meta-data) for pre-defined content-type names, but you can also make up your own. The content type is the preferred way to find streams (as opposed to searching by name).
channel_count	Number of channels per sample. This stays constant for the lifetime of the stream.
nominal_srate	The sampling rate (in Hz) as advertised by the data source, if regular (otherwise set to IRREGULAR_RATE).
channel_format	Format/type of each channel. If your channels have different formats, consider supplying multiple streams or use the largest type that can hold them all (such as cft_double64). A good default is cft_float32.
source_id	Unique identifier of the source or device, if available (such as the serial number). Allows recipients to recover from failure even after the serving app or device crashes. May in some cases also be constructed from device settings.

Returns

A newly created streaminfo handle or NULL in the event that an error occurred.

7.1.4.19 lsl_destroy_continuous_resolver()

```
LIBLSL_C_API void lsl::lsl_destroy_continuous_resolver (
            lsl_continuous_resolver res )
```

Destructor for the continuous resolver.

7.1.4.20 lsl_destroy_inlet()

```
LIBLSL_C_API void lsl::lsl_destroy_inlet (
             lsl_inlet in )
```

Destructor. The inlet will automatically disconnect if destroyed.

7.1.4.21 lsl_destroy_outlet()

```
LIBLSL_C_API void lsl::lsl_destroy_outlet (
             lsl_outlet out )
```

Destroy an outlet. The outlet will no longer be discoverable after destruction and all connected inlets will stop delivering data.

7.1.4.22 lsl_destroy_streaminfo()

```
LIBLSL_C_API void lsl::lsl_destroy_streaminfo (
             lsl_streaminfo info )
```

Destroy a previously created streaminfo object.

7.1.4.23 lsl_destroy_string()

```
LIBLSL_C_API void lsl::lsl_destroy_string (
             char * s )
```

Deallocate a string that has been transferred to the application. Rarely used: the only use case is to deallocate the contents of string-valued samples received from LSL in an application where no free() method is available (e.g., in some scripting languages).

7.1.4.24 lsl_empty()

```
LIBLSL_C_API int32_t lsl::lsl_empty (
             lsl_xml_ptr e )
```

Whether this node is empty.

7.1.4.25 lsl_first_child()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_first_child (
             lsl_xml_ptr e )
```

Get the first child of the element.

7.1.4.26 lsl_get_channel_bytes()

```
LIBLSL_C_API int32_t lsl::lsl_get_channel_bytes (
             lsl_streaminfo info )
```

Number of bytes occupied by a channel (0 for string-typed channels).

7.1.4.27 lsl_get_channel_count()

```
LIBLSL_C_API int32_t lsl::lsl_get_channel_count (
             lsl_streaminfo info )
```

Number of channels of the stream. A stream has at least one channels; the channel count stays constant for all samples.

7.1.4.28 lsl_get_channel_format()

```
LIBLSL_C_API lsl_channel_format_t lsl::lsl_get_channel_format (
             lsl_streaminfo info )
```

Channel format of the stream. All channels in a stream have the same format. However, a device might offer multiple time-synched streams each with its own format.

7.1.4.29 lsl_get_created_at()

```
LIBLSL_C_API double lsl::lsl_get_created_at (
             lsl_streaminfo info )
```

Creation time stamp of the stream. This is the time stamp when the stream was first created (as determined via local_clock() on the providing machine).

7.1.4.30 lsl_get_desc()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_get_desc (
             lsl_streaminfo info )
```

Extended description of the stream. It is highly recommended that at least the channel labels are described here. See code examples on the LSL wiki. Other information, such as amplifier settings, measurement units if deviating from defaults, setup information, subject information, etc., can be specified here, as well. Meta-data recommendations follow the XDF file format project (github.com/sccn/xdf/wiki/Meta-Data or web search for: XDF meta-data).

Important: if you use a stream content type for which meta-data recommendations exist, please try to lay out your meta-data in agreement with these recommendations for compatibility with other applications.

7.1.4.31 lsl_get_fullinfo()

```
LIBLSL_C_API lsl_streaminfo lsl::lsl_get_fullinfo (
             lsl_inlet in,
             double timeout,
             int32_t * ec )
```

Retrieve the complete information of the given stream, including the extended description. Can be invoked at any time of the stream's lifetime.

Parameters

in	The lsl_inlet object to act on.
timeout	Timeout of the operation. Use LSL_FOREVER to effectively disable it.
ec	Error code: if nonzero, can be either lsl_timeout_error (if the timeout has expired) or lsl_lost_error (if the stream source has been lost).

Returns

A copy of the full streaminfo of the inlet or NULL in the event that an error happened. Note: it is the user's responsibility to destroy it when it is no longer needed.

7.1.4.32 lsl_get_hostname()

```
LIBLSL_C_API const char* lsl::lsl_get_hostname (
             lsl_streaminfo info )
```

Hostname of the providing machine (once bound to an outlet). Modification is not permitted.

7.1.4.33 lsl_get_info()

```
LIBLSL_C_API lsl_streaminfo lsl::lsl_get_info (
             lsl_outlet out )
```

Retrieve a handle to the stream info provided by this outlet. This is what was used to create the stream (and also has the Additional Network Information fields assigned).

Returns

A copy of the streaminfo of the outlet or NULL in the event that an error occurred. Note: it is the user's responsibility to destroy it when it is no longer needed.

7.1.4.34 lsl_get_name()

```
LIBLSL_C_API const char* lsl::lsl_get_name (
             lsl_streaminfo info )
```

Name of the stream. This is a human-readable name. For streams offered by device modules, it refers to the type of device or product series that is generating the data of the stream. If the source is an application, the name may be a more generic or specific identifier. Multiple streams with the same name can coexist, though potentially at the cost of ambiguity (for the recording app or experimenter).

Returns

A library-owned pointer to the string value. Modification is not permitted.

7.1.4.35 lsl_get_nominal_srate()

```
LIBLSL_C_API double lsl::lsl_get_nominal_srate (
             lsl_streaminfo info )
```

Sampling rate of the stream, according to the source (in Hz). If a stream is irregularly sampled, this should be set to IRREGULAR_RATE.

Note that no data will be lost even if this sampling rate is incorrect or if a device has temporary hiccups, since all samples will be recorded anyway (except for those dropped by the device itself). However, when the recording is imported into an application, a good importer may correct such errors more accurately if the advertised sampling rate was close to the specs of the device.

7.1.4.36 lsl_get_sample_bytes()

```
LIBLSL_C_API int32_t lsl::lsl_get_sample_bytes (
             lsl_streaminfo info )
```

Number of bytes occupied by a sample (0 for string-typed channels).

7.1.4.37 lsl_get_session_id()

```
LIBLSL_C_API const char* lsl::lsl_get_session_id (
             lsl_streaminfo info )
```

Session ID for the given stream. The session id is an optional human-assigned identifier of the recording session. While it is rarely used, it can be used to prevent concurrent recording activitites on the same sub-network (e.g., in multiple experiment areas) from seeing each other's streams (assigned via a configuration file by the experimenter, see Network Connectivity on the LSL wiki).

Returns

A library-owned pointer to the string value. Modification is not permitted.

7.1.4.38 lsl_get_source_id()

```
LIBLSL_C_API const char* lsl::lsl_get_source_id (
             lsl_streaminfo info )
```

Unique identifier of the stream's source, if available. The unique source (or device) identifier is an optional piece of information that, if available, allows that endpoints (such as the recording program) can re-acquire a stream automatically once it is back online.

Returns

A library-owned pointer to the string value. Modification is not permitted.

7.1.4.39 lsl_get_type()

```
LIBLSL_C_API const char* lsl::lsl_get_type (
            lsl_streaminfo info )
```

Content type of the stream. The content type is a short string such as "EEG", "Gaze" which describes the content carried by the channel (if known). If a stream contains mixed content this value need not be assigned but may instead be stored in the description of channel types. To be useful to applications and automated processing systems using the recommended content types is preferred. Content types usually follow those pre-defined in https://github.com/sccn/xdf/wiki/Meta-Data (or web search for: XDF meta-data).

Returns

A library-owned pointer to the string value. Modification is not permitted.

7.1.4.40 lsl_get_uid()

```
LIBLSL_C_API const char* lsl::lsl_get_uid (
            lsl_streaminfo info )
```

Unique ID of the stream outlet (once assigned). This is a unique identifier of the stream outlet, and is guaranteed to be different across multiple instantiations of the same outlet (e.g., after a re-start).

Returns

A library-owned pointer to the string value. Modification is not permitted.

7.1.4.41 lsl_get_version()

```
LIBLSL_C_API int32_t lsl::lsl_get_version (
            lsl_streaminfo info )
```

Protocol version used to deliver the stream.

7.1.4.42 lsl_get_xml()

```
LIBLSL_C_API char* lsl::lsl_get_xml (
            lsl_streaminfo info )
```

Retrieve the entire streaminfo in XML format. This yields an XML document (in string form) whose top-level element is <info>. The info element contains one element for each field of the streaminfo class, including: a) the core elements <name>, <type>, <channel_count>, <nominal_srate>, <channel_format>, <source_id> b) the misc elements <version>, <created_at>, <uid>, <session_id>, <v4address>, <v4data_port>, <v4service_port>, <v6address>, <v6data_port>, <v6service_port> c) the extended description element <desc> with user-defined sub-elements.

Returns

A pointer to a copy of the XML text or NULL in the event that an error occurred. Note: It is the user's responsibility to deallocate this string when it is no longer needed.

7.1.4.43 lsl_have_consumers()

```
LIBLSL_C_API int32_t lsl::lsl_have_consumers (
             lsl_outlet out )
```

Check whether consumers are currently registered. While it does not hurt, there is technically no reason to push samples if there is no consumer.

7.1.4.44 lsl_is_text()

```
LIBLSL_C_API int32_t lsl::lsl_is_text (
             lsl_xml_ptr e )
```

Whether this is a text body (instead of an XML element). True both for plain char data and CData.

7.1.4.45 lsl_last_child()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_last_child (
             lsl_xml_ptr e )
```

Get the last child of the element.

7.1.4.46 lsl_library_info()

```
LIBLSL_C_API const char* lsl::lsl_library_info ( )
```

Get a string containing library information. The format of the string shouldn't be used for anything important except giving a a debugging person a good idea which exact library version is used.

7.1.4.47 lsl_library_version()

```
LIBLSL_C_API int32_t lsl::lsl_library_version ( )
```

Version of the liblsl library. The major version is library_version() / 100; The minor version is library_version() % 100;

7.1.4.48 lsl_local_clock()

```
LIBLSL_C_API double lsl::lsl_local_clock ( )
```

Obtain a local system time stamp in seconds. The resolution is better than a millisecond. This reading can be used to assign time stamps to samples as they are being acquired. If the "age" of a sample is known at a particular time (e.g., from USB transmission delays), it can be used as an offset to lsl_local_clock() to obtain a better estimate of when a sample was actually captured. See lsl_push_sample() for a use case.

7.1.4.49 lsl_name()

```
LIBLSL_C_API const char* lsl::lsl_name (
             lsl_xml_ptr e )
```

Name of the element.

7.1.4.50 lsl_next_sibling()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_next_sibling (
             lsl_xml_ptr e )
```

Get the next sibling in the children list of the parent node.

7.1.4.51 lsl_next_sibling_n()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_next_sibling_n (
             lsl_xml_ptr e,
             const char * name )
```

Get the next sibling with the specified name.

7.1.4.52 lsl_open_stream()

```
LIBLSL_C_API void lsl::lsl_open_stream (
             lsl_inlet in,
             double timeout,
             int32_t * ec )
```

Subscribe to the data stream. All samples pushed in at the other end from this moment onwards will be queued and eventually be delivered in response to pull_sample() calls. Pulling a sample without some preceding lsl_open_↩ stream() is permitted (the stream will then be opened implicitly).

Parameters

in	The lsl_inlet object to act on.
timeout	Optional timeout of the operation. Use LSL_FOREVER to effectively disable it.
ec	Error code: if nonzero, can be either lsl_timeout_error (if the timeout has expired) or lsl_lost_error (if the stream source has been lost).

7.1.4.53 lsl_parent()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_parent (
             lsl_xml_ptr e )
```

Get the parent node.

7.1.4.54 lsl_prepend_child()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_prepend_child (
             lsl_xml_ptr e,
             const char * name )
```

Prepend a child element with the specified name.

7.1.4.55 lsl_prepend_child_value()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_prepend_child_value (
             lsl_xml_ptr e,
             const char * name,
             const char * value )
```

Prepend a child node with a given name, which has a (nameless) plain-text child with the given text value.

7.1.4.56 lsl_prepend_copy()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_prepend_copy (
             lsl_xml_ptr e,
             lsl_xml_ptr e2 )
```

Prepend a child element with the specified name.

7.1.4.57 lsl_previous_sibling()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_previous_sibling (
             lsl_xml_ptr e )
```

Get the previous sibling in the children list of the parent node.

7.1.4.58 lsl_previous_sibling_n()

```
LIBLSL_C_API lsl_xml_ptr lsl::lsl_previous_sibling_n (
             lsl_xml_ptr e,
             const char * name )
```

Get the previous sibling with the specified name.

7.1.4.59 lsl_protocol_version()

```
LIBLSL_C_API int32_t lsl::lsl_protocol_version ( )
```

Protocol version. The major version is protocol_version() / 100; The minor version is protocol_version() % 100; Clients with different minor versions are protocol-compatible with each other while clients with different major versions will refuse to work together.

7.1.4.60 lsl_pull_chunk_buf()

```
LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_buf (
             lsl_inlet in,
             char ** data_buffer,
             uint32_t * lengths_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

Pull a chunk of data from the inlet and read it into an array of binary strings. These strings may contains 0's, therefore the lengths are read into the lengths_buffer array. Handles type checking & conversion. IMPORTANT: Note that the provided data buffer size is measured in channel values (e.g., floats) rather than in samples.

Parameters

in	The lsl_inlet object to act on.
data_buffer	A pointer to a buffer of data values where the results shall be stored.
lengths_buffer	A pointer to an array that holds the resulting lengths for each returned binary string.
timestamp_buffer	A pointer to a buffer of timestamp values where time stamps shall be stored. If this is NULL, no time stamps will be returned.
data_buffer_elements	The size of the data buffer, in channel data elements (of type T). Must be a multiple of the stream's channel count.
timestamp_buffer_elements	The size of the timestamp buffer. If a timestamp buffer is provided then this must correspond to the same number of samples as data_buffer_elements.
timeout	The timeout for this operation, if any. When the timeout expires, the function may return before the entire buffer is filled. The default value of 0.0 will retrieve only data available for immediate pickup.
ec	Error code: can be either no error or lsl_lost_error (if the stream source has been lost). Note: if the timeout expires before a new sample was received the function returns 0.0; ec is *not* set to lsl_timeout_error (because this case is not considered an error condition).

Returns

data_elements_written Number of channel data elements written to the data buffer.

7.1.4.61 lsl_pull_chunk_c()

```
LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_c (
             lsl_inlet in,
             char * data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

7.1.4.62 lsl_pull_chunk_d()

```
LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_d (
            lsl_inlet in,
            double * data_buffer,
            double * timestamp_buffer,
            unsigned long data_buffer_elements,
            unsigned long timestamp_buffer_elements,
            double timeout,
            int32_t * ec )
```

7.1.4.63 lsl_pull_chunk_f()

```
LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_f (
            lsl_inlet in,
            float * data_buffer,
            double * timestamp_buffer,
            unsigned long data_buffer_elements,
            unsigned long timestamp_buffer_elements,
            double timeout,
            int32_t * ec )
```

Pull a chunk of data from the inlet and read it into a buffer. Handles type checking & conversion. IMPORTANT: Note that the provided data buffer size is measured in channel values (e.g., floats) rather than in samples.

Parameters

in	The lsl_inlet object to act on.
data_buffer	A pointer to a buffer of data values where the results shall be stored.
timestamp_buffer	A pointer to a buffer of timestamp values where time stamps shall be stored. If this is NULL, no time stamps will be returned.
data_buffer_elements	The size of the data buffer, in channel data elements (of type T). Must be a multiple of the stream's channel count.
timestamp_buffer_elements	The size of the timestamp buffer. If a timestamp buffer is provided then this must correspond to the same number of samples as data_buffer_elements.
timeout	The timeout for this operation, if any. When the timeout expires, the function may return before the entire buffer is filled. The default value of 0.0 will retrieve only data available for immediate pickup.
ec	Error code: can be either no error or lsl_lost_error (if the stream source has been lost). Note: if the timeout expires before a new sample was received the function returns 0.0; ec is *not* set to lsl_timeout_error (because this case is not considered an error condition).

Returns

data_elements_written Number of channel data elements written to the data buffer.

7.1.4.64 lsl_pull_chunk_i()

```
LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_i (
             lsl_inlet in,
             int32_t * data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

7.1.4.65 lsl_pull_chunk_l()

```
LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_l (
             lsl_inlet in,
             long * data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int * ec )
```

7.1.4.66 lsl_pull_chunk_s()

```
LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_s (
             lsl_inlet in,
             int16_t * data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

7.1.4.67 lsl_pull_chunk_str()

```
LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_str (
             lsl_inlet in,
             char ** data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

7.1.4.68 lsl_pull_sample_buf()

```
LIBLSL_C_API double lsl::lsl_pull_sample_buf (
             lsl_inlet in,
             char ** buffer,
             uint32_t * buffer_lengths,
             int32_t buffer_elements,
             double timeout,
             int32_t * ec )
```

Pull a sample from the inlet and read it into an array of binary strings. These strings may contains 0's, therefore the lengths are read into the buffer_lengths array. Handles type checking & conversion.

Parameters

in	The lsl_inlet object to act on.
buffer	A pointer to hold the resulting data.
buffer_lengths	A pointer to an array that holds the resulting lengths for each returned binary string.
buffer_elements	The number of samples allocated in the buffer and buffer_lengths variables. Note: it is the responsibility of the user to allocate enough memory.
timeout	The timeout for this operation, if any. Use LSL_FOREVER to effectively disable it. It is also permitted to use 0.0 here; in this case a sample is only returned if one is currently buffered.
ec	Error code: can be either no error or lsl_lost_error (if the stream source has been lost). Note: if the timeout expires before a new sample was received the function returns 0.0; ec is *not* set to lsl_timeout_error (because this case is not considered an error condition).

Returns

The capture time of the sample on the remote machine, or 0.0 if no new sample was available. To remap this time stamp to the local clock, add the value returned by lsl_time_correction() to it.

7.1.4.69 lsl_pull_sample_c()

```
LIBLSL_C_API double lsl::lsl_pull_sample_c (
             lsl_inlet in,
             char * buffer,
             int32_t buffer_elements,
             double timeout,
             int32_t * ec )
```

7.1.4.70 lsl_pull_sample_d()

```
LIBLSL_C_API double lsl::lsl_pull_sample_d (
             lsl_inlet in,
             double * buffer,
             int32_t buffer_elements,
             double timeout,
             int32_t * ec )
```

7.1.4.71 lsl_pull_sample_f()

```
LIBLSL_C_API double lsl::lsl_pull_sample_f (
            lsl_inlet in,
            float * buffer,
            int32_t buffer_elements,
            double timeout,
            int32_t * ec )
```

Pull a sample from the inlet and read it into a pointer to values. Handles type checking & conversion.

Parameters

in	The lsl_inlet object to act on.
buffer	A pointer to hold the resulting values.
buffer_elements	The number of samples allocated in the buffer. Note: it is the responsibility of the user to allocate enough memory.
timeout	The timeout for this operation, if any. Use LSL_FOREVER to effectively disable it. It is also permitted to use 0.0 here; in this case a sample is only returned if one is currently buffered.
ec	Error code: can be either no error or lsl_lost_error (if the stream source has been lost). Note: if the timeout expires before a new sample was received the function returns 0.0; ec is *not* set to lsl_timeout_error (because this case is not considered an error condition).

Returns

The capture time of the sample on the remote machine, or 0.0 if no new sample was available. To remap this time stamp to the local clock, add the value returned by lsl_time_correction() to it.

7.1.4.72 lsl_pull_sample_i()

```
LIBLSL_C_API double lsl::lsl_pull_sample_i (
            lsl_inlet in,
            int32_t * buffer,
            int32_t buffer_elements,
            double timeout,
            int32_t * ec )
```

7.1.4.73 lsl_pull_sample_l()

```
LIBLSL_C_API double lsl::lsl_pull_sample_l (
            lsl_inlet in,
            long * buffer,
            int buffer_elements,
            double timeout,
            int * ec )
```

7.1.4.74 lsl_pull_sample_s()

```
LIBLSL_C_API double lsl::lsl_pull_sample_s (
             lsl_inlet in,
             int16_t * buffer,
             int32_t buffer_elements,
             double timeout,
             int32_t * ec )
```

7.1.4.75 lsl_pull_sample_str()

```
LIBLSL_C_API double lsl::lsl_pull_sample_str (
             lsl_inlet in,
             char ** buffer,
             int32_t buffer_elements,
             double timeout,
             int32_t * ec )
```

7.1.4.76 lsl_pull_sample_v()

```
LIBLSL_C_API double lsl::lsl_pull_sample_v (
             lsl_inlet in,
             void * buffer,
             int32_t buffer_bytes,
             double timeout,
             int32_t * ec )
```

Pull a sample from the inlet and read it into a custom struct or buffer. Overall size checking but no type checking or conversion are done. Do not use for variable-size/string-formatted streams.

Parameters

in	The lsl_inlet object to act on.
buffer	Pointer to hold the sample data. Search for #pragma pack for information on how to pack structs appropriately.
buffer_bytes	Length of the array held by buffer in bytes, not items
timeout	The timeout for this operation, if any. Aside from LSL_FOREVER it is also permitted to use 0.0 here; in this case a sample is only returned if one is currently buffered.
ec	Error code: can be either no error or lsl_lost_error (if the stream source has been lost). Note: if the timeout expires before a new sample was received the function returns 0.0; ec is *not* set to lsl_timeout_error (because this case is not considered an error condition).

Returns

The capture time of the sample on the remote machine, or 0.0 if no new sample was available. To remap this time stamp to the local clock, add the value returned by .time_correction() to it.

7.1.4.77 lsl_push_chunk_buf()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_buf (
             lsl_outlet out,
             const char ** data,
             const uint32_t * lengths,
             unsigned long data_elements )
```

7.1.4.78 lsl_push_chunk_buft()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_buft (
             lsl_outlet out,
             const char ** data,
             const uint32_t * lengths,
             unsigned long data_elements,
             double timestamp )
```

7.1.4.79 lsl_push_chunk_buftn()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_buftn (
             lsl_outlet out,
             const char ** data,
             const uint32_t * lengths,
             unsigned long data_elements,
             const double * timestamps )
```

7.1.4.80 lsl_push_chunk_buftnp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_buftnp (
             lsl_outlet out,
             const char ** data,
             const uint32_t * lengths,
             unsigned long data_elements,
             const double * timestamps,
             int32_t pushthrough )
```

7.1.4.81 lsl_push_chunk_buftp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_buftp (
             lsl_outlet out,
             const char ** data,
             const uint32_t * lengths,
             unsigned long data_elements,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.82 lsl_push_chunk_c()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_c (
             lsl_outlet out,
             const char * data,
             unsigned long data_elements )
```

7.1.4.83 lsl_push_chunk_ct()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_ct (
             lsl_outlet out,
             const char * data,
             unsigned long data_elements,
             double timestamp )
```

7.1.4.84 lsl_push_chunk_ctn()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_ctn (
             lsl_outlet out,
             const char * data,
             unsigned long data_elements,
             const double * timestamps )
```

7.1.4.85 lsl_push_chunk_ctnp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_ctnp (
             lsl_outlet out,
             const char * data,
             unsigned long data_elements,
             const double * timestamps,
             int32_t pushthrough )
```

7.1.4.86 lsl_push_chunk_ctp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_ctp (
             lsl_outlet out,
             const char * data,
             unsigned long data_elements,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.87 lsl_push_chunk_d()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_d (
            lsl_outlet out,
            const double * data,
            unsigned long data_elements )
```

7.1.4.88 lsl_push_chunk_dt()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_dt (
            lsl_outlet out,
            const double * data,
            unsigned long data_elements,
            double timestamp )
```

7.1.4.89 lsl_push_chunk_dtn()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_dtn (
            lsl_outlet out,
            const double * data,
            unsigned long data_elements,
            const double * timestamps )
```

7.1.4.90 lsl_push_chunk_dtnp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_dtnp (
            lsl_outlet out,
            const double * data,
            unsigned long data_elements,
            const double * timestamps,
            int32_t pushthrough )
```

7.1.4.91 lsl_push_chunk_dtp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_dtp (
            lsl_outlet out,
            const double * data,
            unsigned long data_elements,
            double timestamp,
            int32_t pushthrough )
```

7.1.4.92 lsl_push_chunk_f()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_f (
            lsl_outlet out,
            const float * data,
            unsigned long data_elements )
```

Push a chunk of multiplexed samples into the outlet. One timestamp per sample is provided. IMPORTANT: Note that the provided buffer size is measured in channel values (e.g., floats) rather than in samples. Handles type checking & conversion.

Parameters

out	The lsl_outlet object through which to push the data.
data	A buffer of channel values holding the data for zero or more successive samples to send.
lengths	For lsl_push_chunk_buf∗, a pointer the number of elements to push for each value (string lengths).
timestamp	Optionally the capture time of the most recent sample, in agreement with local_clock(); if omitted, the current time is used. The time stamps of other samples are automatically derived based on the sampling rate of the stream.
timestamps	Alternatively a buffer of timestamp values holding time stamps for each sample in the data buffer.
data_elements	The number of data values (of type T) in the data buffer. Must be a multiple of the channel count.
pushthrough	Whether to push the chunk through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

Returns

Error code of the operation (usually attributed to the wrong data type).

7.1.4.93 lsl_push_chunk_ft()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_ft (
            lsl_outlet out,
            const float * data,
            unsigned long data_elements,
            double timestamp )
```

7.1.4.94 lsl_push_chunk_ftn()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_ftn (
            lsl_outlet out,
            const float * data,
            unsigned long data_elements,
            const double * timestamps )
```

7.1.4.95 lsl_push_chunk_ftnp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_ftnp (
             lsl_outlet out,
             const float * data,
             unsigned long data_elements,
             const double * timestamps,
             int32_t pushthrough )
```

7.1.4.96 lsl_push_chunk_ftp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_ftp (
             lsl_outlet out,
             const float * data,
             unsigned long data_elements,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.97 lsl_push_chunk_i()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_i (
             lsl_outlet out,
             const int32_t * data,
             unsigned long data_elements )
```

7.1.4.98 lsl_push_chunk_it()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_it (
             lsl_outlet out,
             const int32_t * data,
             unsigned long data_elements,
             double timestamp )
```

7.1.4.99 lsl_push_chunk_itn()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_itn (
             lsl_outlet out,
             const int32_t * data,
             unsigned long data_elements,
             const double * timestamps )
```

7.1.4.100 lsl_push_chunk_itnp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_itnp (
             lsl_outlet out,
             const int32_t * data,
             unsigned long data_elements,
             const double * timestamps,
             int32_t pushthrough )
```

7.1.4.101 lsl_push_chunk_itp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_itp (
             lsl_outlet out,
             const int32_t * data,
             unsigned long data_elements,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.102 lsl_push_chunk_l()

```
LIBLSL_C_API int lsl::lsl_push_chunk_l (
             lsl_outlet out,
             const long * data,
             unsigned long data_elements )
```

7.1.4.103 lsl_push_chunk_lt()

```
LIBLSL_C_API int lsl::lsl_push_chunk_lt (
             lsl_outlet out,
             const long * data,
             unsigned long data_elements,
             double timestamp )
```

7.1.4.104 lsl_push_chunk_ltn()

```
LIBLSL_C_API int lsl::lsl_push_chunk_ltn (
             lsl_outlet out,
             const long * data,
             unsigned long data_elements,
             const double * timestamps )
```

7.1.4.105 lsl_push_chunk_ltnp()

```
LIBLSL_C_API int lsl::lsl_push_chunk_ltnp (
             lsl_outlet out,
             const long * data,
             unsigned long data_elements,
             const double * timestamps,
             int pushthrough )
```

7.1.4.106 lsl_push_chunk_ltp()

```
LIBLSL_C_API int lsl::lsl_push_chunk_ltp (
             lsl_outlet out,
             const long * data,
             unsigned long data_elements,
             double timestamp,
             int pushthrough )
```

7.1.4.107 lsl_push_chunk_s()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_s (
             lsl_outlet out,
             const int16_t * data,
             unsigned long data_elements )
```

7.1.4.108 lsl_push_chunk_st()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_st (
             lsl_outlet out,
             const int16_t * data,
             unsigned long data_elements,
             double timestamp )
```

7.1.4.109 lsl_push_chunk_stn()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_stn (
             lsl_outlet out,
             const int16_t * data,
             unsigned long data_elements,
             const double * timestamps )
```

7.1.4.110 lsl_push_chunk_stnp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_stnp (
             lsl_outlet out,
             const int16_t * data,
             unsigned long data_elements,
             const double * timestamps,
             int32_t pushthrough )
```

7.1.4.111 lsl_push_chunk_stp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_stp (
             lsl_outlet out,
             const int16_t * data,
             unsigned long data_elements,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.112 lsl_push_chunk_str()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_str (
             lsl_outlet out,
             const char ** data,
             unsigned long data_elements )
```

7.1.4.113 lsl_push_chunk_strt()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_strt (
             lsl_outlet out,
             const char ** data,
             unsigned long data_elements,
             double timestamp )
```

7.1.4.114 lsl_push_chunk_strtn()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_strtn (
             lsl_outlet out,
             const char ** data,
             unsigned long data_elements,
             const double * timestamps )
```

7.1.4.115 lsl_push_chunk_strtnp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_strtnp (
             lsl_outlet out,
             const char ** data,
             unsigned long data_elements,
             const double * timestamps,
             int32_t pushthrough )
```

7.1.4.116 lsl_push_chunk_strtp()

```
LIBLSL_C_API int32_t lsl::lsl_push_chunk_strtp (
             lsl_outlet out,
             const char ** data,
             unsigned long data_elements,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.117 lsl_push_sample_buf()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_buf (
             lsl_outlet out,
             const char ** data,
             const uint32_t * lengths )
```

7.1.4.118 lsl_push_sample_buft()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_buft (
             lsl_outlet out,
             const char ** data,
             const uint32_t * lengths,
             double timestamp )
```

7.1.4.119 lsl_push_sample_buftp()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_buftp (
             lsl_outlet out,
             const char ** data,
             const uint32_t * lengths,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.120 lsl_push_sample_c()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_c (
             lsl_outlet out,
             const char * data )
```

7.1.4.121 lsl_push_sample_ct()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_ct (
             lsl_outlet out,
             const char * data,
             double timestamp )
```

7.1.4.122 lsl_push_sample_ctp()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_ctp (
             lsl_outlet out,
             const char * data,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.123 lsl_push_sample_d()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_d (
             lsl_outlet out,
             const double * data )
```

7.1.4.124 lsl_push_sample_dt()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_dt (
             lsl_outlet out,
             const double * data,
             double timestamp )
```

7.1.4.125 lsl_push_sample_dtp()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_dtp (
             lsl_outlet out,
             const double * data,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.126 lsl_push_sample_f()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_f (
             lsl_outlet out,
             const float * data )
```

Push a pointer to some values as a sample into the outlet. Handles type checking & conversion.

Parameters

out	The lsl_outlet object through which to push the data.
data	A pointer to values to push. The number of values pointed to must be no less than the number of channels in the sample.
lengths	For lsl_push_sample_buf∗, a pointer the number of elements to push for each channel (string lengths).
timestamp	Optionally the capture time of the sample, in agreement with lsl_local_clock(); if omitted, the current time is used.
pushthrough	Whether to push the sample through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

Returns

Error code of the operation or lsl_no_error if successful (usually attributed to the wrong data type).

7.1.4.127 lsl_push_sample_ft()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_ft (
             lsl_outlet out,
             const float * data,
             double timestamp )
```

7.1.4.128 lsl_push_sample_ftp()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_ftp (
             lsl_outlet out,
             const float * data,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.129 lsl_push_sample_i()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_i (
             lsl_outlet out,
             const int32_t * data )
```

7.1.4.130 lsl_push_sample_it()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_it (
            lsl_outlet out,
            const int32_t * data,
            double timestamp )
```

7.1.4.131 lsl_push_sample_itp()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_itp (
            lsl_outlet out,
            const int32_t * data,
            double timestamp,
            int32_t pushthrough )
```

7.1.4.132 lsl_push_sample_l()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_l (
            lsl_outlet out,
            const long * data )
```

7.1.4.133 lsl_push_sample_lt()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_lt (
            lsl_outlet out,
            const long * data,
            double timestamp )
```

7.1.4.134 lsl_push_sample_ltp()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_ltp (
            lsl_outlet out,
            const long * data,
            double timestamp,
            int32_t pushthrough )
```

7.1.4.135 lsl_push_sample_s()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_s (
             lsl_outlet out,
             const int16_t * data )
```

7.1.4.136 lsl_push_sample_st()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_st (
             lsl_outlet out,
             const int16_t * data,
             double timestamp )
```

7.1.4.137 lsl_push_sample_stp()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_stp (
             lsl_outlet out,
             const int16_t * data,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.138 lsl_push_sample_str()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_str (
             lsl_outlet out,
             const char ** data )
```

7.1.4.139 lsl_push_sample_strt()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_strt (
             lsl_outlet out,
             const char ** data,
             double timestamp )
```

7.1.4.140 lsl_push_sample_strtp()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_strtp (
             lsl_outlet out,
             const char ** data,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.141 lsl_push_sample_v()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_v (
             lsl_outlet out,
             const void * data )
```

7.1.4.142 lsl_push_sample_vt()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_vt (
             lsl_outlet out,
             const void * data,
             double timestamp )
```

7.1.4.143 lsl_push_sample_vtp()

```
LIBLSL_C_API int32_t lsl::lsl_push_sample_vtp (
             lsl_outlet out,
             const void * data,
             double timestamp,
             int32_t pushthrough )
```

7.1.4.144 lsl_remove_child()

```
LIBLSL_C_API void lsl::lsl_remove_child (
             lsl_xml_ptr e,
             lsl_xml_ptr e2 )
```

Remove a specified child element.

7.1.4.145 lsl_remove_child_n()

```
LIBLSL_C_API void lsl::lsl_remove_child_n (
             lsl_xml_ptr e,
             const char * name )
```

Remove a child element with the specified name.

7.1.4.146 lsl_resolve_all()

```
LIBLSL_C_API int32_t lsl::lsl_resolve_all (
             lsl_streaminfo * buffer,
             uint32_t buffer_elements,
             double wait_time )
```

Resolve all streams on the network. This function returns all currently available streams from any outlet on the network. The network is usually the subnet specified at the local router, but may also include a multicast group of machines (given that the network supports it), or a list of hostnames. These details may optionally be customized by the experimenter in a configuration file (see page Network Connectivity in the LSL wiki). This is the default mechanism used by the browsing programs and the recording program.

Parameters

buffer	A user-allocated buffer to hold the resolve results. Note: it is the user's responsibility to either destroy the resulting streaminfo objects or to pass them back to the LSL during during creation of an inlet. Note 2: The stream_info's returned by the resolver are only short versions that do not include the .desc() field (which can be arbitrarily big). To obtain the full stream information you need to call .info() on the inlet after you have created one.
buffer_elements	The user-provided buffer length.
wait_time	The waiting time for the operation, in seconds, to search for streams. The recommended wait time is 1 second (or 2 for a busy and large recording operation). Warning: If this is too short (<0.5s) only a subset (or none) of the outlets that are present on the network may be returned.

Returns

The number of results written into the buffer (never more than the provided # of slots) or a negative number if an error has occurred (values corresponding to lsl_error_code_t).

7.1.4.147 lsl_resolve_bypred()

```
LIBLSL_C_API int32_t lsl::lsl_resolve_bypred (
             lsl_streaminfo * buffer,
             uint32_t buffer_elements,
             const char * pred,
             int32_t minimum,
             double timeout )
```

Resolve all streams that match a given predicate. Advanced query that allows to impose more conditions on the retrieved streams; the given string is an XPath 1.0 predicate for the <info> node (omitting the surrounding []'s), see also `http://en.wikipedia.org/w/index.php?title=XPath_1.0&oldid=474981951`.

Parameters

buffer	A user-allocated buffer to hold the resolve results. Note: it is the user's responsibility to either destroy the resulting streaminfo objects or to pass them back to the LSL during during creation of an inlet. Note 2: The stream_info's returned by the resolver are only short versions that do not include the .desc() field (which can be arbitrarily big). To obtain the full stream information you need to call .info() on the inlet after you have created one.
buffer_elements	The user-provided buffer length.
pred	The predicate string, e.g. "name='BioSemi'" or "type='EEG' and starts-with(name,'BioSemi') and count(info/desc/channel)=32"
minimum	Return at least this number of streams.
timeout	Optionally a timeout of the operation, in seconds (default: no timeout). If the timeout expires, less than the desired number of streams (possibly none) will be returned.

Returns

The number of results written into the buffer (never more than the provided # of slots) or a negative number if an error has occurred (values corresponding to lsl_error_code_t).

7.1.4.148 lsl_resolve_byprop()

```
LIBLSL_C_API int32_t lsl::lsl_resolve_byprop (
             lsl_streaminfo * buffer,
             uint32_t buffer_elements,
             const char * prop,
             const char * value,
             int32_t minimum,
             double timeout )
```

Resolve all streams with a given value for a property. If the goal is to resolve a specific stream, this method is preferred over resolving all streams and then selecting the desired one.

Parameters

buffer	A user-allocated buffer to hold the resolve results. Note: it is the user's responsibility to either destroy the resulting streaminfo objects or to pass them back to the LSL during during creation of an inlet. Note 2: The stream_info's returned by the resolver are only short versions that do not include the .desc() field (which can be arbitrarily big). To obtain the full stream information you need to call .info() on the inlet after you have created one.
buffer_elements	The user-provided buffer length.
prop	The streaminfo property that should have a specific value ("name", "type", "source_id", or, e.g., "desc/manufaturer" if present).
value	The string value that the property should have (e.g., "EEG" as the type).
minimum	Return at least this number of streams.
timeout	Optionally a timeout of the operation, in seconds (default: no timeout). If the timeout expires, less than the desired number of streams (possibly none) will be returned.

Returns

The number of results written into the buffer (never more than the provided # of slots) or a negative number if an error has occurred (values corresponding to lsl_error_code_t).

7.1.4.149 lsl_resolver_results()

```
LIBLSL_C_API int32_t lsl::lsl_resolver_results (
             lsl_continuous_resolver res,
             lsl_streaminfo * buffer,
             uint32_t buffer_elements )
```

Obtain the set of currently present streams on the network (i.e. resolve result).

Parameters

res	A continuous resolver (previously created with one of the lsl_create_continuous_resolver functions).
buffer	A user-allocated buffer to hold the current resolve results. Note: it is the user's responsibility to either destroy the resulting streaminfo objects or to pass them back to the LSL during during creation of an inlet. Note 2: The stream_info's returned by the resolver are only short versions that do not include the .desc() field (which can be arbitrarily big). To obtain the full stream information you need to call .info() on the inlet after you have created one.
buffer_elements	The user-provided buffer length.

Returns

The number of results written into the buffer (never more than the provided # of slots) or a negative number if an error has occurred (values corresponding to lsl_error_code_t).

7.1.4.150 lsl_samples_available()

```
LIBLSL_C_API uint32_t lsl::lsl_samples_available (
             lsl_inlet in )
```

Query whether samples are currently available for immediate pickup. Note that it is not a good idea to use samples↩_available() to determine whether a pull_∗() call would block: to be sure, set the pull timeout to 0.0 or an acceptably low value. If the underlying implementation supports it, the value will be the number of samples available (otherwise it will be 1 or 0).

7.1.4.151 lsl_set_child_value()

