# LibMuse Windows 8.0.0

Generated by Doxygen 1.13.2

# Chapter 1

# Muse SDK Documentation

This library enables communication with and control of Muse headbands from client software.

## 1.0.1 Quick start

### 1.0.1.1 Windows

Start with interaxon::bridge::MuseManager and explore from there, or use the Classes tab

## 1.0.2 Threading

Most LibMuse methods are thread safe, except execute() and run_asynchronously() . Each method describes its threading concerns if there are any.

## 1.0.3 Exceptions

The LibMuse native library catches and handles every exception that it can reasonably be expected to do so. There are, however, some situations in which native code can still throw an exception. Any API method that can throw an exception documents this behaviour.

# Chapter 2

# Deprecated List

**Member interaxon::bridge::DROPPED_ACCELEROMETER**

This is never emitted in an actual Muse session; instead, NaN-filled packets of the basic type (EEG or ACCELEROMETER) are emitted to stand in for dropped packets. This can only appear when reading Muse files written with older versions of the library.

**Member interaxon::bridge::DROPPED_EEG**

This is never emitted in an actual Muse session; instead, NaN-filled packets of the basic type (EEG or ACCELEROMETER) are emitted to stand in for dropped packets. This can only appear when reading Muse files written with older versions of the library.

**Member interaxon::bridge::MuseDataPacket::values ()=0**

Use MuseDataPacket::get_accelerometer_value() , MuseDataPacket::get_battery_value() , MuseDataPacket::get_drl_ref_value() , MuseDataPacket::get_eeg_channel_value() , MuseDataPacket::get_ppg_channel_value() , MuseDataPacket::get_gyro_value() instead.

**Member interaxon::bridge::ReaderMuse::run ()=0**

Use playback() with the playback setting ReaderMusePlaybackSettings::AS_FAST_AS_POSSIBLE_WITH_SAVED_TIMESTAMP to replicate this behaviour.

**Member interaxon::bridge::ReaderMuse::run_in_real_timespan ()=0**

Use playback() with the playback setting ReaderMusePlaybackSettings::SIMULATED_WITH_SAVED_TIMESTAMP to replicate this behaviour.

**Member interaxon::bridge::ReaderMuseBuilder::build_with_async (const std::shared_ptr< MuseFile↩ Reader > &reader, const std::shared_ptr< EventLoop > &async_loop)=0**

Set the EventLoop with ReaderMuseBuilder::with_event_loop() and then call ReaderMuseBuilder::build() instead.

# 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 interaxon Namespace Reference

### Namespaces

- namespace bridge

## 7.2 interaxon::bridge Namespace Reference

### Classes

- class Action
- struct AdvertisingStats
- struct AnnotationData
- class ApiVersion
- struct ComputingDeviceConfiguration
- class ComputingDeviceConfigurationFactory
- class Convert
- struct DspData
- struct Error
- class EventLoop
- class EventLoopFactory
- class LibmuseVersion
- class LogListener
- class LogManager
- struct LogPacket
- class Muse
- struct MuseArtifactPacket
- class MuseConfiguration
- class MuseConnectionListener
- struct MuseConnectionPacket
- class MuseDataListener
- class MuseDataPacket
- class MuseErrorListener
- class MuseFile
- class MuseFileFactory

- class MuseFileReader
- class MuseFileWriter
- class MuseListener
- class MuseManager
- class MuseManagerWindows
- class MuseVersion
- class ReaderListener
- class ReaderMuse
- class ReaderMuseBuilder
- class ReaderPlaybackListener
- struct Result
- class Stringify

## Enumerations

- enum class Accelerometer : int {
  X ,
  Y ,
  Z }
- enum class AnnotationFormat : int {
  PLAIN_STRING ,
  JSON ,
  OSC }
- enum class Battery : int {
  CHARGE_PERCENTAGE_REMAINING ,
  MILLIVOLTS ,
  TEMPERATURE_CELSIUS ,
  AVERAGE_CURRENT ,
  TIME_TO_EMPTY ,
  TIME_TO_FULL ,
  BATTERY_CAPACITY ,
  REMAINING_CAPACITY ,
  BATTERY_AGE ,
  TOTAL_CYCLES }
- enum class ConnectionState : int {
  UNKNOWN ,
  CONNECTED ,
  CONNECTING ,
  DISCONNECTED ,
  NEEDS_UPDATE ,
  NEEDS_LICENSE }
- enum class DrlRef : int {
  DRL ,
  REF }
- enum class Eeg : int {
  EEG1 ,
  EEG2 ,
  EEG3 ,
  EEG4 ,
  AUX_LEFT ,
  AUX_RIGHT ,
  AUX1 ,
  AUX2 ,
  AUX3 ,
  AUX4 }

- enum class ErrorType : int {
  FAILURE ,
  TIMEOUT ,
  OVERLOADED ,
  UNIMPLEMENTED }
- enum class Gyro : int {
  X ,
  Y ,
  Z }
- enum class Magnetometer : int {
  X ,
  Y ,
  Z }
- enum class MessageType : int {
  EEG ,
  QUANTIZATION ,
  ACCELEROMETER ,
  BATTERY ,
  VERSION ,
  CONFIGURATION ,
  ANNOTATION ,
  HISTOGRAM ,
  ALG_VALUE ,
  DSP ,
  COMPUTING_DEVICE ,
  EEG_DROPPED ,
  ACC_DROPPED ,
  CALM_APP ,
  CALM_ALG ,
  MUSE_ELEMENTS ,
  GYRO ,
  ARTIFACT ,
  PRESSURE ,
  TEMPERATURE ,
  ULTRA_VIOLET ,
  MAGNETOMETER ,
  PPG ,
  THERMISTOR ,
  OPTICS ,
  ALGORITHM }
- enum class MuseDataPacketType : int {
  ACCELEROMETER ,
  GYRO ,
  EEG ,
  DROPPED_ACCELEROMETER ,
  DROPPED_EEG ,
  QUANTIZATION ,
  BATTERY ,
  DRL_REF ,
  ALPHA_ABSOLUTE ,
  BETA_ABSOLUTE ,
  DELTA_ABSOLUTE ,
  THETA_ABSOLUTE ,
  GAMMA_ABSOLUTE ,
  ALPHA_RELATIVE ,
  BETA_RELATIVE ,
  DELTA_RELATIVE ,
  THETA_RELATIVE ,

GAMMA_RELATIVE ,
ALPHA_SCORE ,
BETA_SCORE ,
DELTA_SCORE ,
THETA_SCORE ,
GAMMA_SCORE ,
IS_GOOD ,
HSI ,
HSI_PRECISION ,
ARTIFACTS ,
MAGNETOMETER ,
PRESSURE ,
TEMPERATURE ,
ULTRA_VIOLET ,
NOTCH_FILTERED_EEG ,
VARIANCE_EEG ,
VARIANCE_NOTCH_FILTERED_EEG ,
PPG ,
IS_PPG_GOOD ,
IS_HEART_GOOD ,
THERMISTOR ,
IS_THERMISTOR_GOOD ,
AVG_BODY_TEMPERATURE ,
CLOUD_COMPUTED ,
OPTICS ,
TOTAL }

- enum class MuseModel : int {
MU_01 ,
MU_02 ,
MU_03 ,
MU_04 ,
MU_05 ,
MU_06 ,
MS_03 }
- enum class MusePreset : int {
PRESET_10 ,
PRESET_12 ,
PRESET_14 ,
PRESET_20 ,
PRESET_21 ,
PRESET_22 ,
PRESET_23 ,
PRESET_AB ,
PRESET_AD ,
PRESET_31 ,
PRESET_32 ,
PRESET_50 ,
PRESET_51 ,
PRESET_52 ,
PRESET_53 ,
PRESET_54 ,
PRESET_55 ,
PRESET_60 ,
PRESET_61 ,
PRESET_63 ,
PRESET_1021 ,
PRESET_1022 ,
PRESET_1023 ,

PRESET_1024 ,
PRESET_1025 ,
PRESET_1026 ,
PRESET_1027 ,
PRESET_1028 ,
PRESET_1029 ,
PRESET_102A ,
PRESET_1031 ,
PRESET_1032 ,
PRESET_1033 ,
PRESET_1034 ,
PRESET_1035 ,
PRESET_1036 ,
PRESET_1041 ,
PRESET_1042 ,
PRESET_1043 ,
PRESET_1044 ,
PRESET_1045 ,
PRESET_1046 }