```
LIBLSL_C_API int32_t lsl::lsl_set_child_value (
             lsl_xml_ptr e,
             const char * name,
             const char * value )
```

Set the text value of the (nameless) plain-text child of a named child node.

7.1.4.152 lsl_set_name()

```
LIBLSL_C_API int32_t lsl::lsl_set_name (
             lsl_xml_ptr e,
             const char * rhs )
```

Set the element's name.

Returns

0 if the node is empty (or if out of memory).

7.1.4.153 lsl_set_postprocessing()

```
LIBLSL_C_API int32_t lsl::lsl_set_postprocessing (
             lsl_inlet in,
             uint32_t flags )
```

Set post-processing flags to use. By default, the inlet performs NO post-processing and returns the ground-truth time stamps, which can then be manually synchronized using time_correction(), and then smoothed/dejittered if desired. This function allows automating these two and possibly more operations. Warning: when you enable this, you will no longer receive or be able to recover the original time stamps.

Parameters

in	The lsl_inlet object to act on.
flags	An integer that is the result of bitwise OR'ing one or more options from processing_options_t together (e.g., post_clocksync|post_dejitter); a good setting is to use post_ALL.

Returns

The error code: if nonzero, can be lsl_argument_error if an unknown flag was passed in.

7.1.4.154 lsl_set_value()

```
LIBLSL_C_API int32_t lsl::lsl_set_value (
            lsl_xml_ptr e,
            const char * rhs )
```

Set the element's value.

Returns

0 if the node is empty (or if out of memory).

7.1.4.155 lsl_smoothing_halftime()

```
LIBLSL_C_API int32_t lsl::lsl_smoothing_halftime (
            lsl_inlet in,
            float value )
```

Override the half-time (forget factor) of the time-stamp smoothing. The default is 90 seconds unless a different value is set in the config file. Using a longer window will yield lower jitter in the time stamps, but longer windows will have trouble tracking changes in the clock rate (usually due to temperature changes); the default is able to track changes up to 10 degrees C per minute sufficiently well.

Parameters

in	The lsl_inlet object to act on.
value	The new value, in seconds. This is the time after which a past sample will be weighted by 1/2 in the exponential smoothing window.

Returns

The error code: if nonzero, can be lsl_argument_error if an unknown flag was passed in.

7.1.4.156 lsl_stream_info_matches_query()

```
LIBLSL_C_API int lsl::lsl_stream_info_matches_query (
             lsl_streaminfo info,
             const char * query )
```

Tries to match the stream info XML element `info` against an `XPath` query.

Example query strings:

```
channel_count>5 and type='EEG'
type='TestStream' or contains(name,'Brain')
name='ExampleStream'
```

7.1.4.157 lsl_streaminfo_from_xml()

```
LIBLSL_C_API lsl_streaminfo lsl::lsl_streaminfo_from_xml (
             const char * xml )
```

Create a streaminfo object from an XML representation.

7.1.4.158 lsl_time_correction()

```
LIBLSL_C_API double lsl::lsl_time_correction (
             lsl_inlet in,
             double timeout,
             int32_t * ec )
```

Retrieve an estimated time correction offset for the given stream. The first call to this function takes several milliseconds until a reliable first estimate is obtained. Subsequent calls are instantaneous (and rely on periodic background updates). On a well-behaved network, the precision of these estimates should be below 1 ms (empirically it is within +/-0.2 ms). To get a measure of whether the network is well-behaved, use lsl_time_correction_ex and check uncertainty (which maps to round-trip-time). 0.2 ms is typical of wired networks. 2 ms is typical of wireless networks. The number can be much higher on poor networks.

Parameters

in	The lsl_inlet object to act on.
remote_time	The current time of the remote computer that was used to generate this time_correction. If desired, the client can fit time_correction vs remote_time to improve the real-time time_correction further.
uncertainty.	The maximum uncertainty of the given time correction.
timeout	Timeout to acquire the first time-correction estimate. Use LSL_FOREVER to defuse the timeout.
ec	Error code: if nonzero, can be either lsl_timeout_error (if the timeout has expired) or lsl_lost_error (if the stream source has been lost).

Returns

The time correction estimate. This is the number that needs to be added to a time stamp that was remotely generated via lsl_local_clock() to map it into the local clock domain of this machine.

7.1.4.159 lsl_time_correction_ex()

```
LIBLSL_C_API double lsl::lsl_time_correction_ex (
             lsl_inlet in,
             double * remote_time,
             double * uncertainty,
             double timeout,
             int32_t * ec )
```

7.1.4.160 lsl_value()

```
LIBLSL_C_API const char* lsl::lsl_value (
             lsl_xml_ptr e )
```

Value of the element.

7.1.4.161 lsl_wait_for_consumers()

```
LIBLSL_C_API int32_t lsl::lsl_wait_for_consumers (
             lsl_outlet out,
             double timeout )
```

Wait until some consumer shows up (without wasting resources).

Returns

True if the wait was successful, false if the timeout expired.

7.1.4.162 lsl_was_clock_reset()

```
LIBLSL_C_API uint32_t lsl::lsl_was_clock_reset (
             lsl_inlet in )
```

Query whether the clock was potentially reset since the last call to was_clock_reset(). This is rarely-used function is only needed for applications that combine multiple time_correction values to estimate precise clock drift if they should tolerate cases where the source machine was hot-swapped or restarted.

7.1.4.163 protocol_version()

```
int32_t lsl::protocol_version ( )  [inline]
```

Protocol version. The major version is protocol_version() / 100; The minor version is protocol_version() % 100; Clients with different minor versions are protocol-compatible with each other while clients with different major versions will refuse to work together.

7.1.4.164 resolve_stream() [1/2]

```
std::vector<stream_info> lsl::resolve_stream (
             const std::string & prop,
             const std::string & value,
             int32_t minimum = 1,
             double timeout = FOREVER )  [inline]
```

Resolve all streams with a specific value for a given property. If the goal is to resolve a specific stream, this method is preferred over resolving all streams and then selecting the desired one.

Parameters

prop	The stream_info property that should have a specific value (e.g., "name", "type", "source_id", or "desc/manufaturer").
value	The string value that the property should have (e.g., "EEG" as the type property).
minimum	Return at least this number of streams.
timeout	Optionally a timeout of the operation, in seconds (default: no timeout). If the timeout expires, less than the desired number of streams (possibly none) will be returned.

Returns

A vector of matching stream info objects (excluding their meta-data), any of which can subsequently be used to open an inlet.

7.1.4.165 resolve_stream() [2/2]

```
std::vector<stream_info> lsl::resolve_stream (
             const std::string & pred,
             int32_t minimum = 1,
             double timeout = FOREVER )  [inline]
```

Resolve all streams that match a given predicate. Advanced query that allows to impose more conditions on the retrieved streams; the given string is an XPath 1.0 predicate for the <info> node (omitting the surrounding []'s), see also http://en.wikipedia.org/w/index.php?title=XPath_1.0&oldid=474981951.

Parameters

pred	The predicate string, e.g. "name='BioSemi'" or "type='EEG' and starts-with(name,'BioSemi') and count(info/desc/channel)=32"
minimum	Return at least this number of streams.
timeout	Optionally a timeout of the operation, in seconds (default: no timeout). If the timeout expires, less than the desired number of streams (possibly none) will be returned.

Returns

A vector of matching stream info objects (excluding their meta-data), any of which can subsequently be used to open an inlet.

7.1.4.166 resolve_streams()

```
std::vector<stream_info> lsl::resolve_streams (
            double wait_time = 1.0 )  [inline]
```

Resolve all streams on the network. This function returns all currently available streams from any outlet on the network. The network is usually the subnet specified at the local router, but may also include a multicast group of machines (given that the network supports it), or list of hostnames. These details may optionally be customized by the experimenter in a configuration file (see Network Connectivity in the LSL wiki). This is the default mechanism used by the browsing programs and the recording program.

Parameters

wait_time	The waiting time for the operation, in seconds, to search for streams. Warning: If this is too short (<0.5s) only a subset (or none) of the outlets that are present on the network may be returned.

Returns

A vector of stream info objects (excluding their desc field), any of which can subsequently be used to open an inlet. The full info can be retrieve from the inlet.

7.1.5 Variable Documentation

7.1.5.1 DEDUCED_TIMESTAMP

```
const double lsl::DEDUCED_TIMESTAMP = -1.0
```

Constant to indicate that a sample has the next successive time stamp. This is an optional optimization to transmit less data per sample. The stamp is then deduced from the preceding one according to the stream's sampling rate (in the case of an irregular rate, the same time stamp as before will is assumed).

7.1.5.2 FOREVER

```
const double lsl::FOREVER = 32000000.0
```

A very large time duration (> 1 year) for timeout values. Note that significantly larger numbers can cause the timeout to be invalid on some operating systems (e.g., 32-bit UNIX).

7.1.5.3 IRREGULAR_RATE

```
const double lsl::IRREGULAR_RATE = 0.0
```

Constant to indicate that a stream has variable sampling rate.

7.2 Ui Namespace Reference

Classes

- class MainWindow

Chapter 8

Class Documentation

8.1 lsl::continuous_resolver Class Reference

```
#include <lsl_cpp.h>
```

Public Member Functions

- continuous_resolver (double forget_after=5.0)
- continuous_resolver (const std::string &prop, const std::string &value, double forget_after=5.0)
- continuous_resolver (const std::string &pred, double forget_after=5.0)
- std::vector< stream_info > results ()
- ∼continuous_resolver ()

8.1.1 Detailed Description

A convenience class that resolves streams continuously in the background throughout its lifetime and which can be queried at any time for the set of streams that are currently visible on the network.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 continuous_resolver() [1/3]

```
lsl::continuous_resolver::continuous_resolver (
            double forget_after = 5.0 )  [inline]
```

Construct a new continuous_resolver that resolves all streams on the network. This is analogous to the functionality offered by the free function resolve_streams().

Parameters

forget_after	When a stream is no longer visible on the network (e.g., because it was shut down), this is the time in seconds after which it is no longer reported by the resolver.

8.1.2.2 continuous_resolver() [2/3]

```
lsl::continuous_resolver::continuous_resolver (
            const std::string & prop,
            const std::string & value,
            double forget_after = 5.0 )  [inline]
```

Construct a new continuous_resolver that resolves all streams with a specific value for a given property. This is analogous to the functionality provided by the free function resolve_stream(prop,value).

Parameters

prop	The stream_info property that should have a specific value (e.g., "name", "type", "source_id", or "desc/manufaturer").
value	The string value that the property should have (e.g., "EEG" as the type property).
forget_after	When a stream is no longer visible on the network (e.g., because it was shut down), this is the time in seconds after which it is no longer reported by the resolver.

8.1.2.3 continuous_resolver() [3/3]

```
lsl::continuous_resolver::continuous_resolver (
            const std::string & pred,
            double forget_after = 5.0 )  [inline]
```

Construct a new continuous_resolver that resolves all streams that match a given XPath 1.0 predicate. This is analogous to the functionality provided by the free function resolve_stream(pred).

Parameters

pred	The predicate string, e.g. "name='BioSemi'" or "type='EEG' and starts-with(name,'BioSemi') and count(info/desc/channel)=32"
forget_after	When a stream is no longer visible on the network (e.g., because it was shut down), this is the time in seconds after which it is no longer reported by the resolver.

8.1.2.4 ∼continuous_resolver()

```
lsl::continuous_resolver::∼continuous_resolver ( )  [inline]
```

Destructor.

8.1.3 Member Function Documentation

8.1.3.1 results()

```
std::vector<stream_info> lsl::continuous_resolver::results ( )  [inline]
```

Obtain the set of currently present streams on the network (i.e. resolve result).

Returns

A vector of matching stream info objects (excluding their meta-data), any of which can subsequently be used to open an inlet.

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

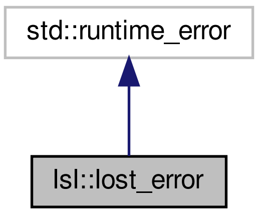
- include/lsl_cpp.h

8.2 lsl::lost_error Class Reference

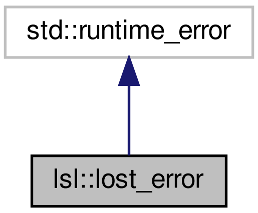
Exception class that indicates that a stream inlet's source has been irrecoverably lost.

```
#include <lsl_cpp.h>
```

Inheritance diagram for lsl::lost_error:



Collaboration diagram for lsl::lost_error:



Public Member Functions

- lost_error (const std::string &msg)

8.2.1 Detailed Description

Exception class that indicates that a stream inlet's source has been irrecoverably lost.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 lost_error()

```
lsl::lost_error::lost_error (
            const std::string & msg )  [inline], [explicit]
```

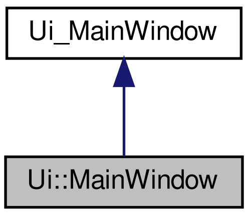
The documentation for this class was generated from the following file:

- include/lsl_cpp.h

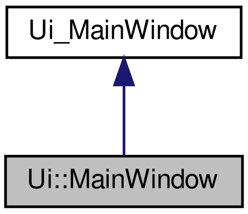
8.3 Ui::MainWindow Class Reference

```
#include <ui_mainwindow.h>
```

Inheritance diagram for Ui::MainWindow:



Collaboration diagram for Ui::MainWindow:



Additional Inherited Members

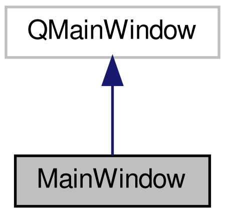
The documentation for this class was generated from the following file:

- OTBconfigGUI/build/ui_mainwindow.h

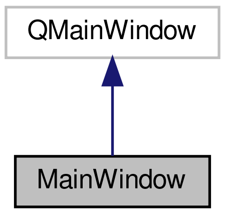
8.4 MainWindow Class Reference

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Public Member Functions

- MainWindow (QWidget ∗parent=nullptr)
- ∼MainWindow ()
- unsigned char crc (unsigned char config[ ])

8.4.1 Constructor & Destructor Documentation

8.4.1.1 MainWindow()

```
MainWindow::MainWindow (
            QWidget ∗ parent = nullptr )  [explicit]
```

8.4.1.2 ∼MainWindow()

```
MainWindow::∼MainWindow ( )
```

8.4.2 Member Function Documentation

8.4.2.1 crc()

```
unsigned char MainWindow::crc (
            unsigned char config[ ] )
```

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

- OTBconfigGUI/include/mainwindow.h
- OTBconfigGUI/src/mainwindow.cpp

8.5 qt_meta_stringdata_MainWindow_t Struct Reference

Public Attributes

- QByteArrayData data [7]
- char stringdata0 [75]

8.5.1 Member Data Documentation

8.5.1.1 data

```
QByteArrayData qt_meta_stringdata_MainWindow_t::data[7]
```

8.5.1.2 stringdata0

```
char qt_meta_stringdata_MainWindow_t::stringdata0[75]
```

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

- OTBconfigGUI/build/moc_mainwindow.cpp

8.6 lsl::stream_info Class Reference

```
#include <lsl_cpp.h>
```

Public Member Functions

- stream_info (const std::string &name, const std::string &type, int32_t channel_count=1, double nominal_↩ srate=IRREGULAR_RATE, channel_format_t channel_format=cf_float32, const std::string &source_id=std↩ ::string())
- stream_info (lsl_streaminfo handle)
- std::string name () const
- std::string type () const
- int32_t channel_count () const
- double nominal_srate () const
- channel_format_t channel_format () const
- std::string source_id () const
- int32_t version () const
- double created_at () const
- std::string uid () const
- std::string session_id () const
- std::string hostname () const

- xml_element desc ()
- bool matches_query (const char ∗query) const
- std::string as_xml () const
- int32_t channel_bytes () const

  Number of bytes occupied by a channel (0 for string-typed channels).
- int32_t sample_bytes () const

  Number of bytes occupied by a sample (0 for string-typed channels).
- lsl_streaminfo handle () const

  Get the implementation handle.
- stream_info ()

  Default contructor.
- stream_info (const stream_info &rhs)

  Copy constructor.
- stream_info & operator= (const stream_info &rhs)

  Assignment operator.
- ∼stream_info ()

  Destructor.

Static Public Member Functions

- static stream_info from_xml (const std::string &xml)

  Utility function to create a stream_info from an XML representation.

8.6.1 Constructor & Destructor Documentation

8.6.1.1 stream_info() [1/4]

```
lsl::stream_info::stream_info (
             const std::string & name,
             const std::string & type,
             int32_t channel_count = 1,
             double nominal_srate = IRREGULAR_RATE,
             channel_format_t channel_format = cf_float32,
             const std::string & source_id = std::string() )  [inline]
```

Construct a new stream_info object. Core stream information is specified here. Any remaining meta-data can be added later.

Parameters

name	Name of the stream. Describes the device (or product series) that this stream makes available (for use by programs, experimenters or data analysts). Cannot be empty.
type	Content type of the stream. Please see `https://github.com/sccn/xdf/wiki/Meta-Data` (or web search for: XDF meta-data) for pre-defined content-type names, but you can also make up your own. The content type is the preferred way to find streams (as opposed to searching by name).
channel_count	Number of channels per sample. This stays constant for the lifetime of the stream.

Parameters

nominal_srate	The sampling rate (in Hz) as advertised by the data source, if regular (otherwise set to IRREGULAR_RATE).
channel_format	Format/type of each channel. If your channels have different formats, consider supplying multiple streams or use the largest type that can hold them all (such as cf_double64).
source_id	Unique identifier of the device or source of the data, if available (such as the serial number). This is critical for system robustness since it allows recipients to recover from failure even after the serving app, device or computer crashes (just by finding a stream with the same source id on the network again). Therefore, it is highly recommended to always try to provide whatever information can uniquely identify the data source itself.

8.6.1.2 stream_info() [2/4]

```
lsl::stream_info::stream_info (
            lsl_streaminfo handle )  [inline]
```

8.6.1.3 stream_info() [3/4]

```
lsl::stream_info::stream_info ( )  [inline]
```

Default contructor.

8.6.1.4 stream_info() [4/4]

```
lsl::stream_info::stream_info (
            const stream_info & rhs )  [inline]
```

Copy constructor.

8.6.1.5 ∼stream_info()

```
lsl::stream_info::∼stream_info ( )  [inline]
```

Destructor.

8.6.2 Member Function Documentation

8.6.2.1 as_xml()

```
std::string lsl::stream_info::as_xml ( ) const  [inline]
```

Retrieve the entire stream_info in XML format. This yields an XML document (in string form) whose top-level element is <info>. The info element contains one element for each field of the stream_info class, including: a) the core elements <name>, <type>, <channel_count>, <nominal_srate>, <channel_format>, <source_id> b) the misc elements <version>, <created_at>, <uid>, <session_id>, <v4address>, <v4data_port>, <v4service←_port>, <v6address>, <v6data_port>, <v6service_port> c) the extended description element <desc> with user-defined sub-elements.

8.6.2.2 channel_bytes()

```
int32_t lsl::stream_info::channel_bytes ( ) const  [inline]
```

Number of bytes occupied by a channel (0 for string-typed channels).

8.6.2.3 channel_count()

```
int32_t lsl::stream_info::channel_count ( ) const  [inline]
```

Number of channels of the stream. A stream has at least one channel; the channel count stays constant for all samples.

8.6.2.4 channel_format()

```
channel_format_t lsl::stream_info::channel_format ( ) const  [inline]
```

Channel format of the stream. All channels in a stream have the same format. However, a device might offer multiple time-synched streams each with its own format.

8.6.2.5 created_at()

```
double lsl::stream_info::created_at ( ) const  [inline]
```

Creation time stamp of the stream. This is the time stamp when the stream was first created (as determined via local_clock() on the providing machine).

8.6.2.6 desc()

```
xml_element lsl::stream_info::desc ( )  [inline]
```

Extended description of the stream. It is highly recommended that at least the channel labels are described here. See code examples on the LSL wiki. Other information, such as amplifier settings, measurement units if deviating from defaults, setup information, subject information, etc., can be specified here, as well. Meta-data recommendations follow the XDF file format project (github.com/sccn/xdf/wiki/Meta-Data or web search for: XDF meta-data).

Important: if you use a stream content type for which meta-data recommendations exist, please try to lay out your meta-data in agreement with these recommendations for compatibility with other applications.

8.6.2.7 from_xml()

```
static stream_info lsl::stream_info::from_xml (
            const std::string & xml )  [inline], [static]
```

Utility function to create a stream_info from an XML representation.

8.6.2.8 handle()

```
lsl_streaminfo lsl::stream_info::handle ( ) const  [inline]
```

Get the implementation handle.

8.6.2.9 hostname()

```
std::string lsl::stream_info::hostname ( ) const  [inline]
```

Hostname of the providing machine.

8.6.2.10 matches_query()

```
bool lsl::stream_info::matches_query (
            const char * query ) const  [inline]
```

Tries to match the stream info XML element `info` against an `XPath` query.

Example query strings:

```
channel_count>5 and type='EEG'
type='TestStream' or contains(name,'Brain')
name='ExampleStream'
```

8.6.2.11 name()

```
std::string lsl::stream_info::name ( ) const  [inline]
```

Name of the stream. This is a human-readable name. For streams offered by device modules, it refers to the type of device or product series that is generating the data of the stream. If the source is an application, the name may be a more generic or specific identifier. Multiple streams with the same name can coexist, though potentially at the cost of ambiguity (for the recording app or experimenter).

8.6.2.12 nominal_srate()

```
double lsl::stream_info::nominal_srate ( ) const  [inline]
```

Sampling rate of the stream, according to the source (in Hz). If a stream is irregularly sampled, this should be set to IRREGULAR_RATE.

Note that no data will be lost even if this sampling rate is incorrect or if a device has temporary hiccups, since all samples will be recorded anyway (except for those dropped by the device itself). However, when the recording is imported into an application, a good importer may correct such errors more accurately if the advertised sampling rate was close to the specs of the device.

8.6.2.13 operator=()

```
stream_info& lsl::stream_info::operator= (
             const stream_info & rhs )  [inline]
```

Assignment operator.

8.6.2.14 sample_bytes()

```
int32_t lsl::stream_info::sample_bytes ( ) const  [inline]
```

Number of bytes occupied by a sample (0 for string-typed channels).

8.6.2.15 session_id()

```
std::string lsl::stream_info::session_id ( ) const  [inline]
```

Session ID for the given stream. The session id is an optional human-assigned identifier of the recording session. While it is rarely used, it can be used to prevent concurrent recording activitites on the same sub-network (e.g., in multiple experiment areas) from seeing each other's streams (assigned via a configuration file by the experimenter, see Network Connectivity in the LSL wiki).

8.6.2.16 source_id()

```
std::string lsl::stream_info::source_id ( ) const  [inline]
```

Unique identifier of the stream's source, if available. The unique source (or device) identifier is an optional piece of information that, if available, allows that endpoints (such as the recording program) can re-acquire a stream automatically once it is back online.

8.6.2.17 type()

```
std::string lsl::stream_info::type ( ) const  [inline]
```

Content type of the stream. The content type is a short string such as "EEG", "Gaze" which describes the content carried by the channel (if known). If a stream contains mixed content this value need not be assigned but may instead be stored in the description of channel types. To be useful to applications and automated processing systems using the recommended content types is preferred. Content types usually follow those pre-defined in `https://github.com/sccn/xdf/wiki/Meta-Data` (or web search for: XDF meta-data).

8.6.2.18 uid()

```
std::string lsl::stream_info::uid ( ) const  [inline]
```

Unique ID of the stream outlet instance (once assigned). This is a unique identifier of the stream outlet, and is guaranteed to be different across multiple instantiations of the same outlet (e.g., after a re-start).

8.6.2.19 version()

```
int32_t lsl::stream_info::version ( ) const  [inline]
```

Protocol version used to deliver the stream.

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

- include/lsl_cpp.h

8.7 lsl::stream_inlet Class Reference

```
#include <lsl_cpp.h>
```

Public Member Functions

- stream_inlet (const stream_info &info, int32_t max_buflen=360, int32_t max_chunklen=0, bool recover=true)
- ∼stream_inlet ()
- stream_info info (double timeout=FOREVER)
- void open_stream (double timeout=FOREVER)
- void close_stream ()
- double time_correction (double timeout=FOREVER)
- double time_correction (double ∗remote_time, double ∗uncertainty, double timeout=FOREVER)
- void set_postprocessing (uint32_t flags=post_ALL)
- template<class T , int N>
  double pull_sample (T sample[N], double timeout=FOREVER)
- double pull_sample (std::vector< float > &sample, double timeout=FOREVER)
- double pull_sample (std::vector< double > &sample, double timeout=FOREVER)
- double pull_sample (std::vector< long > &sample, double timeout=FOREVER)
- double pull_sample (std::vector< int32_t > &sample, double timeout=FOREVER)
- double pull_sample (std::vector< int16_t > &sample, double timeout=FOREVER)
- double pull_sample (std::vector< char > &sample, double timeout=FOREVER)

- double pull_sample (std::vector< std::string > &sample, double timeout=FOREVER)
- double pull_sample (float ∗buffer, int32_t buffer_elements, double timeout=FOREVER)
- double pull_sample (double ∗buffer, int32_t buffer_elements, double timeout=FOREVER)
- double pull_sample (long ∗buffer, int32_t buffer_elements, double timeout=FOREVER)
- double pull_sample (int32_t ∗buffer, int32_t buffer_elements, double timeout=FOREVER)
- double pull_sample (int16_t ∗buffer, int32_t buffer_elements, double timeout=FOREVER)
- double pull_sample (char ∗buffer, int32_t buffer_elements, double timeout=FOREVER)
- double pull_sample (std::string ∗buffer, int32_t buffer_elements, double timeout=FOREVER)
- template<class T >
  double pull_numeric_struct (T &sample, double timeout=FOREVER)
- double pull_numeric_raw (void ∗sample, int32_t buffer_bytes, double timeout=FOREVER)
- template<class T >
  bool pull_chunk (std::vector< std::vector< T > > &chunk, std::vector< double > &timestamps)
- template<class T >
  double pull_chunk (std::vector< std::vector< T > > &chunk)
- template<class T >
  std::vector< std::vector< T > > pull_chunk ()
- std::size_t pull_chunk_multiplexed (float ∗data_buffer, double ∗timestamp_buffer, std::size_t data_buffer_↩
  elements, std::size_t timestamp_buffer_elements, double timeout=0.0)
- std::size_t pull_chunk_multiplexed (double ∗data_buffer, double ∗timestamp_buffer, std::size_t data_buffer↩
  _elements, std::size_t timestamp_buffer_elements, double timeout=0.0)
- std::size_t pull_chunk_multiplexed (long ∗data_buffer, double ∗timestamp_buffer, std::size_t data_buffer_↩
  elements, std::size_t timestamp_buffer_elements, double timeout=0.0)
- std::size_t pull_chunk_multiplexed (int32_t ∗data_buffer, double ∗timestamp_buffer, std::size_t data_buffer↩
  _elements, std::size_t timestamp_buffer_elements, double timeout=0.0)
- std::size_t pull_chunk_multiplexed (int16_t ∗data_buffer, double ∗timestamp_buffer, std::size_t data_buffer↩
  _elements, std::size_t timestamp_buffer_elements, double timeout=0.0)
- std::size_t pull_chunk_multiplexed (char ∗data_buffer, double ∗timestamp_buffer, std::size_t data_buffer_↩
  elements, std::size_t timestamp_buffer_elements, double timeout=0.0)
- std::size_t pull_chunk_multiplexed (std::string ∗data_buffer, double ∗timestamp_buffer, std::size_t data_↩
  buffer_elements, std::size_t timestamp_buffer_elements, double timeout=0.0)
- template<typename T >
  bool pull_chunk_multiplexed (std::vector< T > &chunk, std::vector< double > ∗timestamps=nullptr, double timeout=0.0, bool append=false)

  pull_chunk_multiplexed Pull a multiplexed chunk of samples and optionally the sample timestamps from the inlet.
- template<class T >
  bool pull_chunk_numeric_structs (std::vector< T > &chunk, std::vector< double > &timestamps)
- template<class T >
  double pull_chunk_numeric_structs (std::vector< T > &chunk)
- template<class T >
  std::vector< T > pull_chunk_numeric_structs ()
- std::size_t samples_available ()
- bool was_clock_reset ()
- void smoothing_halftime (float value)
- int get_channel_count () const

8.7.1 Constructor & Destructor Documentation

8.7.1.1 stream_inlet()

```
lsl::stream_inlet::stream_inlet (
            const stream_info & info,
            int32_t max_buflen = 360,
            int32_t max_chunklen = 0,
            bool recover = true )  [inline]
```

Construct a new stream inlet from a resolved stream info.

Parameters

info	A resolved stream info object (as coming from one of the resolver functions). Note: the stream_inlet may also be constructed with a fully-specified stream_info, if the desired channel format and count is already known up-front, but this is strongly discouraged and should only ever be done if there is no time to resolve the stream up-front (e.g., due to limitations in the client program).
max_buflen	Optionally the maximum amount of data to buffer (in seconds if there is a nominal sampling rate, otherwise x100 in samples). Recording applications want to use a fairly large buffer size here, while real-time applications would only buffer as much as they need to perform their next calculation.
max_chunklen	Optionally the maximum size, in samples, at which chunks are transmitted (the default corresponds to the chunk sizes used by the sender). Recording applications can use a generous size here (leaving it to the network how to pack things), while real-time applications may want a finer (perhaps 1-sample) granularity. If left unspecified (=0), the sender determines the chunk granularity.
recover	Try to silently recover lost streams that are recoverable (=those that that have a source_id set). In all other cases (recover is false or the stream is not recoverable) functions may throw a lost_error if the stream's source is lost (e.g., due to an app or computer crash).

8.7.1.2 ∼stream_inlet()

```
lsl::stream_inlet::∼stream_inlet ( )  [inline]
```

Destructor. The inlet will automatically disconnect if destroyed.

8.7.2 Member Function Documentation

8.7.2.1 close_stream()

```
void lsl::stream_inlet::close_stream ( )  [inline]
```

Drop the current data stream. All samples that are still buffered or in flight will be dropped and transmission and buffering of data for this inlet will be stopped. If an application stops being interested in data from a source (temporarily or not) but keeps the outlet alive, it should call close_stream() to not waste unnecessary system and network resources.

8.7.2.2 get_channel_count()

```
int lsl::stream_inlet::get_channel_count ( ) const  [inline]
```

8.7.2.3 info()

```
stream_info lsl::stream_inlet::info (
             double timeout = FOREVER )  [inline]
```

Retrieve the complete information of the given stream, including the extended description. Can be invoked at any time of the stream's lifetime.

Parameters

timeout	Timeout of the operation (default: no timeout).

Exceptions

timeout_error	(if the timeout expires), or lost_error (if the stream source has been lost).

8.7.2.4 open_stream()

```
void lsl::stream_inlet::open_stream (
             double timeout = FOREVER )  [inline]
```

Subscribe to the data stream. All samples pushed in at the other end from this moment onwards will be queued and eventually be delivered in response to pull_sample() or pull_chunk() calls. Pulling a sample without some preceding open_stream is permitted (the stream will then be opened implicitly).

Parameters

timeout	Optional timeout of the operation (default: no timeout).

Exceptions

timeout_error	(if the timeout expires), or lost_error (if the stream source has been lost).

8.7.2.5 pull_chunk() [1/3]

```
template<class T >
bool lsl::stream_inlet::pull_chunk (
             std::vector< std::vector< T > > & chunk,
             std::vector< double > & timestamps )  [inline]
```

Pull a chunk of samples from the inlet. This is the most complete version, returning both the data and a timestamp for each sample.

Parameters

chunk	A vector of vectors to hold the samples.
timestamps	A vector to hold the time stamps.

Returns

True if some data was obtained.

Exceptions

lost_error	(if the stream source has been lost).

8.7.2.6 pull_chunk() [2/3]

```
template<class T >
double lsl::stream_inlet::pull_chunk (
             std::vector< std::vector< T > > & chunk )  [inline]
```

Pull a chunk of samples from the inlet. This version returns only the most recent sample's time stamp.

Parameters

chunk	A vector of vectors to hold the samples.

Returns

The time when the most recent sample was captured on the remote machine, or 0.0 if no new sample was available.

Exceptions

lost_error	(if the stream source has been lost)

8.7.2.7 pull_chunk() [3/3]

```
template<class T >
std::vector<std::vector<T> > lsl::stream_inlet::pull_chunk ( )  [inline]
```

Pull a chunk of samples from the inlet. This function does not return time stamps for the samples. Invoked as: mychunk = pull_chunk<float>();

Returns

A vector of vectors containing the obtained samples; may be empty.

Exceptions

lost_error	(if the stream source has been lost)

8.7.2.8 pull_chunk_multiplexed() [1/8]

```
std::size_t lsl::stream_inlet::pull_chunk_multiplexed (
             float * data_buffer,
             double * timestamp_buffer,
             std::size_t data_buffer_elements,
             std::size_t timestamp_buffer_elements,
             double timeout = 0.0 )  [inline]
```

Pull a chunk of data from the inlet into a pre-allocated buffer. This is a high-performance function that performs no memory allocations (useful for very high data rates or on low-powered devices). IMPORTANT: Note that the provided data buffer size is measured in channel values (e.g., floats) rather than in samples.

Parameters

data_buffer	A pointer to a buffer of data values where the results shall be stored.
timestamp_buffer	A pointer to a buffer of timestamp values where time stamps shall be stored. If this is NULL, no time stamps will be returned.
data_buffer_elements	The size of the data buffer, in channel data elements (of type T). Must be a multiple of the stream's channel count.
timestamp_buffer_elements	The size of the timestamp buffer. If a timestamp buffer is provided then this must correspond to the same number of samples as data_buffer_elements.
timeout	The timeout for this operation, if any. When the timeout expires, the function may return before the entire buffer is filled. The default value of 0.0 will retrieve only data available for immediate pickup.

Returns

data_elements_written Number of channel data elements written to the data buffer.

Exceptions

lost_error	(if the stream source has been lost).

8.7.2.9 pull_chunk_multiplexed() [2/8]

```
std::size_t lsl::stream_inlet::pull_chunk_multiplexed (
             double * data_buffer,
             double * timestamp_buffer,
             std::size_t data_buffer_elements,
             std::size_t timestamp_buffer_elements,
             double timeout = 0.0 )  [inline]
```

8.7.2.10 pull_chunk_multiplexed() [3/8]

```
std::size_t lsl::stream_inlet::pull_chunk_multiplexed (
             long * data_buffer,
             double * timestamp_buffer,
             std::size_t data_buffer_elements,
             std::size_t timestamp_buffer_elements,
             double timeout = 0.0 )  [inline]
```

8.7.2.11 pull_chunk_multiplexed() [4/8]

```
std::size_t lsl::stream_inlet::pull_chunk_multiplexed (
             int32_t * data_buffer,
             double * timestamp_buffer,
             std::size_t data_buffer_elements,
             std::size_t timestamp_buffer_elements,
             double timeout = 0.0 )  [inline]
```

8.7.2.12 pull_chunk_multiplexed() [5/8]

```
std::size_t lsl::stream_inlet::pull_chunk_multiplexed (
             int16_t * data_buffer,
             double * timestamp_buffer,
             std::size_t data_buffer_elements,
             std::size_t timestamp_buffer_elements,
             double timeout = 0.0 )  [inline]
```

8.7.2.13 pull_chunk_multiplexed() [6/8]

```
std::size_t lsl::stream_inlet::pull_chunk_multiplexed (
             char * data_buffer,
             double * timestamp_buffer,
             std::size_t data_buffer_elements,
             std::size_t timestamp_buffer_elements,
             double timeout = 0.0 )  [inline]
```

8.7.2.14 pull_chunk_multiplexed() [7/8]

```
std::size_t lsl::stream_inlet::pull_chunk_multiplexed (
            std::string * data_buffer,
            double * timestamp_buffer,
            std::size_t data_buffer_elements,
            std::size_t timestamp_buffer_elements,
            double timeout = 0.0 ) [inline]
```

8.7.2.15 pull_chunk_multiplexed() [8/8]

```
template<typename T >
bool lsl::stream_inlet::pull_chunk_multiplexed (
            std::vector< T > & chunk,
            std::vector< double > * timestamps = nullptr,
            double timeout = 0.0,
            bool append = false ) [inline]
```

pull_chunk_multiplexed Pull a multiplexed chunk of samples and optionally the sample timestamps from the inlet.

Parameters

chunk	A vector to hold the multiplexed (Sample 1 Channel 1, S1C2, S2C1, S2C2, S3C1, S3C2, ...) samples
timestamps	A vector to hold the timestamps or nullptr
timeout	Time to wait for the first sample. The default value of 0.0 will not wait for data to arrive, pulling only samples already received.
append	(True:) Append data or (false:) clear them first

Returns

True if some data was obtained.

Exceptions

lost_error	(if the stream source has been lost).

8.7.2.16 pull_chunk_numeric_structs() [1/3]

```
template<class T >
bool lsl::stream_inlet::pull_chunk_numeric_structs (
            std::vector< T > & chunk,
            std::vector< double > & timestamps ) [inline]
```

Pull a chunk of samples from the inlet. This is the most complete version, returning both the data and a timestamp for each sample.

Parameters

chunk	A vector of C-style structs to hold the samples.
timestamps	A vector to hold the time stamps.

Returns

True if some data was obtained.

Exceptions

lost_error	(if the stream source has been lost)

8.7.2.17 pull_chunk_numeric_structs() [2/3]

```
template<class T >
double lsl::stream_inlet::pull_chunk_numeric_structs (
             std::vector< T > & chunk )  [inline]
```

Pull a chunk of samples from the inlet. This version returns only the most recent sample's time stamp.

Parameters

chunk	A vector of C-style structs to hold the samples.

Returns

The time when the most recent sample was captured on the remote machine, or 0.0 if no new sample was available.

Exceptions

lost_error	(if the stream source has been lost)

8.7.2.18 pull_chunk_numeric_structs() [3/3]

```
template<class T >
std::vector<T> lsl::stream_inlet::pull_chunk_numeric_structs ( )  [inline]
```

Pull a chunk of samples from the inlet. This function does not return time stamps. Invoked as: mychunk = pull_↩
chunk<mystruct>();

Returns

A vector of C-style structs containing the obtained samples; may be empty.

Exceptions

lost_error	(if the stream source has been lost)

8.7.2.19 pull_numeric_raw()

```
double lsl::stream_inlet::pull_numeric_raw (
              void * sample,
              int32_t buffer_bytes,
              double timeout = FOREVER )  [inline]
```

Pull a sample from the inlet and read it into a pointer to raw data. No type checking or conversions are done (not recommended!). Do not use for variable-size/string-formatted streams.

Parameters

buffer	A pointer to hold the resulting raw sample data.
buffer_bytes	The number of bytes allocated in the buffer. Note: it is the responsibility of the user to allocate enough memory.
timeout	The timeout for this operation, if any. Use 0.0 to make the function non-blocking.

Returns

The capture time of the sample on the remote machine, or 0.0 if no new sample was available. To remap this time stamp to the local clock, add the value returned by .time_correction() to it.

Exceptions

lost_error	(if the stream source has been lost).

8.7.2.20 pull_numeric_struct()

```
template<class T >
double lsl::stream_inlet::pull_numeric_struct (
              T & sample,
              double timeout = FOREVER )  [inline]
```

Pull a sample from the inlet and read it into a custom C-style struct. Overall size checking but no type checking or conversion are done. Do not use for variable-size/string-formatted streams.

Parameters

sample	The raw sample object to hold the data (packed C-style struct). Search for #pragma pack for information on how to pack structs correctly.
timeout	The timeout for this operation, if any. Use 0.0 to make the function non-blocking.

Returns

The capture time of the sample on the remote machine, or 0.0 if no new sample was available. To remap this time stamp to the local clock, add the value returned by .time_correction() to it.

Exceptions

lost_error	(if the stream source has been lost).

8.7.2.21 pull_sample() [1/15]

```
template<class T , int N>
double lsl::stream_inlet::pull_sample (
             T sample[N],
             double timeout = FOREVER )  [inline]
```

Pull a sample from the inlet and read it into an array of values. Handles type checking & conversion.

Parameters

sample	An array to hold the resulting values.
timeout	The timeout for this operation, if any. Use 0.0 to make the function non-blocking.

Returns

The capture time of the sample on the remote machine, or 0.0 if no new sample was available. To remap this time stamp to the local clock, add the value returned by .time_correction() to it.

Exceptions

lost_error	(if the stream source has been lost).

8.7.2.22 pull_sample() [2/15]

```
double lsl::stream_inlet::pull_sample (
             std::vector< float > & sample,
             double timeout = FOREVER )  [inline]
```

Pull a sample from the inlet and read it into a std vector of values. Handles type checking & conversion and allocates the necessary memory in the vector if necessary.

Parameters

sample	An STL vector to hold the resulting values.
timeout	The timeout for this operation, if any. Use 0.0 to make the function non-blocking.

Returns

The capture time of the sample on the remote machine, or 0.0 if no new sample was available. To remap this time stamp to the local clock, add the value returned by .time_correction() to it.

Exceptions

lost_error	(if the stream source has been lost).

8.7.2.23 pull_sample() [3/15]

```
double lsl::stream_inlet::pull_sample (
            std::vector< double > & sample,
            double timeout = FOREVER )  [inline]
```

8.7.2.24 pull_sample() [4/15]

```
double lsl::stream_inlet::pull_sample (
            std::vector< long > & sample,
            double timeout = FOREVER )  [inline]
```

8.7.2.25 pull_sample() [5/15]

```
double lsl::stream_inlet::pull_sample (
            std::vector< int32_t > & sample,
            double timeout = FOREVER )  [inline]
```

8.7.2.26 pull_sample() [6/15]

```
double lsl::stream_inlet::pull_sample (
            std::vector< int16_t > & sample,
            double timeout = FOREVER )  [inline]
```

8.7.2.27 pull_sample() [7/15]

```
double lsl::stream_inlet::pull_sample (
            std::vector< char > & sample,
            double timeout = FOREVER )  [inline]
```

8.7.2.28 pull_sample() [8/15]

```
double lsl::stream_inlet::pull_sample (
            std::vector< std::string > & sample,
            double timeout = FOREVER )   [inline]
```

8.7.2.29 pull_sample() [9/15]

```
double lsl::stream_inlet::pull_sample (
            float * buffer,
            int32_t buffer_elements,
            double timeout = FOREVER )   [inline]
```

Pull a sample from the inlet and read it into a pointer to values. Handles type checking & conversion.

Parameters

buffer	A pointer to hold the resulting values.
buffer_elements	The number of samples allocated in the buffer. Note: it is the responsibility of the user to allocate enough memory.
timeout	The timeout for this operation, if any. Use 0.0 to make the function non-blocking.

Returns

The capture time of the sample on the remote machine, or 0.0 if no new sample was available. To remap this time stamp to the local clock, add the value returned by .time_correction() to it.

Exceptions

lost_error	(if the stream source has been lost).

8.7.2.30 pull_sample() [10/15]

```
double lsl::stream_inlet::pull_sample (
            double * buffer,
            int32_t buffer_elements,
            double timeout = FOREVER )   [inline]
```

8.7.2.31 pull_sample() [11/15]

```
double lsl::stream_inlet::pull_sample (
            long * buffer,
            int32_t buffer_elements,
            double timeout = FOREVER )   [inline]
```

8.7.2.32 pull_sample() [12/15]

```
double lsl::stream_inlet::pull_sample (
            int32_t * buffer,
            int32_t buffer_elements,
            double timeout = FOREVER )  [inline]
```

8.7.2.33 pull_sample() [13/15]

```
double lsl::stream_inlet::pull_sample (
            int16_t * buffer,
            int32_t buffer_elements,
            double timeout = FOREVER )  [inline]
```

8.7.2.34 pull_sample() [14/15]

```
double lsl::stream_inlet::pull_sample (
            char * buffer,
            int32_t buffer_elements,
            double timeout = FOREVER )  [inline]
```

8.7.2.35 pull_sample() [15/15]

```
double lsl::stream_inlet::pull_sample (
            std::string * buffer,
            int32_t buffer_elements,
            double timeout = FOREVER )  [inline]
```

8.7.2.36 samples_available()

```
std::size_t lsl::stream_inlet::samples_available ( )  [inline]
```

Query whether samples are currently available for immediate pickup. Note that it is not a good idea to use samples←_available() to determine whether a pull_∗() call would block: to be sure, set the pull timeout to 0.0 or an acceptably low value. If the underlying implementation supports it, the value will be the number of samples available (otherwise it will be 1 or 0).

8.7.2.37 set_postprocessing()

```
void lsl::stream_inlet::set_postprocessing (
            uint32_t flags = post_ALL )  [inline]
```

Set post-processing flags to use. By default, the inlet performs NO post-processing and returns the ground-truth time stamps, which can then be manually synchronized using time_correction(), and then smoothed/dejittered if desired. This function allows automating these two and possibly more operations. Warning: when you enable this, you will no longer receive or be able to recover the original time stamps.

Parameters

flags	An integer that is the result of bitwise OR'ing one or more options from processing_options_t together (e.g., post_clocksync|post_dejitter); the default is to enable all options.

8.7.2.38 smoothing_halftime()

```
void lsl::stream_inlet::smoothing_halftime (
             float value )  [inline]
```

Override the half-time (forget factor) of the time-stamp smoothing. The default is 90 seconds unless a different value is set in the config file. Using a longer window will yield lower jitter in the time stamps, but longer windows will have trouble tracking changes in the clock rate (usually due to temperature changes); the default is able to track changes up to 10 degrees C per minute sufficiently well.

8.7.2.39 time_correction() [1/2]

```
double lsl::stream_inlet::time_correction (
             double timeout = FOREVER )  [inline]
```

Retrieve an estimated time correction offset for the given stream. The first call to this function takes several milliseconds until a reliable first estimate is obtained. Subsequent calls are instantaneous (and rely on periodic background updates). On a well-behaved network, the precision of these estimates should be below 1 ms (empirically it is within +/-0.2 ms). To get a measure of whether the network is well-behaved, use the extended prototype and check uncertainty (which maps to round-trip-time). 0.2 ms is typical of wired networks. 2 ms is typical of wireless networks. The number can be much higher on poor networks.

Parameters

remote_time	The current time of the remote computer that was used to generate this time_correction. If desired, the client can fit time_correction vs remote_time to improve the real-time time_correction further.
uncertainty.	The maximum uncertainty of the given time correction. Timeout to acquire the first time-correction estimate (default: no timeout).

Returns

The time correction estimate. This is the number that needs to be added to a time stamp that was remotely generated via lsl_local_clock() to map it into the local clock domain of this machine.

Exceptions

timeout_error	(if the timeout expires), or lost_error (if the stream source has been lost).

8.7.2.40 time_correction() [2/2]

```
double lsl::stream_inlet::time_correction (
            double * remote_time,
            double * uncertainty,
            double timeout = FOREVER )  [inline]
```

8.7.2.41 was_clock_reset()

```
bool lsl::stream_inlet::was_clock_reset ( )  [inline]
```

Query whether the clock was potentially reset since the last call to was_clock_reset(). This is a rarely-used function that is only useful to applications that combine multiple time_correction values to estimate precise clock drift; it allows to tolerate cases where the source machine was hot-swapped or restarted in between two measurements.

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

- include/lsl_cpp.h

8.8 lsl::stream_outlet Class Reference

```
#include <lsl_cpp.h>
```

Public Member Functions

- stream_outlet (const stream_info &info, int32_t chunk_size=0, int32_t max_buffered=360)
- template<class T , int32_t N>
  void push_sample (const T data[N], double timestamp=0.0, bool pushthrough=true)
- void push_sample (const std::vector< float > &data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const std::vector< double > &data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const std::vector< long > &data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const std::vector< int32_t > &data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const std::vector< int16_t > &data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const std::vector< char > &data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const std::vector< std::string > &data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const float *data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const double *data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const long *data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const int32_t *data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const int16_t *data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const char *data, double timestamp=0.0, bool pushthrough=true)
- void push_sample (const std::string *data, double timestamp=0.0, bool pushthrough=true)
- template<class T >
  void push_numeric_struct (const T &sample, double timestamp=0.0, bool pushthrough=true)
- void push_numeric_raw (const void *sample, double timestamp=0.0, bool pushthrough=true)
- template<class T >
  void push_chunk (const std::vector< T > &samples, double timestamp=0.0, bool pushthrough=true)

- template<class T >
  void push_chunk (const std::vector< T > &samples, const std::vector< double > &timestamps, bool pushthrough=true)
- template<class T >
  void push_chunk_numeric_structs (const std::vector< T > &samples, double timestamp=0.0, bool pushthrough=true)
- template<class T >
  void push_chunk_numeric_structs (const std::vector< T > &samples, const std::vector< double > &times-tamps, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< float > &buffer, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< double > &buffer, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< long > &buffer, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< int32_t > &buffer, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< int16_t > &buffer, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< char > &buffer, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< std::string > &buffer, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< float > &buffer, const std::vector< double > &timestamps, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< double > &buffer, const std::vector< double > &times-tamps, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< long > &buffer, const std::vector< double > &timestamps, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< int32_t > &buffer, const std::vector< double > &times-tamps, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< int16_t > &buffer, const std::vector< double > &times-tamps, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< char > &buffer, const std::vector< double > &timestamps, bool pushthrough=true)
- void push_chunk_multiplexed (const std::vector< std::string > &buffer, const std::vector< double > &times-tamps, bool pushthrough=true)
- void push_chunk_multiplexed (const float ∗buffer, std::size_t buffer_elements, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const double ∗buffer, std::size_t buffer_elements, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const long ∗buffer, std::size_t buffer_elements, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const int32_t ∗buffer, std::size_t buffer_elements, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const int16_t ∗buffer, std::size_t buffer_elements, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const char ∗buffer, std::size_t buffer_elements, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const std::string ∗buffer, std::size_t buffer_elements, double timestamp=0.0, bool pushthrough=true)
- void push_chunk_multiplexed (const float ∗data_buffer, const double ∗timestamp_buffer, std::size_t data_↩ buffer_elements, bool pushthrough=true)
- void push_chunk_multiplexed (const double ∗data_buffer, const double ∗timestamp_buffer, std::size_t data↩ _buffer_elements, bool pushthrough=true)
- void push_chunk_multiplexed (const long ∗data_buffer, const double ∗timestamp_buffer, std::size_t data_↩ buffer_elements, bool pushthrough=true)
- void push_chunk_multiplexed (const int32_t ∗data_buffer, const double ∗timestamp_buffer, std::size_t data↩ _buffer_elements, bool pushthrough=true)

- void push_chunk_multiplexed (const int16_t ∗data_buffer, const double ∗timestamp_buffer, std::size_t data↩ _buffer_elements, bool pushthrough=true)
- void push_chunk_multiplexed (const char ∗data_buffer, const double ∗timestamp_buffer, std::size_t data_↩ buffer_elements, bool pushthrough=true)
- void push_chunk_multiplexed (const std::string ∗data_buffer, const double ∗timestamp_buffer, std::size_↩ t data_buffer_elements, bool pushthrough=true)
- bool have_consumers ()
- bool wait_for_consumers (double timeout)
- stream_info info () const
- ∼stream_outlet ()

8.8.1 Detailed Description

A stream outlet. Outlets are used to make streaming data (and the meta-data) available on the lab network.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 stream_outlet()

```
lsl::stream_outlet::stream_outlet (
            const stream_info & info,
            int32_t chunk_size = 0,
            int32_t max_buffered = 360 )  [inline]
```

Establish a new stream outlet. This makes the stream discoverable.

Parameters

info	The stream information to use for creating this stream. Stays constant over the lifetime of the outlet.
chunk_size	Optionally the desired chunk granularity (in samples) for transmission. If unspecified, each push operation yields one chunk. Inlets can override this setting.
max_buffered	Optionally the maximum amount of data to buffer (in seconds if there is a nominal sampling rate, otherwise x100 in samples). The default is 6 minutes of data.

8.8.2.2 ∼stream_outlet()

```
lsl::stream_outlet::∼stream_outlet ( )  [inline]
```

Destructor. The stream will no longer be discoverable after destruction and all paired inlets will stop delivering data.

8.8.3 Member Function Documentation

8.8.3.1 have_consumers()

```
bool lsl::stream_outlet::have_consumers ( )  [inline]
```

Check whether consumers are currently registered. While it does not hurt, there is technically no reason to push samples if there is no consumer.

8.8.3.2 info()

```
stream_info lsl::stream_outlet::info ( ) const  [inline]
```

Retrieve the stream info provided by this outlet. This is what was used to create the stream (and also has the Additional Network Information fields assigned).

8.8.3.3 push_chunk() [1/2]

```
template<class T >
void lsl::stream_outlet::push_chunk (
             const std::vector< T > & samples,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

Push a chunk of samples (batched into an STL vector) into the outlet.

Parameters

samples	A vector of samples in some supported format (each sample can be a data pointer, data array, or std vector of data).
timestamp	Optionally the capture time of the most recent sample, in agreement with local_clock(); if omitted, the current time is used. The time stamps of other samples are automatically derived according to the sampling rate of the stream.
pushthrough	Whether to push the chunk through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.4 push_chunk() [2/2]

```
template<class T >
void lsl::stream_outlet::push_chunk (
             const std::vector< T > & samples,
             const std::vector< double > & timestamps,
             bool pushthrough = true )  [inline]
```

Push a chunk of samples (batched into an STL vector) into the outlet. Allows to specify a separate time stamp for each sample (for irregular-rate streams).

Parameters

samples	A vector of samples in some supported format (each sample can be a data pointer, data array, or std vector of data).
timestamps	A vector of capture times for each sample, in agreement with local_clock().
pushthrough	Whether to push the chunk through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.5 push_chunk_multiplexed() [1/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
             const std::vector< float > & buffer,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

Push a chunk of multiplexed data into the outlet.

Parameters

buffer	A buffer of channel values holding the data for zero or more successive samples to send.
timestamp	Optionally the capture time of the most recent sample, in agreement with local_clock(); if omitted, the current time is used. The time stamps of other samples are automatically derived according to the sampling rate of the stream.
pushthrough	Whether to push the chunk through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.6 push_chunk_multiplexed() [2/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
             const std::vector< double > & buffer,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

8.8.3.7 push_chunk_multiplexed() [3/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
             const std::vector< long > & buffer,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

8.8.3.8 push_chunk_multiplexed() [4/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const std::vector< int32_t > & buffer,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.9 push_chunk_multiplexed() [5/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const std::vector< int16_t > & buffer,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.10 push_chunk_multiplexed() [6/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const std::vector< char > & buffer,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.11 push_chunk_multiplexed() [7/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const std::vector< std::string > & buffer,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.12 push_chunk_multiplexed() [8/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const std::vector< float > & buffer,
            const std::vector< double > & timestamps,
            bool pushthrough = true )  [inline]
```

Push a chunk of multiplexed data into the outlet. One timestamp per sample is provided. Allows to specify a separate time stamp for each sample (for irregular-rate streams).

Parameters

data_buffer	A buffer of channel values holding the data for zero or more successive samples to send.
timestamps	A buffer of timestamp values holding time stamps for each sample in the data buffer.
pushthrough	Whether to push the chunk through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.13 push_chunk_multiplexed() [9/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const std::vector< double > & buffer,
            const std::vector< double > & timestamps,
            bool pushthrough = true )  [inline]
```

8.8.3.14 push_chunk_multiplexed() [10/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const std::vector< long > & buffer,
            const std::vector< double > & timestamps,
            bool pushthrough = true )  [inline]
```

8.8.3.15 push_chunk_multiplexed() [11/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const std::vector< int32_t > & buffer,
            const std::vector< double > & timestamps,
            bool pushthrough = true )  [inline]
```

8.8.3.16 push_chunk_multiplexed() [12/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const std::vector< int16_t > & buffer,
            const std::vector< double > & timestamps,
            bool pushthrough = true )  [inline]
```

8.8.3.17 push_chunk_multiplexed() [13/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const std::vector< char > & buffer,
            const std::vector< double > & timestamps,
            bool pushthrough = true )  [inline]
```

8.8.3.18 push_chunk_multiplexed() [14/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const std::vector< std::string > & buffer,
            const std::vector< double > & timestamps,
            bool pushthrough = true ) [inline]
```

8.8.3.19 push_chunk_multiplexed() [15/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const float * buffer,
            std::size_t buffer_elements,
            double timestamp = 0.0,
            bool pushthrough = true ) [inline]
```

Push a chunk of multiplexed samples into the outlet. Single timestamp provided. IMPORTANT: Note that the provided buffer size is measured in channel values (e.g., floats) rather than in samples.

Parameters

buffer	A buffer of channel values holding the data for zero or more successive samples to send.
buffer_elements	The number of channel values (of type T) in the buffer. Must be a multiple of the channel count.
timestamp	Optionally the capture time of the most recent sample, in agreement with local_clock(); if omitted, the current time is used. The time stamps of other samples are automatically derived based on the sampling rate of the stream.
pushthrough	Whether to push the chunk through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.20 push_chunk_multiplexed() [16/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const double * buffer,
            std::size_t buffer_elements,
            double timestamp = 0.0,
            bool pushthrough = true ) [inline]
```

8.8.3.21 push_chunk_multiplexed() [17/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const long * buffer,
            std::size_t buffer_elements,
            double timestamp = 0.0,
            bool pushthrough = true ) [inline]
```

8.8.3.22 push_chunk_multiplexed() [18/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
             const int32_t * buffer,
             std::size_t buffer_elements,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

8.8.3.23 push_chunk_multiplexed() [19/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
             const int16_t * buffer,
             std::size_t buffer_elements,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

8.8.3.24 push_chunk_multiplexed() [20/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
             const char * buffer,
             std::size_t buffer_elements,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

8.8.3.25 push_chunk_multiplexed() [21/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
             const std::string * buffer,
             std::size_t buffer_elements,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

8.8.3.26 push_chunk_multiplexed() [22/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
             const float * data_buffer,
             const double * timestamp_buffer,
             std::size_t data_buffer_elements,
             bool pushthrough = true )  [inline]
```

Push a chunk of multiplexed samples into the outlet. One timestamp per sample is provided. IMPORTANT: Note that the provided buffer size is measured in channel values (e.g., floats) rather than in samples.

Parameters

data_buffer	A buffer of channel values holding the data for zero or more successive samples to send.
timestamp_buffer	A buffer of timestamp values holding time stamps for each sample in the data buffer.
data_buffer_elements	The number of data values (of type T) in the data buffer. Must be a multiple of the channel count.
pushthrough	Whether to push the chunk through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.27 push_chunk_multiplexed() [23/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const double * data_buffer,
            const double * timestamp_buffer,
            std::size_t data_buffer_elements,
            bool pushthrough = true ) [inline]
```

8.8.3.28 push_chunk_multiplexed() [24/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const long * data_buffer,
            const double * timestamp_buffer,
            std::size_t data_buffer_elements,
            bool pushthrough = true ) [inline]
```

8.8.3.29 push_chunk_multiplexed() [25/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const int32_t * data_buffer,
            const double * timestamp_buffer,
            std::size_t data_buffer_elements,
            bool pushthrough = true ) [inline]
```

8.8.3.30 push_chunk_multiplexed() [26/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
            const int16_t * data_buffer,
            const double * timestamp_buffer,
            std::size_t data_buffer_elements,
            bool pushthrough = true ) [inline]
```

8.8.3.31 push_chunk_multiplexed() [27/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
             const char * data_buffer,
             const double * timestamp_buffer,
             std::size_t data_buffer_elements,
             bool pushthrough = true )  [inline]
```

8.8.3.32 push_chunk_multiplexed() [28/28]

```
void lsl::stream_outlet::push_chunk_multiplexed (
             const std::string * data_buffer,
             const double * timestamp_buffer,
             std::size_t data_buffer_elements,
             bool pushthrough = true )  [inline]
```

8.8.3.33 push_chunk_numeric_structs() [1/2]

```
template<class T >
void lsl::stream_outlet::push_chunk_numeric_structs (
             const std::vector< T > & samples,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

Push a chunk of numeric data as C-style structs (batched into an STL vector) into the outlet. This performs some size checking but no type checking. Can not be used for variable-size / string-formatted data.

Parameters

samples	A vector of samples, as C structs.
timestamp	Optionally the capture time of the sample, in agreement with local_clock(); if omitted, the current time is used.
pushthrough	Whether to push the chunk through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.34 push_chunk_numeric_structs() [2/2]

```
template<class T >
void lsl::stream_outlet::push_chunk_numeric_structs (
             const std::vector< T > & samples,
             const std::vector< double > & timestamps,
             bool pushthrough = true )  [inline]
```

Push a chunk of numeric data from C-style structs (batched into an STL vector), into the outlet. This performs some size checking but no type checking. Can not be used for variable-size / string-formatted data.

Parameters

samples	A vector of samples, as C structs.
timestamps	A vector of capture times for each sample, in agreement with local_clock().
pushthrough	Whether to push the chunk through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.35 push_numeric_raw()

```
void lsl::stream_outlet::push_numeric_raw (
             const void * sample,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

Push a pointer to raw numeric data as one sample into the outlet. This is the lowest-level function; performns no checking whatsoever. Can not be used for variable-size / string-formatted channels.

Parameters

sample	A pointer to the raw sample data to push.
timestamp	Optionally the capture time of the sample, in agreement with local_clock(); if omitted, the current time is used.
pushthrough	Whether to push the sample through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.36 push_numeric_struct()

```
template<class T >
void lsl::stream_outlet::push_numeric_struct (
             const T & sample,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

Push a packed C struct (of numeric data) as one sample into the outlet (search for #pragma pack for information on packing structs appropriately). Overall size checking but no type checking or conversion are done. Dan not be used for variable-size / string-formatted data.

Parameters

sample	The sample struct to push.
timestamp	Optionally the capture time of the sample, in agreement with local_clock(); if omitted, the current time is used.
pushthrough	Whether to push the sample through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.37 push_sample() [1/15]

```
template<class T , int32_t N>
void lsl::stream_outlet::push_sample (
            const T data[N],
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

Push a C array of values as a sample into the outlet. Each entry in the array corresponds to one channel. The function handles type checking & conversion.

Parameters

data	An array of values to push (one per channel).
timestamp	Optionally the capture time of the sample, in agreement with local_clock(); if omitted, the current time is used.
pushthrough	Whether to push the sample through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.38 push_sample() [2/15]

```
void lsl::stream_outlet::push_sample (
            const std::vector< float > & data,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

Push a std vector of values as a sample into the outlet. Each entry in the vector corresponds to one channel. The function handles type checking & conversion.

Parameters

data	A vector of values to push (one for each channel).
timestamp	Optionally the capture time of the sample, in agreement with local_clock(); if omitted, the current time is used.
pushthrough	Whether to push the sample through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.39 push_sample() [3/15]

```
void lsl::stream_outlet::push_sample (
            const std::vector< double > & data,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.40 push_sample() [4/15]

```
void lsl::stream_outlet::push_sample (
            const std::vector< long > & data,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.41 push_sample() [5/15]

```
void lsl::stream_outlet::push_sample (
            const std::vector< int32_t > & data,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.42 push_sample() [6/15]

```
void lsl::stream_outlet::push_sample (
            const std::vector< int16_t > & data,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.43 push_sample() [7/15]

```
void lsl::stream_outlet::push_sample (
            const std::vector< char > & data,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.44 push_sample() [8/15]

```
void lsl::stream_outlet::push_sample (
            const std::vector< std::string > & data,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.45 push_sample() [9/15]

```
void lsl::stream_outlet::push_sample (
            const float * data,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

Push a pointer to some values as a sample into the outlet. This is a lower-level function for cases where data is available in some buffer. Handles type checking & conversion.

Parameters

data	A pointer to values to push. The number of values pointed to must not be less than the number of channels in the sample.
timestamp	Optionally the capture time of the sample, in agreement with local_clock(); if omitted, the current time is used.
pushthrough	Whether to push the sample through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

8.8.3.46 push_sample() [10/15]

```
void lsl::stream_outlet::push_sample (
             const double * data,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

8.8.3.47 push_sample() [11/15]

```
void lsl::stream_outlet::push_sample (
             const long * data,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

8.8.3.48 push_sample() [12/15]

```
void lsl::stream_outlet::push_sample (
             const int32_t * data,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

8.8.3.49 push_sample() [13/15]

```
void lsl::stream_outlet::push_sample (
             const int16_t * data,
             double timestamp = 0.0,
             bool pushthrough = true )  [inline]
```

8.8.3.50 push_sample() [14/15]

```
void lsl::stream_outlet::push_sample (
            const char * data,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.51 push_sample() [15/15]

```
void lsl::stream_outlet::push_sample (
            const std::string * data,
            double timestamp = 0.0,
            bool pushthrough = true )  [inline]
```

8.8.3.52 wait_for_consumers()

```
bool lsl::stream_outlet::wait_for_consumers (
            double timeout )  [inline]
```

Wait until some consumer shows up (without wasting resources).

Returns

True if the wait was successful, false if the timeout expired.

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

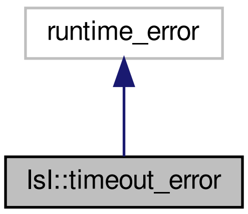
- include/lsl_cpp.h

8.9 lsl::timeout_error Class Reference

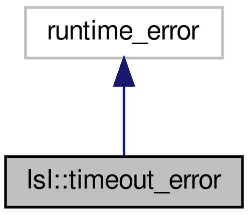
Exception class that indicates that an operation failed due to a timeout.

`#include <lsl_cpp.h>`

Inheritance diagram for lsl::timeout_error:



Collaboration diagram for lsl::timeout_error:



Public Member Functions

- timeout_error (const std::string &msg)

8.9.1 Detailed Description

Exception class that indicates that an operation failed due to a timeout.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 timeout_error()

```
lsl::timeout_error::timeout_error (
            const std::string & msg )  [inline], [explicit]
```

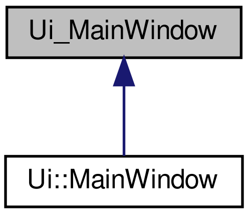
The documentation for this class was generated from the following file:

- include/lsl_cpp.h

8.10 Ui_MainWindow Class Reference

```
#include <ui_mainwindow.h>
```

Inheritance diagram for Ui_MainWindow:



Public Member Functions

- void setupUi (QMainWindow ∗MainWindow)
- void retranslateUi (QMainWindow ∗MainWindow)

Public Attributes

- QWidget ∗ centralWidget
- QGridLayout ∗ gridLayout
- QCheckBox ∗ checkBox_REC_ON
- QTreeWidget ∗ treeWidget
- QLabel ∗ label_6
- QLabel ∗ label
- QFrame ∗ line
- QFrame ∗ line_4
- QCheckBox ∗ checkBox_DECIM
- QLabel ∗ label_4
- QLabel ∗ label_5
- QComboBox ∗ comboBox_NCH
- QGridLayout ∗ gridLayout_2
- QLabel ∗ label_8
- QLabel ∗ label_9
- QLabel ∗ label_7
- QComboBox ∗ comboBox_INSEL
- QComboBox ∗ comboBox_CHSEL
- QComboBox ∗ comboBox_ANOUT_GAIN
- QGridLayout ∗ gridLayout_4
- QComboBox ∗ comboBox_HPF
- QLabel ∗ label_15
- QLabel ∗ label_10
- QComboBox ∗ comboBox_selectedIN
- QComboBox ∗ comboBox_MODE

- QComboBox ∗ comboBox_LPF
- QFrame ∗ line_3
- QComboBox ∗ comboBox_ADAPT
- QComboBox ∗ comboBox_SENS
- QLabel ∗ label_14
- QLabel ∗ label_16
- QLabel ∗ label_11
- QLabel ∗ label_13
- QLabel ∗ label_12
- QComboBox ∗ comboBox_SIDE
- QLabel ∗ label_17
- QComboBox ∗ comboBox_MUS
- QLabel ∗ label_2
- QGridLayout ∗ gridLayout_5
- QPushButton ∗ pushButton_save
- QLineEdit ∗ lineEdit_filepath
- QLabel ∗ label_18
- QPushButton ∗ pushButton_open
- QComboBox ∗ comboBox_FSAMP
- QFrame ∗ line_2
- QLabel ∗ label_3
- QFrame ∗ line_5
- QMenuBar ∗ menuBar
- QToolBar ∗ mainToolBar
- QStatusBar ∗ statusBar

8.10.1 Member Function Documentation

8.10.1.1 retranslateUi()

```
void Ui_MainWindow::retranslateUi (
            QMainWindow ∗ MainWindow )  [inline]
```

8.10.1.2 setupUi()

```
void Ui_MainWindow::setupUi (
            QMainWindow ∗ MainWindow )  [inline]
```

8.10.2 Member Data Documentation

8.10.2.1 centralWidget

```
QWidget* Ui_MainWindow::centralWidget
```

8.10.2.2 checkBox_DECIM

```
QCheckBox* Ui_MainWindow::checkBox_DECIM
```

8.10.2.3 checkBox_REC_ON

```
QCheckBox* Ui_MainWindow::checkBox_REC_ON
```

8.10.2.4 comboBox_ADAPT

```
QComboBox* Ui_MainWindow::comboBox_ADAPT
```

8.10.2.5 comboBox_ANOUT_GAIN

```
QComboBox* Ui_MainWindow::comboBox_ANOUT_GAIN
```

8.10.2.6 comboBox_CHSEL

```
QComboBox* Ui_MainWindow::comboBox_CHSEL
```

8.10.2.7 comboBox_FSAMP

```
QComboBox* Ui_MainWindow::comboBox_FSAMP
```

8.10.2.8 comboBox_HPF

```
QComboBox* Ui_MainWindow::comboBox_HPF
```

8.10.2.9 comboBox_INSEL

```
QComboBox* Ui_MainWindow::comboBox_INSEL
```

8.10.2.10 comboBox_LPF

```
QComboBox* Ui_MainWindow::comboBox_LPF
```

8.10.2.11 comboBox_MODE

```
QComboBox* Ui_MainWindow::comboBox_MODE
```

8.10.2.12 comboBox_MUS

```
QComboBox* Ui_MainWindow::comboBox_MUS
```

8.10.2.13 comboBox_NCH

```
QComboBox* Ui_MainWindow::comboBox_NCH
```

8.10.2.14 comboBox_selectedIN

```
QComboBox* Ui_MainWindow::comboBox_selectedIN
```

8.10.2.15 comboBox_SENS

```
QComboBox* Ui_MainWindow::comboBox_SENS
```

8.10.2.16 comboBox_SIDE

```
QComboBox* Ui_MainWindow::comboBox_SIDE
```

8.10.2.17 gridLayout

```
QGridLayout* Ui_MainWindow::gridLayout
```

8.10.2.18 gridLayout_2

```
QGridLayout* Ui_MainWindow::gridLayout_2
```

8.10.2.19 gridLayout_4

```
QGridLayout* Ui_MainWindow::gridLayout_4
```

8.10.2.20 gridLayout_5

```
QGridLayout* Ui_MainWindow::gridLayout_5
```

8.10.2.21 label

```
QLabel* Ui_MainWindow::label
```

8.10.2.22 label_10

```
QLabel* Ui_MainWindow::label_10
```

8.10.2.23 label_11

```
QLabel* Ui_MainWindow::label_11
```

8.10.2.24 label_12

```
QLabel* Ui_MainWindow::label_12
```

8.10.2.25 label_13

```
QLabel* Ui_MainWindow::label_13
```

8.10.2.26 label_14

```
QLabel* Ui_MainWindow::label_14
```

8.10.2.27 label_15

```
QLabel* Ui_MainWindow::label_15
```

8.10.2.28 label_16

```
QLabel* Ui_MainWindow::label_16
```

8.10.2.29 label_17

```
QLabel* Ui_MainWindow::label_17
```

8.10.2.30 label_18

```
QLabel* Ui_MainWindow::label_18
```

8.10.2.31 label_2

```
QLabel* Ui_MainWindow::label_2
```

8.10.2.32 label_3

```
QLabel* Ui_MainWindow::label_3
```

8.10.2.33 label_4

```
QLabel* Ui_MainWindow::label_4
```

8.10.2.34 label_5

```
QLabel* Ui_MainWindow::label_5
```

8.10.2.35 label_6

```
QLabel* Ui_MainWindow::label_6
```

8.10.2.36 label_7

```
QLabel* Ui_MainWindow::label_7
```

8.10.2.37 label_8

```
QLabel* Ui_MainWindow::label_8
```

8.10.2.38 label_9

```
QLabel* Ui_MainWindow::label_9
```

8.10.2.39 line

```
QFrame* Ui_MainWindow::line
```

8.10.2.40 line_2

```
QFrame* Ui_MainWindow::line_2
```

8.10.2.41 line_3

```
QFrame* Ui_MainWindow::line_3
```

8.10.2.42 line_4

```
QFrame* Ui_MainWindow::line_4
```

8.10.2.43 line_5

```
QFrame* Ui_MainWindow::line_5
```

8.10.2.44 lineEdit_filepath

```
QLineEdit* Ui_MainWindow::lineEdit_filepath
```

8.10.2.45 mainToolBar

```
QToolBar* Ui_MainWindow::mainToolBar
```

8.10.2.46 menuBar

```
QMenuBar* Ui_MainWindow::menuBar
```

8.10.2.47 pushButton_open

```
QPushButton* Ui_MainWindow::pushButton_open
```

8.10.2.48 pushButton_save

```
QPushButton* Ui_MainWindow::pushButton_save
```

8.10.2.49 statusBar

```
QStatusBar* Ui_MainWindow::statusBar
```

8.10.2.50 treeWidget

```
QTreeWidget* Ui_MainWindow::treeWidget
```

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

- OTBconfigGUI/build/ui_mainwindow.h

8.11 lsl::xml_element Class Reference

```
#include <lsl_cpp.h>
```

Public Member Functions

- xml_element (lsl_xml_ptr obj=0)

  Constructor.
- xml_element first_child () const

  Get the first child of the element.
- xml_element last_child () const

  Get the last child of the element.
- xml_element next_sibling () const

  Get the next sibling in the children list of the parent node.
- xml_element previous_sibling () const

  Get the previous sibling in the children list of the parent node.
- xml_element parent () const

  Get the parent node.
- xml_element child (const std::string &name) const

  Get a child with a specified name.
- xml_element next_sibling (const std::string &name) const

  Get the next sibling with the specified name.
- xml_element previous_sibling (const std::string &name) const

  Get the previous sibling with the specified name.
- bool empty () const

  Whether this node is empty.
- bool is_text () const

  Whether this is a text body (instead of an XML element). True both for plain char data and CData.
- const char ∗ name () const

  Name of the element.
- const char ∗ value () const

  Value of the element.
- const char ∗ child_value () const

Get child value (value of the first child that is text).
- const char ∗ child_value (const std::string &name) const

  Get child value of a child with a specified name.
- xml_element append_child_value (const std::string &name, const std::string &value)
- xml_element prepend_child_value (const std::string &name, const std::string &value)
- bool set_child_value (const std::string &name, const std::string &value)
- bool set_name (const std::string &rhs)
- bool set_value (const std::string &rhs)
- xml_element append_child (const std::string &name)

  Append a child element with the specified name.
- xml_element prepend_child (const std::string &name)

  Prepend a child element with the specified name.
- xml_element append_copy (const xml_element &e)

  Append a copy of the specified element as a child.
- xml_element prepend_copy (const xml_element &e)

  Prepend a child element with the specified name.
- void remove_child (const std::string &name)

  Remove a child element with the specified name.
- void remove_child (const xml_element &e)

  Remove a specified child element.

8.11.1 Detailed Description

A lightweight XML element tree; models the .desc() field of stream_info. Has a name and can have multiple named children or have text content as value; attributes are omitted. Insider note: The interface is modeled after a subset of pugixml's node type and is compatible with it. See also `http://pugixml.googlecode.com/svn/tags/latest/docs/manual/access.html` for additional documentation.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 xml_element()

```
lsl::xml_element::xml_element (
            lsl_xml_ptr obj = 0 )  [inline]
```

Constructor.

8.11.3 Member Function Documentation

8.11.3.1 append_child()

```
xml_element lsl::xml_element::append_child (
            const std::string & name )  [inline]
```

Append a child element with the specified name.

8.11.3.2 append_child_value()

```
xml_element lsl::xml_element::append_child_value (
            const std::string & name,
            const std::string & value )  [inline]
```

Append a child node with a given name, which has a (nameless) plain-text child with the given text value.

8.11.3.3 append_copy()

```
xml_element lsl::xml_element::append_copy (
            const xml_element & e )  [inline]
```

Append a copy of the specified element as a child.

8.11.3.4 child()

```
xml_element lsl::xml_element::child (
            const std::string & name ) const  [inline]
```

Get a child with a specified name.

8.11.3.5 child_value() [1/2]

```
const char* lsl::xml_element::child_value ( ) const  [inline]
```

Get child value (value of the first child that is text).

8.11.3.6 child_value() [2/2]

```
const char* lsl::xml_element::child_value (
            const std::string & name ) const  [inline]
```

Get child value of a child with a specified name.

8.11.3.7 empty()

```
bool lsl::xml_element::empty ( ) const  [inline]
```

Whether this node is empty.

8.11.3.8 first_child()

```
xml_element lsl::xml_element::first_child ( ) const  [inline]
```

Get the first child of the element.

8.11.3.9 is_text()

```
bool lsl::xml_element::is_text ( ) const  [inline]
```

Whether this is a text body (instead of an XML element). True both for plain char data and CData.

8.11.3.10 last_child()

```
xml_element lsl::xml_element::last_child ( ) const  [inline]
```

Get the last child of the element.

8.11.3.11 name()

```
const char* lsl::xml_element::name ( ) const  [inline]
```

Name of the element.

8.11.3.12 next_sibling() [1/2]

```
xml_element lsl::xml_element::next_sibling ( ) const  [inline]
```

Get the next sibling in the children list of the parent node.

8.11.3.13 next_sibling() [2/2]

```
xml_element lsl::xml_element::next_sibling (
             const std::string & name ) const  [inline]
```

Get the next sibling with the specified name.

8.11.3.14 parent()

```
xml_element lsl::xml_element::parent ( ) const  [inline]
```

Get the parent node.

8.11.3.15 prepend_child()

```
xml_element lsl::xml_element::prepend_child (
             const std::string & name )  [inline]
```

Prepend a child element with the specified name.

8.11.3.16 prepend_child_value()

```
xml_element lsl::xml_element::prepend_child_value (
             const std::string & name,
             const std::string & value )  [inline]
```

Prepend a child node with a given name, which has a (nameless) plain-text child with the given text value.

8.11.3.17 prepend_copy()

```
xml_element lsl::xml_element::prepend_copy (
             const xml_element & e )  [inline]
```

Prepend a child element with the specified name.

8.11.3.18 previous_sibling() [1/2]

```
xml_element lsl::xml_element::previous_sibling ( ) const  [inline]
```

Get the previous sibling in the children list of the parent node.

8.11.3.19 previous_sibling() [2/2]

```
xml_element lsl::xml_element::previous_sibling (
             const std::string & name ) const  [inline]
```

Get the previous sibling with the specified name.

8.11.3.20 remove_child() [1/2]

```
void lsl::xml_element::remove_child (
             const std::string & name )  [inline]
```

Remove a child element with the specified name.