- enum class NotchFrequency : int {
  NOTCH_NONE ,
  NOTCH_50HZ ,
  NOTCH_60HZ }
- enum class Optics : int {
  OPTICS1 ,
  OPTICS2 ,
  OPTICS3 ,
  OPTICS4 ,
  OPTICS5 ,
  OPTICS6 ,
  OPTICS7 ,
  OPTICS8 ,
  OPTICS9 ,
  OPTICS10 ,
  OPTICS11 ,
  OPTICS12 ,
  OPTICS13 ,
  OPTICS14 ,
  OPTICS15 ,
  OPTICS16 }
- enum class Ppg : int {
  AMBIENT ,
  IR ,
  RED }
- enum class Pressure : int {
  RAW ,
  AVERAGED }
- enum class ReaderMusePlaybackSettings : int {
  AS_FAST_AS_POSSIBLE_WITH_SAVED_TIMESTAMP ,
  SIMULATED_WITH_SAVED_TIMESTAMP ,
  SIMULATED_WITH_SYSTEM_CLOCK_TIMESTAMP }
- enum class ResultLevel : int {
  R_NONE ,
  R_SUCCESS ,
  R_INFO ,
  R_WARN ,
  R_ERROR ,
  R_DEBUG }

- enum class Severity : int {
  SEV_VERBOSE ,
  SEV_INFO ,
  SEV_WARN ,
  SEV_ERROR ,
  SEV_FATAL ,
  SEV_DEBUG ,
  TOTAL }
- enum class TimestampMode : int {
  LEGACY ,
  CURRENT ,
  EXPLICIT }
- enum class UltraViolet : int {
  UV_INDEX ,
  UV_A ,
  UV_B }

## 7.2.1 Enumeration Type Documentation

### 7.2.1.1 Accelerometer

```
enum class interaxon::bridge::Accelerometer :  int  [strong]
```

Represents the data mapping in an Accelerometer data packet.

The accelerometer data is measured on 3 axes as shown in the pictures below:

The axes are oriented to a Right Hand Coordinate System along the headband axes. Values are given in g (9.81 m/s^2) and are negated to align with the orientation of the headband in Earth's gravitational field. This convention is described in more detail in the following application note.
 `https://www.nxp.com/docs/en/application-note/AN3461.pdf`
Values along the X axis increase as the head tilts down aligning the X axis with the downward force of gravity. Negative values indicate a tilt upwards.
Values along the Y axis increase as the head tilts to the right. Negative values indicate a tilt to the left.
When worn on a level head, or held in the level position shown in the figure above, the net acceleration of the device will only be caused from gravity. It will be in the direction of the ground aligned with the Z axis. This will give a_x =0, a_y = 0 and a_z = 1 g. As the headband tilts out of this orientation, the value of Z will decrease. -1 in Z represents a headband that is upside down.

**See also**

MuseDataPacketType::ACCELEROMETER

MuseDataPacket::get_accelerometer_value()

**Enumerator**

| | |
|---|---|
| X | Orientation of the X axis relative to gravity in g. Values along the X axis increase as the head tilts down. Negative values indicate a tilt up. |
| Y | Orientation of the Y axis relative to gravity in g. Values along the Y axis increase as the head tilts to the right. Negative values indicate a tilt to the left. |
| Z | Orientation of the Z axis relative to gravity in g. |

#### 7.2.1.2 AnnotationFormat

```
enum class interaxon::bridge::AnnotationFormat : int [strong]
```

Represents all possible data types in the annotation data. These fields correspond to our Muse protobuf schema.

**Enumerator**

| | |
|---|---|
| PLAIN_STRING | The data is a string with no inherrent format. |
| JSON | The data is a string of JSON name, value pairs. |
| OSC | The data is formatted as OSC (open sound control) data. |

#### 7.2.1.3 Battery

```
enum class interaxon::bridge::Battery : int [strong]
```

Represents the data mapping in a Battery packet.

**See also**

MuseDataPacketType::BATTERY
MuseDataPacket::get_battery_value()

**Enumerator**

| | |
|---|---|
| CHARGE_PERCENTAGE_REMAINING | Charge percentage remaining of battery. |
| MILLIVOLTS | Millivolts of battery from the view of the fuel gauge. |
| TEMPERATURE_CELSIUS | Temperature in degrees Celsius. |
| AVERAGE_CURRENT | Average current in microamps. |
| TIME_TO_EMPTY | Remaining seconds until charge depleted (while discharging). |
| TIME_TO_FULL | Remaining seconds until fully charged (while charging). |
| BATTERY_CAPACITY | Full capacity of battery in mAh. |
| REMAINING_CAPACITY | Remaining capacity of battery in mAh. |
| BATTERY_AGE | Percentage battery age calculated from capacity. |
| TOTAL_CYCLES | Total number of charge/discharge cycles. |

#### 7.2.1.4 ConnectionState

```
enum class interaxon::bridge::ConnectionState : int [strong]
```

Lists all possible connection states

**Enumerator**

| | |
|---|---|
| UNKNOWN | Initial state |
| CONNECTED | This state is set if the connection was correctly established. |
| CONNECTING | This state is set while trying to establish a connection. |
| DISCONNECTED | This state is set in case of an unsuccessful connection operation or after execution of disconnect method. |
| NEEDS_UPDATE | This state is set when the connection succeeded but the headband's firmware is out of date – if this occurs, you should instruct your users to use the Muse app to upgrade their firmware. |
| NEEDS_LICENSE | This state is set when the connection succeeded but the headband's license is invalid. |

#### 7.2.1.5 DrlRef

```
enum class interaxon::bridge::DrlRef : int [strong]
```

Represents the data mapping in a DRL_REF packet.

**See also**

MuseDataPacketType::DRL_REF

MuseDataPacket::get_drl_ref_value()

**Enumerator**

| | |
|---|---|
| DRL | DRL sensor |
| REF | REF sensor |

#### 7.2.1.6 Eeg

```
enum class interaxon::bridge::Eeg : int [strong]
```

Represents the data mapping in an EEG packet. Enum values correspond to head locations.

Raw EEG values are given in microvolts. EEG derived values may have different units as defined in MuseDataPacketType

**See also**

MuseDataPacketType::EEG

MuseDataPacket::get_eeg_channel_value()

**Enumerator**

| | |
|---|---|
| EEG1 | Left ear |
| EEG2 | Left forehead |
| EEG3 | Right forehead |
| EEG4 | Right ear |
| AUX_LEFT | Left auxiliary (maps to AUX1 on MuseS 2025 ( MS_03 )). |
| AUX_RIGHT | Right auxiliary (maps to AUX2 on MuseS 2025 ( MS_03 )). |
| AUX1 | Auxiliary input 1 on MuseS 2025 ( MS_03 ). |
| AUX2 | Auxiliary input 2 on MuseS 2025 ( MS_03 ). |
| AUX3 | Auxiliary input 3 on MuseS 2025 ( MS_03 ) |
| AUX4 | Auxiliary input 4 on MuseS 2025 ( MS_03 ) |

#### 7.2.1.7 ErrorType

```
enum class interaxon::bridge::ErrorType : int [strong]
```

Error classifications.

These are designed to provide users with courses of action based on the code that's received: e.g. retry on timeout, try a different way of doing the same thing on unimplemented, show the user an error message on failure.

**Enumerator**

| | |
|---|---|
| FAILURE | A generic failure occurred without any further information. Retrying the operation is unlikely to result in success. |
| TIMEOUT | Some timeout was exceeded; the operation might succeed if retried. |
| OVERLOADED | Some resource (queue space, memory, bandwidth) was exhausted. Retry with backoff. |
| UNIMPLEMENTED | Something was tried that isn't implemented. |

#### 7.2.1.8 Gyro

```
enum class interaxon::bridge::Gyro :  int  [strong]
```

Represents the data mapping in a Gyro data packet

The gyro data is measured as the rotation about 3 axes which is shown in the pictures below:

The axes are oriented to a Right Hand Coordinate System.
Rotation about the X axis corresponds to tilting the head side to side. Positive values increase when tilting to the right. This is also known as roll.
Rotation about the Y axis corresponds to tilting the head up and down. Positive values increase when looking up. This is also known as pitch.
Rotation about the Z axis corresponds to looking left and right. Positive values increase when looking to the right. This is also known as yaw.

**See also**

MuseDataPacketType::GYRO
MuseDataPacket::get_gyro_value()

**Enumerator**

| | |
|---|---|
| X | Rotation about the X axis in degrees per second. Rotation about the X axis corresponds to tilting the head side to side. Positive values increase when tilting to the right. This is also known as roll. |
| Y | Rotation about the Y axis in degrees per second. Rotation about the Y axis corresponds to tilting the head up and down. Positive values increase when looking up. This is also known as pitch. |
| Z | Rotation about the Z axis in degrees per second. Rotation about the Z axis corresponds to looking left and right. Positive values increase when looking to the right. This is also known as yaw. |

#### 7.2.1.9 Magnetometer

```
enum class interaxon::bridge::Magnetometer :  int  [strong]
```

Represents the data mapping in a Magnetometer data packet

The magnetometer data is measured on 3 axes as shown in the picture below:

The axes are oriented to a Right Hand Coordinate System.