8.11.3.21 remove_child() [2/2]

```
void lsl::xml_element::remove_child (
             const xml_element & e )  [inline]
```

Remove a specified child element.

8.11.3.22 set_child_value()

```
bool lsl::xml_element::set_child_value (
             const std::string & name,
             const std::string & value )  [inline]
```

Set the text value of the (nameless) plain-text child of a named child node.

8.11.3.23 set_name()

```
bool lsl::xml_element::set_name (
             const std::string & rhs )  [inline]
```

Set the element's name.

Returns

False if the node is empty (or if out of memory).

8.11.3.24 set_value()

```
bool lsl::xml_element::set_value (
             const std::string & rhs )  [inline]
```

Set the element's value.

Returns

False if the node is empty (or if out of memory).

8.11.3.25 value()

```
const char* lsl::xml_element::value ( ) const  [inline]
```

Value of the element.

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

- include/lsl_cpp.h

Chapter 9

File Documentation

9.1 build/CMakeFiles/3.10.2/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define C_DIALECT

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char const ∗ info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const ∗ info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const ∗ info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char ∗ info_language_dialect_default

9.1.1 Macro Definition Documentation

9.1.1.1 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

9.1.1.2 C_DIALECT

```
#define C_DIALECT
```

9.1.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

9.1.1.4 DEC

```
#define DEC(
              n )
```

Value:

```
('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

9.1.1.5 HEX

```
#define HEX(
              n )
```

Value:

```
('0' + ((n)>>28 & 0xF)), \
  ('0' + ((n)>>24 & 0xF)), \
  ('0' + ((n)>>20 & 0xF)), \
  ('0' + ((n)>>16 & 0xF)), \
  ('0' + ((n)>>12 & 0xF)), \
  ('0' + ((n)>>8  & 0xF)), \
  ('0' + ((n)>>4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

9.1.1.6 PLATFORM_ID

```
#define PLATFORM_ID
```

9.1.1.7 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

9.1.1.8 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

9.1.2 Function Documentation

9.1.2.1 main()

```
int main (
             int argc,
             char * argv[] )
```

9.1.3 Variable Documentation

9.1.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

9.1.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

9.1.3.3 info_language_dialect_default

```
const char* info_language_dialect_default
```

Initial value:

```
=
  "INFO" ":" "dialect_default[" C_DIALECT "]"
```

9.1.3.4 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

9.2 build/CMakeFiles/3.10.2/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX_STD __cplusplus

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char const ∗ info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const ∗ info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const ∗ info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char ∗ info_language_dialect_default

9.2.1 Macro Definition Documentation

9.2.1.1 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

9.2.1.2 COMPILER_ID

```
#define COMPILER_ID ""
```

9.2.1.3 CXX_STD

```
#define CXX_STD __cplusplus
```

9.2.1.4 DEC

```
#define DEC(
              n )
```

Value:

```
('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

9.2.1.5 HEX

```
#define HEX(
              n )
```

Value:

```
('0' + ((n)>>28 & 0xF)), \
  ('0' + ((n)>>24 & 0xF)), \
  ('0' + ((n)>>20 & 0xF)), \
  ('0' + ((n)>>16 & 0xF)), \
  ('0' + ((n)>>12 & 0xF)), \
  ('0' + ((n)>>8  & 0xF)), \
  ('0' + ((n)>>4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

9.2.1.6 PLATFORM_ID

```
#define PLATFORM_ID
```

9.2.1.7 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

9.2.1.8 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

9.2.2 Function Documentation

9.2.2.1 main()

```
int main (
             int argc,
             char * argv[] )
```

9.2.3 Variable Documentation

9.2.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

9.2.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

9.2.3.3 info_language_dialect_default

```
const char* info_language_dialect_default
```

Initial value:

```
= "INFO" ":" "dialect_default["




  "98"

"]"
```

9.2.3.4 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

9.3 build/CMakeFiles/feature_tests.c File Reference

Functions

- int main (int argc, char **argv)

Variables

- const char features [ ]

9.3.1 Function Documentation

9.3.1.1 main()

```
int main (
             int argc,
             char ** argv )
```

9.3.2 Variable Documentation

9.3.2.1 features

```
const char features[ ]
```

9.4 build/CMakeFiles/feature_tests.cxx File Reference

Functions

- int main (int argc, char **argv)

Variables

- const char features [ ]

9.4.1 Function Documentation

9.4.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

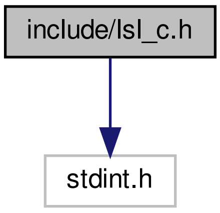
9.4.2 Variable Documentation

9.4.2.1 features

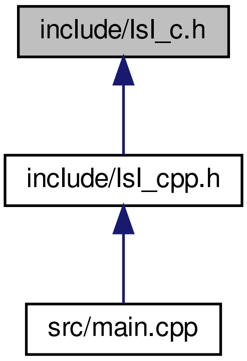
```
const char features[]
```

9.5 include/lsl_c.h File Reference

`#include <stdint.h>`
Include dependency graph for lsl_c.h:



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



Macros

- #define LIBLSL_C_API __attribute__((visibility("default")))
- #define LSL_IRREGULAR_RATE 0.0
- #define LSL_DEDUCED_TIMESTAMP -1.0
- #define LSL_FOREVER 32000000.0
- #define LSL_NO_PREFERENCE 0
- #define LIBLSL_COMPILE_HEADER_VERSION = 113;
- #define LSL_C_H
- #define LIBLSL_C_API __attribute__((visibility("default")))
- #define LSL_IRREGULAR_RATE 0.0
- #define LSL_DEDUCED_TIMESTAMP -1.0
- #define LSL_FOREVER 32000000.0
- #define LSL_NO_PREFERENCE 0
- #define LIBLSL_COMPILE_HEADER_VERSION = 113;

Typedefs

- typedef struct lsl_streaminfo_struct_ ∗ lsl_streaminfo
- typedef struct lsl_outlet_struct_ ∗ lsl_outlet
- typedef struct lsl_inlet_struct_ ∗ lsl_inlet
- typedef struct lsl_xml_ptr_struct_ ∗ lsl_xml_ptr
- typedef struct lsl_continuous_resolver_ ∗ lsl_continuous_resolver

Enumerations

- enum lsl_channel_format_t {
  cft_float32 = 1, cft_double64 = 2, cft_string = 3, cft_int32 = 4,
  cft_int16 = 5, cft_int8 = 6, cft_int64 = 7, cft_undefined = 0 }
- enum lsl_processing_options_t {
  proc_none = 0, proc_clocksync = 1, proc_dejitter = 2, proc_monotonize = 4,
  proc_threadsafe = 8, proc_ALL = 1|2|4|8 }
- enum lsl_error_code_t {
  lsl_no_error = 0, lsl_timeout_error = -1, lsl_lost_error = -2, lsl_argument_error = -3,
  lsl_internal_error = -4 }

Functions

- LIBLSL_C_API int32_t lsl_protocol_version ()
- LIBLSL_C_API int32_t lsl_library_version ()
- LIBLSL_C_API const char ∗ lsl_library_info ()
- LIBLSL_C_API double lsl_local_clock ()
- LIBLSL_C_API int32_t lsl_resolve_all (lsl_streaminfo ∗buffer, uint32_t buffer_elements, double wait_time)
- LIBLSL_C_API int32_t lsl_resolve_byprop (lsl_streaminfo ∗buffer, uint32_t buffer_elements, const char ∗prop, const char ∗value, int32_t minimum, double timeout)
- LIBLSL_C_API int32_t lsl_resolve_bypred (lsl_streaminfo ∗buffer, uint32_t buffer_elements, const char ∗pred, int32_t minimum, double timeout)
- LIBLSL_C_API void lsl_destroy_string (char ∗s)
- LIBLSL_C_API lsl_streaminfo lsl_create_streaminfo (const char ∗name, const char ∗type, int32_t channel←_count, double nominal_srate, lsl_channel_format_t channel_format, const char ∗source_id)
- LIBLSL_C_API void lsl_destroy_streaminfo (lsl_streaminfo info)
- LIBLSL_C_API lsl_streaminfo lsl_copy_streaminfo (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_name (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_type (lsl_streaminfo info)
- LIBLSL_C_API int32_t lsl_get_channel_count (lsl_streaminfo info)
- LIBLSL_C_API double lsl_get_nominal_srate (lsl_streaminfo info)
- LIBLSL_C_API lsl_channel_format_t lsl_get_channel_format (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_source_id (lsl_streaminfo info)
- LIBLSL_C_API int32_t lsl_get_version (lsl_streaminfo info)
- LIBLSL_C_API double lsl_get_created_at (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_uid (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_session_id (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl_get_hostname (lsl_streaminfo info)
- LIBLSL_C_API lsl_xml_ptr lsl_get_desc (lsl_streaminfo info)
- LIBLSL_C_API char ∗ lsl_get_xml (lsl_streaminfo info)
- LIBLSL_C_API int32_t lsl_get_channel_bytes (lsl_streaminfo info)

    Number of bytes occupied by a channel (0 for string-typed channels).
- LIBLSL_C_API int32_t lsl_get_sample_bytes (lsl_streaminfo info)

    Number of bytes occupied by a sample (0 for string-typed channels).
- LIBLSL_C_API int lsl_stream_info_matches_query (lsl_streaminfo info, const char ∗query)
- LIBLSL_C_API lsl_streaminfo lsl_streaminfo_from_xml (const char ∗xml)

    Create a streaminfo object from an XML representation.
- LIBLSL_C_API lsl_outlet lsl_create_outlet (lsl_streaminfo info, int32_t chunk_size, int32_t max_buffered)
- LIBLSL_C_API void lsl_destroy_outlet (lsl_outlet out)
- LIBLSL_C_API int32_t lsl_push_sample_f (lsl_outlet out, const float ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_ft (lsl_outlet out, const float ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_ftp (lsl_outlet out, const float ∗data, double timestamp, int32_←t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_d (lsl_outlet out, const double ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_dt (lsl_outlet out, const double ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_dtp (lsl_outlet out, const double ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_l (lsl_outlet out, const long ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_lt (lsl_outlet out, const long ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_ltp (lsl_outlet out, const long ∗data, double timestamp, int32_←t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_i (lsl_outlet out, const int32_t ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_it (lsl_outlet out, const int32_t ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_itp (lsl_outlet out, const int32_t ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_s (lsl_outlet out, const int16_t ∗data)

- LIBLSL_C_API int32_t lsl_push_sample_st (lsl_outlet out, const int16_t ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_stp (lsl_outlet out, const int16_t ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_c (lsl_outlet out, const char ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_ct (lsl_outlet out, const char ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_ctp (lsl_outlet out, const char ∗data, double timestamp, int32_↩ t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_str (lsl_outlet out, const char ∗∗data)
- LIBLSL_C_API int32_t lsl_push_sample_strt (lsl_outlet out, const char ∗∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_strtp (lsl_outlet out, const char ∗∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_buf (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths)
- LIBLSL_C_API int32_t lsl_push_sample_buft (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_buftp (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_sample_v (lsl_outlet out, const void ∗data)
- LIBLSL_C_API int32_t lsl_push_sample_vt (lsl_outlet out, const void ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl_push_sample_vtp (lsl_outlet out, const void ∗data, double timestamp, int32_↩ t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_f (lsl_outlet out, const float ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_ft (lsl_outlet out, const float ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_ftp (lsl_outlet out, const float ∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_ftn (lsl_outlet out, const float ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_ftnp (lsl_outlet out, const float ∗data, unsigned long data_elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_d (lsl_outlet out, const double ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_dt (lsl_outlet out, const double ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_dtp (lsl_outlet out, const double ∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_dtn (lsl_outlet out, const double ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_dtnp (lsl_outlet out, const double ∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int lsl_push_chunk_l (lsl_outlet out, const long ∗data, unsigned long data_elements)
- LIBLSL_C_API int lsl_push_chunk_lt (lsl_outlet out, const long ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int lsl_push_chunk_ltp (lsl_outlet out, const long ∗data, unsigned long data_elements, double timestamp, int pushthrough)
- LIBLSL_C_API int lsl_push_chunk_ltn (lsl_outlet out, const long ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int lsl_push_chunk_ltnp (lsl_outlet out, const long ∗data, unsigned long data_elements, const double ∗timestamps, int pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_i (lsl_outlet out, const int32_t ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_it (lsl_outlet out, const int32_t ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_itp (lsl_outlet out, const int32_t ∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_itn (lsl_outlet out, const int32_t ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_itnp (lsl_outlet out, const int32_t ∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)

- LIBLSL_C_API int32_t lsl_push_chunk_s (lsl_outlet out, const int16_t ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_st (lsl_outlet out, const int16_t ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_stp (lsl_outlet out, const int16_t ∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_stn (lsl_outlet out, const int16_t ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_stnp (lsl_outlet out, const int16_t ∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_c (lsl_outlet out, const char ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_ct (lsl_outlet out, const char ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_ctp (lsl_outlet out, const char ∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_ctn (lsl_outlet out, const char ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_ctnp (lsl_outlet out, const char ∗data, unsigned long data_elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_str (lsl_outlet out, const char ∗∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_strt (lsl_outlet out, const char ∗∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_strtp (lsl_outlet out, const char ∗∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_strtn (lsl_outlet out, const char ∗∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_strtnp (lsl_outlet out, const char ∗∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_buf (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl_push_chunk_buft (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl_push_chunk_buftp (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_push_chunk_buftn (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl_push_chunk_buftnp (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl_have_consumers (lsl_outlet out)
- LIBLSL_C_API int32_t lsl_wait_for_consumers (lsl_outlet out, double timeout)
- LIBLSL_C_API lsl_streaminfo lsl_get_info (lsl_outlet out)
- LIBLSL_C_API lsl_inlet lsl_create_inlet (lsl_streaminfo info, int32_t max_buflen, int32_t max_chunklen, int32_t recover)
- LIBLSL_C_API void lsl_destroy_inlet (lsl_inlet in)
- LIBLSL_C_API lsl_streaminfo lsl_get_fullinfo (lsl_inlet in, double timeout, int32_t ∗ec)
- LIBLSL_C_API void lsl_open_stream (lsl_inlet in, double timeout, int32_t ∗ec)
- LIBLSL_C_API void lsl_close_stream (lsl_inlet in)
- LIBLSL_C_API double lsl_time_correction (lsl_inlet in, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_time_correction_ex (lsl_inlet in, double ∗remote_time, double ∗uncertainty, double timeout, int32_t ∗ec)
- LIBLSL_C_API int32_t lsl_set_postprocessing (lsl_inlet in, uint32_t flags)
- LIBLSL_C_API double lsl_pull_sample_f (lsl_inlet in, float ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_d (lsl_inlet in, double ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_l (lsl_inlet in, long ∗buffer, int buffer_elements, double timeout, int ∗ec)

- LIBLSL_C_API double lsl_pull_sample_i (lsl_inlet in, int32_t ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_s (lsl_inlet in, int16_t ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_c (lsl_inlet in, char ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_str (lsl_inlet in, char ∗∗buffer, int32_t buffer_elements, double time-out, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_buf (lsl_inlet in, char ∗∗buffer, uint32_t ∗buffer_lengths, int32_↩ t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl_pull_sample_v (lsl_inlet in, void ∗buffer, int32_t buffer_bytes, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_f (lsl_inlet in, float ∗data_buffer, double ∗timestamp_buffer, un-signed long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_d (lsl_inlet in, double ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_l (lsl_inlet in, long ∗data_buffer, double ∗timestamp_buffer, un-signed long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_i (lsl_inlet in, int32_t ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_s (lsl_inlet in, int16_t ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_c (lsl_inlet in, char ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_↩ t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_str (lsl_inlet in, char ∗∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl_pull_chunk_buf (lsl_inlet in, char ∗∗data_buffer, uint32_t ∗lengths_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API uint32_t lsl_samples_available (lsl_inlet in)
- LIBLSL_C_API uint32_t lsl_was_clock_reset (lsl_inlet in)
- LIBLSL_C_API int32_t lsl_smoothing_halftime (lsl_inlet in, float value)
- LIBLSL_C_API lsl_xml_ptr lsl_first_child (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl_last_child (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl_next_sibling (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl_previous_sibling (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl_parent (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl_child (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl_next_sibling_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl_previous_sibling_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API int32_t lsl_empty (lsl_xml_ptr e)
- LIBLSL_C_API int32_t lsl_is_text (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl_name (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl_value (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl_child_value (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl_child_value_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl_append_child_value (lsl_xml_ptr e, const char ∗name, const char ∗value)
- LIBLSL_C_API lsl_xml_ptr lsl_prepend_child_value (lsl_xml_ptr e, const char ∗name, const char ∗value)
- LIBLSL_C_API int32_t lsl_set_child_value (lsl_xml_ptr e, const char ∗name, const char ∗value)
- LIBLSL_C_API int32_t lsl_set_name (lsl_xml_ptr e, const char ∗rhs)
- LIBLSL_C_API int32_t lsl_set_value (lsl_xml_ptr e, const char ∗rhs)
- LIBLSL_C_API lsl_xml_ptr lsl_append_child (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl_prepend_child (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl_append_copy (lsl_xml_ptr e, lsl_xml_ptr e2)
- LIBLSL_C_API lsl_xml_ptr lsl_prepend_copy (lsl_xml_ptr e, lsl_xml_ptr e2)

- LIBLSL_C_API void lsl_remove_child_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API void lsl_remove_child (lsl_xml_ptr e, lsl_xml_ptr e2)
- LIBLSL_C_API lsl_continuous_resolver lsl_create_continuous_resolver (double forget_after)
- LIBLSL_C_API lsl_continuous_resolver lsl_create_continuous_resolver_byprop (const char ∗prop, const char ∗value, double forget_after)
- LIBLSL_C_API lsl_continuous_resolver lsl_create_continuous_resolver_bypred (const char ∗pred, double forget_after)
- LIBLSL_C_API int32_t lsl_resolver_results (lsl_continuous_resolver res, lsl_streaminfo ∗buffer, uint32_↩ t buffer_elements)
- LIBLSL_C_API void lsl_destroy_continuous_resolver (lsl_continuous_resolver res)

9.5.1 Macro Definition Documentation

9.5.1.1 LIBLSL_C_API [1/2]

```
#define LIBLSL_C_API __attribute__((visibility("default")))
```

C API for the lab streaming layer.

The lab streaming layer provides a set of functions to make instrument data accessible in real time within a lab network. From there, streams can be picked up by recording programs, viewing programs or custom experiment applications that access data streams in real time.

The API covers two areas:

- The "push API" allows to create stream outlets and to push data (regular or irregular measurement time series, event data, coded audio/video frames, etc.) into them.
- The "pull API" allows to create stream inlets and read time-synched experiment data from them (for recording, viewing or experiment control).

To use this library you need to link to either the liblsl32 or liblsl64 shared library that comes with this header. Under Visual Studio the library is linked in automatically.

9.5.1.2 LIBLSL_C_API [2/2]

```
#define LIBLSL_C_API __attribute__((visibility("default")))
```

9.5.1.3 LIBLSL_COMPILE_HEADER_VERSION [1/2]

```
#define LIBLSL_COMPILE_HEADER_VERSION = 113;
```

LSL version the binary was compiled against Used either to check if the same version is used (if(lsl_protocol_↩ version()!=LIBLSL_COMPILE_HEADER_VERSION) . . . or to require a certain set of features: #if LIBLSL_COM↩ PILE_HEADER_VERSION > 113 do_stuff(); #endif

9.5.1.4 LIBLSL_COMPILE_HEADER_VERSION [2/2]

```
#define LIBLSL_COMPILE_HEADER_VERSION = 113;
```

9.5.1.5 LSL_C_H

```
#define LSL_C_H
```

9.5.1.6 LSL_DEDUCED_TIMESTAMP [1/2]

```
#define LSL_DEDUCED_TIMESTAMP -1.0
```

Constant to indicate that a sample has the next successive time stamp. This is an optional optimization to transmit less data per sample. The stamp is then deduced from the preceding one according to the stream's sampling rate (in the case of an irregular rate, the same time stamp as before will is assumed).

9.5.1.7 LSL_DEDUCED_TIMESTAMP [2/2]

```
#define LSL_DEDUCED_TIMESTAMP -1.0
```

9.5.1.8 LSL_FOREVER [1/2]

```
#define LSL_FOREVER 32000000.0
```

A very large time value (ca. 1 year); can be used in timeouts.

9.5.1.9 LSL_FOREVER [2/2]

```
#define LSL_FOREVER 32000000.0
```

9.5.1.10 LSL_IRREGULAR_RATE [1/2]

```
#define LSL_IRREGULAR_RATE 0.0
```

9.5.1.11 LSL_IRREGULAR_RATE [2/2]

```
#define LSL_IRREGULAR_RATE 0.0
```

Constant to indicate that a stream has variable sampling rate.

9.5.1.12 LSL_NO_PREFERENCE [1/2]

```
#define LSL_NO_PREFERENCE 0
```

9.5.1.13 LSL_NO_PREFERENCE [2/2]

```
#define LSL_NO_PREFERENCE 0
```

Constant to indicate that there is no preference about how a data stream shall be chunked for transmission. (can be used for the chunking parameters in the inlet or the outlet).

9.5.2 Typedef Documentation

9.5.2.1 lsl_continuous_resolver

```
typedef struct lsl_continuous_resolver_* lsl_continuous_resolver
```

Handle to a convenience object that resolves streams continuously in the background throughout its lifetime and which can be queried at any time for the set of streams that are currently visible on the network.

9.5.2.2 lsl_inlet

```
typedef struct lsl_inlet_struct_* lsl_inlet
```

A stream inlet handle. Inlets are used to receive streaming data (and meta-data) from the lab network.

9.5.2.3 lsl_outlet

```
typedef struct lsl_outlet_struct_* lsl_outlet
```

A stream outlet handle. Outlets are used to make streaming data (and the meta-data) available on the lab network.

9.5.2.4 lsl_streaminfo

```
typedef struct lsl_streaminfo_struct_* lsl_streaminfo
```

Handle to a stream info object. Stores the declaration of a data stream. Represents the following information: a) stream data format (#channels, channel format) b) core information (stream name, content type, sampling rate) c) optional meta-data about the stream content (channel labels, measurement units, etc.)

Whenever a program wants to provide a new stream on the lab network it will typically first create an lsl_streaminfo to describe its properties and then construct an lsl_outlet with it to create the stream on the network. Other parties who discover/resolve the outlet on the network can query the stream info; it is also written to disk when recording the stream (playing a similar role as a file header).

9.5.2.5 lsl_xml_ptr

```
typedef struct lsl_xml_ptr_struct_* lsl_xml_ptr
```

A lightweight XML element tree handle; models the description of a streaminfo object. XML elements behave like advanced pointers into memory that is owned by some respective streaminfo. Has a name and can have multiple named children or have text content as value; attributes are omitted. Insider note: The interface is modeled after a subset of pugixml's node type and is compatible with it. Type-casts between pugi::xml_node_struct∗ and lsl_↩ xml_ptr are permitted (in both directions) since the types are binary compatible. See also pugixml.googlecode.↩ com/svn/tags/latest/docs/manual/access.html for additional documentation.

9.5.3 Enumeration Type Documentation

9.5.3.1 lsl_channel_format_t

```
enum lsl_channel_format_t
```

Data format of a channel (each transmitted sample holds an array of channels).

Enumerator

cft_float32	
cft_double64	
cft_string	
cft_int32	
cft_int16	
cft_int8	
cft_int64	
cft_undefined	

9.5.3.2 lsl_error_code_t

```
enum lsl_error_code_t
```

Possible error codes.

Enumerator

lsl_no_error	
lsl_timeout_error	
lsl_lost_error	
lsl_argument_error	
lsl_internal_error	

9.5.3.3 lsl_processing_options_t

```
enum lsl_processing_options_t
```

Post-processing options for stream inlets.

Enumerator

proc_none	
proc_clocksync	
proc_dejitter	
proc_monotonize	
proc_threadsafe	
proc_ALL	

9.5.4 Function Documentation

9.5.4.1 lsl_append_child()

```
LIBLSL_C_API lsl_xml_ptr lsl_append_child (
            lsl_xml_ptr e,
            const char * name )
```

Append a child element with the specified name.

9.5.4.2 lsl_append_child_value()

```
LIBLSL_C_API lsl_xml_ptr lsl_append_child_value (
            lsl_xml_ptr e,
            const char * name,
            const char * value )
```

Append a child node with a given name, which has a (nameless) plain-text child with the given text value.

9.5.4.3 lsl_append_copy()

```
LIBLSL_C_API lsl_xml_ptr lsl_append_copy (
             lsl_xml_ptr e,
             lsl_xml_ptr e2 )
```

Append a copy of the specified element as a child.

9.5.4.4 lsl_child()

```
LIBLSL_C_API lsl_xml_ptr lsl_child (
             lsl_xml_ptr e,
             const char * name )
```

Get a child with a specified name.

9.5.4.5 lsl_child_value()

```
LIBLSL_C_API const char* lsl_child_value (
             lsl_xml_ptr e )
```

Get child value (value of the first child that is text).

9.5.4.6 lsl_child_value_n()

```
LIBLSL_C_API const char* lsl_child_value_n (
             lsl_xml_ptr e,
             const char * name )
```

Get child value of a child with a specified name.

9.5.4.7 lsl_close_stream()

```
LIBLSL_C_API void lsl_close_stream (
             lsl_inlet in )
```

Drop the current data stream. All samples that are still buffered or in flight will be dropped and transmission and buffering of data for this inlet will be stopped. If an application stops being interested in data from a source (temporarily or not) but keeps the outlet alive, it should call lsl_close_stream() to not waste unnecessary system and network resources.

9.5.4.8 lsl_copy_streaminfo()

```
LIBLSL_C_API lsl_streaminfo lsl_copy_streaminfo (
             lsl_streaminfo info )
```

Copy an existing streaminfo object (rarely used).

9.5.4.9 lsl_create_continuous_resolver()

```
LIBLSL_C_API lsl_continuous_resolver lsl_create_continuous_resolver (
             double forget_after )
```

Construct a new continuous_resolver that resolves all streams on the network. This is analogous to the functionality offered by the free function resolve_streams().

Parameters

forget_after	When a stream is no longer visible on the network (e.g., because it was shut down), this is the time in seconds after which it is no longer reported by the resolver. The recommended default value is 5.0.

9.5.4.10 lsl_create_continuous_resolver_bypred()

```
LIBLSL_C_API lsl_continuous_resolver lsl_create_continuous_resolver_bypred (
            const char * pred,
            double forget_after )
```

Construct a new continuous_resolver that resolves all streams that match a given XPath 1.0 predicate. This is analogous to the functionality provided by the free function resolve_stream(pred).

Parameters

pred	The predicate string, e.g. "name='BioSemi'" or "type='EEG' and starts-with(name,'BioSemi') and count(info/desc/channel)=32"
forget_after	When a stream is no longer visible on the network (e.g., because it was shut down), this is the time in seconds after which it is no longer reported by the resolver. The recommended default value is 5.0.

9.5.4.11 lsl_create_continuous_resolver_byprop()

```
LIBLSL_C_API lsl_continuous_resolver lsl_create_continuous_resolver_byprop (
            const char * prop,
            const char * value,
            double forget_after )
```

Construct a new continuous_resolver that resolves all streams with a specific value for a given property. This is analogous to the functionality provided by the free function resolve_stream(prop,value).

Parameters

prop	The stream_info property that should have a specific value (e.g., "name", "type", "source_id", or "desc/manufaturer").
value	The string value that the property should have (e.g., "EEG" as the type property).
forget_after	When a stream is no longer visible on the network (e.g., because it was shut down), this is the time in seconds after which it is no longer reported by the resolver. The recommended default value is 5.0.

9.5.4.12 lsl_create_inlet()

```
LIBLSL_C_API lsl_inlet lsl_create_inlet (
              lsl_streaminfo info,
              int32_t max_buflen,
              int32_t max_chunklen,
              int32_t recover )
```

Construct a new stream inlet from a resolved stream info.

Parameters

info	A resolved stream info object (as coming from one of the resolver functions). Note: the inlet makes a copy of the info object at its construction. Note: the stream_inlet may also be constructed with a fully-specified stream_info, if the desired channel format and count is already known up-front, but this is strongly discouraged and should only ever be done if there is no time to resolve the stream up-front (e.g., due to limitations in the client program).
max_buflen	Optionally the maximum amount of data to buffer (in seconds if there is a nominal sampling rate, otherwise x100 in samples). Recording applications want to use a fairly large buffer size here, while real-time applications would only buffer as much as they need to perform their next calculation. A good default is 360, which corresponds to 6 minutes of data.
max_chunklen	Optionally the maximum size, in samples, at which chunks are transmitted. If specified as 0, the chunk sizes preferred by the sender are used. Recording applications can use a generous size here (leaving it to the network how to pack things), while real-time applications may want a finer (perhaps 1-sample) granularity.
recover	Try to silently recover lost streams that are recoverable (=those that that have a source_id set). It is generally a good idea to enable this, unless the application wants to act in a special way when a data provider has temporarily crashed. If recover is 0 or the stream is not recoverable, most outlet functions will return an lsl_lost_error if the stream's source is lost.

Returns

A newly created lsl_inlet handle or NULL in the event that an error occurred.

9.5.4.13 lsl_create_outlet()

```
LIBLSL_C_API lsl_outlet lsl_create_outlet (
              lsl_streaminfo info,
              int32_t chunk_size,
              int32_t max_buffered )
```

Establish a new stream outlet. This makes the stream discoverable.

Parameters

info	The stream information to use for creating this stream. Stays constant over the lifetime of the outlet. Note: the outlet makes a copy of the streaminfo object upon construction (so the old info should still be destroyed.)
chunk_size	Optionally the desired chunk granularity (in samples) for transmission. If specified as 0, each push operation yields one chunk. Stream recipients can have this setting bypassed.
max_buffered	Optionally the maximum amount of data to buffer (in seconds if there is a nominal sampling rate, otherwise x100 in samples). A good default is 360, which corresponds to 6 minutes of data. Note that, for high-bandwidth data you will almost certainly want to use a lower value here to avoid running out of RAM.

Returns

A newly created lsl_outlet handle or NULL in the event that an error occurred.

9.5.4.14 lsl_create_streaminfo()

```
LIBLSL_C_API lsl_streaminfo lsl_create_streaminfo (
             const char * name,
             const char * type,
             int32_t channel_count,
             double nominal_srate,
             lsl_channel_format_t channel_format,
             const char * source_id )
```

Construct a new streaminfo object. Core stream information is specified here. Any remaining meta-data can be added later.

Parameters

name	Name of the stream. Describes the device (or product series) that this stream makes available (for use by programs, experimenters or data analysts). Cannot be empty.
type	Content type of the stream. Please see `https://github.com/sccn/xdf/wiki/Meta-Data` (or web search for: XDF meta-data) for pre-defined content-type names, but you can also make up your own. The content type is the preferred way to find streams (as opposed to searching by name).
channel_count	Number of channels per sample. This stays constant for the lifetime of the stream.
nominal_srate	The sampling rate (in Hz) as advertised by the data source, if regular (otherwise set to IRREGULAR_RATE).
channel_format	Format/type of each channel. If your channels have different formats, consider supplying multiple streams or use the largest type that can hold them all (such as cft_double64). A good default is cft_float32.
source_id	Unique identifier of the source or device, if available (such as the serial number). Allows recipients to recover from failure even after the serving app or device crashes. May in some cases also be constructed from device settings.

Returns

A newly created streaminfo handle or NULL in the event that an error occurred.

9.5.4.15 lsl_destroy_continuous_resolver()

```
LIBLSL_C_API void lsl_destroy_continuous_resolver (
             lsl_continuous_resolver res )
```

Destructor for the continuous resolver.

9.5.4.16 lsl_destroy_inlet()

```
LIBLSL_C_API void lsl_destroy_inlet (
             lsl_inlet in )
```

Destructor. The inlet will automatically disconnect if destroyed.

9.5.4.17 lsl_destroy_outlet()

```
LIBLSL_C_API void lsl_destroy_outlet (
             lsl_outlet out )
```

Destroy an outlet. The outlet will no longer be discoverable after destruction and all connected inlets will stop delivering data.

9.5.4.18 lsl_destroy_streaminfo()

```
LIBLSL_C_API void lsl_destroy_streaminfo (
             lsl_streaminfo info )
```

Destroy a previously created streaminfo object.

9.5.4.19 lsl_destroy_string()

```
LIBLSL_C_API void lsl_destroy_string (
             char * s )
```

Deallocate a string that has been transferred to the application. Rarely used: the only use case is to deallocate the contents of string-valued samples received from LSL in an application where no free() method is available (e.g., in some scripting languages).

9.5.4.20 lsl_empty()

```
LIBLSL_C_API int32_t lsl_empty (
             lsl_xml_ptr e )
```

Whether this node is empty.

9.5.4.21 lsl_first_child()

```
LIBLSL_C_API lsl_xml_ptr lsl_first_child (
             lsl_xml_ptr e )
```

Get the first child of the element.

9.5.4.22 lsl_get_channel_bytes()

```
LIBLSL_C_API int32_t lsl_get_channel_bytes (
             lsl_streaminfo info )
```

Number of bytes occupied by a channel (0 for string-typed channels).

9.5.4.23 lsl_get_channel_count()

```
LIBLSL_C_API int32_t lsl_get_channel_count (
             lsl_streaminfo info )
```

Number of channels of the stream. A stream has at least one channels; the channel count stays constant for all samples.

9.5.4.24 lsl_get_channel_format()

```
LIBLSL_C_API lsl_channel_format_t lsl_get_channel_format (
             lsl_streaminfo info )
```

Channel format of the stream. All channels in a stream have the same format. However, a device might offer multiple time-synched streams each with its own format.

9.5.4.25 lsl_get_created_at()

```
LIBLSL_C_API double lsl_get_created_at (
             lsl_streaminfo info )
```

Creation time stamp of the stream. This is the time stamp when the stream was first created (as determined via local_clock() on the providing machine).

9.5.4.26 lsl_get_desc()

```
LIBLSL_C_API lsl_xml_ptr lsl_get_desc (
             lsl_streaminfo info )
```

Extended description of the stream. It is highly recommended that at least the channel labels are described here. See code examples on the LSL wiki. Other information, such as amplifier settings, measurement units if deviating from defaults, setup information, subject information, etc., can be specified here, as well. Meta-data recommendations follow the XDF file format project (github.com/sccn/xdf/wiki/Meta-Data or web search for: XDF meta-data).

Important: if you use a stream content type for which meta-data recommendations exist, please try to lay out your meta-data in agreement with these recommendations for compatibility with other applications.

9.5.4.27 lsl_get_fullinfo()

```
LIBLSL_C_API lsl_streaminfo lsl_get_fullinfo (
             lsl_inlet in,
             double timeout,
             int32_t * ec )
```

Retrieve the complete information of the given stream, including the extended description. Can be invoked at any time of the stream's lifetime.

Parameters

in	The lsl_inlet object to act on.
timeout	Timeout of the operation. Use LSL_FOREVER to effectively disable it.
ec	Error code: if nonzero, can be either lsl_timeout_error (if the timeout has expired) or lsl_lost_error (if the stream source has been lost).

Returns

A copy of the full streaminfo of the inlet or NULL in the event that an error happened. Note: it is the user's responsibility to destroy it when it is no longer needed.

9.5.4.28 lsl_get_hostname()

```
LIBLSL_C_API const char* lsl_get_hostname (
            lsl_streaminfo info )
```

Hostname of the providing machine (once bound to an outlet). Modification is not permitted.

9.5.4.29 lsl_get_info()

```
LIBLSL_C_API lsl_streaminfo lsl_get_info (
            lsl_outlet out )
```

Retrieve a handle to the stream info provided by this outlet. This is what was used to create the stream (and also has the Additional Network Information fields assigned).

Returns

A copy of the streaminfo of the outlet or NULL in the event that an error occurred. Note: it is the user's responsibility to destroy it when it is no longer needed.

9.5.4.30 lsl_get_name()

```
LIBLSL_C_API const char* lsl_get_name (
            lsl_streaminfo info )
```

Name of the stream. This is a human-readable name. For streams offered by device modules, it refers to the type of device or product series that is generating the data of the stream. If the source is an application, the name may be a more generic or specific identifier. Multiple streams with the same name can coexist, though potentially at the cost of ambiguity (for the recording app or experimenter).

Returns

A library-owned pointer to the string value. Modification is not permitted.

9.5.4.31 lsl_get_nominal_srate()

```
LIBLSL_C_API double lsl_get_nominal_srate (
            lsl_streaminfo info )
```

Sampling rate of the stream, according to the source (in Hz). If a stream is irregularly sampled, this should be set to IRREGULAR_RATE.

Note that no data will be lost even if this sampling rate is incorrect or if a device has temporary hiccups, since all samples will be recorded anyway (except for those dropped by the device itself). However, when the recording is imported into an application, a good importer may correct such errors more accurately if the advertised sampling rate was close to the specs of the device.

9.5.4.32 lsl_get_sample_bytes()

```
LIBLSL_C_API int32_t lsl_get_sample_bytes (
            lsl_streaminfo info )
```

Number of bytes occupied by a sample (0 for string-typed channels).

9.5.4.33 lsl_get_session_id()

```
LIBLSL_C_API const char* lsl_get_session_id (
            lsl_streaminfo info )
```

Session ID for the given stream. The session id is an optional human-assigned identifier of the recording session. While it is rarely used, it can be used to prevent concurrent recording activitites on the same sub-network (e.g., in multiple experiment areas) from seeing each other's streams (assigned via a configuration file by the experimenter, see Network Connectivity on the LSL wiki).

Returns

A library-owned pointer to the string value. Modification is not permitted.

9.5.4.34 lsl_get_source_id()

```
LIBLSL_C_API const char* lsl_get_source_id (
            lsl_streaminfo info )
```

Unique identifier of the stream's source, if available. The unique source (or device) identifier is an optional piece of information that, if available, allows that endpoints (such as the recording program) can re-acquire a stream automatically once it is back online.

Returns

A library-owned pointer to the string value. Modification is not permitted.

9.5.4.35 lsl_get_type()

```
LIBLSL_C_API const char* lsl_get_type (
             lsl_streaminfo info )
```

Content type of the stream. The content type is a short string such as "EEG", "Gaze" which describes the content carried by the channel (if known). If a stream contains mixed content this value need not be assigned but may instead be stored in the description of channel types. To be useful to applications and automated processing systems using the recommended content types is preferred. Content types usually follow those pre-defined in `https://github.com/sccn/xdf/wiki/Meta-Data` (or web search for: XDF meta-data).

Returns

A library-owned pointer to the string value. Modification is not permitted.

9.5.4.36 lsl_get_uid()

```
LIBLSL_C_API const char* lsl_get_uid (
             lsl_streaminfo info )
```

Unique ID of the stream outlet (once assigned). This is a unique identifier of the stream outlet, and is guaranteed to be different across multiple instantiations of the same outlet (e.g., after a re-start).

Returns

A library-owned pointer to the string value. Modification is not permitted.

9.5.4.37 lsl_get_version()

```
LIBLSL_C_API int32_t lsl_get_version (
             lsl_streaminfo info )
```

Protocol version used to deliver the stream.

9.5.4.38 lsl_get_xml()

```
LIBLSL_C_API char* lsl_get_xml (
             lsl_streaminfo info )
```

Retrieve the entire streaminfo in XML format. This yields an XML document (in string form) whose top-level element is <info>. The info element contains one element for each field of the streaminfo class, including: a) the core elements <name>, <type>, <channel_count>, <nominal_srate>, <channel_format>, <source_id> b) the misc elements <version>, <created_at>, <uid>, <session_id>, <v4address>, <v4data_port>, <v4service_port>, <v6address>, <v6data_port>, <v6service_port> c) the extended description element <desc> with user-defined sub-elements.

Returns

A pointer to a copy of the XML text or NULL in the event that an error occurred. Note: It is the user's responsibility to deallocate this string when it is no longer needed.

9.5.4.39 lsl_have_consumers()

```
LIBLSL_C_API int32_t lsl_have_consumers (
            lsl_outlet out )
```

Check whether consumers are currently registered. While it does not hurt, there is technically no reason to push samples if there is no consumer.

9.5.4.40 lsl_is_text()

```
LIBLSL_C_API int32_t lsl_is_text (
            lsl_xml_ptr e )
```

Whether this is a text body (instead of an XML element). True both for plain char data and CData.

9.5.4.41 lsl_last_child()

```
LIBLSL_C_API lsl_xml_ptr lsl_last_child (
            lsl_xml_ptr e )
```

Get the last child of the element.

9.5.4.42 lsl_library_info()

```
LIBLSL_C_API const char* lsl_library_info ( )
```

Get a string containing library information. The format of the string shouldn't be used for anything important except giving a a debugging person a good idea which exact library version is used.

9.5.4.43 lsl_library_version()

```
LIBLSL_C_API int32_t lsl_library_version ( )
```

Version of the liblsl library. The major version is library_version() / 100; The minor version is library_version() % 100;

9.5.4.44 lsl_local_clock()

```
LIBLSL_C_API double lsl_local_clock ( )
```

Obtain a local system time stamp in seconds. The resolution is better than a millisecond. This reading can be used to assign time stamps to samples as they are being acquired. If the "age" of a sample is known at a particular time (e.g., from USB transmission delays), it can be used as an offset to lsl_local_clock() to obtain a better estimate of when a sample was actually captured. See lsl_push_sample() for a use case.

9.5.4.45 lsl_name()

```
LIBLSL_C_API const char* lsl_name (
             lsl_xml_ptr e )
```

Name of the element.

9.5.4.46 lsl_next_sibling()

```
LIBLSL_C_API lsl_xml_ptr lsl_next_sibling (
             lsl_xml_ptr e )
```

Get the next sibling in the children list of the parent node.

9.5.4.47 lsl_next_sibling_n()

```
LIBLSL_C_API lsl_xml_ptr lsl_next_sibling_n (
             lsl_xml_ptr e,
             const char * name )
```

Get the next sibling with the specified name.

9.5.4.48 lsl_open_stream()

```
LIBLSL_C_API void lsl_open_stream (
             lsl_inlet in,
             double timeout,
             int32_t * ec )
```

Subscribe to the data stream. All samples pushed in at the other end from this moment onwards will be queued and eventually be delivered in response to pull_sample() calls. Pulling a sample without some preceding lsl_open_↩ stream() is permitted (the stream will then be opened implicitly).

Parameters

in	The lsl_inlet object to act on.
timeout	Optional timeout of the operation. Use LSL_FOREVER to effectively disable it.
ec	Error code: if nonzero, can be either lsl_timeout_error (if the timeout has expired) or lsl_lost_error (if the stream source has been lost).

9.5.4.49 lsl_parent()

```
LIBLSL_C_API lsl_xml_ptr lsl_parent (
             lsl_xml_ptr e )
```

Get the parent node.

9.5.4.50 lsl_prepend_child()

```
LIBLSL_C_API lsl_xml_ptr lsl_prepend_child (
             lsl_xml_ptr e,
             const char * name )
```

Prepend a child element with the specified name.

9.5.4.51 lsl_prepend_child_value()

```
LIBLSL_C_API lsl_xml_ptr lsl_prepend_child_value (
             lsl_xml_ptr e,
             const char * name,
             const char * value )
```

Prepend a child node with a given name, which has a (nameless) plain-text child with the given text value.

9.5.4.52 lsl_prepend_copy()

```
LIBLSL_C_API lsl_xml_ptr lsl_prepend_copy (
             lsl_xml_ptr e,
             lsl_xml_ptr e2 )
```

Prepend a child element with the specified name.

9.5.4.53 lsl_previous_sibling()

```
LIBLSL_C_API lsl_xml_ptr lsl_previous_sibling (
             lsl_xml_ptr e )
```

Get the previous sibling in the children list of the parent node.

9.5.4.54 lsl_previous_sibling_n()

```
LIBLSL_C_API lsl_xml_ptr lsl_previous_sibling_n (
             lsl_xml_ptr e,
             const char * name )
```

Get the previous sibling with the specified name.

9.5.4.55 lsl_protocol_version()

```
LIBLSL_C_API int32_t lsl_protocol_version ( )
```

Protocol version. The major version is protocol_version() / 100; The minor version is protocol_version() % 100; Clients with different minor versions are protocol-compatible with each other while clients with different major versions will refuse to work together.

9.5.4.56 lsl_pull_chunk_buf()

```
LIBLSL_C_API unsigned long lsl_pull_chunk_buf (
             lsl_inlet in,
             char ** data_buffer,
             uint32_t * lengths_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

Pull a chunk of data from the inlet and read it into an array of binary strings. These strings may contains 0's, therefore the lengths are read into the lengths_buffer array. Handles type checking & conversion. IMPORTANT: Note that the provided data buffer size is measured in channel values (e.g., floats) rather than in samples.

Parameters

in	The lsl_inlet object to act on.
data_buffer	A pointer to a buffer of data values where the results shall be stored.
lengths_buffer	A pointer to an array that holds the resulting lengths for each returned binary string.
timestamp_buffer	A pointer to a buffer of timestamp values where time stamps shall be stored. If this is NULL, no time stamps will be returned.
data_buffer_elements	The size of the data buffer, in channel data elements (of type T). Must be a multiple of the stream's channel count.
timestamp_buffer_elements	The size of the timestamp buffer. If a timestamp buffer is provided then this must correspond to the same number of samples as data_buffer_elements.
timeout	The timeout for this operation, if any. When the timeout expires, the function may return before the entire buffer is filled. The default value of 0.0 will retrieve only data available for immediate pickup.
ec	Error code: can be either no error or lsl_lost_error (if the stream source has been lost). Note: if the timeout expires before a new sample was received the function returns 0.0; ec is *not* set to lsl_timeout_error (because this case is not considered an error condition).

Returns

data_elements_written Number of channel data elements written to the data buffer.

9.5.4.57 lsl_pull_chunk_c()

```
LIBLSL_C_API unsigned long lsl_pull_chunk_c (
             lsl_inlet in,
             char * data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

9.5.4.58 lsl_pull_chunk_d()

```
LIBLSL_C_API unsigned long lsl_pull_chunk_d (
             lsl_inlet in,
             double * data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

9.5.4.59 lsl_pull_chunk_f()

```
LIBLSL_C_API unsigned long lsl_pull_chunk_f (
             lsl_inlet in,
             float * data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

Pull a chunk of data from the inlet and read it into a buffer. Handles type checking & conversion. IMPORTANT: Note that the provided data buffer size is measured in channel values (e.g., floats) rather than in samples.

Parameters

in	The lsl_inlet object to act on.
data_buffer	A pointer to a buffer of data values where the results shall be stored.
timestamp_buffer	A pointer to a buffer of timestamp values where time stamps shall be stored. If this is NULL, no time stamps will be returned.
data_buffer_elements	The size of the data buffer, in channel data elements (of type T). Must be a multiple of the stream's channel count.
timestamp_buffer_elements	The size of the timestamp buffer. If a timestamp buffer is provided then this must correspond to the same number of samples as data_buffer_elements.
timeout	The timeout for this operation, if any. When the timeout expires, the function may return before the entire buffer is filled. The default value of 0.0 will retrieve only data available for immediate pickup.
ec	Error code: can be either no error or lsl_lost_error (if the stream source has been lost). Note: if the timeout expires before a new sample was received the function returns 0.0; ec is *not* set to lsl_timeout_error (because this case is not considered an error condition).

Returns

data_elements_written Number of channel data elements written to the data buffer.

9.5.4.60 lsl_pull_chunk_i()

```
LIBLSL_C_API unsigned long lsl_pull_chunk_i (
             lsl_inlet in,
             int32_t * data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

9.5.4.61 lsl_pull_chunk_l()

```
LIBLSL_C_API unsigned long lsl_pull_chunk_l (
             lsl_inlet in,
             long * data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int * ec )
```

9.5.4.62 lsl_pull_chunk_s()

```
LIBLSL_C_API unsigned long lsl_pull_chunk_s (
             lsl_inlet in,
             int16_t * data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

9.5.4.63 lsl_pull_chunk_str()

```
LIBLSL_C_API unsigned long lsl_pull_chunk_str (
             lsl_inlet in,
             char ** data_buffer,
             double * timestamp_buffer,
             unsigned long data_buffer_elements,
             unsigned long timestamp_buffer_elements,
             double timeout,
             int32_t * ec )
```

9.5.4.64 lsl_pull_sample_buf()

```
LIBLSL_C_API double lsl_pull_sample_buf (
            lsl_inlet in,
            char ** buffer,
            uint32_t * buffer_lengths,
            int32_t buffer_elements,
            double timeout,
            int32_t * ec )
```

Pull a sample from the inlet and read it into an array of binary strings. These strings may contains 0's, therefore the lengths are read into the buffer_lengths array. Handles type checking & conversion.

Parameters

in	The lsl_inlet object to act on.
buffer	A pointer to hold the resulting data.
buffer_lengths	A pointer to an array that holds the resulting lengths for each returned binary string.
buffer_elements	The number of samples allocated in the buffer and buffer_lengths variables. Note: it is the responsibility of the user to allocate enough memory.
timeout	The timeout for this operation, if any. Use LSL_FOREVER to effectively disable it. It is also permitted to use 0.0 here; in this case a sample is only returned if one is currently buffered.
ec	Error code: can be either no error or lsl_lost_error (if the stream source has been lost). Note: if the timeout expires before a new sample was received the function returns 0.0; ec is *not* set to lsl_timeout_error (because this case is not considered an error condition).

Returns

The capture time of the sample on the remote machine, or 0.0 if no new sample was available. To remap this time stamp to the local clock, add the value returned by lsl_time_correction() to it.

9.5.4.65 lsl_pull_sample_c()

```
LIBLSL_C_API double lsl_pull_sample_c (
            lsl_inlet in,
            char * buffer,
            int32_t buffer_elements,
            double timeout,
            int32_t * ec )
```

9.5.4.66 lsl_pull_sample_d()

```
LIBLSL_C_API double lsl_pull_sample_d (
            lsl_inlet in,
            double * buffer,
            int32_t buffer_elements,
            double timeout,
            int32_t * ec )
```

9.5.4.67 lsl_pull_sample_f()

```
LIBLSL_C_API double lsl_pull_sample_f (
              lsl_inlet in,
              float * buffer,
              int32_t buffer_elements,
              double timeout,
              int32_t * ec )
```

Pull a sample from the inlet and read it into a pointer to values. Handles type checking & conversion.

Parameters

in	The lsl_inlet object to act on.
buffer	A pointer to hold the resulting values.
buffer_elements	The number of samples allocated in the buffer. Note: it is the responsibility of the user to allocate enough memory.
timeout	The timeout for this operation, if any. Use LSL_FOREVER to effectively disable it. It is also permitted to use 0.0 here; in this case a sample is only returned if one is currently buffered.
ec	Error code: can be either no error or lsl_lost_error (if the stream source has been lost). Note: if the timeout expires before a new sample was received the function returns 0.0; ec is *not* set to lsl_timeout_error (because this case is not considered an error condition).

Returns

The capture time of the sample on the remote machine, or 0.0 if no new sample was available. To remap this time stamp to the local clock, add the value returned by lsl_time_correction() to it.

9.5.4.68 lsl_pull_sample_i()

```
LIBLSL_C_API double lsl_pull_sample_i (
              lsl_inlet in,
              int32_t * buffer,
              int32_t buffer_elements,
              double timeout,
              int32_t * ec )
```

9.5.4.69 lsl_pull_sample_l()

```
LIBLSL_C_API double lsl_pull_sample_l (
              lsl_inlet in,
              long * buffer,
              int buffer_elements,
              double timeout,
              int * ec )
```

9.5.4.70 lsl_pull_sample_s()

```
LIBLSL_C_API double lsl_pull_sample_s (
             lsl_inlet in,
             int16_t * buffer,
             int32_t buffer_elements,
             double timeout,
             int32_t * ec )
```

9.5.4.71 lsl_pull_sample_str()

```
LIBLSL_C_API double lsl_pull_sample_str (
             lsl_inlet in,
             char ** buffer,
             int32_t buffer_elements,
             double timeout,
             int32_t * ec )
```

9.5.4.72 lsl_pull_sample_v()

```
LIBLSL_C_API double lsl_pull_sample_v (
             lsl_inlet in,
             void * buffer,
             int32_t buffer_bytes,
             double timeout,
             int32_t * ec )
```

Pull a sample from the inlet and read it into a custom struct or buffer. Overall size checking but no type checking or conversion are done. Do not use for variable-size/string-formatted streams.

Parameters

in	The lsl_inlet object to act on.
buffer	Pointer to hold the sample data. Search for #pragma pack for information on how to pack structs appropriately.
buffer_bytes	Length of the array held by buffer in bytes, not items
timeout	The timeout for this operation, if any. Aside from LSL_FOREVER it is also permitted to use 0.0 here; in this case a sample is only returned if one is currently buffered.
ec	Error code: can be either no error or lsl_lost_error (if the stream source has been lost). Note: if the timeout expires before a new sample was received the function returns 0.0; ec is *not* set to lsl_timeout_error (because this case is not considered an error condition).

Returns

The capture time of the sample on the remote machine, or 0.0 if no new sample was available. To remap this time stamp to the local clock, add the value returned by .time_correction() to it.

9.5.4.73 lsl_push_chunk_buf()

```
LIBLSL_C_API int32_t lsl_push_chunk_buf (
            lsl_outlet out,
            const char ** data,
            const uint32_t * lengths,
            unsigned long data_elements )
```

9.5.4.74 lsl_push_chunk_buft()

```
LIBLSL_C_API int32_t lsl_push_chunk_buft (
            lsl_outlet out,
            const char ** data,
            const uint32_t * lengths,
            unsigned long data_elements,
            double timestamp )
```

9.5.4.75 lsl_push_chunk_buftn()

```
LIBLSL_C_API int32_t lsl_push_chunk_buftn (
            lsl_outlet out,
            const char ** data,
            const uint32_t * lengths,
            unsigned long data_elements,
            const double * timestamps )
```

9.5.4.76 lsl_push_chunk_buftnp()

```
LIBLSL_C_API int32_t lsl_push_chunk_buftnp (
            lsl_outlet out,
            const char ** data,
            const uint32_t * lengths,
            unsigned long data_elements,
            const double * timestamps,
            int32_t pushthrough )
```

9.5.4.77 lsl_push_chunk_buftp()

```
LIBLSL_C_API int32_t lsl_push_chunk_buftp (
            lsl_outlet out,
            const char ** data,
            const uint32_t * lengths,
            unsigned long data_elements,
            double timestamp,
            int32_t pushthrough )
```

9.5.4.78 lsl_push_chunk_c()

```
LIBLSL_C_API int32_t lsl_push_chunk_c (
            lsl_outlet out,
            const char * data,
            unsigned long data_elements )
```

9.5.4.79 lsl_push_chunk_ct()

```
LIBLSL_C_API int32_t lsl_push_chunk_ct (
            lsl_outlet out,
            const char * data,
            unsigned long data_elements,
            double timestamp )
```

9.5.4.80 lsl_push_chunk_ctn()

```
LIBLSL_C_API int32_t lsl_push_chunk_ctn (
            lsl_outlet out,
            const char * data,
            unsigned long data_elements,
            const double * timestamps )
```

9.5.4.81 lsl_push_chunk_ctnp()

```
LIBLSL_C_API int32_t lsl_push_chunk_ctnp (
            lsl_outlet out,
            const char * data,
            unsigned long data_elements,
            const double * timestamps,
            int32_t pushthrough )
```

9.5.4.82 lsl_push_chunk_ctp()

```
LIBLSL_C_API int32_t lsl_push_chunk_ctp (
            lsl_outlet out,
            const char * data,
            unsigned long data_elements,
            double timestamp,
            int32_t pushthrough )
```

9.5.4.83 lsl_push_chunk_d()

```
LIBLSL_C_API int32_t lsl_push_chunk_d (
            lsl_outlet out,
            const double * data,
            unsigned long data_elements )
```

9.5.4.84 lsl_push_chunk_dt()

```
LIBLSL_C_API int32_t lsl_push_chunk_dt (
            lsl_outlet out,
            const double * data,
            unsigned long data_elements,
            double timestamp )
```

9.5.4.85 lsl_push_chunk_dtn()

```
LIBLSL_C_API int32_t lsl_push_chunk_dtn (
            lsl_outlet out,
            const double * data,
            unsigned long data_elements,
            const double * timestamps )
```

9.5.4.86 lsl_push_chunk_dtnp()

```
LIBLSL_C_API int32_t lsl_push_chunk_dtnp (
            lsl_outlet out,
            const double * data,
            unsigned long data_elements,
            const double * timestamps,
            int32_t pushthrough )
```

9.5.4.87 lsl_push_chunk_dtp()

```
LIBLSL_C_API int32_t lsl_push_chunk_dtp (
            lsl_outlet out,
            const double * data,
            unsigned long data_elements,
            double timestamp,
            int32_t pushthrough )
```

9.5.4.88 lsl_push_chunk_f()

```
LIBLSL_C_API int32_t lsl_push_chunk_f (
            lsl_outlet out,
            const float * data,
            unsigned long data_elements )
```

Push a chunk of multiplexed samples into the outlet. One timestamp per sample is provided. IMPORTANT: Note that the provided buffer size is measured in channel values (e.g., floats) rather than in samples. Handles type checking & conversion.

Parameters

out	The lsl_outlet object through which to push the data.
data	A buffer of channel values holding the data for zero or more successive samples to send.
lengths	For lsl_push_chunk_buf∗, a pointer the number of elements to push for each value (string lengths).
timestamp	Optionally the capture time of the most recent sample, in agreement with local_clock(); if omitted, the current time is used. The time stamps of other samples are automatically derived based on the sampling rate of the stream.
timestamps	Alternatively a buffer of timestamp values holding time stamps for each sample in the data buffer.
data_elements	The number of data values (of type T) in the data buffer. Must be a multiple of the channel count.
pushthrough	Whether to push the chunk through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

Returns

Error code of the operation (usually attributed to the wrong data type).

9.5.4.89 lsl_push_chunk_ft()

```
LIBLSL_C_API int32_t lsl_push_chunk_ft (
            lsl_outlet out,
            const float * data,
            unsigned long data_elements,
            double timestamp )
```

9.5.4.90 lsl_push_chunk_ftn()

```
LIBLSL_C_API int32_t lsl_push_chunk_ftn (
            lsl_outlet out,
            const float * data,
            unsigned long data_elements,
            const double * timestamps )
```

9.5.4.91 lsl_push_chunk_ftnp()

```
LIBLSL_C_API int32_t lsl_push_chunk_ftnp (
             lsl_outlet out,
             const float * data,
             unsigned long data_elements,
             const double * timestamps,
             int32_t pushthrough )
```

9.5.4.92 lsl_push_chunk_ftp()

```
LIBLSL_C_API int32_t lsl_push_chunk_ftp (
             lsl_outlet out,
             const float * data,
             unsigned long data_elements,
             double timestamp,
             int32_t pushthrough )
```

9.5.4.93 lsl_push_chunk_i()

```
LIBLSL_C_API int32_t lsl_push_chunk_i (
             lsl_outlet out,
             const int32_t * data,
             unsigned long data_elements )
```

9.5.4.94 lsl_push_chunk_it()

```
LIBLSL_C_API int32_t lsl_push_chunk_it (
             lsl_outlet out,
             const int32_t * data,
             unsigned long data_elements,
             double timestamp )
```

9.5.4.95 lsl_push_chunk_itn()

```
LIBLSL_C_API int32_t lsl_push_chunk_itn (
             lsl_outlet out,
             const int32_t * data,
             unsigned long data_elements,
             const double * timestamps )
```

9.5.4.96 lsl_push_chunk_itnp()

```
LIBLSL_C_API int32_t lsl_push_chunk_itnp (
            lsl_outlet out,
            const int32_t * data,
            unsigned long data_elements,
            const double * timestamps,
            int32_t pushthrough )
```

9.5.4.97 lsl_push_chunk_itp()

```
LIBLSL_C_API int32_t lsl_push_chunk_itp (
            lsl_outlet out,
            const int32_t * data,
            unsigned long data_elements,
            double timestamp,
            int32_t pushthrough )
```

9.5.4.98 lsl_push_chunk_l()

```
LIBLSL_C_API int lsl_push_chunk_l (
            lsl_outlet out,
            const long * data,
            unsigned long data_elements )
```

9.5.4.99 lsl_push_chunk_lt()

```
LIBLSL_C_API int lsl_push_chunk_lt (
            lsl_outlet out,
            const long * data,
            unsigned long data_elements,
            double timestamp )
```

9.5.4.100 lsl_push_chunk_ltn()

```
LIBLSL_C_API int lsl_push_chunk_ltn (
            lsl_outlet out,
            const long * data,
            unsigned long data_elements,
            const double * timestamps )
```

9.5.4.101 lsl_push_chunk_ltnp()

```
LIBLSL_C_API int lsl_push_chunk_ltnp (
            lsl_outlet out,
            const long * data,
            unsigned long data_elements,
            const double * timestamps,
            int pushthrough )
```

9.5.4.102 lsl_push_chunk_ltp()

```
LIBLSL_C_API int lsl_push_chunk_ltp (
            lsl_outlet out,
            const long * data,
            unsigned long data_elements,
            double timestamp,
            int pushthrough )
```

9.5.4.103 lsl_push_chunk_s()

```
LIBLSL_C_API int32_t lsl_push_chunk_s (
            lsl_outlet out,
            const int16_t * data,
            unsigned long data_elements )
```

9.5.4.104 lsl_push_chunk_st()

```
LIBLSL_C_API int32_t lsl_push_chunk_st (
            lsl_outlet out,
            const int16_t * data,
            unsigned long data_elements,
            double timestamp )
```

9.5.4.105 lsl_push_chunk_stn()

```
LIBLSL_C_API int32_t lsl_push_chunk_stn (
            lsl_outlet out,
            const int16_t * data,
            unsigned long data_elements,
            const double * timestamps )
```

9.5.4.106 lsl_push_chunk_stnp()

```
LIBLSL_C_API int32_t lsl_push_chunk_stnp (
             lsl_outlet out,
             const int16_t * data,
             unsigned long data_elements,
             const double * timestamps,
             int32_t pushthrough )
```

9.5.4.107 lsl_push_chunk_stp()

```
LIBLSL_C_API int32_t lsl_push_chunk_stp (
             lsl_outlet out,
             const int16_t * data,
             unsigned long data_elements,
             double timestamp,
             int32_t pushthrough )
```

9.5.4.108 lsl_push_chunk_str()

```
LIBLSL_C_API int32_t lsl_push_chunk_str (
             lsl_outlet out,
             const char ** data,
             unsigned long data_elements )
```

9.5.4.109 lsl_push_chunk_strt()

```
LIBLSL_C_API int32_t lsl_push_chunk_strt (
             lsl_outlet out,
             const char ** data,
             unsigned long data_elements,
             double timestamp )
```

9.5.4.110 lsl_push_chunk_strtn()

```
LIBLSL_C_API int32_t lsl_push_chunk_strtn (
             lsl_outlet out,
             const char ** data,
             unsigned long data_elements,
             const double * timestamps )
```

9.5.4.111 lsl_push_chunk_strtnp()

```
LIBLSL_C_API int32_t lsl_push_chunk_strtnp (
             lsl_outlet out,
             const char ** data,
             unsigned long data_elements,
             const double * timestamps,
             int32_t pushthrough )
```

9.5.4.112 lsl_push_chunk_strtp()

```
LIBLSL_C_API int32_t lsl_push_chunk_strtp (
             lsl_outlet out,
             const char ** data,
             unsigned long data_elements,
             double timestamp,
             int32_t pushthrough )
```

9.5.4.113 lsl_push_sample_buf()

```
LIBLSL_C_API int32_t lsl_push_sample_buf (
             lsl_outlet out,
             const char ** data,
             const uint32_t * lengths )
```

9.5.4.114 lsl_push_sample_buft()

```
LIBLSL_C_API int32_t lsl_push_sample_buft (
             lsl_outlet out,
             const char ** data,
             const uint32_t * lengths,
             double timestamp )
```

9.5.4.115 lsl_push_sample_buftp()

```
LIBLSL_C_API int32_t lsl_push_sample_buftp (
             lsl_outlet out,
             const char ** data,
             const uint32_t * lengths,
             double timestamp,
             int32_t pushthrough )
```

9.5.4.116 lsl_push_sample_c()

```
LIBLSL_C_API int32_t lsl_push_sample_c (
            lsl_outlet out,
            const char * data )
```

9.5.4.117 lsl_push_sample_ct()

```
LIBLSL_C_API int32_t lsl_push_sample_ct (
            lsl_outlet out,
            const char * data,
            double timestamp )
```

9.5.4.118 lsl_push_sample_ctp()

```
LIBLSL_C_API int32_t lsl_push_sample_ctp (
            lsl_outlet out,
            const char * data,
            double timestamp,
            int32_t pushthrough )
```

9.5.4.119 lsl_push_sample_d()

```
LIBLSL_C_API int32_t lsl_push_sample_d (
            lsl_outlet out,
            const double * data )
```

9.5.4.120 lsl_push_sample_dt()

```
LIBLSL_C_API int32_t lsl_push_sample_dt (
            lsl_outlet out,
            const double * data,
            double timestamp )
```

9.5.4.121 lsl_push_sample_dtp()

```
LIBLSL_C_API int32_t lsl_push_sample_dtp (
            lsl_outlet out,
            const double * data,
            double timestamp,
            int32_t pushthrough )
```

9.5.4.122 lsl_push_sample_f()

```
LIBLSL_C_API int32_t lsl_push_sample_f (
             lsl_outlet out,
             const float * data )
```

Push a pointer to some values as a sample into the outlet. Handles type checking & conversion.

Parameters

out	The lsl_outlet object through which to push the data.
data	A pointer to values to push. The number of values pointed to must be no less than the number of channels in the sample.
lengths	For lsl_push_sample_buf∗, a pointer the number of elements to push for each channel (string lengths).
timestamp	Optionally the capture time of the sample, in agreement with lsl_local_clock(); if omitted, the current time is used.
pushthrough	Whether to push the sample through to the receivers instead of buffering it with subsequent samples. Note that the chunk_size, if specified at outlet construction, takes precedence over the pushthrough flag.

Returns

Error code of the operation or lsl_no_error if successful (usually attributed to the wrong data type).

9.5.4.123 lsl_push_sample_ft()

```
LIBLSL_C_API int32_t lsl_push_sample_ft (
             lsl_outlet out,
             const float * data,
             double timestamp )
```

9.5.4.124 lsl_push_sample_ftp()

```
LIBLSL_C_API int32_t lsl_push_sample_ftp (
             lsl_outlet out,
             const float * data,
             double timestamp,
             int32_t pushthrough )
```

9.5.4.125 lsl_push_sample_i()

```
LIBLSL_C_API int32_t lsl_push_sample_i (
             lsl_outlet out,
             const int32_t * data )
```

9.5.4.126 lsl_push_sample_it()

```
LIBLSL_C_API int32_t lsl_push_sample_it (
             lsl_outlet out,
             const int32_t * data,
             double timestamp )
```

9.5.4.127 lsl_push_sample_itp()

```
LIBLSL_C_API int32_t lsl_push_sample_itp (
             lsl_outlet out,
             const int32_t * data,
             double timestamp,
             int32_t pushthrough )
```

9.5.4.128 lsl_push_sample_l()

```
LIBLSL_C_API int32_t lsl_push_sample_l (
             lsl_outlet out,
             const long * data )
```

9.5.4.129 lsl_push_sample_lt()

```
LIBLSL_C_API int32_t lsl_push_sample_lt (
             lsl_outlet out,
             const long * data,
             double timestamp )
```

9.5.4.130 lsl_push_sample_ltp()

```
LIBLSL_C_API int32_t lsl_push_sample_ltp (
             lsl_outlet out,
             const long * data,
             double timestamp,
             int32_t pushthrough )
```

9.5.4.131 lsl_push_sample_s()

```
LIBLSL_C_API int32_t lsl_push_sample_s (
             lsl_outlet out,
             const int16_t * data )
```

9.5.4.132 lsl_push_sample_st()

```
LIBLSL_C_API int32_t lsl_push_sample_st (
             lsl_outlet out,
             const int16_t * data,
             double timestamp )
```

9.5.4.133 lsl_push_sample_stp()

```
LIBLSL_C_API int32_t lsl_push_sample_stp (
             lsl_outlet out,
             const int16_t * data,
             double timestamp,
             int32_t pushthrough )
```

9.5.4.134 lsl_push_sample_str()

```
LIBLSL_C_API int32_t lsl_push_sample_str (
             lsl_outlet out,
             const char ** data )
```

9.5.4.135 lsl_push_sample_strt()

```
LIBLSL_C_API int32_t lsl_push_sample_strt (
             lsl_outlet out,
             const char ** data,
             double timestamp )
```

9.5.4.136 lsl_push_sample_strtp()

```
LIBLSL_C_API int32_t lsl_push_sample_strtp (
             lsl_outlet out,
             const char ** data,
             double timestamp,
             int32_t pushthrough )
```

9.5.4.137 lsl_push_sample_v()

```
LIBLSL_C_API int32_t lsl_push_sample_v (
             lsl_outlet out,
             const void * data )
```

9.5.4.138 lsl_push_sample_vt()

```
LIBLSL_C_API int32_t lsl_push_sample_vt (
             lsl_outlet out,
             const void * data,
             double timestamp )
```

9.5.4.139 lsl_push_sample_vtp()

```
LIBLSL_C_API int32_t lsl_push_sample_vtp (
             lsl_outlet out,
             const void * data,
             double timestamp,
             int32_t pushthrough )
```

9.5.4.140 lsl_remove_child()

```
LIBLSL_C_API void lsl_remove_child (
             lsl_xml_ptr e,
             lsl_xml_ptr e2 )
```

Remove a specified child element.

9.5.4.141 lsl_remove_child_n()

```
LIBLSL_C_API void lsl_remove_child_n (
             lsl_xml_ptr e,
             const char * name )
```

Remove a child element with the specified name.

9.5.4.142 lsl_resolve_all()

```
LIBLSL_C_API int32_t lsl_resolve_all (
             lsl_streaminfo * buffer,
             uint32_t buffer_elements,
             double wait_time )
```

Resolve all streams on the network. This function returns all currently available streams from any outlet on the network. The network is usually the subnet specified at the local router, but may also include a multicast group of machines (given that the network supports it), or a list of hostnames. These details may optionally be customized by the experimenter in a configuration file (see page Network Connectivity in the LSL wiki). This is the default mechanism used by the browsing programs and the recording program.

Parameters

buffer	A user-allocated buffer to hold the resolve results. Note: it is the user's responsibility to either destroy the resulting streaminfo objects or to pass them back to the LSL during during creation of an inlet. Note 2: The stream_info's returned by the resolver are only short versions that do not include the .desc() field (which can be arbitrarily big). To obtain the full stream information you need to call .info() on the inlet after you have created one.
buffer_elements	The user-provided buffer length.
wait_time	The waiting time for the operation, in seconds, to search for streams. The recommended wait time is 1 second (or 2 for a busy and large recording operation). Warning: If this is too short (<0.5s) only a subset (or none) of the outlets that are present on the network may be returned.

Returns

The number of results written into the buffer (never more than the provided # of slots) or a negative number if an error has occurred (values corresponding to lsl_error_code_t).

9.5.4.143 lsl_resolve_bypred()

```
LIBLSL_C_API int32_t lsl_resolve_bypred (
            lsl_streaminfo * buffer,
            uint32_t buffer_elements,
            const char * pred,
            int32_t minimum,
            double timeout )
```

Resolve all streams that match a given predicate. Advanced query that allows to impose more conditions on the retrieved streams; the given string is an XPath 1.0 predicate for the <info> node (omitting the surrounding []'s), see also http://en.wikipedia.org/w/index.php?title=XPath_1.0&oldid=474981951.

Parameters

buffer	A user-allocated buffer to hold the resolve results. Note: it is the user's responsibility to either destroy the resulting streaminfo objects or to pass them back to the LSL during during creation of an inlet. Note 2: The stream_info's returned by the resolver are only short versions that do not include the .desc() field (which can be arbitrarily big). To obtain the full stream information you need to call .info() on the inlet after you have created one.
buffer_elements	The user-provided buffer length.
pred	The predicate string, e.g. "name='BioSemi'" or "type='EEG' and starts-with(name,'BioSemi') and count(info/desc/channel)=32"
minimum	Return at least this number of streams.
timeout	Optionally a timeout of the operation, in seconds (default: no timeout). If the timeout expires, less than the desired number of streams (possibly none) will be returned.

Returns

The number of results written into the buffer (never more than the provided # of slots) or a negative number if an error has occurred (values corresponding to lsl_error_code_t).

9.5.4.144 lsl_resolve_byprop()

```
LIBLSL_C_API int32_t lsl_resolve_byprop (
             lsl_streaminfo * buffer,
             uint32_t buffer_elements,
             const char * prop,
             const char * value,
             int32_t minimum,
             double timeout )
```

Resolve all streams with a given value for a property. If the goal is to resolve a specific stream, this method is preferred over resolving all streams and then selecting the desired one.

Parameters

buffer	A user-allocated buffer to hold the resolve results. Note: it is the user's responsibility to either destroy the resulting streaminfo objects or to pass them back to the LSL during during creation of an inlet. Note 2: The stream_info's returned by the resolver are only short versions that do not include the .desc() field (which can be arbitrarily big). To obtain the full stream information you need to call .info() on the inlet after you have created one.
buffer_elements	The user-provided buffer length.
prop	The streaminfo property that should have a specific value ("name", "type", "source_id", or, e.g., "desc/manufaturer" if present).
value	The string value that the property should have (e.g., "EEG" as the type).
minimum	Return at least this number of streams.
timeout	Optionally a timeout of the operation, in seconds (default: no timeout). If the timeout expires, less than the desired number of streams (possibly none) will be returned.

Returns

The number of results written into the buffer (never more than the provided # of slots) or a negative number if an error has occurred (values corresponding to lsl_error_code_t).

9.5.4.145 lsl_resolver_results()

```
LIBLSL_C_API int32_t lsl_resolver_results (
             lsl_continuous_resolver res,
             lsl_streaminfo * buffer,
             uint32_t buffer_elements )
```

Obtain the set of currently present streams on the network (i.e. resolve result).

Parameters

res	A continuous resolver (previously created with one of the lsl_create_continuous_resolver functions).
buffer	A user-allocated buffer to hold the current resolve results. Note: it is the user's responsibility to either destroy the resulting streaminfo objects or to pass them back to the LSL during during creation of an inlet. Note 2: The stream_info's returned by the resolver are only short versions that do not include the .desc() field (which can be arbitrarily big). To obtain the full stream information you need to call .info() on the inlet after you have created one.
buffer_elements	The user-provided buffer length.

Returns

The number of results written into the buffer (never more than the provided # of slots) or a negative number if an error has occurred (values corresponding to lsl_error_code_t).

9.5.4.146 lsl_samples_available()

```
LIBLSL_C_API uint32_t lsl_samples_available (
             lsl_inlet in )
```

Query whether samples are currently available for immediate pickup. Note that it is not a good idea to use samples↩
available() to determine whether a pull∗() call would block: to be sure, set the pull timeout to 0.0 or an acceptably low value. If the underlying implementation supports it, the value will be the number of samples available (otherwise it will be 1 or 0).

9.5.4.147 lsl_set_child_value()

```
LIBLSL_C_API int32_t lsl_set_child_value (
             lsl_xml_ptr e,
             const char ∗ name,
             const char ∗ value )
```

Set the text value of the (nameless) plain-text child of a named child node.

9.5.4.148 lsl_set_name()

```
LIBLSL_C_API int32_t lsl_set_name (
             lsl_xml_ptr e,
             const char ∗ rhs )
```

Set the element's name.

Returns

0 if the node is empty (or if out of memory).

9.5.4.149 lsl_set_postprocessing()

```
LIBLSL_C_API int32_t lsl_set_postprocessing (
             lsl_inlet in,
             uint32_t flags )
```

Set post-processing flags to use. By default, the inlet performs NO post-processing and returns the ground-truth time stamps, which can then be manually synchronized using time_correction(), and then smoothed/dejittered if desired. This function allows automating these two and possibly more operations. Warning: when you enable this, you will no longer receive or be able to recover the original time stamps.

Parameters

in	The lsl_inlet object to act on.
flags	An integer that is the result of bitwise OR'ing one or more options from processing_options_t together (e.g., post_clocksync|post_dejitter); a good setting is to use post_ALL.

Returns

The error code: if nonzero, can be lsl_argument_error if an unknown flag was passed in.

9.5.4.150 lsl_set_value()

```
LIBLSL_C_API int32_t lsl_set_value (
             lsl_xml_ptr e,
             const char * rhs )
```

Set the element's value.

Returns

0 if the node is empty (or if out of memory).

9.5.4.151 lsl_smoothing_halftime()

```
LIBLSL_C_API int32_t lsl_smoothing_halftime (
             lsl_inlet in,
             float value )
```

Override the half-time (forget factor) of the time-stamp smoothing. The default is 90 seconds unless a different value is set in the config file. Using a longer window will yield lower jitter in the time stamps, but longer windows will have trouble tracking changes in the clock rate (usually due to temperature changes); the default is able to track changes up to 10 degrees C per minute sufficiently well.

Parameters

in	The lsl_inlet object to act on.
value	The new value, in seconds. This is the time after which a past sample will be weighted by 1/2 in the exponential smoothing window.

Returns

The error code: if nonzero, can be lsl_argument_error if an unknown flag was passed in.

9.5.4.152 lsl_stream_info_matches_query()

```
LIBLSL_C_API int lsl_stream_info_matches_query (
            lsl_streaminfo info,
            const char * query )
```

Tries to match the stream info XML element `info` against an `XPath` query.

Example query strings:

```
channel_count>5 and type='EEG'
type='TestStream' or contains(name,'Brain')
name='ExampleStream'
```

9.5.4.153 lsl_streaminfo_from_xml()

```
LIBLSL_C_API lsl_streaminfo lsl_streaminfo_from_xml (
            const char * xml )
```

Create a streaminfo object from an XML representation.

9.5.4.154 lsl_time_correction()

```
LIBLSL_C_API double lsl_time_correction (
            lsl_inlet in,
            double timeout,
            int32_t * ec )
```

Retrieve an estimated time correction offset for the given stream. The first call to this function takes several milliseconds until a reliable first estimate is obtained. Subsequent calls are instantaneous (and rely on periodic background updates). On a well-behaved network, the precision of these estimates should be below 1 ms (empirically it is within +/-0.2 ms). To get a measure of whether the network is well-behaved, use lsl_time_correction_ex and check uncertainty (which maps to round-trip-time). 0.2 ms is typical of wired networks. 2 ms is typical of wireless networks. The number can be much higher on poor networks.

Parameters

in	The lsl_inlet object to act on.
remote_time	The current time of the remote computer that was used to generate this time_correction. If desired, the client can fit time_correction vs remote_time to improve the real-time time_correction further.
uncertainty.	The maximum uncertainty of the given time correction.
timeout	Timeout to acquire the first time-correction estimate. Use LSL_FOREVER to defuse the timeout.
ec	Error code: if nonzero, can be either lsl_timeout_error (if the timeout has expired) or lsl_lost_error (if the stream source has been lost).

Returns

The time correction estimate. This is the number that needs to be added to a time stamp that was remotely generated via lsl_local_clock() to map it into the local clock domain of this machine.