**See also**

MuseDataPacketType::MAGNETOMETER
MuseDataPacket::get_magnetometer_value()

**Enumerator**

| | |
|---|---|
| X | |
| Y | |
| Z | |

#### 7.2.1.10 MessageType

```
enum class interaxon::bridge::MessageType :  int  [strong]
```

Represents all possible MuseData message data types in a .muse file. This enum corresponds to the DataType enum in the protobuf schema.

**Enumerator**

| | |
|---|---|
| EEG | A message containing eeg data. |
| QUANTIZATION | A message containing quantization data. |
| ACCELEROMETER | A message containing accelerometer data. |
| BATTERY | A message containing battery data. |
| VERSION | A message containing version data. |
| CONFIGURATION | A message containing configuration data. |
| ANNOTATION | A message containing annotation data. |
| HISTOGRAM | A message containing histogram data. |
| ALG_VALUE | A message containing algorithm data. |
| DSP | A message containing dsp data. |
| COMPUTING_DEVICE | A message containing device data. |
| EEG_DROPPED | A message containing dropped eeg data. |
| ACC_DROPPED | A message containing dropped acc data. |
| CALM_APP | A message containing data from the calm application. |
| CALM_ALG | A message containing data from the calm algorithm. |
| MUSE_ELEMENTS | A message containing muse element data. |
| GYRO | A message containing gyro data. |
| ARTIFACT | A message containing artifact packet. |
| PRESSURE | A message containing pressure data. |
| TEMPERATURE | A message containing temperature data. |
| ULTRA_VIOLET | A message containing ultra violet data. |
| MAGNETOMETER | A message containing magnetometer data. |
| PPG | A message containing ppg data. |
| THERMISTOR | A message containing thermistor data. |
| OPTICS | A message containing optics data. |
| ALGORITHM | A message containing algorithm data. |

#### 7.2.1.11 MuseDataPacketType

```
enum class interaxon::bridge::MuseDataPacketType :  int  [strong]
```

This Enum represents all possible packet types. The type of the packet tells you information about what data it contains. When you know the packet type, look at the corresponding enum for information about data mapping (e.g.: Accelerometer enum, EEG enum, etc).

### EEG derived signals

#### Absolute band powers

The absolute band power for a given frequency range (for instance, alpha, i.e. 7.5-13Hz) is the logarithm of the sum of the Power Spectral Density of the EEG data over that frequency range. They are provided for each of the channels/electrode sites on Muse. Since it is a logarithm, some of the values will be negative (i.e. when the absolute power is less than 1). They are given on a log scale, units are Bels.

#### Relative band powers

The relative band powers are calculated by dividing the absolute linear-scale power in one band over the sum of the absolute linear-scale powers in all bands. The linear-scale band power can be calculated from the log-scale band power.

linear-scale band power = 10^ (log-scale band power)

Therefore, the relative band powers can be calculated as percentages of linear-scale band powers in each band. For example:

alpha_relative = (10^alpha_absolute / (10^alpha_absolute + 10^beta_absolute + 10^delta_absolute + 10^gamma_absolute + 10^theta_absolute))

#### Band power scores

The band session score is computed by comparing the current value of a band power to its history. This current value is mapped to a score between 0 and 1 using a linear function that returns 0 if the current value is equal to or below the 10th percentile of the distribution of band powers, and returns 1 if it's equal to or above the 90th percentile. Linear scoring between 0 and 1 is done for any value between these two percentiles.

Be advised that these scores are based on recent history and it will take a few seconds before having a stable distribution to score the power against. The estimated distribution is continuously updated as long as the headband is on the head. However, every time it's updated, the newest values are weighted to have more importance than the historical values. This means that eventually old values will not be present anymore in the estimated distribution. The half-life of the estimated distribution at any given point is around 10 seconds.

The score will start being calculated as soon as the headband has established a good connection with the skin. Whenever the headset loses connection with the head (as determined by the DRL/REF contact quality) the estimated distributions are reset. This means that when the headband is removed, the session data from any previous user will be cleared.

**Enumerator**

| | |
|---|---|
| ACCELEROMETER | 3-axis accelerometer data packet An Accelerometer packet provides 3 pieces of data.<br><br>**See also**<br><br>• Accelerometer for mapping details. |
| GYRO | 3-axis gyro data packet A Gyro packet provides 3 pieces of data.<br><br>**See also**<br><br>• Gyro for mapping details. |

**Enumerator**

| | |
|---|---|
| EEG | Specifies raw EEG samples.<br>Values in this packet correspond to EEG data read from the different sensor locations on the headband. The accessors in the Eeg enum define the mapping from packet values to sensor locations. The units of EEG values are microvolts.<br>The size of the data is unspecified, but it is large enough to hold all the EEG channels emitted by the current preset.<br>In the future new Muse Presets may be added, which will have extra values.<br>**See also**<br>• Eeg for mapping details. |
| DROPPED_ACCELEROMETER | Packet stands in for n dropped samples of the accelerometer type. Size of the values array for this packet is always 1.<br>**Deprecated** This is never emitted in an actual Muse session; instead, NaN-filled packets of the basic type (EEG or ACCELEROMETER) are emitted to stand in for dropped packets. This can only appear when reading Muse files written with older versions of the library. |
| DROPPED_EEG | Packet stands in for n dropped samples of the eeg type. Size of the values array for this packet is always 1.<br>**Deprecated** This is never emitted in an actual Muse session; instead, NaN-filled packets of the basic type (EEG or ACCELEROMETER) are emitted to stand in for dropped packets. This can only appear when reading Muse files written with older versions of the library. |
| QUANTIZATION | Packet contains information about signal quantization. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br>Each index in this packet corresponds to the same index in an EEG packet. Quantization occurs when there is a particularly noisy signal, which generally happens when there is not a good contact between the headband and the skin.<br>Higher numbers are worse; 1 is no quantization, and 16 is maximum quantization.<br>These values are used under the hood by the library and by Muse Elements in reconstructing the EEG signal and contributing to an overall measure of noise; it is extremely unlikely that you will be interested in them. For measuring noise, it is recommended to instead use the more useful computed values like 'headband_on' or 'hsi'.<br>Each quantization packet applies to the next 16 EEG packets.<br>**See also**<br>• Eeg for mapping details. |
| BATTERY | Muse headband battery data packet. This packet provides 3 pieces of data.<br>**See also**<br>• Battery for mapping details. |

**Enumerator**

| | |
|---|---|
| DRL_REF | Packet contains raw data from Muse DRL and REF sensors. This packet provides 2 pieces of data. The units of both values are in microvolts.<br>**See also**<br>• DrlRef for mapping details. |
| ALPHA_ABSOLUTE | EEG derived value. Absolute alpha band powers for each channel. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br>**See also**<br>• Eeg for mapping details. |
| BETA_ABSOLUTE | EEG derived value. Absolute beta band powers for each channel. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br>**See also**<br>• Eeg for mapping details. |
| DELTA_ABSOLUTE | EEG derived value. Absolute delta band powers for each channel. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br>**See also**<br>• Eeg for mapping details. |
| THETA_ABSOLUTE | EEG derived value. Absolute theta band powers for each channel. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br>**See also**<br>• Eeg for mapping details. |
| GAMMA_ABSOLUTE | EEG derived value. Absolute gamma band powers for each channel. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br>**See also**<br>• Eeg for mapping details. |
| ALPHA_RELATIVE | EEG derived value. Relative alpha band powers for each channel. Values in this packet are in range [0; 1]. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br>**See also**<br>• Eeg for mapping details. |

**Enumerator**

| | |
|---|---|
| BETA_RELATIVE | EEG derived value. Relative beta band powers for each channel. Values in this packet are in range [0; 1]. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br><br>**See also**<br><br>• Eeg for mapping details. |
| DELTA_RELATIVE | EEG derived value. Relative delta band powers for each channel. Values in this packet are in range [0; 1]. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br><br>**See also**<br><br>• Eeg for mapping details. |
| THETA_RELATIVE | EEG derived value. Relative band powers for each channel. Values in this packet are in range [0; 1]. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br><br>**See also**<br><br>• Eeg for mapping details. |
| GAMMA_RELATIVE | EEG derived value. Relative band powers for each channel. Values in this packet are in range [0; 1]. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br><br>**See also**<br><br>• Eeg for mapping details. |
| ALPHA_SCORE | EEG derived value. Alpha band power scores for each channel. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br><br>**See also**<br><br>• Eeg for mapping details. |
| BETA_SCORE | EEG derived value. Beta band power scores for each channel. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br><br>**See also**<br><br>• Eeg for mapping details. |
| DELTA_SCORE | EEG derived value. Delta band power scores for each channel. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br><br>**See also**<br><br>• Eeg for mapping details. |