9.5.4.155 lsl_time_correction_ex()

```
LIBLSL_C_API double lsl_time_correction_ex (
             lsl_inlet in,
             double * remote_time,
             double * uncertainty,
             double timeout,
             int32_t * ec )
```

9.5.4.156 lsl_value()

```
LIBLSL_C_API const char* lsl_value (
             lsl_xml_ptr e )
```

Value of the element.

9.5.4.157 lsl_wait_for_consumers()

```
LIBLSL_C_API int32_t lsl_wait_for_consumers (
             lsl_outlet out,
             double timeout )
```

Wait until some consumer shows up (without wasting resources).

Returns

True if the wait was successful, false if the timeout expired.

9.5.4.158 lsl_was_clock_reset()

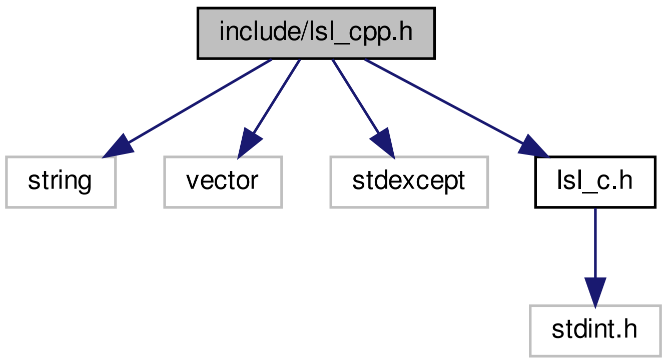
```
LIBLSL_C_API uint32_t lsl_was_clock_reset (
             lsl_inlet in )
```

Query whether the clock was potentially reset since the last call to was_clock_reset(). This is rarely-used function is only needed for applications that combine multiple time_correction values to estimate precise clock drift if they should tolerate cases where the source machine was hot-swapped or restarted.

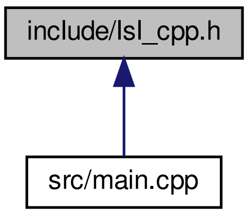
9.6 include/lsl_cpp.h File Reference

```
#include <string>
#include <vector>
#include <stdexcept>
#include "lsl_c.h"
```

Include dependency graph for lsl_cpp.h:



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



Classes

- class lsl::stream_info
- class lsl::stream_outlet
- class lsl::stream_inlet
- class lsl::xml_element
- class lsl::continuous_resolver
- class lsl::lost_error

  Exception class that indicates that a stream inlet's source has been irrecoverably lost.
- class lsl::timeout_error

  Exception class that indicates that an operation failed due to a timeout.

Namespaces

- lsl

Typedefs

- typedef struct lsl_streaminfo_struct_ ∗ lsl::lsl_streaminfo
- typedef struct lsl_outlet_struct_ ∗ lsl::lsl_outlet
- typedef struct lsl_inlet_struct_ ∗ lsl::lsl_inlet
- typedef struct lsl_xml_ptr_struct_ ∗ lsl::lsl_xml_ptr
- typedef struct lsl_continuous_resolver_ ∗ lsl::lsl_continuous_resolver

Enumerations

- enum lsl::lsl_channel_format_t {
  lsl::cft_float32 = 1, lsl::cft_double64 = 2, lsl::cft_string = 3, lsl::cft_int32 = 4,
  lsl::cft_int16 = 5, lsl::cft_int8 = 6, lsl::cft_int64 = 7, lsl::cft_undefined = 0 }
- enum lsl::lsl_processing_options_t {
  lsl::proc_none = 0, lsl::proc_clocksync = 1, lsl::proc_dejitter = 2, lsl::proc_monotonize = 4,
  lsl::proc_threadsafe = 8, lsl::proc_ALL = 1|2|4|8 }
- enum lsl::lsl_error_code_t {
  lsl::lsl_no_error = 0, lsl::lsl_timeout_error = -1, lsl::lsl_lost_error = -2, lsl::lsl_argument_error = -3,
  lsl::lsl_internal_error = -4 }
- enum lsl::channel_format_t {
  lsl::cf_float32 = 1, lsl::cf_double64 = 2, lsl::cf_string = 3, lsl::cf_int32 = 4,
  lsl::cf_int16 = 5, lsl::cf_int8 = 6, lsl::cf_int64 = 7, lsl::cf_undefined = 0 }
- enum lsl::processing_options_t {
  lsl::post_none = 0, lsl::post_clocksync = 1, lsl::post_dejitter = 2, lsl::post_monotonize = 4,
  lsl::post_threadsafe = 8, lsl::post_ALL = 1|2|4|8 }

Functions

- LIBLSL_C_API int32_t lsl::lsl_protocol_version ()
- LIBLSL_C_API int32_t lsl::lsl_library_version ()
- LIBLSL_C_API const char ∗ lsl::lsl_library_info ()
- LIBLSL_C_API double lsl::lsl_local_clock ()
- LIBLSL_C_API int32_t lsl::lsl_resolve_all (lsl_streaminfo ∗buffer, uint32_t buffer_elements, double wait_time)
- LIBLSL_C_API int32_t lsl::lsl_resolve_byprop (lsl_streaminfo ∗buffer, uint32_t buffer_elements, const char ∗prop, const char ∗value, int32_t minimum, double timeout)
- LIBLSL_C_API int32_t lsl::lsl_resolve_bypred (lsl_streaminfo ∗buffer, uint32_t buffer_elements, const char ∗pred, int32_t minimum, double timeout)
- LIBLSL_C_API void lsl::lsl_destroy_string (char ∗s)
- LIBLSL_C_API lsl_streaminfo lsl::lsl_create_streaminfo (const char ∗name, const char ∗type, int32_↩
  t channel_count, double nominal_srate, lsl_channel_format_t channel_format, const char ∗source_id)
- LIBLSL_C_API void lsl::lsl_destroy_streaminfo (lsl_streaminfo info)
- LIBLSL_C_API lsl_streaminfo lsl::lsl_copy_streaminfo (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl::lsl_get_name (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl::lsl_get_type (lsl_streaminfo info)
- LIBLSL_C_API int32_t lsl::lsl_get_channel_count (lsl_streaminfo info)
- LIBLSL_C_API double lsl::lsl_get_nominal_srate (lsl_streaminfo info)
- LIBLSL_C_API lsl_channel_format_t lsl::lsl_get_channel_format (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl::lsl_get_source_id (lsl_streaminfo info)
- LIBLSL_C_API int32_t lsl::lsl_get_version (lsl_streaminfo info)

- LIBLSL_C_API double lsl::lsl_get_created_at (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl::lsl_get_uid (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl::lsl_get_session_id (lsl_streaminfo info)
- LIBLSL_C_API const char ∗ lsl::lsl_get_hostname (lsl_streaminfo info)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_get_desc (lsl_streaminfo info)
- LIBLSL_C_API char ∗ lsl::lsl_get_xml (lsl_streaminfo info)
- LIBLSL_C_API int32_t lsl::lsl_get_channel_bytes (lsl_streaminfo info)

  Number of bytes occupied by a channel (0 for string-typed channels).
- LIBLSL_C_API int32_t lsl::lsl_get_sample_bytes (lsl_streaminfo info)

  Number of bytes occupied by a sample (0 for string-typed channels).
- LIBLSL_C_API int lsl::lsl_stream_info_matches_query (lsl_streaminfo info, const char ∗query)
- LIBLSL_C_API lsl_streaminfo lsl::lsl_streaminfo_from_xml (const char ∗xml)

  Create a streaminfo object from an XML representation.
- LIBLSL_C_API lsl_outlet lsl::lsl_create_outlet (lsl_streaminfo info, int32_t chunk_size, int32_t max_buffered)
- LIBLSL_C_API void lsl::lsl_destroy_outlet (lsl_outlet out)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_f (lsl_outlet out, const float ∗data)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_ft (lsl_outlet out, const float ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_ftp (lsl_outlet out, const float ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_d (lsl_outlet out, const double ∗data)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_dt (lsl_outlet out, const double ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_dtp (lsl_outlet out, const double ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_l (lsl_outlet out, const long ∗data)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_lt (lsl_outlet out, const long ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_ltp (lsl_outlet out, const long ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_i (lsl_outlet out, const int32_t ∗data)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_it (lsl_outlet out, const int32_t ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_itp (lsl_outlet out, const int32_t ∗data, double timestamp, int32↩_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_s (lsl_outlet out, const int16_t ∗data)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_st (lsl_outlet out, const int16_t ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_stp (lsl_outlet out, const int16_t ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_c (lsl_outlet out, const char ∗data)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_ct (lsl_outlet out, const char ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_ctp (lsl_outlet out, const char ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_str (lsl_outlet out, const char ∗∗data)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_strt (lsl_outlet out, const char ∗∗data, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_strtp (lsl_outlet out, const char ∗∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_buf (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_buft (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_buftp (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_v (lsl_outlet out, const void ∗data)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_vt (lsl_outlet out, const void ∗data, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_sample_vtp (lsl_outlet out, const void ∗data, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_f (lsl_outlet out, const float ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_ft (lsl_outlet out, const float ∗data, unsigned long data_elements, double timestamp)

- LIBLSL_C_API int32_t lsl::lsl_push_chunk_ftp (lsl_outlet out, const float ∗data, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_ftn (lsl_outlet out, const float ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_ftnp (lsl_outlet out, const float ∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_d (lsl_outlet out, const double ∗data, unsigned long data_↩ elements)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_dt (lsl_outlet out, const double ∗data, unsigned long data_↩ elements, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_dtp (lsl_outlet out, const double ∗data, unsigned long data_↩ elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_dtn (lsl_outlet out, const double ∗data, unsigned long data_↩ elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_dtnp (lsl_outlet out, const double ∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int lsl::lsl_push_chunk_l (lsl_outlet out, const long ∗data, unsigned long data_elements)
- LIBLSL_C_API int lsl::lsl_push_chunk_lt (lsl_outlet out, const long ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int lsl::lsl_push_chunk_ltp (lsl_outlet out, const long ∗data, unsigned long data_elements, double timestamp, int pushthrough)
- LIBLSL_C_API int lsl::lsl_push_chunk_ltn (lsl_outlet out, const long ∗data, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int lsl::lsl_push_chunk_ltnp (lsl_outlet out, const long ∗data, unsigned long data_elements, const double ∗timestamps, int pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_i (lsl_outlet out, const int32_t ∗data, unsigned long data_↩ elements)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_it (lsl_outlet out, const int32_t ∗data, unsigned long data_↩ elements, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_itp (lsl_outlet out, const int32_t ∗data, unsigned long data_↩ elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_itn (lsl_outlet out, const int32_t ∗data, unsigned long data_↩ elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_itnp (lsl_outlet out, const int32_t ∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_s (lsl_outlet out, const int16_t ∗data, unsigned long data_↩ elements)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_st (lsl_outlet out, const int16_t ∗data, unsigned long data_↩ elements, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_stp (lsl_outlet out, const int16_t ∗data, unsigned long data_↩ elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_stn (lsl_outlet out, const int16_t ∗data, unsigned long data_↩ elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_stnp (lsl_outlet out, const int16_t ∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_c (lsl_outlet out, const char ∗data, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_ct (lsl_outlet out, const char ∗data, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_ctp (lsl_outlet out, const char ∗data, unsigned long data_↩ elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_ctn (lsl_outlet out, const char ∗data, unsigned long data_↩ elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_ctnp (lsl_outlet out, const char ∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_str (lsl_outlet out, const char ∗∗data, unsigned long data_↩ elements)

- LIBLSL_C_API int32_t lsl::lsl_push_chunk_strt (lsl_outlet out, const char ∗∗data, unsigned long data_↩ elements, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_strtp (lsl_outlet out, const char ∗∗data, unsigned long data_↩ elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_strtn (lsl_outlet out, const char ∗∗data, unsigned long data_↩ elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_strtnp (lsl_outlet out, const char ∗∗data, unsigned long data_↩ elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_buf (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_buft (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, double timestamp)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_buftp (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, double timestamp, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_buftn (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, const double ∗timestamps)
- LIBLSL_C_API int32_t lsl::lsl_push_chunk_buftnp (lsl_outlet out, const char ∗∗data, const uint32_t ∗lengths, unsigned long data_elements, const double ∗timestamps, int32_t pushthrough)
- LIBLSL_C_API int32_t lsl::lsl_have_consumers (lsl_outlet out)
- LIBLSL_C_API int32_t lsl::lsl_wait_for_consumers (lsl_outlet out, double timeout)
- LIBLSL_C_API lsl_streaminfo lsl::lsl_get_info (lsl_outlet out)
- LIBLSL_C_API lsl_inlet lsl::lsl_create_inlet (lsl_streaminfo info, int32_t max_buflen, int32_t max_chunklen, int32_t recover)
- LIBLSL_C_API void lsl::lsl_destroy_inlet (lsl_inlet in)
- LIBLSL_C_API lsl_streaminfo lsl::lsl_get_fullinfo (lsl_inlet in, double timeout, int32_t ∗ec)
- LIBLSL_C_API void lsl::lsl_open_stream (lsl_inlet in, double timeout, int32_t ∗ec)
- LIBLSL_C_API void lsl::lsl_close_stream (lsl_inlet in)
- LIBLSL_C_API double lsl::lsl_time_correction (lsl_inlet in, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl::lsl_time_correction_ex (lsl_inlet in, double ∗remote_time, double ∗uncertainty, double timeout, int32_t ∗ec)
- LIBLSL_C_API int32_t lsl::lsl_set_postprocessing (lsl_inlet in, uint32_t flags)
- LIBLSL_C_API double lsl::lsl_pull_sample_f (lsl_inlet in, float ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl::lsl_pull_sample_d (lsl_inlet in, double ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl::lsl_pull_sample_l (lsl_inlet in, long ∗buffer, int buffer_elements, double timeout, int ∗ec)
- LIBLSL_C_API double lsl::lsl_pull_sample_i (lsl_inlet in, int32_t ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl::lsl_pull_sample_s (lsl_inlet in, int16_t ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl::lsl_pull_sample_c (lsl_inlet in, char ∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl::lsl_pull_sample_str (lsl_inlet in, char ∗∗buffer, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl::lsl_pull_sample_buf (lsl_inlet in, char ∗∗buffer, uint32_t ∗buffer_lengths, int32_t buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API double lsl::lsl_pull_sample_v (lsl_inlet in, void ∗buffer, int32_t buffer_bytes, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_f (lsl_inlet in, float ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_d (lsl_inlet in, double ∗data_buffer, double ∗timestamp↩ _buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_l (lsl_inlet in, long ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int ∗ec)

- LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_i (lsl_inlet in, int32_t ∗data_buffer, double ∗timestamp←_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_s (lsl_inlet in, int16_t ∗data_buffer, double ∗timestamp←_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_c (lsl_inlet in, char ∗data_buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_str (lsl_inlet in, char ∗∗data_buffer, double ∗timestamp←_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API unsigned long lsl::lsl_pull_chunk_buf (lsl_inlet in, char ∗∗data_buffer, uint32_t ∗lengths_←buffer, double ∗timestamp_buffer, unsigned long data_buffer_elements, unsigned long timestamp_buffer_←elements, double timeout, int32_t ∗ec)
- LIBLSL_C_API uint32_t lsl::lsl_samples_available (lsl_inlet in)
- LIBLSL_C_API uint32_t lsl::lsl_was_clock_reset (lsl_inlet in)
- LIBLSL_C_API int32_t lsl::lsl_smoothing_halftime (lsl_inlet in, float value)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_first_child (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_last_child (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_next_sibling (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_previous_sibling (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_parent (lsl_xml_ptr e)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_child (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_next_sibling_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_previous_sibling_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API int32_t lsl::lsl_empty (lsl_xml_ptr e)
- LIBLSL_C_API int32_t lsl::lsl_is_text (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl::lsl_name (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl::lsl_value (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl::lsl_child_value (lsl_xml_ptr e)
- LIBLSL_C_API const char ∗ lsl::lsl_child_value_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_append_child_value (lsl_xml_ptr e, const char ∗name, const char ∗value)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_prepend_child_value (lsl_xml_ptr e, const char ∗name, const char ∗value)
- LIBLSL_C_API int32_t lsl::lsl_set_child_value (lsl_xml_ptr e, const char ∗name, const char ∗value)
- LIBLSL_C_API int32_t lsl::lsl_set_name (lsl_xml_ptr e, const char ∗rhs)
- LIBLSL_C_API int32_t lsl::lsl_set_value (lsl_xml_ptr e, const char ∗rhs)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_append_child (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_prepend_child (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_append_copy (lsl_xml_ptr e, lsl_xml_ptr e2)
- LIBLSL_C_API lsl_xml_ptr lsl::lsl_prepend_copy (lsl_xml_ptr e, lsl_xml_ptr e2)
- LIBLSL_C_API void lsl::lsl_remove_child_n (lsl_xml_ptr e, const char ∗name)
- LIBLSL_C_API void lsl::lsl_remove_child (lsl_xml_ptr e, lsl_xml_ptr e2)
- LIBLSL_C_API lsl_continuous_resolver lsl::lsl_create_continuous_resolver (double forget_after)
- LIBLSL_C_API lsl_continuous_resolver lsl::lsl_create_continuous_resolver_byprop (const char ∗prop, const char ∗value, double forget_after)
- LIBLSL_C_API lsl_continuous_resolver lsl::lsl_create_continuous_resolver_bypred (const char ∗pred, double forget_after)
- LIBLSL_C_API int32_t lsl::lsl_resolver_results (lsl_continuous_resolver res, lsl_streaminfo ∗buffer, uint32_t buffer_elements)
- LIBLSL_C_API void lsl::lsl_destroy_continuous_resolver (lsl_continuous_resolver res)
- int32_t lsl::protocol_version ()
- int32_t lsl::library_version ()
- const char ∗ lsl::library_info ()
- double lsl::local_clock ()
- std::vector< stream_info > lsl::resolve_streams (double wait_time=1.0)

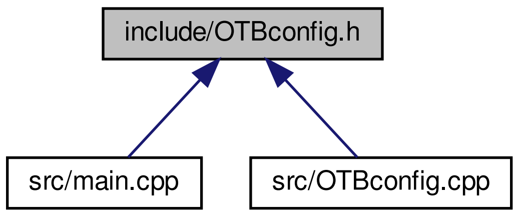
- std::vector< stream_info > lsl::resolve_stream (const std::string &prop, const std::string &value, int32_↩ t minimum=1, double timeout=FOREVER)
- std::vector< stream_info > lsl::resolve_stream (const std::string &pred, int32_t minimum=1, double timeout=FOREVER)
- void lsl::check_error (int32_t ec)

Variables

- const double lsl::IRREGULAR_RATE = 0.0
- const double lsl::DEDUCED_TIMESTAMP = -1.0
- const double lsl::FOREVER = 32000000.0

9.7 include/OTBconfig.h File Reference

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



Macros

- #define CONFIG_SIZE 40
- #define ACQ_SETT 0b10000000
- #define DECIM 0b01000000
- #define REC_ON 0b00100000
- #define FSAMP1 0b00010000
- #define FSAMP0 0b00001000
- #define NCH1 0b00000100
- #define NCH0 0b00000010
- #define ACQ_ON 0b00000001
- #define ACQ_OFF 0b00000000
- #define FSAMP_10240 FSAMP0 | FSAMP1
- #define FSAMP_5120 FSAMP1
- #define FSAMP_2048 FSAMP0
- #define FSAMP_512 0b00000000
- #define NCH_IN1to8_MIN1to4 NCH1 | NCH0
- #define NCH_IN1to6_MIN1to3 NCH1
- #define NCH_IN1to4_MIN1to2 NCH0
- #define NCH_IN1to2_MIN1 0b00000000

- #define AN_OUT_IN_SET 0b00000000
- #define ANOUT_GAIN1 0b00100000
- #define ANOUT_GAIN0 0b00010000
- #define INSEL3 0b00001000
- #define INSEL2 0b00000100
- #define INSEL1 0b00000010
- #define INSEL0 0b00000001
- #define ANOUT_GAIN_16 ANOUT_GAIN1 | ANOUT_GAIN0
- #define ANOUT_GAIN_4 ANOUT_GAIN1
- #define ANOUT_GAIN_2 ANOUT_GAIN0
- #define ANOUT_GAIN_1 0b00000000
- #define AN_OUT_CH_SET 0b00000000
- #define CHSEL5 0b00100000
- #define CHSEL4 0b00010000
- #define CHSEL3 0b00001000
- #define CHSEL2 0b00000100
- #define CHSEL1 0b00000010
- #define CHSEL0 0b00000001
- #define INX_CONF0 0b00000000
- #define MUS6 0b01000000
- #define MUS5 0b00100000
- #define MUS4 0b00010000
- #define MUS3 0b00001000
- #define MUS2 0b00000100
- #define MUS1 0b00000010
- #define MUS0 0b00000001
- #define Not_defined 0
- #define Temporalis_Anterior 1
- #define Superfic_Masseter 2
- #define Splenius_Capitis 3
- #define Upper_Trapezius 4
- #define Middle_Trapezius 5
- #define Lower_Trapezius 6
- #define Rhomboideus_Major 7
- #define Rhomboideus_Minor 8
- #define Anterior_Deltoid 9
- #define Posterior_Deltoid 10
- #define Lateral_Deltoid 11
- #define Infraspinatus 12
- #define Teres_Major 13
- #define Erector_Spinae 14
- #define Latissimus_Dorsi 15
- #define Bic_Br_Long_Head 16
- #define Bic_Br_Short_Head 17
- #define Tric_Br_Lat_Head 18
- #define Tric_Br_Med_Head 19
- #define Pronator_Teres 20
- #define Flex_Carpi_Radial 21
- #define Flex_Carpi_Ulnaris 22
- #define Palmaris_Longus 23
- #define Ext_Carpi_Radialis 24
- #define Ext_Carpi_Ulnaris 25
- #define Ext_Dig_Communis 26
- #define Brachioradialis 27
- #define Abd_Pollicis_Brev 28

- #define Abd_Pollicis_Long 29
- #define Opponens_Pollicis 30
- #define Adductor_Pollicis 31
- #define Flex_Poll_Brevis 32
- #define Abd_Digiti_Minimi 33
- #define Flex_Digiti_Minimi 34
- #define Opp_Digiti_Minimi 35
- #define Dorsal_Interossei 36
- #define Palmar_Interossei 37
- #define Lumbrical 38
- #define Rectus_Abdominis 39
- #define Ext_Abdom_Obliq 40
- #define Serratus_Anterior 41
- #define Pectoralis_Major 42
- #define Sternoc_Ster_Head 43
- #define Sternoc_Clav_Head 44
- #define Anterior_Scalenus 45
- #define Tensor_Fascia Latae 46
- #define Gastrocn_Lateralis 47
- #define Gastrocn_Medialis 48
- #define Biceps_Femoris 49
- #define Soleus 50
- #define Semitendinosus 51
- #define Gluteus_maximus 52
- #define Gluteus_medius 53
- #define Vastus_lateralis 54
- #define Vastus_medialis 55
- #define Rectus_femoris 56
- #define Tibialis_anterior 57
- #define Peroneus_longus 58
- #define Semimembranosus 59
- #define Gracilis 60
- #define Ext_Anal_Sphincter 61
- #define Puborectalis 62
- #define Urethral_Sphincter 63
- #define Not_a_Muscle 64
- #define INX_CONF1 0b00000000
- #define SENS4 0b10000000
- #define SENS3 0b01000000
- #define SENS2 0b00100000
- #define SENS1 0b00010000
- #define SENS0 0b00001000
- #define ADAPT2 0b00000100
- #define ADAPT1 0b00000010
- #define ADAPT0 0b00000001
- #define INX_CONF2 0b00000000
- #define SIDE1 0b10000000
- #define SIDE0 0b01000000
- #define HPF1 0b00100000
- #define HPF0 0b00010000
- #define LPF1 0b00001000
- #define LPF0 0b00000100
- #define MODE1 0b00000010
- #define MODE0 0b00000001
- #define SIDE_NONE SIDE1 | SIDE0

- #define SIDE_RIGHT SIDE1
- #define SIDE_LEFT SIDE0
- #define SIDE_UNDEFINED 0b00000000
- #define HIGH_PASS_FILTER_200 HPF1 | HPF0
- #define HIGH_PASS_FILTER_100 HPF1
- #define HIGH_PASS_FILTER_10 HPF0
- #define HIGH_PASS_FILTER_03 0b00000000
- #define LOW_PASS_FILTER_4400 LPF1 | LPF0
- #define LOW_PASS_FILTER_900 LPF1
- #define LOW_PASS_FILTER_500 LPF0
- #define LOW_PASS_FILTER_130 0b00000000
- #define DETECTION_MODE_BIPOLAR MODE1
- #define DETECTION_MODE_DIFFERENCIAL MODE0
- #define DETECTION_MODE_MONOPOLAR 0b00000000
- #define CRC_CODE 0b10001100

Functions

- unsigned char crc (unsigned char config[ ])
- void printBIN (char)

9.7.1 Macro Definition Documentation

9.7.1.1 Abd_Digiti_Minimi

```
#define Abd_Digiti_Minimi 33
```

9.7.1.2 Abd_Pollicis_Brev

```
#define Abd_Pollicis_Brev 28
```

9.7.1.3 Abd_Pollicis_Long

```
#define Abd_Pollicis_Long 29
```

9.7.1.4 ACQ_OFF

```
#define ACQ_OFF 0b00000000
```

9.7.1.5 ACQ_ON

```
#define ACQ_ON 0b00000001
```

9.7.1.6 ACQ_SETT

```
#define ACQ_SETT 0b10000000
```

9.7.1.7 ADAPT0

```
#define ADAPT0 0b00000001
```

9.7.1.8 ADAPT1

```
#define ADAPT1 0b00000010
```

9.7.1.9 ADAPT2

```
#define ADAPT2 0b00000100
```

9.7.1.10 Adductor_Pollicis

```
#define Adductor_Pollicis 31
```

9.7.1.11 AN_OUT_CH_SET

```
#define AN_OUT_CH_SET 0b00000000
```

9.7.1.12 AN_OUT_IN_SET

```
#define AN_OUT_IN_SET 0b00000000
```

9.7.1.13 ANOUT_GAIN0

```
#define ANOUT_GAIN0 0b00010000
```

9.7.1.14 ANOUT_GAIN1

```
#define ANOUT_GAIN1 0b00100000
```

9.7.1.15 ANOUT_GAIN_1

```
#define ANOUT_GAIN_1 0b00000000
```

9.7.1.16 ANOUT_GAIN_16

```
#define ANOUT_GAIN_16 ANOUT_GAIN1 | ANOUT_GAIN0
```

9.7.1.17 ANOUT_GAIN_2

```
#define ANOUT_GAIN_2 ANOUT_GAIN0
```

9.7.1.18 ANOUT_GAIN_4

```
#define ANOUT_GAIN_4 ANOUT_GAIN1
```

9.7.1.19 Anterior_Deltoid

```
#define Anterior_Deltoid 9
```

9.7.1.20 Anterior_Scalenus

```
#define Anterior_Scalenus 45
```

9.7.1.21 Bic_Br_Long_Head

```
#define Bic_Br_Long_Head 16
```

9.7.1.22 Bic_Br_Short_Head

```
#define Bic_Br_Short_Head 17
```

9.7.1.23 Biceps_Femoris

```
#define Biceps_Femoris 49
```

9.7.1.24 Brachioradialis

```
#define Brachioradialis 27
```

9.7.1.25 CHSEL0

```
#define CHSEL0 0b00000001
```

9.7.1.26 CHSEL1

```
#define CHSEL1 0b00000010
```

9.7.1.27 CHSEL2

```
#define CHSEL2 0b00000100
```

9.7.1.28 CHSEL3

```
#define CHSEL3 0b00001000
```

9.7.1.29 CHSEL4

```
#define CHSEL4 0b00010000
```

9.7.1.30 CHSEL5

```
#define CHSEL5 0b00100000
```

9.7.1.31 CONFIG_SIZE

```
#define CONFIG_SIZE 40
```

9.7.1.32 CRC_CODE

```
#define CRC_CODE 0b10001100
```

9.7.1.33 DECIM

```
#define DECIM 0b01000000
```

9.7.1.34 DETECTION_MODE_BIPOLAR

```
#define DETECTION_MODE_BIPOLAR MODE1
```

9.7.1.35 DETECTION_MODE_DIFFERENCIAL

```
#define DETECTION_MODE_DIFFERENCIAL MODE0
```

9.7.1.36 DETECTION_MODE_MONOPOLAR

```
#define DETECTION_MODE_MONOPOLAR 0b00000000
```

9.7.1.37 Dorsal_Interossei

```
#define Dorsal_Interossei 36
```

9.7.1.38 Erector_Spinae

```
#define Erector_Spinae 14
```

9.7.1.39 Ext_Abdom_Obliq

```
#define Ext_Abdom_Obliq 40
```

9.7.1.40 Ext_Anal_Sphincter

```
#define Ext_Anal_Sphincter 61
```

9.7.1.41 Ext_Carpi_Radialis

```
#define Ext_Carpi_Radialis 24
```

9.7.1.42 Ext_Carpi_Ulnaris

```
#define Ext_Carpi_Ulnaris 25
```

9.7.1.43 Ext_Dig_Communis

```
#define Ext_Dig_Communis 26
```

9.7.1.44 Flex_Carpi_Radial

```
#define Flex_Carpi_Radial 21
```

9.7.1.45 Flex_Carpi_Ulnaris

```
#define Flex_Carpi_Ulnaris 22
```

9.7.1.46 Flex_Digiti_Minimi

```
#define Flex_Digiti_Minimi 34
```

9.7.1.47 Flex_Poll_Brevis

```
#define Flex_Poll_Brevis 32
```

9.7.1.48 FSAMP0

```
#define FSAMP0 0b00001000
```

9.7.1.49 FSAMP1

```
#define FSAMP1 0b00010000
```

9.7.1.50 FSAMP_10240

```
#define FSAMP_10240 FSAMP0 | FSAMP1
```

9.7.1.51 FSAMP_2048

```
#define FSAMP_2048 FSAMP0
```

9.7.1.52 FSAMP_512

```
#define FSAMP_512 0b00000000
```

9.7.1.53 FSAMP_5120

```
#define FSAMP_5120 FSAMP1
```

9.7.1.54 Gastrocn_Lateralis

```
#define Gastrocn_Lateralis 47
```

9.7.1.55 Gastrocn_Medialis

```
#define Gastrocn_Medialis 48
```

9.7.1.56 Gluteus_maximus

```
#define Gluteus_maximus 52
```

9.7.1.57 Gluteus_medius

```
#define Gluteus_medius 53
```

9.7.1.58 Gracilis

```
#define Gracilis 60
```

9.7.1.59 HIGH_PASS_FILTER_03

```
#define HIGH_PASS_FILTER_03 0b00000000
```

9.7.1.60 HIGH_PASS_FILTER_10

```
#define HIGH_PASS_FILTER_10 HPF0
```

9.7.1.61 HIGH_PASS_FILTER_100

```
#define HIGH_PASS_FILTER_100 HPF1
```

9.7.1.62 HIGH_PASS_FILTER_200

```
#define HIGH_PASS_FILTER_200 HPF1 | HPF0
```

9.7.1.63 HPF0

```
#define HPF0 0b00010000
```

9.7.1.64 HPF1

```
#define HPF1 0b00100000
```

9.7.1.65 Infraspinatus

```
#define Infraspinatus 12
```

9.7.1.66 INSEL0

```
#define INSEL0 0b00000001
```

9.7.1.67 INSEL1

```
#define INSEL1 0b00000010
```

9.7.1.68 INSEL2

```
#define INSEL2 0b00000100
```

9.7.1.69 INSEL3

```
#define INSEL3 0b00001000
```

9.7.1.70 INX_CONF0

```
#define INX_CONF0 0b00000000
```

9.7.1.71 INX_CONF1

```
#define INX_CONF1 0b00000000
```

9.7.1.72 INX_CONF2

```
#define INX_CONF2 0b00000000
```

9.7.1.73 Lateral_Deltoid

```
#define Lateral_Deltoid 11
```

9.7.1.74 Latissimus_Dorsi

```
#define Latissimus_Dorsi 15
```

9.7.1.75 LOW_PASS_FILTER_130

```
#define LOW_PASS_FILTER_130 0b00000000
```

9.7.1.76 LOW_PASS_FILTER_4400

```
#define LOW_PASS_FILTER_4400 LPF1 | LPF0
```

9.7.1.77 LOW_PASS_FILTER_500

```
#define LOW_PASS_FILTER_500 LPF0
```

9.7.1.78 LOW_PASS_FILTER_900

```
#define LOW_PASS_FILTER_900 LPF1
```

9.7.1.79 Lower_Trapezius

```
#define Lower_Trapezius 6
```

9.7.1.80 LPF0

```
#define LPF0 0b00000100
```

9.7.1.81 LPF1

```
#define LPF1 0b00001000
```

9.7.1.82 Lumbrical

```
#define Lumbrical 38
```

9.7.1.83 Middle_Trapezius

```
#define Middle_Trapezius 5
```

9.7.1.84 MODE0

```
#define MODE0 0b00000001
```

9.7.1.85 MODE1

```
#define MODE1 0b00000010
```

9.7.1.86 MUS0

```
#define MUS0 0b00000001
```

9.7.1.87 MUS1

```
#define MUS1 0b00000010
```

9.7.1.88 MUS2

```
#define MUS2 0b00000100
```

9.7.1.89 MUS3

```
#define MUS3 0b00001000
```

9.7.1.90 MUS4

```
#define MUS4 0b00010000
```

9.7.1.91 MUS5

```
#define MUS5 0b00100000
```

9.7.1.92 MUS6

```
#define MUS6 0b01000000
```

9.7.1.93 NCH0

```
#define NCH0 0b00000010
```

9.7.1.94 NCH1

```
#define NCH1 0b00000100
```

9.7.1.95 NCH_IN1to2_MIN1

```
#define NCH_IN1to2_MIN1 0b00000000
```

9.7.1.96 NCH_IN1to4_MIN1to2

```
#define NCH_IN1to4_MIN1to2 NCH0
```

9.7.1.97 NCH_IN1to6_MIN1to3

```
#define NCH_IN1to6_MIN1to3 NCH1
```

9.7.1.98 NCH_IN1to8_MIN1to4

```
#define NCH_IN1to8_MIN1to4 NCH1 | NCH0
```

9.7.1.99 Not_a_Muscle

```
#define Not_a_Muscle 64
```

9.7.1.100 Not_defined

```
#define Not_defined 0
```

9.7.1.101 Opp_Digiti_Minimi

```
#define Opp_Digiti_Minimi 35
```

9.7.1.102 Opponens_Pollicis

```
#define Opponens_Pollicis 30
```

9.7.1.103 Palmar_Interossei

```
#define Palmar_Interossei 37
```

9.7.1.104 Palmaris_Longus

```
#define Palmaris_Longus 23
```

9.7.1.105 Pectoralis_Major

```
#define Pectoralis_Major 42
```

9.7.1.106 Peroneus_longus

```
#define Peroneus_longus 58
```

9.7.1.107 Posterior_Deltoid

```
#define Posterior_Deltoid 10
```

9.7.1.108 Pronator_Teres

```
#define Pronator_Teres 20
```

9.7.1.109 Puborectalis

```
#define Puborectalis 62
```

9.7.1.110 REC_ON

```
#define REC_ON 0b00100000
```

9.7.1.111 Rectus_Abdominis

```
#define Rectus_Abdominis 39
```

9.7.1.112 Rectus_femoris

```
#define Rectus_femoris 56
```

9.7.1.113 Rhomboideus_Major

```
#define Rhomboideus_Major 7
```

9.7.1.114 Rhomboideus_Minor

```
#define Rhomboideus_Minor 8
```

9.7.1.115 Semimembranosus

```
#define Semimembranosus 59
```

9.7.1.116 Semitendinosus

```
#define Semitendinosus 51
```

9.7.1.117 SENS0

```
#define SENS0 0b00001000
```

9.7.1.118 SENS1

```
#define SENS1 0b00010000
```

9.7.1.119 SENS2

```
#define SENS2 0b00100000
```

9.7.1.120 SENS3

```
#define SENS3 0b01000000
```

9.7.1.121 SENS4

```
#define SENS4 0b10000000
```

9.7.1.122 Serratus_Anterior

```
#define Serratus_Anterior 41
```

9.7.1.123 SIDE0

```
#define SIDE0 0b01000000
```

9.7.1.124 SIDE1

```
#define SIDE1 0b10000000
```

9.7.1.125 SIDE_LEFT

```
#define SIDE_LEFT SIDE0
```

9.7.1.126 SIDE_NONE

```
#define SIDE_NONE SIDE1 | SIDE0
```

9.7.1.127 SIDE_RIGHT

```
#define SIDE_RIGHT SIDE1
```

9.7.1.128 SIDE_UNDEFINED

```
#define SIDE_UNDEFINED 0b00000000
```

9.7.1.129 Soleus

```
#define Soleus 50
```

9.7.1.130 Splenius_Capitis

```
#define Splenius_Capitis 3
```

9.7.1.131 Sternoc_Clav_Head

```
#define Sternoc_Clav_Head 44
```

9.7.1.132 Sternoc_Ster_Head

```
#define Sternoc_Ster_Head 43
```

9.7.1.133 Superfic_Masseter

```
#define Superfic_Masseter 2
```

9.7.1.134 Temporalis_Anterior

```
#define Temporalis_Anterior 1
```

9.7.1.135 Tensor_Fascia

```
#define Tensor_Fascia Latae 46
```

9.7.1.136 Teres_Major

```
#define Teres_Major 13
```

9.7.1.137 Tibialis_anterior

```
#define Tibialis_anterior 57
```

9.7.1.138 Tric_Br_Lat_Head

```
#define Tric_Br_Lat_Head 18
```

9.7.1.139 Tric_Br_Med_Head

```
#define Tric_Br_Med_Head 19
```

9.7.1.140 Upper_Trapezius

```
#define Upper_Trapezius 4
```

9.7.1.141 Urethral_Sphincter

```
#define Urethral_Sphincter 63
```

9.7.1.142 Vastus_lateralis

```
#define Vastus_lateralis 54
```

9.7.1.143 Vastus_medialis

```
#define Vastus_medialis 55
```

9.7.2 Function Documentation

9.7.2.1 crc()

```
unsigned char crc (
             unsigned char config[] )
```

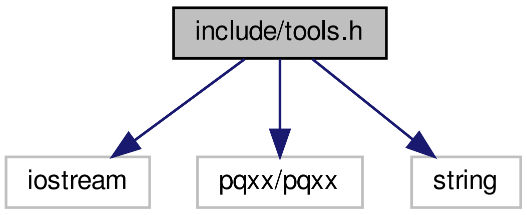
9.7.2.2 printBIN()

```
void printBIN (
             char  )
```

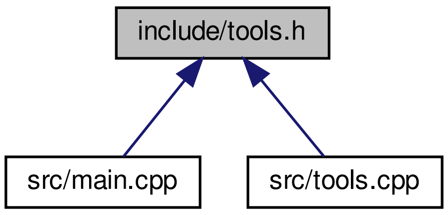
9.8 include/tools.h File Reference

```
#include <iostream>
#include <pqxx/pqxx>
#include <string>
```

Include dependency graph for tools.h:



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



Functions

- void error (std::string str)

    error Display the passed string thne exit the program.
- void usage (std::vector< std::string > &optf, std::vector< std::string > &optl, std::vector< std::string > &optv)

    usage Display the usage, then exit the program.
- void get_arg (int argc, char **argv, std::vector< std::string > &optf, std::vector< std::string > &optl, std↩ ::vector< std::string > &optv)

    get_arg Search for the potential argument in the argument passed to the program.

9.8.1 Function Documentation

9.8.1.1 error()

```
void error (
             std::string str )
```

error Display the passed string thne exit the program.