**Enumerator**

| | |
|---|---|
| THETA_SCORE | EEG derived value. Theta band power scores for each channel. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br><br>**See also**<br><br>• Eeg for mapping details. |
| GAMMA_SCORE | EEG derived value. Gamma band power scores for each channel. This packet contains the same amount of data as an EEG packet and has the same channel mapping.<br><br>**See also**<br><br>• Eeg for mapping details. |
| IS_GOOD | EEG derived value. Is Good indicates whether or not the last 1 second of raw EEG data on each channel was good or not. Eye blinks or muscle movement can interfere with EEG data and cause Is Good to report that the data is not good. This is emitted every 1/10 of a second to represent the rolling window of the last second of EEG data. This is only useful for real time EEG analysis. This packet only contains 4 values for the 4 sensors on the headband, there is no support for the auxillary channels.<br><br>**See also**<br><br>• Eeg for mapping details. |
| HSI | EEG derived value. HSI values represent the fit of the headband. (known as headband status indicator). This value is not emitted by the LibMuse SDK. |
| HSI_PRECISION | EEG derived value. HSI precision values represent the fit of the headband. This packet contains 4 values corresponding to Eeg::EEG1 , Eeg::EEG2 , Eeg::EEG3 and Eeg::EEG4 There are no MuseDataPacketType::HSI_PRECISION values for the auxillary channels.<br><br>Each channel represents the fit at that location. A value of `1` represents a good fit, `2` represents a mediocre fit, and a value or `4` represents a poor fit.<br><br>**See also**<br><br>• Eeg for mapping details. |
| ARTIFACTS | Artifacts packet type will be sent.<br>Note that this will result in your listener receiving MuseArtifactPacket We never emit a MuseDataPacket with MuseDataPacketType::ARTIFACTS ; this is only here for use in register / unregister methods. |
| MAGNETOMETER | 3-axis magnetometer data packet A Magnetometer packet provides 3 pieces of data.<br><br>**See also**<br><br>• Magnetometer for mapping details. |

**Enumerator**

| | |
|---|---|
| PRESSURE | Pressure packet provides both a raw and averaged ambient pressure value.<br>**See also**<br>• Pressure for mapping details. |
| TEMPERATURE | Temperature packet provides ambient temperature value. |
| ULTRA_VIOLET | UltraViolet packet provides both UVA and UVB index value and the average of the 2.<br>**See also**<br>• UltraViolet for mapping details. |
| NOTCH_FILTERED_EEG | EEG derived value. Notch filtered EEG is the raw EEG passed through a band stop filter to remove frequencies between 45 and 65 Hz inclusive. This packet contains 4 values corresponding to Eeg::EEG1 , Eeg::EEG2 , Eeg::EEG3 and Eeg::EEG4<br>**See also**<br>• Eeg for mapping details. |
| VARIANCE_EEG | EEG derived value. Signal variance for raw EEG values. Variance is the numerical value that measures how widely a set of numbers within the interval are spread out from the average value. This packet contains the variance value of raw EEG over the last second.<br>**See also**<br>• Eeg for mapping details.<br>• `https://en.wikipedia.org/wiki/↩ Variance#Discrete_random_variable` |
| VARIANCE_NOTCH_FILTERED_EEG | EEG derived value. Signal variance for notch filtered EEG. Variance is the numerical value that measures how widely a set of numbers within the interval are spread out from the average value. This packet contains the variance value of notch filtered EEG over the last second.<br>**See also**<br>• Eeg for mapping details.<br>• `https://en.wikipedia.org/wiki/↩ Variance#Discrete_random_variable` |
| PPG | Specifies PPG samples.<br>Values in this packet correspond to PPG data read from supported hardware. The accessors in the Ppg enum define the mapping from packet data to 3 different values Ambient, IR, and RED. The units of PPG values are arbitrary.<br>**See also**<br>• Ppg for mapping details. |
| IS_PPG_GOOD | PPG derived value.<br>**See also**<br>• Ppg for mapping details. |

**Enumerator**

| | |
|---|---|
| IS_HEART_GOOD | PPG derived value.<br>**See also**<br>• Ppg for mapping details. |
| THERMISTOR | Provides temperature values from the thermistor that is in contact with the user's skin. |
| IS_THERMISTOR_GOOD | Thermistor derived value. |
| AVG_BODY_TEMPERATURE | Thermistor derived value. |
| CLOUD_COMPUTED | Specifies a cloud computed value.<br>Values in this packet have been computed on the cloud remotely. This packet type allows the libmuse library to expose new packet type that has been generated remotely on the cloud. |
| OPTICS | Specifies OPTICS samples.<br>Values in this packet correspond to optics data read from supported hardware. The accessors in the Optics enum define the mapping from packet data to 16 different values. The units of OPTICS values are microamps.<br>**See also**<br>• Optics for mapping details. |
| TOTAL | The total number of possible data packet types |

#### 7.2.1.12 MuseModel

```
enum class interaxon::bridge::MuseModel : int [strong]
```

The model identifier of the headbands. The model is laser printed on the inside of the left pod of the headband as either "MU-01", "MU-02", "MU-03" or "MU-06". MuseS 2019 ( MU_04 ) MuseS 2021 ( MU_05 ) and MuseS 2025 ( MS_03 ) models are marked on the underside as "MS-01", "MS-02" or "MS-03".

**Enumerator**

| | |
|---|---|
| MU_01 | First model of Muse, Muse 2014 |
| MU_02 | Muse 2016 with Bluetooth Low Energy support. |
| MU_03 | Muse2 2018 adding PPG sensor, Accelerometer and Gyroscope. |
| MU_04 | MuseS 2019 softband (MS-01). |
| MU_05 | MuseS 2021 softband refresh (MS-02). |
| MU_06 | Muse2 2024 with USB-C connector. |
| MS_03 | MuseS 2025 softband with USB-C, Bluetooth 5.3, improved EEG and Optics (MS-03). |

#### 7.2.1.13 MusePreset

```
enum class interaxon::bridge::MusePreset : int [strong]
```

Defines all possible Muse presets.

Presets are specific to the model of headband. For example, setting a preset that is intended for a Muse 2014 ( MU_01 ) headband on a Muse 2016 ( MU_02 ) headband will not work. Attempting to set an incorrect preset on a headband will result in a warning in the log and the incorrect preset will be ignored. To avoid this, you should first connect to the headband without setting a preset and check the MuseConfiguration to get the model of the headband and then set the appropriate preset. You can also obtain the current preset of the headband from the MuseConfiguration

If you change the preset while the headband is connected, the headband will disconnect. If the preset is valid for that headband model, then the headband will automatically reconnect. If the preset is invalid for that headband model, then the headband will remain disconnected.

**See also**

MuseConfiguration

**Enumerator**

| | |
|---|---|
| PRESET_10 | 4 CH EEG, 10 bit @ 220 Hz, compression ON, no accelerometer, no battery data, no error data, no DRL/REF data.<br><br>Availability: Muse 2014 ( MU_01 ) only |
| PRESET_12 | 4 CH EEG, 10 bit @ 220 Hz, compression ON, 50 Hz accelerometer data, 0.1 Hz battery data, no error data, no DRL/REF data<br><br>Availability: Muse 2014 ( MU_01 ) only |
| PRESET_14 | 4 CH EEG, 10 bit @ 220 Hz, compression ON, 50 Hz accelerometer data, 0.1 Hz battery data, real-time error data, 10 bit @ 10 Hz DRL/REF data<br><br>This is the default for Muse 2014 ( MU_01 ).<br><br>Availability: Muse 2014 ( MU_01 ) only |
| PRESET_20 | 5 CH EEG, 12 bit @ 256 Hz, 52 Hz accelerometer/gyro, 0.1 Hz battery, 32 Hz DRL/REF.<br><br>Availability: Muse 2016 ( MU_02 ), Muse2 2018 ( MU_03 ), MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ), Muse2 2024 ( MU_06 ) |
| PRESET_21 | 4 CH EEG, 12 bit @ 256 Hz, 52 Hz accelerometer/gyro, 0.1 Hz battery, 32 Hz DRL/REF.<br><br>This is the default for Muse 2016 ( MU_02 ), Muse2 2018 ( MU_03 ), MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ), Muse2 2024 ( MU_06 ).<br><br>Availability: Muse 2016 ( MU_02 ), Muse2 2018 ( MU_03 ), MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ), Muse2 2024 ( MU_06 ) |
| PRESET_22 | 4 CH EEG, 12 bit @ 256 Hz, 0.1 Hz battery, 32 Hz DRL/REF.<br><br>Availability: Muse 2016 ( MU_02 ), Muse2 2018 ( MU_03 ), MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ), Muse2 2024 ( MU_06 ) |
| PRESET_23 | 5 CH EEG, 12 bit @ 256 Hz, 0.1 Hz battery, 32 Hz DRL/REF.<br><br>Availability: Muse2 2018 ( MU_03 ), MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ), Muse2 2024 ( MU_06 ) |