Parameters

str	String to display.

9.8.1.2 get_arg()

```
void get_arg (
             int argc,
             char ** argv,
             std::vector< std::string > & optf,
             std::vector< std::string > & optl,
             std::vector< std::string > & optv )
```

get_arg Search for the potential argument in the argument passed to the program.

Parameters

argc	Argument counter
argv	Argument array
optf	List of option flags
optf	List of option labels
optf	List of option values

9.8.1.3 usage()

```
void usage (
             std::vector< std::string > & optf,
             std::vector< std::string > & optl,
             std::vector< std::string > & optv )
```

usage Display the usage, then exit the program.

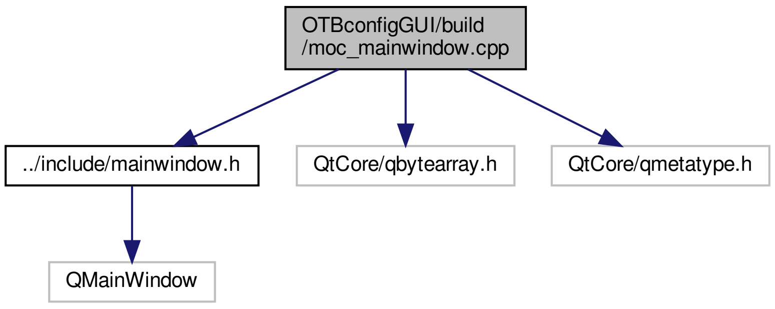
Parameters

optf	List of option flags
optf	List of option labels
optf	List of option values

9.9 OTBconfigGUI/build/moc_mainwindow.cpp File Reference

```
#include "../include/mainwindow.h"
#include <QtCore/qbytearray.h>
#include <QtCore/qmetatype.h>
```

Include dependency graph for moc_mainwindow.cpp:



Classes

- struct qt_meta_stringdata_MainWindow_t

Macros

- #define QT_MOC_LITERAL(idx, ofs, len)

9.9.1 Macro Definition Documentation

9.9.1.1 QT_MOC_LITERAL

```
#define QT_MOC_LITERAL(
                idx,
                ofs,
                len )
```

Value:

```
Q_STATIC_BYTE_ARRAY_DATA_HEADER_INITIALIZER_WITH_OFFSET(len, \
    qptrdiff(offsetof(qt_meta_stringdata_MainWindow_t, stringdata0) + ofs \
        - idx * sizeof(QByteArrayData)) \
    )
```

9.10 OTBconfigGUI/build/moc_predefs.h File Reference

Macros

- #define __SSP_STRONG__ 3
- #define __DBL_MIN_EXP__ (-1021)
- #define __FLT32X_MAX_EXP__ 1024
- #define __cpp_attributes 200809
- #define __UINT_LEAST16_MAX__ 0xffff
- #define __ATOMIC_ACQUIRE 2
- #define __FLT128_MAX_10_EXP__ 4932
- #define __FLT_MIN__ 1.17549435082228750796873653722224568e-38F
- #define __GCC_IEC_559_COMPLEX 2
- #define __UINT_LEAST8_TYPE__ unsigned char
- #define __SIZEOF_FLOAT80__ 16
- #define __INTMAX_C(c) c ## L
- #define __CHAR_BIT__ 8
- #define __UINT8_MAX__ 0xff
- #define __WINT_MAX__ 0xffffffffU
- #define __FLT32_MIN_EXP__ (-125)
- #define __cpp_static_assert 200410
- #define __ORDER_LITTLE_ENDIAN__ 1234
- #define __SIZE_MAX__ 0xffffffffffffffffUL
- #define __WCHAR_MAX__ 0x7fffffff
- #define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_1 1
- #define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_2 1
- #define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_4 1
- #define __DBL_DENORM_MIN__ double(4.94065645841246544176568792868221372e-324L)
- #define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_8 1
- #define __GCC_ATOMIC_CHAR_LOCK_FREE 2
- #define __GCC_IEC_559 2
- #define __FLT32X_DECIMAL_DIG__ 17
- #define __FLT_EVAL_METHOD__ 0
- #define __unix__ 1
- #define __cpp_binary_literals 201304
- #define __FLT64_DECIMAL_DIG__ 17
- #define __GCC_ATOMIC_CHAR32_T_LOCK_FREE 2
- #define __x86_64 1
- #define __cpp_variadic_templates 200704
- #define __UINT_FAST64_MAX__ 0xffffffffffffffffUL
- #define __SIG_ATOMIC_TYPE__ int
- #define __DBL_MIN_10_EXP__ (-307)
- #define __FINITE_MATH_ONLY__ 0
- #define __GNUC_PATCHLEVEL__ 0
- #define __FLT32_HAS_DENORM__ 1
- #define __UINT_FAST8_MAX__ 0xff
- #define __has_include(STR) __has_include__(STR)
- #define __DEC64_MAX_EXP__ 385
- #define __INT8_C(c) c
- #define __INT_LEAST8_WIDTH__ 8
- #define __UINT_LEAST64_MAX__ 0xffffffffffffffffUL
- #define __SHRT_MAX__ 0x7fff
- #define __LDBL_MAX__ 1.18973149535723176502126385303097021e+4932L
- #define __FLT64X_MAX_10_EXP__ 4932

- #define __UINT_LEAST8_MAX__ 0xff
- #define __GCC_ATOMIC_BOOL_LOCK_FREE 2
- #define __FLT128_DENORM_MIN__ 6.47517511943802511092443895822764655e-4966F128
- #define __UINTMAX_TYPE__ long unsigned int
- #define __linux 1
- #define __DEC32_EPSILON__ 1E-6DF
- #define __FLT_EVAL_METHOD_TS_18661_3__ 0
- #define __OPTIMIZE__ 1
- #define __unix 1
- #define __UINT32_MAX__ 0xffffffffU
- #define __GXX_EXPERIMENTAL_CXX0X__ 1
- #define __LDBL_MAX_EXP__ 16384
- #define __FLT128_MIN_EXP__ (-16381)
- #define __WINT_MIN__ 0U
- #define __linux__ 1
- #define __FLT128_MIN_10_EXP__ (-4931)
- #define __INT_LEAST16_WIDTH__ 16
- #define __SCHAR_MAX__ 0x7f
- #define __FLT128_MANT_DIG__ 113
- #define __WCHAR_MIN__ (-__WCHAR_MAX__ - 1)
- #define __INT64_C(c) c ## L
- #define __DBL_DIG__ 15
- #define __GCC_ATOMIC_POINTER_LOCK_FREE 2
- #define __FLT64X_MANT_DIG__ 64
- #define _FORTIFY_SOURCE 2
- #define __SIZEOF_INT__ 4
- #define __SIZEOF_POINTER__ 8
- #define __GCC_ATOMIC_CHAR16_T_LOCK_FREE 2
- #define __USER_LABEL_PREFIX__
- #define __FLT64X_EPSILON__ 1.08420217248550443400745280086994171e-19F64x
- #define __STDC_HOSTED__ 1
- #define __LDBL_HAS_INFINITY__ 1
- #define __FLT32_DIG__ 6
- #define __FLT_EPSILON__ 1.19209289550781250000000000000000000e-7F
- #define __GXX_WEAK__ 1
- #define __SHRT_WIDTH__ 16
- #define __LDBL_MIN__ 3.36210314311209350626267781732175260e-4932L
- #define __DEC32_MAX__ 9.999999E96DF
- #define __cpp_threadsafe_static_init 200806
- #define __FLT64X_DENORM_MIN__ 3.64519953188247460252840593361941982e-4951F64x
- #define __FLT32X_HAS_INFINITY__ 1
- #define __INT32_MAX__ 0x7fffffff
- #define __INT_WIDTH__ 32
- #define __SIZEOF_LONG__ 8
- #define __STDC_IEC_559__ 1
- #define __STDC_ISO_10646__ 201706L
- #define __UINT16_C(c) c
- #define __PTRDIFF_WIDTH__ 64
- #define __DECIMAL_DIG__ 21
- #define __FLT64_EPSILON__ 2.22044604925031308084726333618164062e-16F64
- #define __gnu_linux__ 1
- #define __INTMAX_WIDTH__ 64
- #define __FLT64_MIN_EXP__ (-1021)
- #define __has_include_next(STR) __has_include_next__(STR)
- #define __FLT64X_MIN_10_EXP__ (-4931)

- #define __LDBL_HAS_QUIET_NAN__ 1
- #define __FLT64_MANT_DIG__ 53
- #define __GNUC__ 7
- #define __GXX_RTTI 1
- #define __pie__ 2
- #define __MMX__ 1
- #define __cpp_delegating_constructors 200604
- #define __FLT_HAS_DENORM__ 1
- #define __SIZEOF_LONG_DOUBLE__ 16
- #define __BIGGEST_ALIGNMENT__ 16
- #define __STDC_UTF_16__ 1
- #define __FLT64_MAX_10_EXP__ 308
- #define __FLT32_HAS_INFINITY__ 1
- #define __DBL_MAX__ double(1.79769313486231570814527423731704357e+308L)
- #define __cpp_raw_strings 200710
- #define __INT_FAST32_MAX__ 0x7fffffffffffffffL
- #define __DBL_HAS_INFINITY__ 1
- #define __INT64_MAX__ 0x7fffffffffffffffL
- #define __DEC32_MIN_EXP__ (-94)
- #define __INTPTR_WIDTH__ 64
- #define __FLT32X_HAS_DENORM__ 1
- #define __INT_FAST16_TYPE__ long int
- #define __LDBL_HAS_DENORM__ 1
- #define __cplusplus 201103L
- #define __cpp_ref_qualifiers 200710
- #define __DEC128_MAX__ 9.999999999999999999999999999999999E6144DL
- #define __INT_LEAST32_MAX__ 0x7fffffff
- #define __DEC32_MIN__ 1E-95DF
- #define __DEPRECATED 1
- #define __cpp_rvalue_references 200610
- #define __DBL_MAX_EXP__ 1024
- #define __WCHAR_WIDTH__ 32
- #define __FLT32_MAX__ 3.40282346638528859811704183484516925e+38F32
- #define __DEC128_EPSILON__ 1E-33DL
- #define __SSE2_MATH__ 1
- #define __ATOMIC_HLE_RELEASE 131072
- #define __PTRDIFF_MAX__ 0x7fffffffffffffffL
- #define __amd64 1
- #define __STDC_NO_THREADS__ 1
- #define __ATOMIC_HLE_ACQUIRE 65536
- #define __FLT32_HAS_QUIET_NAN__ 1
- #define __GNUG__ 7
- #define __LONG_LONG_MAX__ 0x7fffffffffffffffLL
- #define __SIZEOF_SIZE_T__ 8
- #define __cpp_rvalue_reference 200610
- #define __cpp_nsdmi 200809
- #define __FLT64X_MIN_EXP__ (-16381)
- #define __SIZEOF_WINT_T__ 4
- #define __LONG_LONG_WIDTH__ 64
- #define __cpp_initializer_lists 200806
- #define __FLT32_MAX_EXP__ 128
- #define __cpp_hex_float 201603
- #define __GCC_HAVE_DWARF2_CFI_ASM 1
- #define __GXX_ABI_VERSION 1011
- #define __FLT128_HAS_INFINITY__ 1

- #define __FLT_MIN_EXP__ (-125)
- #define __cpp_lambdas 200907
- #define __FLT64X_HAS_QUIET_NAN__ 1
- #define __INT_FAST64_TYPE__ long int
- #define __FLT64_DENORM_MIN__ 4.94065645841246544176568792868221372e-324F64
- #define __DBL_MIN__ double(2.22507385850720138309023271733240406e-308L)
- #define __PIE__ 2
- #define __LP64__ 1
- #define __FLT32X_EPSILON__ 2.22044604925031308084726333618164062e-16F32x
- #define __DECIMAL_BID_FORMAT__ 1
- #define __FLT64_MIN_10_EXP__ (-307)
- #define __FLT64X_DECIMAL_DIG__ 21
- #define __DEC128_MIN__ 1E-6143DL
- #define __REGISTER_PREFIX__
- #define __UINT16_MAX__ 0xffff
- #define __DBL_HAS_DENORM__ 1
- #define __FLT32_MIN__ 1.17549435082228750796873653722224568e-38F32
- #define __UINT8_TYPE__ unsigned char
- #define __FLT_MANT_DIG__ 24
- #define __LDBL_DECIMAL_DIG__ 21
- #define __VERSION__ "7.3.0"
- #define __UINT64_C(c) c ## UL
- #define __cpp_unicode_characters 200704
- #define _STDC_PREDEF_H 1
- #define __GCC_ATOMIC_INT_LOCK_FREE 2
- #define __FLT128_MAX_EXP__ 16384
- #define __FLT32_MANT_DIG__ 24
- #define __FLOAT_WORD_ORDER__ __ORDER_LITTLE_ENDIAN__
- #define __STDC_IEC_559_COMPLEX__ 1
- #define __FLT128_HAS_DENORM__ 1
- #define __FLT128_DIG__ 33
- #define __SCHAR_WIDTH__ 8
- #define __INT32_C(c) c
- #define __DEC64_EPSILON__ 1E-15DD
- #define __ORDER_PDP_ENDIAN__ 3412
- #define __DEC128_MIN_EXP__ (-6142)
- #define __FLT32_MAX_10_EXP__ 38
- #define __INT_FAST32_TYPE__ long int
- #define __UINT_LEAST16_TYPE__ short unsigned int
- #define __FLT64X_HAS_INFINITY__ 1
- #define unix 1
- #define __INT16_MAX__ 0x7fff
- #define __cpp_rtti 199711
- #define __SIZE_TYPE__ long unsigned int
- #define __UINT64_MAX__ 0xffffffffffffffffUL
- #define __FLT64X_DIG__ 18
- #define __INT8_TYPE__ signed char
- #define __ELF__ 1
- #define __GCC_ASM_FLAG_OUTPUTS__ 1
- #define __FLT_RADIX__ 2
- #define __INT_LEAST16_TYPE__ short int
- #define __LDBL_EPSILON__ 1.08420217248550443400745280086994171e-19L
- #define __UINTMAX_C(c) c ## UL
- #define __GLIBCXX_BITSIZE_INT_N_0 128
- #define __k8 1

- #define __SIG_ATOMIC_MAX__ 0x7fffffff
- #define __GCC_ATOMIC_WCHAR_T_LOCK_FREE 2
- #define __SIZEOF_PTRDIFF_T__ 8
- #define __FLT32X_MANT_DIG__ 53
- #define __x86_64__ 1
- #define __FLT32X_MIN_EXP__ (-1021)
- #define __DEC32_SUBNORMAL_MIN__ 0.000001E-95DF
- #define __INT_FAST16_MAX__ 0x7fffffffffffffffL
- #define __FLT64_DIG__ 15
- #define __UINT_FAST32_MAX__ 0xffffffffffffffffUL
- #define __UINT_LEAST64_TYPE__ long unsigned int
- #define __FLT_HAS_QUIET_NAN__ 1
- #define __FLT_MAX_10_EXP__ 38
- #define __LONG_MAX__ 0x7fffffffffffffffL
- #define __FLT64X_HAS_DENORM__ 1
- #define __DEC128_SUBNORMAL_MIN__ 0.000000000000000000000000000000001E-6143DL
- #define __FLT_HAS_INFINITY__ 1
- #define __cpp_unicode_literals 200710
- #define __UINT_FAST16_TYPE__ long unsigned int
- #define __DEC64_MAX__ 9.999999999999999E384DD
- #define __INT_FAST32_WIDTH__ 64
- #define __CHAR16_TYPE__ short unsigned int
- #define __PRAGMA_REDEFINE_EXTNAME 1
- #define __SIZE_WIDTH__ 64
- #define __SEG_FS 1
- #define __INT_LEAST16_MAX__ 0x7fff
- #define __DEC64_MANT_DIG__ 16
- #define __UINT_LEAST32_MAX__ 0xffffffffU
- #define __SEG_GS 1
- #define __FLT32_DENORM_MIN__ 1.40129846432481707092372958328991613e-45F32
- #define __GCC_ATOMIC_LONG_LOCK_FREE 2
- #define __SIG_ATOMIC_WIDTH__ 32
- #define __INT_LEAST64_TYPE__ long int
- #define __INT16_TYPE__ short int
- #define __INT_LEAST8_TYPE__ signed char
- #define __DEC32_MAX_EXP__ 97
- #define __INT_FAST8_MAX__ 0x7f
- #define __FLT128_MAX__ 1.18973149535723176508575932662800702e+4932F128
- #define __INTPTR_MAX__ 0x7fffffffffffffffL
- #define linux 1
- #define __cpp_range_based_for 200907
- #define __FLT64_HAS_QUIET_NAN__ 1
- #define __FLT32_MIN_10_EXP__ (-37)
- #define __SSE2__ 1
- #define __EXCEPTIONS 1
- #define __LDBL_MANT_DIG__ 64
- #define __DBL_HAS_QUIET_NAN__ 1
- #define __FLT64_HAS_INFINITY__ 1
- #define __FLT64X_MAX__ 1.18973149535723176502126385303097021e+4932F64x
- #define __SIG_ATOMIC_MIN__ (-__SIG_ATOMIC_MAX__ - 1)
- #define __code_model_small__ 1
- #define __k8__ 1
- #define __INTPTR_TYPE__ long int
- #define __UINT16_TYPE__ short unsigned int
- #define __WCHAR_TYPE__ int

- #define __SIZEOF_FLOAT__ 4
- #define __pic__ 2
- #define __UINTPTR_MAX__ 0xffffffffffffffffUL
- #define __INT_FAST64_WIDTH__ 64
- #define __DEC64_MIN_EXP__ (-382)
- #define __cpp_decltype 200707
- #define __FLT32_DECIMAL_DIG__ 9
- #define __INT_FAST64_MAX__ 0x7fffffffffffffffL
- #define __GCC_ATOMIC_TEST_AND_SET_TRUEVAL 1
- #define __FLT_DIG__ 6
- #define __FLT64X_MAX_EXP__ 16384
- #define __UINT_FAST64_TYPE__ long unsigned int
- #define __INT_MAX__ 0x7fffffff
- #define __amd64__ 1
- #define __INT64_TYPE__ long int
- #define __FLT_MAX_EXP__ 128
- #define __ORDER_BIG_ENDIAN__ 4321
- #define __DBL_MANT_DIG__ 53
- #define __cpp_inheriting_constructors 201511
- #define __SIZEOF_FLOAT128__ 16
- #define __INT_LEAST64_MAX__ 0x7fffffffffffffffL
- #define __DEC64_MIN__ 1E-383DD
- #define __WINT_TYPE__ unsigned int
- #define __UINT_LEAST32_TYPE__ unsigned int
- #define __SIZEOF_SHORT__ 2
- #define __SSE__ 1
- #define __LDBL_MIN_EXP__ (-16381)
- #define __FLT64_MAX__ 1.79769313486231570814527423731704357e+308F64
- #define __WINT_WIDTH__ 32
- #define __INT_LEAST8_MAX__ 0x7f
- #define __FLT32X_MAX_10_EXP__ 308
- #define __SIZEOF_INT128__ 16
- #define __LDBL_MAX_10_EXP__ 4932
- #define __ATOMIC_RELAXED 0
- #define __DBL_EPSILON__ double(2.22044604925031308084726333618164062e-16L)
- #define __FLT128_MIN__ 3.36210314311209350626267781732175260e-4932F128
- #define _LP64 1
- #define __UINT8_C(c) c
- #define __FLT64_MAX_EXP__ 1024
- #define __INT_LEAST32_TYPE__ int
- #define __SIZEOF_WCHAR_T__ 4
- #define __FLT128_HAS_QUIET_NAN__ 1
- #define __INT_FAST8_TYPE__ signed char
- #define __FLT64X_MIN__ 3.36210314311209350626267781732175260e-4932F64x
- #define __GNUC_STDC_INLINE__ 1
- #define __FLT64_HAS_DENORM__ 1
- #define __FLT32_EPSILON__ 1.19209289550781250000000000000000000e-7F32
- #define __DBL_DECIMAL_DIG__ 17
- #define __STDC_UTF_32__ 1
- #define __INT_FAST8_WIDTH__ 8
- #define __FXSR__ 1
- #define __DEC_EVAL_METHOD__ 2
- #define __FLT32X_MAX__ 1.79769313486231570814527423731704357e+308F32x
- #define __cpp_runtime_arrays 198712
- #define __UINT64_TYPE__ long unsigned int

- #define __UINT32_C(c) c ## U
- #define __INTMAX_MAX__ 0x7fffffffffffffffL
- #define __cpp_alias_templates 200704
- #define __BYTE_ORDER__ __ORDER_LITTLE_ENDIAN__
- #define __FLT_DENORM_MIN__ 1.40129846432481707092372958328991613e-45F
- #define __INT8_MAX__ 0x7f
- #define __LONG_WIDTH__ 64
- #define __PIC__ 2
- #define __UINT_FAST32_TYPE__ long unsigned int
- #define __CHAR32_TYPE__ unsigned int
- #define __FLT_MAX__ 3.40282346638528859811704183484516925e+38F
- #define __cpp_constexpr 200704
- #define __INT32_TYPE__ int
- #define __SIZEOF_DOUBLE__ 8
- #define __cpp_exceptions 199711
- #define __FLT_MIN_10_EXP__ (-37)
- #define __FLT64_MIN__ 2.22507385850720138309023271733240406e-308F64
- #define __INT_LEAST32_WIDTH__ 32
- #define __INTMAX_TYPE__ long int
- #define __DEC128_MAX_EXP__ 6145
- #define __FLT32X_HAS_QUIET_NAN__ 1
- #define __ATOMIC_CONSUME 1
- #define __GNUC_MINOR__ 3
- #define __GLIBCXX_TYPE_INT_N_0 __int128
- #define __INT_FAST16_WIDTH__ 64
- #define __UINTMAX_MAX__ 0xffffffffffffffffUL
- #define __DEC32_MANT_DIG__ 7
- #define __FLT32X_DENORM_MIN__ 4.94065645841246544176568792868221372e-324F32x
- #define __DBL_MAX_10_EXP__ 308
- #define __LDBL_DENORM_MIN__ 3.64519953188247460252840593361941982e-4951L
- #define __INT16_C(c) c
- #define __STDC__ 1
- #define __FLT32X_DIG__ 15
- #define __PTRDIFF_TYPE__ long int
- #define __ATOMIC_SEQ_CST 5
- #define __UINT32_TYPE__ unsigned int
- #define __FLT32X_MIN_10_EXP__ (-307)
- #define __UINTPTR_TYPE__ long unsigned int
- #define __DEC64_SUBNORMAL_MIN__ 0.000000000000001E-383DD
- #define __DEC128_MANT_DIG__ 34
- #define __LDBL_MIN_10_EXP__ (-4931)
- #define __FLT128_EPSILON__ 1.92592994438723585305597794258492732e-34F128
- #define __SSE_MATH__ 1
- #define __SIZEOF_LONG_LONG__ 8
- #define __cpp_user_defined_literals 200809
- #define __FLT128_DECIMAL_DIG__ 36
- #define __GCC_ATOMIC_LLONG_LOCK_FREE 2
- #define __FLT32X_MIN__ 2.22507385850720138309023271733240406e-308F32x
- #define __LDBL_DIG__ 18
- #define __FLT_DECIMAL_DIG__ 9
- #define __UINT_FAST16_MAX__ 0xffffffffffffffffUL
- #define __GCC_ATOMIC_SHORT_LOCK_FREE 2
- #define __INT_LEAST64_WIDTH__ 64
- #define __UINT_FAST8_TYPE__ unsigned char
- #define _GNU_SOURCE 1
- #define __ATOMIC_ACQ_REL 4
- #define __ATOMIC_RELEASE 3

9.10.1 Macro Definition Documentation

9.10.1.1 __amd64

```
#define __amd64 1
```

9.10.1.2 __amd64__

```
#define __amd64__ 1
```

9.10.1.3 __ATOMIC_ACQ_REL

```
#define __ATOMIC_ACQ_REL 4
```

9.10.1.4 __ATOMIC_ACQUIRE

```
#define __ATOMIC_ACQUIRE 2
```

9.10.1.5 __ATOMIC_CONSUME

```
#define __ATOMIC_CONSUME 1
```

9.10.1.6 __ATOMIC_HLE_ACQUIRE

```
#define __ATOMIC_HLE_ACQUIRE 65536
```

9.10.1.7 __ATOMIC_HLE_RELEASE

```
#define __ATOMIC_HLE_RELEASE 131072
```

9.10.1.8 __ATOMIC_RELAXED

```
#define __ATOMIC_RELAXED 0
```

9.10.1.9 __ATOMIC_RELEASE

```
#define __ATOMIC_RELEASE 3
```

9.10.1.10 __ATOMIC_SEQ_CST

```
#define __ATOMIC_SEQ_CST 5
```

9.10.1.11 __BIGGEST_ALIGNMENT__

```
#define __BIGGEST_ALIGNMENT__ 16
```

9.10.1.12 __BYTE_ORDER__

```
#define __BYTE_ORDER__ __ORDER_LITTLE_ENDIAN__
```

9.10.1.13 __CHAR16_TYPE__

```
#define __CHAR16_TYPE__ short unsigned int
```

9.10.1.14 __CHAR32_TYPE__

```
#define __CHAR32_TYPE__ unsigned int
```

9.10.1.15 __CHAR_BIT__

```
#define __CHAR_BIT__ 8
```

9.10.1.16 __code_model_small__

```
#define __code_model_small__ 1
```

9.10.1.17 __cplusplus

```
#define __cplusplus 201103L
```

9.10.1.18 __cpp_alias_templates

```
#define __cpp_alias_templates 200704
```

9.10.1.19 __cpp_attributes

```
#define __cpp_attributes 200809
```

9.10.1.20 __cpp_binary_literals

```
#define __cpp_binary_literals 201304
```

9.10.1.21 __cpp_constexpr

```
#define __cpp_constexpr 200704
```

9.10.1.22 __cpp_decltype

```
#define __cpp_decltype 200707
```

9.10.1.23 __cpp_delegating_constructors

```
#define __cpp_delegating_constructors 200604
```

9.10.1.24 __cpp_exceptions

```
#define __cpp_exceptions 199711
```

9.10.1.25 __cpp_hex_float

```
#define __cpp_hex_float 201603
```

9.10.1.26 __cpp_inheriting_constructors

```
#define __cpp_inheriting_constructors 201511
```

9.10.1.27 __cpp_initializer_lists

```
#define __cpp_initializer_lists 200806
```

9.10.1.28 __cpp_lambdas

```
#define __cpp_lambdas 200907
```

9.10.1.29 __cpp_nsdmi

```
#define __cpp_nsdmi 200809
```

9.10.1.30 __cpp_range_based_for

```
#define __cpp_range_based_for 200907
```

9.10.1.31 __cpp_raw_strings

```
#define __cpp_raw_strings 200710
```

9.10.1.32 __cpp_ref_qualifiers

```
#define __cpp_ref_qualifiers 200710
```

9.10.1.33 __cpp_rtti

```
#define __cpp_rtti 199711
```

9.10.1.34 __cpp_runtime_arrays

```
#define __cpp_runtime_arrays 198712
```

9.10.1.35 __cpp_rvalue_reference

```
#define __cpp_rvalue_reference 200610
```

9.10.1.36 __cpp_rvalue_references

```
#define __cpp_rvalue_references 200610
```

9.10.1.37 __cpp_static_assert

```
#define __cpp_static_assert 200410
```

9.10.1.38 __cpp_threadsafe_static_init

```
#define __cpp_threadsafe_static_init 200806
```

9.10.1.39 __cpp_unicode_characters

```
#define __cpp_unicode_characters 200704
```

9.10.1.40 __cpp_unicode_literals

```
#define __cpp_unicode_literals 200710
```

9.10.1.41 __cpp_user_defined_literals

```
#define __cpp_user_defined_literals 200809
```

9.10.1.42 __cpp_variadic_templates

```
#define __cpp_variadic_templates 200704
```

9.10.1.43 __DBL_DECIMAL_DIG__

```
#define __DBL_DECIMAL_DIG__ 17
```

9.10.1.44 __DBL_DENORM_MIN__

```
#define __DBL_DENORM_MIN__ double(4.94065645841246544176568792868221372e-324L)
```

9.10.1.45 __DBL_DIG__

```
#define __DBL_DIG__ 15
```

9.10.1.46 __DBL_EPSILON__

```
#define __DBL_EPSILON__ double(2.22044604925031308084726333618164062e-16L)
```

9.10.1.47 __DBL_HAS_DENORM__

```
#define __DBL_HAS_DENORM__ 1
```

9.10.1.48 __DBL_HAS_INFINITY__

```
#define __DBL_HAS_INFINITY__ 1
```

9.10.1.49 __DBL_HAS_QUIET_NAN__

```
#define __DBL_HAS_QUIET_NAN__ 1
```

9.10.1.50 __DBL_MANT_DIG__

```
#define __DBL_MANT_DIG__ 53
```

9.10.1.51 __DBL_MAX_10_EXP__

```
#define __DBL_MAX_10_EXP__ 308
```

9.10.1.52 __DBL_MAX__

```
#define __DBL_MAX__ double(1.79769313486231570814527423731704357e+308L)
```

9.10.1.53 __DBL_MAX_EXP__

```
#define __DBL_MAX_EXP__ 1024
```

9.10.1.54 __DBL_MIN_10_EXP__

```
#define __DBL_MIN_10_EXP__ (-307)
```

9.10.1.55 __DBL_MIN__

```
#define __DBL_MIN__ double(2.22507385850720138309023271733240406e-308L)
```

9.10.1.56 __DBL_MIN_EXP__

```
#define __DBL_MIN_EXP__ (-1021)
```

9.10.1.57 __DEC128_EPSILON__

```
#define __DEC128_EPSILON__ 1E-33DL
```

9.10.1.58 __DEC128_MANT_DIG__

```
#define __DEC128_MANT_DIG__ 34
```

9.10.1.59 __DEC128_MAX__

```
#define __DEC128_MAX__ 9.999999999999999999999999999999999E6144DL
```

9.10.1.60 __DEC128_MAX_EXP__

```
#define __DEC128_MAX_EXP__ 6145
```

9.10.1.61 __DEC128_MIN__

```
#define __DEC128_MIN__ 1E-6143DL
```

9.10.1.62 __DEC128_MIN_EXP__

```
#define __DEC128_MIN_EXP__ (-6142)
```

9.10.1.63 __DEC128_SUBNORMAL_MIN__

```
#define __DEC128_SUBNORMAL_MIN__ 0.000000000000000000000000000000001E-6143DL
```

9.10.1.64 __DEC32_EPSILON__

```
#define __DEC32_EPSILON__ 1E-6DF
```

9.10.1.65 __DEC32_MANT_DIG__

```
#define __DEC32_MANT_DIG__ 7
```

9.10.1.66 __DEC32_MAX__

```
#define __DEC32_MAX__ 9.999999E96DF
```

9.10.1.67 __DEC32_MAX_EXP__

```
#define __DEC32_MAX_EXP__ 97
```

9.10.1.68 __DEC32_MIN__

```
#define __DEC32_MIN__ 1E-95DF
```

9.10.1.69 __DEC32_MIN_EXP__

```
#define __DEC32_MIN_EXP__ (-94)
```

9.10.1.70 __DEC32_SUBNORMAL_MIN__

```
#define __DEC32_SUBNORMAL_MIN__ 0.000001E-95DF
```

9.10.1.71 __DEC64_EPSILON__

```
#define __DEC64_EPSILON__ 1E-15DD
```

9.10.1.72 __DEC64_MANT_DIG__

```
#define __DEC64_MANT_DIG__ 16
```

9.10.1.73 __DEC64_MAX__

```
#define __DEC64_MAX__ 9.999999999999999E384DD
```

9.10.1.74 __DEC64_MAX_EXP__

```
#define __DEC64_MAX_EXP__ 385
```

9.10.1.75 __DEC64_MIN__

```
#define __DEC64_MIN__ 1E-383DD
```

9.10.1.76 __DEC64_MIN_EXP__

```
#define __DEC64_MIN_EXP__ (-382)
```

9.10.1.77 __DEC64_SUBNORMAL_MIN__

```
#define __DEC64_SUBNORMAL_MIN__ 0.000000000000001E-383DD
```

9.10.1.78 __DEC_EVAL_METHOD__

```
#define __DEC_EVAL_METHOD__ 2
```

9.10.1.79 __DECIMAL_BID_FORMAT__

```
#define __DECIMAL_BID_FORMAT__ 1
```

9.10.1.80 __DECIMAL_DIG__

```
#define __DECIMAL_DIG__ 21
```

9.10.1.81 __DEPRECATED

```
#define __DEPRECATED 1
```

9.10.1.82 __ELF__

```
#define __ELF__ 1
```

9.10.1.83 __EXCEPTIONS

```
#define __EXCEPTIONS 1
```

9.10.1.84 __FINITE_MATH_ONLY__

```
#define __FINITE_MATH_ONLY__ 0
```

9.10.1.85 __FLOAT_WORD_ORDER__

```
#define __FLOAT_WORD_ORDER__ __ORDER_LITTLE_ENDIAN__
```

9.10.1.86 __FLT128_DECIMAL_DIG__

```
#define __FLT128_DECIMAL_DIG__ 36
```

9.10.1.87 __FLT128_DENORM_MIN__

```
#define __FLT128_DENORM_MIN__ 6.47517511943802511092443895822764655e-4966F128
```

9.10.1.88 __FLT128_DIG__

```
#define __FLT128_DIG__ 33
```

9.10.1.89 __FLT128_EPSILON__

```
#define __FLT128_EPSILON__ 1.92592994438723585305597794258492732e-34F128
```

9.10.1.90 __FLT128_HAS_DENORM__

```
#define __FLT128_HAS_DENORM__ 1
```

9.10.1.91 __FLT128_HAS_INFINITY__

```
#define __FLT128_HAS_INFINITY__ 1
```

9.10.1.92 __FLT128_HAS_QUIET_NAN__

```
#define __FLT128_HAS_QUIET_NAN__ 1
```

9.10.1.93 __FLT128_MANT_DIG__

```
#define __FLT128_MANT_DIG__ 113
```

9.10.1.94 __FLT128_MAX_10_EXP__

```
#define __FLT128_MAX_10_EXP__ 4932
```

9.10.1.95 __FLT128_MAX__

```
#define __FLT128_MAX__ 1.18973149535723176508575932662800702e+4932F128
```

9.10.1.96 __FLT128_MAX_EXP__

```
#define __FLT128_MAX_EXP__ 16384
```

9.10.1.97 __FLT128_MIN_10_EXP__

```
#define __FLT128_MIN_10_EXP__ (-4931)
```

9.10.1.98 __FLT128_MIN__

```
#define __FLT128_MIN__ 3.36210314311209350626267781732175260e-4932F128
```

9.10.1.99 __FLT128_MIN_EXP__

```
#define __FLT128_MIN_EXP__ (-16381)
```

9.10.1.100 __FLT32_DECIMAL_DIG__

```
#define __FLT32_DECIMAL_DIG__ 9
```

9.10.1.101 __FLT32_DENORM_MIN__

```
#define __FLT32_DENORM_MIN__ 1.40129846432481707092372958328991613e-45F32
```

9.10.1.102 __FLT32_DIG__

```
#define __FLT32_DIG__ 6
```

9.10.1.103 __FLT32_EPSILON__

```
#define __FLT32_EPSILON__ 1.19209289550781250000000000000000000e-7F32
```

9.10.1.104 __FLT32_HAS_DENORM__

```
#define __FLT32_HAS_DENORM__ 1
```

9.10.1.105 __FLT32_HAS_INFINITY__

```
#define __FLT32_HAS_INFINITY__ 1
```

9.10.1.106 __FLT32_HAS_QUIET_NAN__

```
#define __FLT32_HAS_QUIET_NAN__ 1
```

9.10.1.107 __FLT32_MANT_DIG__

```
#define __FLT32_MANT_DIG__ 24
```

9.10.1.108 __FLT32_MAX_10_EXP__

```
#define __FLT32_MAX_10_EXP__ 38
```

9.10.1.109 __FLT32_MAX__

```
#define __FLT32_MAX__ 3.40282346638528859811704183484516925e+38F32
```

9.10.1.110 __FLT32_MAX_EXP__

```
#define __FLT32_MAX_EXP__ 128
```

9.10.1.111 __FLT32_MIN_10_EXP__

```
#define __FLT32_MIN_10_EXP__ (-37)
```

9.10.1.112 __FLT32_MIN__

```
#define __FLT32_MIN__ 1.17549435082228750796873653722224568e-38F32
```

9.10.1.113 __FLT32_MIN_EXP__

```
#define __FLT32_MIN_EXP__ (-125)
```

9.10.1.114 __FLT32X_DECIMAL_DIG__

```
#define __FLT32X_DECIMAL_DIG__ 17
```

9.10.1.115 __FLT32X_DENORM_MIN__

```
#define __FLT32X_DENORM_MIN__ 4.94065645841246544176568792868221372e-324F32x
```

9.10.1.116 __FLT32X_DIG__

```
#define __FLT32X_DIG__ 15
```

9.10.1.117 __FLT32X_EPSILON__

```
#define __FLT32X_EPSILON__ 2.22044604925031308084726333618164062e-16F32x
```

9.10.1.118 __FLT32X_HAS_DENORM__

```
#define __FLT32X_HAS_DENORM__ 1
```

9.10.1.119 __FLT32X_HAS_INFINITY__

```
#define __FLT32X_HAS_INFINITY__ 1
```

9.10.1.120 __FLT32X_HAS_QUIET_NAN__

```
#define __FLT32X_HAS_QUIET_NAN__ 1
```

9.10.1.121 __FLT32X_MANT_DIG__

```
#define __FLT32X_MANT_DIG__ 53
```

9.10.1.122 __FLT32X_MAX_10_EXP__

```
#define __FLT32X_MAX_10_EXP__ 308
```

9.10.1.123 __FLT32X_MAX__

```
#define __FLT32X_MAX__ 1.79769313486231570814527423731704357e+308F32x
```

9.10.1.124 __FLT32X_MAX_EXP__

```
#define __FLT32X_MAX_EXP__ 1024
```

9.10.1.125 __FLT32X_MIN_10_EXP__

```
#define __FLT32X_MIN_10_EXP__ (-307)
```

9.10.1.126 __FLT32X_MIN__

```
#define __FLT32X_MIN__ 2.22507385850720138309023271733240406e-308F32x
```

9.10.1.127 __FLT32X_MIN_EXP__

```
#define __FLT32X_MIN_EXP__ (-1021)
```

9.10.1.128 __FLT64_DECIMAL_DIG__

```
#define __FLT64_DECIMAL_DIG__ 17
```

9.10.1.129 __FLT64_DENORM_MIN__

```
#define __FLT64_DENORM_MIN__ 4.94065645841246544176568792868221372e-324F64
```

9.10.1.130 __FLT64_DIG__

```
#define __FLT64_DIG__ 15
```

9.10.1.131 __FLT64_EPSILON__

```
#define __FLT64_EPSILON__ 2.22044604925031308084726333618164062e-16F64
```

9.10.1.132 __FLT64_HAS_DENORM__

```
#define __FLT64_HAS_DENORM__ 1
```

9.10.1.133 __FLT64_HAS_INFINITY__

```
#define __FLT64_HAS_INFINITY__ 1
```

9.10.1.134 __FLT64_HAS_QUIET_NAN__

```
#define __FLT64_HAS_QUIET_NAN__ 1
```

9.10.1.135 __FLT64_MANT_DIG__

```
#define __FLT64_MANT_DIG__ 53
```

9.10.1.136 __FLT64_MAX_10_EXP__

```
#define __FLT64_MAX_10_EXP__ 308
```

9.10.1.137 __FLT64_MAX__

```
#define __FLT64_MAX__ 1.79769313486231570814527423731704357e+308F64
```