**Enumerator**

| | |
|---|---|
| PRESET_AB | 6 CH EEG, 16 bit @ 500 Hz, 50 Hz accelerometer, 0.1 Hz battery, compression OFF, notch filter OFF, no error data, no DRL/REF data<br><br>Research preset:<br>Only the following data packets are generated with this preset: MuseDataPacketType::EEG , MuseDataPacketType::ACCELEROMETER , MuseDataPacketType::BATTERY<br>Artifacts are not generated with this preset.<br><br>Availability: Muse 2014 ( MU_01 ) only |
| PRESET_AD | 4 CH EEG, 16 bit @ 500 Hz, 50 Hz accelerometer, 0.1 Hz battery, compression OFF, notch filter OFF, no error data, no DRL/REF data<br><br>Research preset:<br>Only the following data packets are generated with this preset: MuseDataPacketType::EEG , MuseDataPacketType::ACCELEROMETER , MuseDataPacketType::BATTERY<br>Artifacts are not generated with this preset.<br><br>Availability: Muse 2014 ( MU_01 ) only |
| PRESET_31 | 4 CH EEG, 12 bit @ 256 Hz, 0.1 Hz battery, 32 Hz DRL/REF, 52 Hz Acc, Gyro and Magnetometer, 0.1 Hz for UV and Pressure.<br><br>Availability: glasses only |
| PRESET_32 | No EEG data, only other sensors. 32 Hz DRL/REF, 52 Hz Acc, Gyro and Magnetometer, 0.1 Hz for battery, UV and Pressure.<br><br>Availability: glasses only |
| PRESET_50 | 5 CH EEG, 12 bit @ 256 Hz, 52 Hz accelerometer/gyro, 0.1 Hz battery, 32 Hz DRL/REF, PPG @ 64 Hz<br><br>Availability: Muse2 2018 ( MU_03 ), MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ), Muse2 2024 ( MU_06 ) |
| PRESET_51 | 4 CH EEG, 12 bit @ 256 Hz, 52 Hz accelerometer/gyro, 0.1 Hz battery, 32 Hz DRL/REF, PPG @ 64 Hz<br><br>Availability: Muse2 2018 ( MU_03 ), MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ), Muse2 2024 ( MU_06 ) |
| PRESET_52 | 4 CH EEG, 12 bit @ 256 Hz, 0.1 Hz battery, 32 Hz DRL/REF, PPG @ 64 Hz<br><br>Availability: Muse2 2018 ( MU_03 ), MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ), Muse2 2024 ( MU_06 ) |
| PRESET_53 | 6 CH EEG, 12 bit @ 256 Hz, 52 Hz accelerometer/gyro, 0.1 Hz battery, 32 Hz DRL/REF, PPG @ 64 Hz<br><br>Availability: MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ), Muse2 2024 ( MU_06 ) |
| PRESET_54 | 6 CH EEG, 12 bit @ 128 Hz, 52 Hz accelerometer/gyro, 0.1 Hz battery, 16 Hz DRL/REF<br><br>Availability: MuseS 2021 ( MU_05 ) |
| PRESET_55 | 4 CH EEG, 12 bit @ 128 Hz, 0.1 Hz battery, 16 Hz DRL/REF, PPG @ 64 Hz<br><br>Availability: MuseS 2021 ( MU_05 ) |
| PRESET_60 | 5 CH EEG, 12 bit @ 256 Hz, 52 Hz accelerometer/gyro, 0.1 Hz battery, 32 Hz DRL/REF, PPG @ 64 Hz, THERMISTOR @ 16 Hz<br><br>Availability: MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ) |

**Enumerator**

| | |
|---|---|
| PRESET_61 | 4 CH EEG, 12 bit @ 256 Hz, 52 Hz accelerometer/gyro, 0.1 Hz battery, 32 Hz DRL/REF, PPG @ 64 Hz, THERMISTOR @ 16 Hz<br><br>Availability: MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ) |
| PRESET_63 | 6 CH EEG, 12 bit @ 256 Hz, 52 Hz accelerometer/gyro, 0.1 Hz battery, 32 Hz DRL/REF, PPG @ 64 Hz, THERMISTOR @ 16 Hz<br><br>Availability: MuseS 2019 ( MU_04 ), MuseS 2021 ( MU_05 ) |
| PRESET_1021 | 4 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1022 | 8 CH EEG, 14 bit @ 256 Hz, 1 Hz battery, 32 Hz DRL/REF<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1023 | Battery only @ 5 Hz<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1024 | 52 Hz accelerometer/gyro<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1025 | 16 CH Optics @ 64 Hz, low power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1026 | 16 CH Optics @ 64 Hz, high power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1027 | 8 CH Optics @ 64 Hz, low power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1028 | 8 CH Optics @ 64 Hz, high power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1029 | 4 CH Optics @ 64 Hz, low power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_102A | 4 CH Optics @ 64 Hz, high power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1031 | 4 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 16 CH Optics @ 64 Hz, low power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1032 | 4 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 16 CH Optics @ 64 Hz, high power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1033 | 4 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 8 CH Optics @ 64 Hz, low power<br><br>Availability: MuseS 2025 ( MS_03 ) |

**Enumerator**

| | |
|---|---|
| PRESET_1034 | 4 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 8 CH Optics @ 64 Hz, high power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1035 | 4 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 4 CH Optics @ 64 Hz, low power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1036 | 4 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 4 CH Optics @ 64 Hz, high power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1041 | 8 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 16 CH Optics @ 64 Hz, low power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1042 | 8 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 16 CH Optics @ 64 Hz, high power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1043 | 8 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 8 CH Optics @ 64 Hz, low power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1044 | 8 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 8 CH Optics @ 64 Hz, high power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1045 | 8 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 4 CH Optics @ 64 Hz, low power<br><br>Availability: MuseS 2025 ( MS_03 ) |
| PRESET_1046 | 8 CH EEG, 14 bit @ 256 Hz, 52 Hz accelerometer/gyro, 1 Hz battery, 32 Hz DRL/REF, 4 CH Optics @ 64 Hz, high power<br><br>Availability: MuseS 2025 ( MS_03 ) |

### 7.2.1.14 NotchFrequency

```
enum class interaxon::bridge::NotchFrequency : int [strong]
```

Notch Frequencies refer to the possible power line frequencies in a geographic area that may create noise in raw EEG data.

Muse 2014 ( MU_01 ) headbands are equipped with a hardware filter to remove this noise in certain preset configurations.

For Muse 2016 ( MU_02 ) or later headbands or Muse 2014 ( MU_01 ) headbands using a research preset there is no hardware filtering.

**See also**

- MusePreset
- `http://en.wikipedia.org/wiki/Utility_frequency`

**Enumerator**

| | |
|---|---|
| NOTCH_NONE | The notch filter is not available on some hardware versions. |
| NOTCH_50HZ | 50 Hz frequency is used in most parts of the world: Europe, Russia, Africa |
| NOTCH_60HZ | 60 Hz frequency is used in North America and Asia. This is the default setting. |

#### 7.2.1.15 Optics

```
enum class interaxon::bridge::Optics : int [strong]
```

Represents the data mapping in an Optics packet.

Functional near-infrared spectroscopy (fNIRS) is a non-invasive imaging technique that measures brain activity by detecting changes in blood oxygenation.

The light absorbtion of the blood varies by wavelength. The 750nm wavelength is sensitive to deoxygenated hemoglobin (HbR), while the 850nm wavelength is sensitive to oxygenated hemoglobin (HbO). By analyzing the absorption of these wavelengths, we can determine changes in blood oxygenation levels.

For more information on fNIRS, see the following Wikipedia article:
`https://en.wikipedia.org/wiki/Functional_near-infrared_spectroscopy`

The Optics data is also used to measure PPG.

Optics data values are given in microamps (uA).

**See also**

MuseDataPacketType::OPTICS
MuseDataPacket::get_optics_channel_value()

**Enumerator**

| | |
|---|---|
| OPTICS1 | 730nm left outer (8 and 16 channel modes) or 730nm left inner (4 channel mode) |
| OPTICS2 | 730nm right outer (8 and 16 channel modes) or 730nm right inner (4 channel mode) |
| OPTICS3 | 850nm left outer (8 and 16 channel modes) or 850nm left inner (4 channel mode) |
| OPTICS4 | 850nm right outer (8 and 16 channel modes) or 850nm right inner (4 channel mode) |
| OPTICS5 | 730nm left inner (8 and 16 channel modes) |
| OPTICS6 | 730nm right inner (8 and 16 channel modes) |
| OPTICS7 | 850nm left inner (8 and 16 channel modes) |
| OPTICS8 | 850nm right inner (8 and 16 channel modes) |
| OPTICS9 | Red left outer (16 channel mode) |
| OPTICS10 | Red right outer (16 channel mode) |
| OPTICS11 | Ambient left outer (16 channel mode) |
| OPTICS12 | Ambient right outer (16 channel mode) |
| OPTICS13 | Red left inner (16 channel mode) |
| OPTICS14 | Red right inner (16 channel mode) |
| OPTICS15 | Ambient left inner (16 channel mode) |
| OPTICS16 | Ambient right inner (16 channel mode) |

#### 7.2.1.16 Ppg

`enum class interaxon::bridge::Ppg : int [strong]`

Represents the data mapping in a PPG packet. Enum values for IR and RED correspond to the Infrared and Red values. The AMBIENT enum value is repurposed on models MuseS 2019 ( MU_04 ) and MuseS 2021 ( MU_05 ).

Muse2 2018 ( MU_03 ) AMBIENT represents the Ambient value.
MuseS 2019 ( MU_04 ) AMBIENT represents the Green value. There is no Ambient available and Red is not used.
MuseS 2021 ( MU_05 ) AMBIENT represents the IR-H16 value. There is no Ambient available.

MuseS 2025 ( MS_03 ) uses Optics data for PPG.

Raw PPG values are given in arbitrary units.

**See also**

MuseDataPacketType::PPG
MuseDataPacket::get_ppg_channel_value()

**Enumerator**

| | |
|---|---|
| AMBIENT | Ambient, Green or IR-H16. |
| IR | IR. |
| RED | Red. |

#### 7.2.1.17 Pressure

`enum class interaxon::bridge::Pressure : int [strong]`

Represents the data mapping in a Pressure packet.

**See also**

MuseDataPacketType::PRESSURE
MuseDataPacket::get_pressure_value()

**Enumerator**

| | |
|---|---|
| RAW | The raw pressure value returned by the pressure sensor in mBar |
| AVERAGED | The averaged pressure value in mBar based on the last 10 readings. This provides a smoother curve for the pressure values. |

#### 7.2.1.18 ReaderMusePlaybackSettings

`enum class interaxon::bridge::ReaderMusePlaybackSettings : int [strong]`

Configures the playback speed and timestamps of a ReaderMuse

**See also**

ReaderMuse
ReaderMuseBuilder

**Enumerator**

| | |
|---|---|
| AS_FAST_AS_POSSIBLE_WITH_SAVED_↩ TIMESTAMP | The ReaderMuse should playback the data as fast as it can be read from the file. There will be no simulated time between packets and the timestamps of the packets created during playback will match the timestamps that are saved in the file. |
| SIMULATED_WITH_SAVED_TIMESTAMP | The ReaderMuse should playback the data simulating the time between packets. For example, if the second packet in the file has a timestamp that is 20ms later than the timestamp of the first packet, then LibMuse will wait for 20ms after the first packet is sent before sending the second packet. The timestamps of the packets created during playback will match the timestamps that are saved in the file.<br><br>Using this setting requires that the ReaderMuse be constructed with an EventLoop that will be used to simulate the time between packets. If this setting is used, but the ReaderMuse does not have an EventLoop then playback will not read the file and a warning will be logged. |
| SIMULATED_WITH_SYSTEM_CLOCK_TIMESTAMP | The ReaderMuse should playback the data simulating the time between packets. For example, if the second packet in the file has a timestamp that is 20ms later than the timestamp of the first packet, then LibMuse will wait for 20ms after the first packet is sent before sending the second packet. The timestamps of the packets created during playback will use the current times of the system clock rather than the timestamp of the packet saved in the file.<br><br>Using this setting requires that the ReaderMuse be constructed with an EventLoop that will be used to simulate the time between packets. If this setting is used, but the ReaderMuse does not have an EventLoop then playback will not read the file and a warning will be logged. |

### 7.2.1.19 ResultLevel

```
enum class interaxon::bridge::ResultLevel : int [strong]
```

Represents the level of a result

**Enumerator**

| | |
|---|---|
| R_NONE | default |
| R_SUCCESS | success |
| R_INFO | info |
| R_WARN | warn |
| R_ERROR | error |
| R_DEBUG | debug |

#### 7.2.1.20 Severity

```
enum class interaxon::bridge::Severity : int [strong]
```

The log message severity.

**Enumerator**

| | |
|---|---|
| SEV_VERBOSE | Verbose logs. These provide lots of details that are probably irrelevant except for tracing or debugging problems with the headband or library. |
| SEV_INFO | Informational logs. These messages are sent when significant but expected events happen (e.g. a connection started or was completed successfully.) |
| SEV_WARN | Warning logs. These messages indicate that something out of the ordinary but recoverable happened (e.g. a connection attempt failed but will be retried automatically.) |
| SEV_ERROR | Error logs. These messages indicate that something has gone wrong – for instance, a connection terminated unexpectedly or a corrupted packet was received. |
| SEV_FATAL | Fatal logs. These are sent when the library is in a completely unrecoverable state from which the only reasonable outcome is termination of the running process. The process will be aborted as soon as the log handler returns. |
| SEV_DEBUG | Debug-only logs. These are only interesting to developers trying to track down problems in the library. |
| TOTAL | Enum size sentinel; not an actual severity. |

#### 7.2.1.21 TimestampMode

```
enum class interaxon::bridge::TimestampMode : int [strong]
```

File writer timestamp mode.

**Enumerator**

| | |
|---|---|
| LEGACY | Legacy mode.<br>Use the current time for everything except data packets. Use the data packet's timestamp field for those. |
| CURRENT | Use the current time for timestamps. |
| EXPLICIT | Use set_timestamp for timestamps. |

#### 7.2.1.22 UltraViolet

```
enum class interaxon::bridge::UltraViolet : int [strong]
```

Represents the data mapping in a UltraViolet data packet

The ultra violet sensor will measure UVA and UVB wavelength. The UV index is the average of UVA and UVB calculated index values from raw sensor data.

**See also**

MuseDataPacketType::ULTRA_VIOLET

MuseDataPacket::get_uv_value()

**Enumerator**

| | |
|---|---|
| UV_INDEX | |
| UV_A | |
| UV_B | |

## 7.3 std Namespace Reference

STL namespace.

### Classes

- class **allocator**
  *STL class.*
- class **array**
  *STL class.*
- class **atomic**
  *STL class.*
- class **atomic_ref**
  *STL class.*
- class **auto_ptr**
  *STL class.*
- class **bad_alloc**
  *STL class.*
- class **bad_cast**
  *STL class.*
- class **bad_exception**
  *STL class.*
- class **bad_typeid**
  *STL class.*
- class **basic_fstream**
  *STL class.*
- class **basic_ifstream**
  *STL class.*
- class **basic_ios**
  *STL class.*
- class **basic_iostream**
  *STL class.*
- class **basic_istream**
  *STL class.*
- class **basic_istringstream**
  *STL class.*
- class **basic_ofstream**
  *STL class.*
- class **basic_ostream**
  *STL class.*
- class **basic_ostringstream**

*STL class.*
- class **basic_string**

  *STL class.*
- class **basic_string_view**

  *STL class.*
- class **basic_stringstream**

  *STL class.*
- class **bitset**

  *STL class.*
- class **complex**

  *STL class.*
- class **deque**

  *STL class.*
- class **domain_error**

  *STL class.*
- class **error_category**

  *STL class.*
- class **error_code**

  *STL class.*
- class **error_condition**

  *STL class.*
- class **exception**

  *STL class.*
- class **forward_list**

  *STL class.*
- class **fstream**

  *STL class.*
- struct hash< ::interaxon::bridge::Accelerometer >
- struct hash< ::interaxon::bridge::AnnotationFormat >
- struct hash< ::interaxon::bridge::Battery >
- struct hash< ::interaxon::bridge::ConnectionState >
- struct hash< ::interaxon::bridge::DrlRef >
- struct hash< ::interaxon::bridge::Eeg >
- struct hash< ::interaxon::bridge::ErrorType >
- struct hash< ::interaxon::bridge::Gyro >
- struct hash< ::interaxon::bridge::Magnetometer >
- struct hash< ::interaxon::bridge::MessageType >
- struct hash< ::interaxon::bridge::MuseDataPacketType >
- struct hash< ::interaxon::bridge::MuseModel >
- struct hash< ::interaxon::bridge::MusePreset >
- struct hash< ::interaxon::bridge::NotchFrequency >
- struct hash< ::interaxon::bridge::Optics >
- struct hash< ::interaxon::bridge::Ppg >
- struct hash< ::interaxon::bridge::Pressure >
- struct hash< ::interaxon::bridge::ReaderMusePlaybackSettings >
- struct hash< ::interaxon::bridge::ResultLevel >
- struct hash< ::interaxon::bridge::Severity >
- struct hash< ::interaxon::bridge::TimestampMode >
- struct hash< ::interaxon::bridge::UltraViolet >
- class **ifstream**