9.10.1.138 __FLT64_MAX_EXP__

```
#define __FLT64_MAX_EXP__ 1024
```

9.10.1.139 __FLT64_MIN_10_EXP__

```
#define __FLT64_MIN_10_EXP__ (-307)
```

9.10.1.140 __FLT64_MIN__

```
#define __FLT64_MIN__ 2.22507385850720138309023271733240406e-308F64
```

9.10.1.141 __FLT64_MIN_EXP__

```
#define __FLT64_MIN_EXP__ (-1021)
```

9.10.1.142 __FLT64X_DECIMAL_DIG__

```
#define __FLT64X_DECIMAL_DIG__ 21
```

9.10.1.143 __FLT64X_DENORM_MIN__

```
#define __FLT64X_DENORM_MIN__ 3.64519953188247460252840593361941982e-4951F64x
```

9.10.1.144 __FLT64X_DIG__

```
#define __FLT64X_DIG__ 18
```

9.10.1.145 __FLT64X_EPSILON__

```
#define __FLT64X_EPSILON__ 1.08420217248550443400745280086994171e-19F64x
```

9.10.1.146 __FLT64X_HAS_DENORM__

```
#define __FLT64X_HAS_DENORM__ 1
```

9.10.1.147 __FLT64X_HAS_INFINITY__

```
#define __FLT64X_HAS_INFINITY__ 1
```

9.10.1.148 __FLT64X_HAS_QUIET_NAN__

```
#define __FLT64X_HAS_QUIET_NAN__ 1
```

9.10.1.149 __FLT64X_MANT_DIG__

```
#define __FLT64X_MANT_DIG__ 64
```

9.10.1.150 __FLT64X_MAX_10_EXP__

```
#define __FLT64X_MAX_10_EXP__ 4932
```

9.10.1.151 __FLT64X_MAX__

```
#define __FLT64X_MAX__ 1.18973149535723176502126385303097021e+4932F64x
```

9.10.1.152 __FLT64X_MAX_EXP__

```
#define __FLT64X_MAX_EXP__ 16384
```

9.10.1.153 __FLT64X_MIN_10_EXP__

```
#define __FLT64X_MIN_10_EXP__ (-4931)
```

9.10.1.154 __FLT64X_MIN__

```
#define __FLT64X_MIN__ 3.36210314311209350626267781732175260e-4932F64x
```

9.10.1.155 __FLT64X_MIN_EXP__

```
#define __FLT64X_MIN_EXP__ (-16381)
```

9.10.1.156 __FLT_DECIMAL_DIG__

```
#define __FLT_DECIMAL_DIG__ 9
```

9.10.1.157 __FLT_DENORM_MIN__

```
#define __FLT_DENORM_MIN__ 1.40129846432481707092372958328991613e-45F
```

9.10.1.158 __FLT_DIG__

```
#define __FLT_DIG__ 6
```

9.10.1.159 __FLT_EPSILON__

```
#define __FLT_EPSILON__ 1.19209289550781250000000000000000000e-7F
```

9.10.1.160 __FLT_EVAL_METHOD__

```
#define __FLT_EVAL_METHOD__ 0
```

9.10.1.161 __FLT_EVAL_METHOD_TS_18661_3__

```
#define __FLT_EVAL_METHOD_TS_18661_3__ 0
```

9.10.1.162 __FLT_HAS_DENORM__

```
#define __FLT_HAS_DENORM__ 1
```

9.10.1.163 __FLT_HAS_INFINITY__

```
#define __FLT_HAS_INFINITY__ 1
```

9.10.1.164 __FLT_HAS_QUIET_NAN__

```
#define __FLT_HAS_QUIET_NAN__ 1
```

9.10.1.165 __FLT_MANT_DIG__

```
#define __FLT_MANT_DIG__ 24
```

9.10.1.166 __FLT_MAX_10_EXP__

```
#define __FLT_MAX_10_EXP__ 38
```

9.10.1.167 __FLT_MAX__

```
#define __FLT_MAX__ 3.40282346638528859811704183484516925e+38F
```

9.10.1.168 __FLT_MAX_EXP__

```
#define __FLT_MAX_EXP__ 128
```

9.10.1.169 __FLT_MIN_10_EXP__

```
#define __FLT_MIN_10_EXP__ (-37)
```

9.10.1.170 __FLT_MIN__

```
#define __FLT_MIN__ 1.17549435082228750796873653722224568e-38F
```

9.10.1.171 __FLT_MIN_EXP__

```
#define __FLT_MIN_EXP__ (-125)
```

9.10.1.172 __FLT_RADIX__

```
#define __FLT_RADIX__ 2
```

9.10.1.173 __FXSR__

```
#define __FXSR__ 1
```

9.10.1.174 __GCC_ASM_FLAG_OUTPUTS__

```
#define __GCC_ASM_FLAG_OUTPUTS__ 1
```

9.10.1.175 __GCC_ATOMIC_BOOL_LOCK_FREE

```
#define __GCC_ATOMIC_BOOL_LOCK_FREE 2
```

9.10.1.176 __GCC_ATOMIC_CHAR16_T_LOCK_FREE

```
#define __GCC_ATOMIC_CHAR16_T_LOCK_FREE 2
```

9.10.1.177 __GCC_ATOMIC_CHAR32_T_LOCK_FREE

```
#define __GCC_ATOMIC_CHAR32_T_LOCK_FREE 2
```

9.10.1.178 __GCC_ATOMIC_CHAR_LOCK_FREE

```
#define __GCC_ATOMIC_CHAR_LOCK_FREE 2
```

9.10.1.179 __GCC_ATOMIC_INT_LOCK_FREE

```
#define __GCC_ATOMIC_INT_LOCK_FREE 2
```

9.10.1.180 __GCC_ATOMIC_LLONG_LOCK_FREE

```
#define __GCC_ATOMIC_LLONG_LOCK_FREE 2
```

9.10.1.181 __GCC_ATOMIC_LONG_LOCK_FREE

```
#define __GCC_ATOMIC_LONG_LOCK_FREE 2
```

9.10.1.182 __GCC_ATOMIC_POINTER_LOCK_FREE

```
#define __GCC_ATOMIC_POINTER_LOCK_FREE 2
```

9.10.1.183 __GCC_ATOMIC_SHORT_LOCK_FREE

```
#define __GCC_ATOMIC_SHORT_LOCK_FREE 2
```

9.10.1.184 __GCC_ATOMIC_TEST_AND_SET_TRUEVAL

```
#define __GCC_ATOMIC_TEST_AND_SET_TRUEVAL 1
```

9.10.1.185 __GCC_ATOMIC_WCHAR_T_LOCK_FREE

```
#define __GCC_ATOMIC_WCHAR_T_LOCK_FREE 2
```

9.10.1.186 __GCC_HAVE_DWARF2_CFI_ASM

```
#define __GCC_HAVE_DWARF2_CFI_ASM 1
```

9.10.1.187 __GCC_HAVE_SYNC_COMPARE_AND_SWAP_1

```
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_1 1
```

9.10.1.188 __GCC_HAVE_SYNC_COMPARE_AND_SWAP_2

```
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_2 1
```

9.10.1.189 __GCC_HAVE_SYNC_COMPARE_AND_SWAP_4

```
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_4 1
```

9.10.1.190 __GCC_HAVE_SYNC_COMPARE_AND_SWAP_8

```
#define __GCC_HAVE_SYNC_COMPARE_AND_SWAP_8 1
```

9.10.1.191 __GCC_IEC_559

```
#define __GCC_IEC_559 2
```

9.10.1.192 __GCC_IEC_559_COMPLEX

```
#define __GCC_IEC_559_COMPLEX 2
```

9.10.1.193 __GLIBCXX_BITSIZE_INT_N_0

```
#define __GLIBCXX_BITSIZE_INT_N_0 128
```

9.10.1.194 __GLIBCXX_TYPE_INT_N_0

```
#define __GLIBCXX_TYPE_INT_N_0 __int128
```

9.10.1.195 __gnu_linux__

```
#define __gnu_linux__ 1
```

9.10.1.196 __GNUC__

```
#define __GNUC__ 7
```

9.10.1.197 __GNUC_MINOR__

```
#define __GNUC_MINOR__ 3
```

9.10.1.198 __GNUC_PATCHLEVEL__

```
#define __GNUC_PATCHLEVEL__ 0
```

9.10.1.199 __GNUC_STDC_INLINE__

```
#define __GNUC_STDC_INLINE__ 1
```

9.10.1.200 __GNUG__

```
#define __GNUG__ 7
```

9.10.1.201 __GXX_ABI_VERSION

```
#define __GXX_ABI_VERSION 1011
```

9.10.1.202 __GXX_EXPERIMENTAL_CXX0X__

```
#define __GXX_EXPERIMENTAL_CXX0X__ 1
```

9.10.1.203 __GXX_RTTI

```
#define __GXX_RTTI 1
```

9.10.1.204 __GXX_WEAK__

```
#define __GXX_WEAK__ 1
```

9.10.1.205 __has_include

```
#define __has_include(
              STR ) __has_include__(STR)
```

9.10.1.206 __has_include_next

```
#define __has_include_next(
              STR ) __has_include_next__(STR)
```

9.10.1.207 __INT16_C

```
#define __INT16_C(
              c ) c
```

9.10.1.208 __INT16_MAX__

```
#define __INT16_MAX__ 0x7fff
```

9.10.1.209 __INT16_TYPE__

```
#define __INT16_TYPE__ short int
```

9.10.1.210 __INT32_C

```
#define __INT32_C(
              c ) c
```

9.10.1.211 __INT32_MAX__

```
#define __INT32_MAX__ 0x7fffffff
```

9.10.1.212 __INT32_TYPE__

```
#define __INT32_TYPE__ int
```

9.10.1.213 __INT64_C

```
#define __INT64_C(
              c ) c ## L
```

9.10.1.214 __INT64_MAX__

```
#define __INT64_MAX__ 0x7fffffffffffffffL
```

9.10.1.215 __INT64_TYPE__

```
#define __INT64_TYPE__ long int
```

9.10.1.216 __INT8_C

```
#define __INT8_C(
              c ) c
```

9.10.1.217 __INT8_MAX__

```
#define __INT8_MAX__ 0x7f
```

9.10.1.218 __INT8_TYPE__

```
#define __INT8_TYPE__ signed char
```

9.10.1.219 __INT_FAST16_MAX__

```
#define __INT_FAST16_MAX__ 0x7fffffffffffffffL
```

9.10.1.220 __INT_FAST16_TYPE__

```
#define __INT_FAST16_TYPE__ long int
```

9.10.1.221 __INT_FAST16_WIDTH__

```
#define __INT_FAST16_WIDTH__ 64
```

9.10.1.222 __INT_FAST32_MAX__

```
#define __INT_FAST32_MAX__ 0x7fffffffffffffffL
```

9.10.1.223 __INT_FAST32_TYPE__

```
#define __INT_FAST32_TYPE__ long int
```

9.10.1.224 __INT_FAST32_WIDTH__

```
#define __INT_FAST32_WIDTH__ 64
```

9.10.1.225 __INT_FAST64_MAX__

```
#define __INT_FAST64_MAX__ 0x7fffffffffffffffL
```

9.10.1.226 __INT_FAST64_TYPE__

```
#define __INT_FAST64_TYPE__ long int
```

9.10.1.227 __INT_FAST64_WIDTH__

```
#define __INT_FAST64_WIDTH__ 64
```

9.10.1.228 __INT_FAST8_MAX__

```
#define __INT_FAST8_MAX__ 0x7f
```

9.10.1.229 __INT_FAST8_TYPE__

```
#define __INT_FAST8_TYPE__ signed char
```

9.10.1.230 __INT_FAST8_WIDTH__

```
#define __INT_FAST8_WIDTH__ 8
```

9.10.1.231 __INT_LEAST16_MAX__

```
#define __INT_LEAST16_MAX__ 0x7fff
```

9.10.1.232 __INT_LEAST16_TYPE__

```
#define __INT_LEAST16_TYPE__ short int
```

9.10.1.233 __INT_LEAST16_WIDTH__

```
#define __INT_LEAST16_WIDTH__ 16
```

9.10.1.234 __INT_LEAST32_MAX__

```
#define __INT_LEAST32_MAX__ 0x7fffffff
```

9.10.1.235 __INT_LEAST32_TYPE__

```
#define __INT_LEAST32_TYPE__ int
```

9.10.1.236 __INT_LEAST32_WIDTH__

```
#define __INT_LEAST32_WIDTH__ 32
```

9.10.1.237 __INT_LEAST64_MAX__

```
#define __INT_LEAST64_MAX__ 0x7fffffffffffffffL
```

9.10.1.238 __INT_LEAST64_TYPE__

```
#define __INT_LEAST64_TYPE__ long int
```

9.10.1.239 __INT_LEAST64_WIDTH__

```
#define __INT_LEAST64_WIDTH__ 64
```

9.10.1.240 __INT_LEAST8_MAX__

```
#define __INT_LEAST8_MAX__ 0x7f
```

9.10.1.241 __INT_LEAST8_TYPE__

```
#define __INT_LEAST8_TYPE__ signed char
```

9.10.1.242 __INT_LEAST8_WIDTH__

```
#define __INT_LEAST8_WIDTH__ 8
```

9.10.1.243 __INT_MAX__

```
#define __INT_MAX__ 0x7fffffff
```

9.10.1.244 __INT_WIDTH__

```
#define __INT_WIDTH__ 32
```

9.10.1.245 __INTMAX_C

```
#define __INTMAX_C(
              c ) c ## L
```

9.10.1.246 __INTMAX_MAX__

```
#define __INTMAX_MAX__ 0x7fffffffffffffffL
```

9.10.1.247 __INTMAX_TYPE__

```
#define __INTMAX_TYPE__ long int
```

9.10.1.248 __INTMAX_WIDTH__

```
#define __INTMAX_WIDTH__ 64
```

9.10.1.249 __INTPTR_MAX__

```
#define __INTPTR_MAX__ 0x7fffffffffffffffL
```

9.10.1.250 __INTPTR_TYPE__

```
#define __INTPTR_TYPE__ long int
```

9.10.1.251 __INTPTR_WIDTH__

```
#define __INTPTR_WIDTH__ 64
```

9.10.1.252 __k8

```
#define __k8 1
```

9.10.1.253 __k8__

```
#define __k8__ 1
```

9.10.1.254 __LDBL_DECIMAL_DIG__

```
#define __LDBL_DECIMAL_DIG__ 21
```

9.10.1.255 __LDBL_DENORM_MIN__

```
#define __LDBL_DENORM_MIN__ 3.64519953188247460252840593361941982e-4951L
```

9.10.1.256 __LDBL_DIG__

```
#define __LDBL_DIG__ 18
```

9.10.1.257 __LDBL_EPSILON__

```
#define __LDBL_EPSILON__ 1.08420217248550443400745280086994171e-19L
```

9.10.1.258 __LDBL_HAS_DENORM__

```
#define __LDBL_HAS_DENORM__ 1
```

9.10.1.259 __LDBL_HAS_INFINITY__

```
#define __LDBL_HAS_INFINITY__ 1
```

9.10.1.260 __LDBL_HAS_QUIET_NAN__

```
#define __LDBL_HAS_QUIET_NAN__ 1
```

9.10.1.261 __LDBL_MANT_DIG__

```
#define __LDBL_MANT_DIG__ 64
```

9.10.1.262 __LDBL_MAX_10_EXP__

```
#define __LDBL_MAX_10_EXP__ 4932
```

9.10.1.263 __LDBL_MAX__

```
#define __LDBL_MAX__ 1.18973149535723176502126385303097021e+4932L
```

9.10.1.264 __LDBL_MAX_EXP__

```
#define __LDBL_MAX_EXP__ 16384
```

9.10.1.265 __LDBL_MIN_10_EXP__

```
#define __LDBL_MIN_10_EXP__ (-4931)
```

9.10.1.266 __LDBL_MIN__

```
#define __LDBL_MIN__ 3.36210314311209350626267781732175260e-4932L
```

9.10.1.267 __LDBL_MIN_EXP__

```
#define __LDBL_MIN_EXP__ (-16381)
```

9.10.1.268 __linux

```
#define __linux 1
```

9.10.1.269 __linux__

```
#define __linux__ 1
```

9.10.1.270 __LONG_LONG_MAX__

```
#define __LONG_LONG_MAX__ 0x7fffffffffffffffLL
```

9.10.1.271 __LONG_LONG_WIDTH__

```
#define __LONG_LONG_WIDTH__ 64
```

9.10.1.272 __LONG_MAX__

```
#define __LONG_MAX__ 0x7fffffffffffffffL
```

9.10.1.273 __LONG_WIDTH__

```
#define __LONG_WIDTH__ 64
```

9.10.1.274 __LP64__

```
#define __LP64__ 1
```

9.10.1.275 __MMX__

```
#define __MMX__ 1
```

9.10.1.276 __OPTIMIZE__

```
#define __OPTIMIZE__ 1
```

9.10.1.277 __ORDER_BIG_ENDIAN__

```
#define __ORDER_BIG_ENDIAN__ 4321
```

9.10.1.278 __ORDER_LITTLE_ENDIAN__

```
#define __ORDER_LITTLE_ENDIAN__ 1234
```

9.10.1.279 __ORDER_PDP_ENDIAN__

```
#define __ORDER_PDP_ENDIAN__ 3412
```

9.10.1.280 __pic__

```
#define __pic__ 2
```

9.10.1.281 __PIC__

```
#define __PIC__ 2
```

9.10.1.282 __pie__

```
#define __pie__ 2
```

9.10.1.283 __PIE__

```
#define __PIE__ 2
```

9.10.1.284 __PRAGMA_REDEFINE_EXTNAME

```
#define __PRAGMA_REDEFINE_EXTNAME 1
```

9.10.1.285 __PTRDIFF_MAX__

```
#define __PTRDIFF_MAX__ 0x7fffffffffffffffL
```

9.10.1.286 __PTRDIFF_TYPE__

```
#define __PTRDIFF_TYPE__ long int
```

9.10.1.287 __PTRDIFF_WIDTH__

```
#define __PTRDIFF_WIDTH__ 64
```

9.10.1.288 __REGISTER_PREFIX__

```
#define __REGISTER_PREFIX__
```

9.10.1.289 __SCHAR_MAX__

```
#define __SCHAR_MAX__ 0x7f
```

9.10.1.290 __SCHAR_WIDTH__

```
#define __SCHAR_WIDTH__ 8
```

9.10.1.291 __SEG_FS

```
#define __SEG_FS 1
```

9.10.1.292 __SEG_GS

```
#define __SEG_GS 1
```

9.10.1.293 __SHRT_MAX__

```
#define __SHRT_MAX__ 0x7fff
```

9.10.1.294 __SHRT_WIDTH__

```
#define __SHRT_WIDTH__ 16
```

9.10.1.295 __SIG_ATOMIC_MAX__

```
#define __SIG_ATOMIC_MAX__ 0x7fffffff
```

9.10.1.296 __SIG_ATOMIC_MIN__

```
#define __SIG_ATOMIC_MIN__ (-__SIG_ATOMIC_MAX__ - 1)
```

9.10.1.297 __SIG_ATOMIC_TYPE__

```
#define __SIG_ATOMIC_TYPE__ int
```

9.10.1.298 __SIG_ATOMIC_WIDTH__

```
#define __SIG_ATOMIC_WIDTH__ 32
```

9.10.1.299 __SIZE_MAX__

```
#define __SIZE_MAX__ 0xffffffffffffffffUL
```

9.10.1.300 __SIZE_TYPE__

```
#define __SIZE_TYPE__ long unsigned int
```

9.10.1.301 __SIZE_WIDTH__

```
#define __SIZE_WIDTH__ 64
```

9.10.1.302 __SIZEOF_DOUBLE__

```
#define __SIZEOF_DOUBLE__ 8
```

9.10.1.303 __SIZEOF_FLOAT128__

```
#define __SIZEOF_FLOAT128__ 16
```

9.10.1.304 __SIZEOF_FLOAT80__

```
#define __SIZEOF_FLOAT80__ 16
```

9.10.1.305 __SIZEOF_FLOAT__

```
#define __SIZEOF_FLOAT__ 4
```

9.10.1.306 __SIZEOF_INT128__

```
#define __SIZEOF_INT128__ 16
```

9.10.1.307 __SIZEOF_INT__

```
#define __SIZEOF_INT__ 4
```

9.10.1.308 __SIZEOF_LONG__

```
#define __SIZEOF_LONG__ 8
```

9.10.1.309 __SIZEOF_LONG_DOUBLE__

```
#define __SIZEOF_LONG_DOUBLE__ 16
```

9.10.1.310 __SIZEOF_LONG_LONG__

```
#define __SIZEOF_LONG_LONG__ 8
```

9.10.1.311 __SIZEOF_POINTER__

```
#define __SIZEOF_POINTER__ 8
```

9.10.1.312 __SIZEOF_PTRDIFF_T__

```
#define __SIZEOF_PTRDIFF_T__ 8
```

9.10.1.313 __SIZEOF_SHORT__

```
#define __SIZEOF_SHORT__ 2
```

9.10.1.314 __SIZEOF_SIZE_T__

```
#define __SIZEOF_SIZE_T__ 8
```

9.10.1.315 __SIZEOF_WCHAR_T__

```
#define __SIZEOF_WCHAR_T__ 4
```

9.10.1.316 __SIZEOF_WINT_T__

```
#define __SIZEOF_WINT_T__ 4
```

9.10.1.317 __SSE2__

```
#define __SSE2__ 1
```

9.10.1.318 __SSE2_MATH__

```
#define __SSE2_MATH__ 1
```

9.10.1.319 __SSE__

```
#define __SSE__ 1
```

9.10.1.320 __SSE_MATH__

```
#define __SSE_MATH__ 1
```

9.10.1.321 __SSP_STRONG__

```
#define __SSP_STRONG__ 3
```

9.10.1.322 __STDC__

```
#define __STDC__ 1
```

9.10.1.323 __STDC_HOSTED__

```
#define __STDC_HOSTED__ 1
```

9.10.1.324 __STDC_IEC_559__

```
#define __STDC_IEC_559__ 1
```

9.10.1.325 __STDC_IEC_559_COMPLEX__

```
#define __STDC_IEC_559_COMPLEX__ 1
```

9.10.1.326 __STDC_ISO_10646__

```
#define __STDC_ISO_10646__ 201706L
```

9.10.1.327 __STDC_NO_THREADS__

```
#define __STDC_NO_THREADS__ 1
```

9.10.1.328 __STDC_UTF_16__

```
#define __STDC_UTF_16__ 1
```

9.10.1.329 __STDC_UTF_32__

```
#define __STDC_UTF_32__ 1
```

9.10.1.330 __UINT16_C

```
#define __UINT16_C(
              c ) c
```

9.10.1.331 __UINT16_MAX__

```
#define __UINT16_MAX__ 0xffff
```

9.10.1.332 __UINT16_TYPE__

```
#define __UINT16_TYPE__ short unsigned int
```

9.10.1.333 __UINT32_C

```
#define __UINT32_C(
              c ) c ## U
```

9.10.1.334 __UINT32_MAX__

```
#define __UINT32_MAX__ 0xffffffffU
```

9.10.1.335 __UINT32_TYPE__

```
#define __UINT32_TYPE__ unsigned int
```

9.10.1.336 __UINT64_C

```
#define __UINT64_C(
              c ) c ## UL
```

9.10.1.337 __UINT64_MAX__

```
#define __UINT64_MAX__ 0xffffffffffffffffUL
```

9.10.1.338 __UINT64_TYPE__

```
#define __UINT64_TYPE__ long unsigned int
```

9.10.1.339 __UINT8_C

```
#define __UINT8_C(
              c ) c
```

9.10.1.340 __UINT8_MAX__

```
#define __UINT8_MAX__ 0xff
```

9.10.1.341 __UINT8_TYPE__

```
#define __UINT8_TYPE__ unsigned char
```

9.10.1.342 __UINT_FAST16_MAX__

```
#define __UINT_FAST16_MAX__ 0xffffffffffffffffUL
```

9.10.1.343 __UINT_FAST16_TYPE__

```
#define __UINT_FAST16_TYPE__ long unsigned int
```

9.10.1.344 __UINT_FAST32_MAX__

```
#define __UINT_FAST32_MAX__ 0xffffffffffffffffUL
```

9.10.1.345 __UINT_FAST32_TYPE__

```
#define __UINT_FAST32_TYPE__ long unsigned int
```

9.10.1.346 __UINT_FAST64_MAX__

```
#define __UINT_FAST64_MAX__ 0xffffffffffffffffUL
```

9.10.1.347 __UINT_FAST64_TYPE__

```
#define __UINT_FAST64_TYPE__ long unsigned int
```

9.10.1.348 __UINT_FAST8_MAX__

```
#define __UINT_FAST8_MAX__ 0xff
```

9.10.1.349 __UINT_FAST8_TYPE__

```
#define __UINT_FAST8_TYPE__ unsigned char
```

9.10.1.350 __UINT_LEAST16_MAX__

```
#define __UINT_LEAST16_MAX__ 0xffff
```

9.10.1.351 __UINT_LEAST16_TYPE__

```
#define __UINT_LEAST16_TYPE__ short unsigned int
```

9.10.1.352 __UINT_LEAST32_MAX__

```
#define __UINT_LEAST32_MAX__ 0xffffffffU
```

9.10.1.353 __UINT_LEAST32_TYPE__

```
#define __UINT_LEAST32_TYPE__ unsigned int
```

9.10.1.354 __UINT_LEAST64_MAX__

```
#define __UINT_LEAST64_MAX__ 0xffffffffffffffffUL
```

9.10.1.355 __UINT_LEAST64_TYPE__

```
#define __UINT_LEAST64_TYPE__ long unsigned int
```

9.10.1.356 __UINT_LEAST8_MAX__

```
#define __UINT_LEAST8_MAX__ 0xff
```

9.10.1.357 __UINT_LEAST8_TYPE__

```
#define __UINT_LEAST8_TYPE__ unsigned char
```

9.10.1.358 __UINTMAX_C

```
#define __UINTMAX_C(
              c ) c ## UL
```

9.10.1.359 __UINTMAX_MAX__

```
#define __UINTMAX_MAX__ 0xffffffffffffffffUL
```

9.10.1.360 __UINTMAX_TYPE__

```
#define __UINTMAX_TYPE__ long unsigned int
```

9.10.1.361 __UINTPTR_MAX__

```
#define __UINTPTR_MAX__ 0xffffffffffffffffUL
```

9.10.1.362 __UINTPTR_TYPE__

```
#define __UINTPTR_TYPE__ long unsigned int
```

9.10.1.363 __unix

```
#define __unix 1
```

9.10.1.364 __unix__

```
#define __unix__ 1
```

9.10.1.365 __USER_LABEL_PREFIX__

```
#define __USER_LABEL_PREFIX__
```

9.10.1.366 __VERSION__

```
#define __VERSION__ "7.3.0"
```

9.10.1.367 __WCHAR_MAX__

```
#define __WCHAR_MAX__ 0x7fffffff
```

9.10.1.368 __WCHAR_MIN__

```
#define __WCHAR_MIN__ (-__WCHAR_MAX__ - 1)
```

9.10.1.369 __WCHAR_TYPE__

```
#define __WCHAR_TYPE__ int
```

9.10.1.370 __WCHAR_WIDTH__

```
#define __WCHAR_WIDTH__ 32
```

9.10.1.371 __WINT_MAX__

```
#define __WINT_MAX__ 0xffffffffU
```

9.10.1.372 __WINT_MIN__

```
#define __WINT_MIN__ 0U
```

9.10.1.373 __WINT_TYPE__

```
#define __WINT_TYPE__ unsigned int
```

9.10.1.374 __WINT_WIDTH__

```
#define __WINT_WIDTH__ 32
```

9.10.1.375 __x86_64

```
#define __x86_64 1
```

9.10.1.376 __x86_64__

```
#define __x86_64__ 1
```

9.10.1.377 _FORTIFY_SOURCE

```
#define _FORTIFY_SOURCE 2
```

9.10.1.378 _GNU_SOURCE

```
#define _GNU_SOURCE 1
```

9.10.1.379 _LP64

```
#define _LP64 1
```

9.10.1.380 _STDC_PREDEF_H

```
#define _STDC_PREDEF_H 1
```

9.10.1.381 linux

```
#define linux 1
```

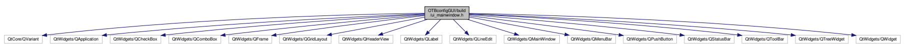
9.10.1.382 unix

```
#define unix 1
```

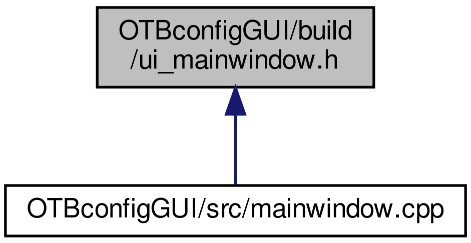
9.11 OTBconfigGUI/build/ui_mainwindow.h File Reference

```
#include <QtCore/QVariant>
#include <QtWidgets/QApplication>
#include <QtWidgets/QCheckBox>
#include <QtWidgets/QComboBox>
#include <QtWidgets/QFrame>
#include <QtWidgets/QGridLayout>
#include <QtWidgets/QHeaderView>
#include <QtWidgets/QLabel>
#include <QtWidgets/QLineEdit>
#include <QtWidgets/QMainWindow>
#include <QtWidgets/QMenuBar>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QStatusBar>
#include <QtWidgets/QToolBar>
#include <QtWidgets/QTreeWidget>
#include <QtWidgets/QWidget>
```

Include dependency graph for ui_mainwindow.h:



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



Classes

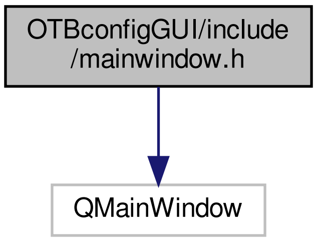
- class Ui_MainWindow
- class Ui::MainWindow

Namespaces

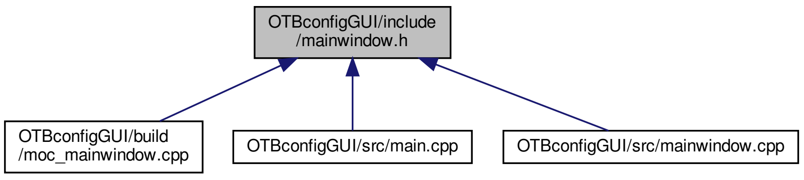
- Ui

9.12 OTBconfigGUI/include/mainwindow.h File Reference

`#include <QMainWindow>`
Include dependency graph for mainwindow.h:



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



Classes

- class MainWindow

Namespaces

- Ui

Macros

- #define ACQ_SETT 0b10000000
- #define DECIM 0b01000000
- #define REC_ON 0b00100000
- #define FSAMP1 0b00010000
- #define FSAMP0 0b00001000
- #define NCH1 0b00000100
- #define NCH0 0b00000010
- #define ACQ_ON 0b00000001
- #define ACQ_OFF 0b00000000
- #define CRC_CODE 0b10001100
- #define CONFIG_SIZE 40

9.12.1 Macro Definition Documentation

9.12.1.1 ACQ_OFF

```
#define ACQ_OFF 0b00000000
```

9.12.1.2 ACQ_ON

```
#define ACQ_ON 0b00000001
```

9.12.1.3 ACQ_SETT

```
#define ACQ_SETT 0b10000000
```

9.12.1.4 CONFIG_SIZE

```
#define CONFIG_SIZE 40
```

9.12.1.5 CRC_CODE

```
#define CRC_CODE 0b10001100
```

9.12.1.6 DECIM

```
#define DECIM 0b01000000
```

9.12.1.7 FSAMP0

```
#define FSAMP0 0b00001000
```

9.12.1.8 FSAMP1

```
#define FSAMP1 0b00010000
```

9.12.1.9 NCH0

```
#define NCH0 0b00000010
```

9.12.1.10 NCH1

```
#define NCH1 0b00000100
```

9.12.1.11 REC_ON

```
#define REC_ON 0b00100000
```

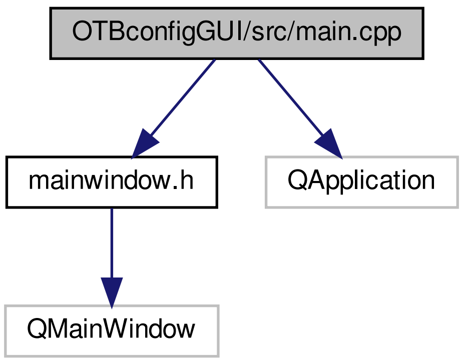
9.13 OTBconfigGUI/README.md File Reference

9.14 README.md File Reference

9.15 OTBconfigGUI/src/main.cpp File Reference

```
#include "mainwindow.h"
#include <QApplication>
```

Include dependency graph for main.cpp:



Functions

- int main (int argc, char ∗argv[ ])

9.15.1 Function Documentation

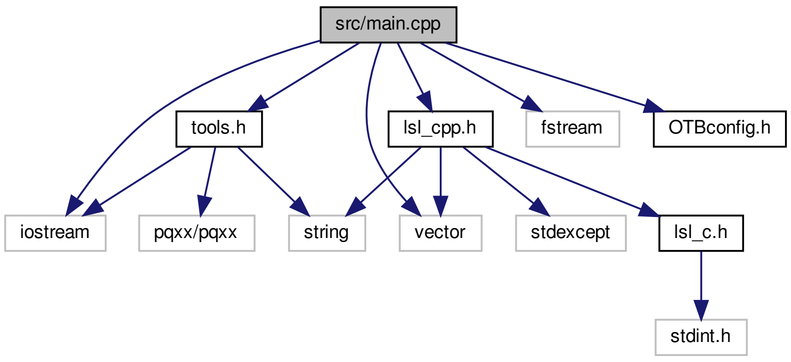
9.15.1.1 main()

```
int main (
            int argc,
            char * argv[ ] )
```

9.16 src/main.cpp File Reference

```
#include <vector>
#include <fstream>
#include <iostream>
#include <lsl_cpp.h>
#include "OTBconfig.h"
#include "tools.h"
```

Include dependency graph for main.cpp:



Macros

- #define SAMPLING_FREQUENCY 2048
- #define CHUNK_SIZE 1

Functions

- template<class T >
  void fill_chunk (unsigned char ∗from, std::vector< std::vector< T >> &to, int nb_ch, int n=CHUNK_SIZE)

  fill_chunk transform a unsigned char array into a typed vector of vector.
- void getConf (std::string path, unsigned char ∗config)

  fill_chunk transform a unsigned char array into a typed vector of vector.
- int get_sampling_rate (unsigned char ∗config)

  get_sampling_rate Get and interpret the sampling frequency bites of the config array and return the coresponding rate.
- int get_nbChannels (unsigned char ∗config)

  get_sampling_rate Get and interpret the number of channels bites of the config array and return the coresponding rate
- int main (int argc, char ∗∗argv)

9.16.1 Macro Definition Documentation

9.16.1.1 CHUNK_SIZE

```
#define CHUNK_SIZE 1
```

9.16.1.2 SAMPLING_FREQUENCY

```
#define SAMPLING_FREQUENCY 2048
```

9.16.2 Function Documentation

9.16.2.1 fill_chunk()

```
template<class T >
void fill_chunk (
             unsigned char ∗ from,
             std::vector< std::vector< T >> & to,
             int nb_ch,
             int n = CHUNK_SIZE )
```

fill_chunk transform a unsigned char array into a typed vector of vector.

Template Parameters

T	Type of the vector.

Parameters

from	Unsigned char array to transform.
to	Resulting vector of vector of type T.
nb_ch	Number of channel of the stream.
n	Number of sample in the array.

9.16.2.2 get_nbChannels()

```
int get_nbChannels (
            unsigned char * config )
```

get_sampling_rate Get and interpret the number of channels bites of the config array and return the coresponding rate

Parameters

config	The configuration array used in the program.

9.16.2.3 get_sampling_rate()

```
int get_sampling_rate (
            unsigned char * config )
```

get_sampling_rate Get and interpret the sampling frequency bites of the config array and return the coresponding rate.

Parameters

config	The configuration array used in the program.

9.16.2.4 getConf()

```
void getConf (
            std::string path,
            unsigned char * config )
```

fill_chunk transform a unsigned char array into a typed vector of vector.

Parameters

path	Path of the configuration file..
config	The configuration array used in the program.

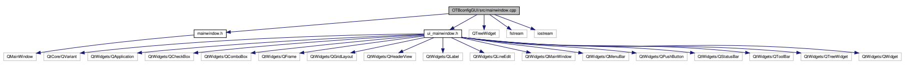
9.16.2.5 main()

```
int main (
            int argc,
            char ** argv )
```

9.17 OTBconfigGUI/src/mainwindow.cpp File Reference

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QTreeWidget>
#include <fstream>
#include <iostream>
```

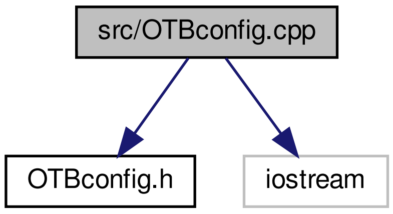
Include dependency graph for mainwindow.cpp:



9.18 src/OTBconfig.cpp File Reference

```
#include "OTBconfig.h"
#include <iostream>
```

Include dependency graph for OTBconfig.cpp:



Functions

- unsigned char crc (unsigned char config[ ])
- void printBIN (char n)

9.18.1 Function Documentation

9.18.1.1 crc()

```
unsigned char crc (
            unsigned char config[ ] )
```

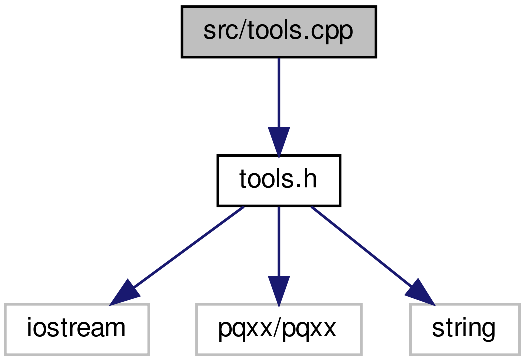
9.18.1.2 printBIN()

```
void printBIN (
            char n )
```

9.19 src/tools.cpp File Reference

TODO.

`#include "tools.h"`
Include dependency graph for tools.cpp:



Functions

- void error (std::string str)

  error Display the passed string thne exit the program.
- void usage (std::vector< std::string > &optf, std::vector< std::string > &optl, std::vector< std::string > &optv)

  usage Display the usage, then exit the program.
- void get_arg (int argc, char ∗∗argv, std::vector< std::string > &optf, std::vector< std::string > &optl, std↩ ::vector< std::string > &optv)

  get_arg Search for the potential argument in the argument passed to the program.

9.19.1 Detailed Description

TODO.

Author

Alexis Devillard

Version

1.0

Date

08 may 2019

9.19.2 Function Documentation

9.19.2.1 error()

```
void error (
             std::string str )
```

error Display the passed string thne exit the program.

Parameters

str	String to display.

9.19.2.2 get_arg()

```
void get_arg (
             int argc,
             char ** argv,
             std::vector< std::string > & optf,
             std::vector< std::string > & optl,
             std::vector< std::string > & optv )
```

get_arg Search for the potential argument in the argument passed to the program.

Parameters

argc	Argument counter
argv	Argument array
optf	List of option flags
optf	List of option labels
optf	List of option values

9.19.2.3 usage()

```
void usage (
            std::vector< std::string > & optf,
            std::vector< std::string > & optl,
            std::vector< std::string > & optv )
```

usage Display the usage, then exit the program.

Parameters

optf	List of option flags
optf	List of option labels
optf	List of option values

Index