  *STL class.*
- class **invalid_argument**

*STL class.*
- class **ios**
    *STL class.*
- class **ios_base**
    *STL class.*
- class **istream**
    *STL class.*
- class **istringstream**
    *STL class.*
- class **jthread**
    *STL class.*
- class **length_error**
    *STL class.*
- class **list**
    *STL class.*
- class **lock_guard**
    *STL class.*
- class **logic_error**
    *STL class.*
- class **map**
    *STL class.*
- class **multimap**
    *STL class.*
- class **multiset**
    *STL class.*
- class **mutex**
    *STL class.*
- class **ofstream**
    *STL class.*
- class **ostream**
    *STL class.*
- class **ostringstream**
    *STL class.*
- class **out_of_range**
    *STL class.*
- class **overflow_error**
    *STL class.*
- class **pair**
    *STL class.*
- class **priority_queue**
    *STL class.*
- class **queue**
    *STL class.*
- class **range_error**
    *STL class.*
- class **recursive_mutex**
    *STL class.*
- class **recursive_timed_mutex**
    *STL class.*
- class **runtime_error**
    *STL class.*

- class **set**

    *STL class.*
- class **shared_lock**

    *STL class.*
- class **shared_mutex**

    *STL class.*
- class **shared_ptr**

    *STL class.*
- class **shared_timed_mutex**

    *STL class.*
- class **smart_ptr**

    *STL class.*
- class **span**

    *STL class.*
- class **stack**

    *STL class.*
- class **string**

    *STL class.*
- class **string_view**

    *STL class.*
- class **stringstream**

    *STL class.*
- class **system_error**

    *STL class.*
- class **thread**

    *STL class.*
- class **timed_mutex**

    *STL class.*
- class **u16string**

    *STL class.*
- class **u16string_view**

    *STL class.*
- class **u32string**

    *STL class.*
- class **u32string_view**

    *STL class.*
- class **u8string**

    *STL class.*
- class **u8string_view**

    *STL class.*
- class **underflow_error**

    *STL class.*
- class **unique_lock**

    *STL class.*
- class **unique_ptr**

    *STL class.*
- class **unordered_map**

    *STL class.*
- class **unordered_multimap**

    *STL class.*
- class **unordered_multiset**

*STL class.*

- class **unordered_set**

  *STL class.*
- class **valarray**

  *STL class.*
- class **vector**

  *STL class.*
- class **weak_ptr**

  *STL class.*
- class **wfstream**

  *STL class.*
- class **wifstream**

  *STL class.*
- class **wios**

  *STL class.*
- class **wistream**

  *STL class.*
- class **wistringstream**

  *STL class.*
- class **wofstream**

  *STL class.*
- class **wostream**

  *STL class.*
- class **wostringstream**

  *STL class.*
- class **wstring**

  *STL class.*
- class **wstring_view**

  *STL class.*
- class **wstringstream**

  *STL class.*

### 7.3.1 Detailed Description

STL namespace.

# Chapter 8

# Class Documentation

## 8.1 interaxon::bridge::Action Class Reference

```
#include <bridge_action.h>
```

#### Public Member Functions

- virtual ∼Action ()
- virtual void run ()=0

### 8.1.1 Detailed Description

A wrapper that represents a block or runnable action.

**See also**

EventLoop

### 8.1.2 Constructor & Destructor Documentation

#### 8.1.2.1 ∼Action()

```
virtual interaxon::bridge::Action::∼Action ()  [inline], [virtual]
```

### 8.1.3 Member Function Documentation

#### 8.1.3.1 run()

```
virtual void interaxon::bridge::Action::run ()  [pure virtual]
```

Performs the action.

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

- bridge_action.h

## 8.2 interaxon::bridge::AdvertisingStats Struct Reference

```
#include <bridge_advertising_stats.h>
```

**Public Member Functions**

- AdvertisingStats (int32_t numAdvertisingPackets_, double avgAdvertisingInterval_, double sigma↵ AdvertisingInterval_, double maxAdvertisingInterval_, bool isLost_, bool hasBadMac_)

**Public Attributes**

- int32_t numAdvertisingPackets
- double avgAdvertisingInterval
- double sigmaAdvertisingInterval
- double maxAdvertisingInterval
- bool isLost
- bool hasBadMac

### 8.2.1 Detailed Description

Statistics about the advertising packets that LibMuse sees from Muse 2016 ( MU_02 ) or later headbands.

**Warning**

This is only available on Android devices.

### 8.2.2 Constructor & Destructor Documentation

#### 8.2.2.1 AdvertisingStats()

```
interaxon::bridge::AdvertisingStats::AdvertisingStats (
             int32_t numAdvertisingPackets_,
             double avgAdvertisingInterval_,
             double sigmaAdvertisingInterval_,
             double maxAdvertisingInterval_,
             bool isLost_,
             bool hasBadMac_)  [inline]
```

### 8.2.3 Member Data Documentation

#### 8.2.3.1 avgAdvertisingInterval

```
double interaxon::bridge::AdvertisingStats::avgAdvertisingInterval
```

Returns the average interval in seconds between advertising packets. This is a running average since the time the statistics were created or reset.

**Returns**

the average interval between advertising packets.

#### 8.2.3.2 hasBadMac

```
bool interaxon::bridge::AdvertisingStats::hasBadMac
```

Indicates an issue with the MAC address of the headband.

**Returns**

`true` if for some reason the MAC address of a headband has changed since initial detection. `false` under normal circumstances.

#### 8.2.3.3 isLost

```
bool interaxon::bridge::AdvertisingStats::isLost
```

Indicates if the phone has lost contact with the Muse headband. This is a subjective measurement. Even if this is true, the phone may still be able to connect with the headband if a connection is requested.

**Returns**

`true` if the phone has lost contact with the headband. `false` otherwise.

#### 8.2.3.4 maxAdvertisingInterval

```
double interaxon::bridge::AdvertisingStats::maxAdvertisingInterval
```

Returns the maximum interval in seconds between receiving 2 advertising packets.

**Returns**

the maximum interval between advertising packets.

#### 8.2.3.5 numAdvertisingPackets

```
int32_t interaxon::bridge::AdvertisingStats::numAdvertisingPackets
```

Returns the number of advertising packets seen.

**Returns**

the number of advertising packets seen.

#### 8.2.3.6 sigmaAdvertisingInterval

```
double interaxon::bridge::AdvertisingStats::sigmaAdvertisingInterval
```

Returns the standard deviation of the average advertising interval.

**Returns**

the standard deviation of the average advertising interval.

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

- bridge_advertising_stats.h

## 8.3 interaxon::bridge::AnnotationData Struct Reference

```
#include <bridge_annotation_data.h>
```

**Public Member Functions**

- AnnotationData (std::string data_, AnnotationFormat format_, std::string event_type_, std::string event_id_, std::string parent_id_)

**Public Attributes**

- std::string data
- AnnotationFormat format
- std::string event_type
- std::string event_id
- std::string parent_id

### 8.3.1 Detailed Description

A data structure that corresponds to an Annotation message in the .muse file format.

### 8.3.2 Constructor & Destructor Documentation

#### 8.3.2.1 AnnotationData()

```
interaxon::bridge::AnnotationData::AnnotationData (
            std::string data_,
            AnnotationFormat format_,
            std::string event_type_,
            std::string event_id_,
            std::string parent_id_)  [inline]
```

## 8.3.3 Member Data Documentation

### 8.3.3.1 data

```
std::string interaxon::bridge::AnnotationData::data
```

This is additional data for the event, can be in any format specified by AnnotationFormat

### 8.3.3.2 event_id

```
std::string interaxon::bridge::AnnotationData::event_id
```

This can be used to pair events together or give them unique IDs. If you do not need this value, set it to the empty string "".

### 8.3.3.3 event_type

```
std::string interaxon::bridge::AnnotationData::event_type
```

This can be used to classify events, for example "blink" or "error". If you do not need this value, set it to the empty string "".

### 8.3.3.4 format

```
AnnotationFormat interaxon::bridge::AnnotationData::format
```

This specifies the string format of the event data.

### 8.3.3.5 parent_id

```
std::string interaxon::bridge::AnnotationData::parent_id
```

This can be used to pair events together or give them unique IDs. If you do not need this value, set it to the empty string "".

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

- bridge_annotation_data.h

## 8.4 interaxon::bridge::ApiVersion Class Reference

```
#include <bridge_api_version.h>
```

**Public Member Functions**

- virtual ∼ApiVersion ()
- virtual int64_t get_monotonic ()=0
- virtual int64_t get_major ()=0
- virtual int64_t get_minor ()=0
- virtual int64_t get_patch ()=0
- virtual int64_t get_api ()=0
- virtual std::string get_string ()=0

## 8.4.1 Detailed Description

Library version interface.

Describes the version of LibMuse.

## 8.4.2 Constructor & Destructor Documentation

### 8.4.2.1 ∼ApiVersion()

```
virtual interaxon::bridge::ApiVersion::~ApiVersion ()  [inline], [virtual]
```

## 8.4.3 Member Function Documentation

### 8.4.3.1 get_api()

```
virtual int64_t interaxon::bridge::ApiVersion::get_api ()  [pure virtual]
```

API

This number is incremented when changes are introduced to the API that are not backwards compatible.

**Returns**

The API version.

### 8.4.3.2 get_major()

```
virtual int64_t interaxon::bridge::ApiVersion::get_major ()  [pure virtual]
```

Major release.

This number is incremented on a major release of new functionality.

**Returns**

The major version.

#### 8.4.3.3 get_minor()

```
virtual int64_t interaxon::bridge::ApiVersion::get_minor ()  [pure virtual]
```

Minor release.

This number is incremented when minor changes are made that are backwards compatible.

**Returns**

The minor version.

#### 8.4.3.4 get_monotonic()

```
virtual int64_t interaxon::bridge::ApiVersion::get_monotonic ()  [pure virtual]
```

Returns a monotonically increasing version number.

This number is guaranteed to increase from release to release. Currently it is equivalent to:

```
get_major() * 1000000 + get_minor() * 10000 + get_patch() * 100 + get_api()
```

So for major = 5, minor = 1, patch = 3, api = 2, it would be 5010302.

#### 8.4.3.5 get_patch()

```
virtual int64_t interaxon::bridge::ApiVersion::get_patch ()  [pure virtual]
```

Patch release.

This number is incremented when backwards-compatible bugfixes are made.

**Returns**

The patch version.

#### 8.4.3.6 get_string()

```
virtual std::string interaxon::bridge::ApiVersion::get_string ()  [pure virtual]
```

String representation of the version.

This is a string like "x.y.z API Version j" where x is get_major(), y is get_minor(), z is get_patch() and j is get_api(). The format may change from release to release. In particular, don't make assumptions about it or try to parse it. Use the other API calls on this interface to get the numerical versions.

**Returns**

The version string.

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

- bridge_api_version.h

## 8.5 interaxon::bridge::ComputingDeviceConfiguration Struct Reference

```
#include <bridge_computing_device_configuration.h>
```

**Public Member Functions**

- ComputingDeviceConfiguration (std::string os_type_, std::string os_version_, std::string hardware_model_↩ name_, std::string hardware_model_id_, std::string processor_name_, std::string processor_speed_, int32↩ _t number_of_processors_, std::string memory_size_, std::string bluetooth_version_, std::string time_zone↩ _, int32_t time_zone_offset_seconds_)

**Public Attributes**

- std::string os_type
- std::string os_version
- std::string hardware_model_name
- std::string hardware_model_id
- std::string processor_name
- std::string processor_speed
- int32_t number_of_processors
- std::string memory_size
- std::string bluetooth_version
- std::string time_zone
- int32_t time_zone_offset_seconds

### 8.5.1 Detailed Description

This data structure contains fields, which describe the running machine. To prevent confusion, field names should correspond to names in the protobuf schema of the .muse file format. If the computing device configuration cannot be obtained for any field, they will be empty strings.

### 8.5.2 Constructor & Destructor Documentation

#### 8.5.2.1 ComputingDeviceConfiguration()

```
interaxon::bridge::ComputingDeviceConfiguration::ComputingDeviceConfiguration (
             std::string os_type_,
             std::string os_version_,
             std::string hardware_model_name_,
             std::string hardware_model_id_,
             std::string processor_name_,
             std::string processor_speed_,
             int32_t number_of_processors_,
             std::string memory_size_,
             std::string bluetooth_version_,
             std::string time_zone_,
             int32_t time_zone_offset_seconds_ ) [inline]
```

### 8.5.3 Member Data Documentation

#### 8.5.3.1 bluetooth_version

```
std::string interaxon::bridge::ComputingDeviceConfiguration::bluetooth_version
```

Bluetooth version

#### 8.5.3.2 hardware_model_id

```
std::string interaxon::bridge::ComputingDeviceConfiguration::hardware_model_id
```

Hardware model id. For ex.: "SM-N900W8", "MacBookPro-101"

#### 8.5.3.3 hardware_model_name

```
std::string interaxon::bridge::ComputingDeviceConfiguration::hardware_model_name
```

Hardware model. For ex.: "Samsung Galaxy Note 3", "Macbook Pro"

#### 8.5.3.4 memory_size

```
std::string interaxon::bridge::ComputingDeviceConfiguration::memory_size
```

Memory size. For ex.: "500MB", "2000MB"

#### 8.5.3.5 number_of_processors

```
int32_t interaxon::bridge::ComputingDeviceConfiguration::number_of_processors
```

Number of cores

#### 8.5.3.6 os_type

```
std::string interaxon::bridge::ComputingDeviceConfiguration::os_type
```

Operation system type. For ex.: "Android", "iOS", "Windows"

#### 8.5.3.7 os_version

```
std::string interaxon::bridge::ComputingDeviceConfiguration::os_version
```

Operation system version. For ex.: "10.1", "4.4.2"

#### 8.5.3.8 processor_name

```
std::string interaxon::bridge::ComputingDeviceConfiguration::processor_name
```

Processor name. For ex.: "Intel", "ARM"

#### 8.5.3.9 processor_speed

```
std::string interaxon::bridge::ComputingDeviceConfiguration::processor_speed
```

Processor frequency in Hz

#### 8.5.3.10 time_zone

```
std::string interaxon::bridge::ComputingDeviceConfiguration::time_zone
```

time zone indicator

#### 8.5.3.11 time_zone_offset_seconds

```
int32_t interaxon::bridge::ComputingDeviceConfiguration::time_zone_offset_seconds
```

time zone offset in seconds

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

- bridge_computing_device_configuration.h

## 8.6 interaxon::bridge::ComputingDeviceConfigurationFactory Class Reference

```
#include <computing_device_configuration_factory.h>
```

### Public Member Functions

- ComputingDeviceConfigurationFactory (const ComputingDeviceConfigurationFactory &rhs)=delete
- ComputingDeviceConfigurationFactory & operator= (const ComputingDeviceConfigurationFactory &rhs)=delete
- ComputingDeviceConfiguration get_computing_device_configuration ()

### Static Public Member Functions

- static std::shared_ptr< ComputingDeviceConfigurationFactory > get_instance ()

### 8.6.1 Detailed Description

Provides access to the ComputingDeviceConfiguration object containing information about the computing device.

### 8.6.2 Constructor & Destructor Documentation

#### 8.6.2.1 ComputingDeviceConfigurationFactory()

```
interaxon::bridge::ComputingDeviceConfigurationFactory::ComputingDeviceConfigurationFactory (
            const ComputingDeviceConfigurationFactory & rhs)  [delete]
```

### 8.6.3 Member Function Documentation

#### 8.6.3.1 get_computing_device_configuration()

```
ComputingDeviceConfiguration interaxon::bridge::ComputingDeviceConfigurationFactory::get_↩
computing_device_configuration ()
```

Retrieves the appropriate ComputingDeviceConfiguration structure for the current computing device.

**Returns**

A ComputingDeviceConfiguration structure for the current computing device.

#### 8.6.3.2 get_instance()

```
static std::shared_ptr< ComputingDeviceConfigurationFactory > interaxon::bridge::Computing↩
DeviceConfigurationFactory::get_instance ()  [static]
```

Static constructor for the singleton object.

**Returns**

An instance of the ComputingDeviceConfigurationFactory object.

#### 8.6.3.3 operator=()

```
ComputingDeviceConfigurationFactory & interaxon::bridge::ComputingDeviceConfigurationFactory↩
::operator= (
            const ComputingDeviceConfigurationFactory & rhs)  [delete]
```

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

- computing_device_configuration_factory.h

## 8.7 interaxon::bridge::Convert Class Reference

```
#include <conversions.h>
```

**Static Public Member Functions**

- static Platform::String to_platform_string (const std::string &str)
- static std::string to_std_string (Platform::String^ str)

### 8.7.1 Detailed Description

A utility class that provides conversions to and from C++ and UWP objects.

### 8.7.2 Member Function Documentation

#### 8.7.2.1 to_platform_string()

```
static Platform::String interaxon::bridge::Convert::to_platform_string (
            const std::string & str) [static]
```

Converts a std::string object to a Platform::String object.

**Parameters**

| | |
|---|---|
| *str* | The std::string object to convert. |

**Returns**

The Platform::String object with the same text representation.

#### 8.7.2.2 to_std_string()

```
static std::string interaxon::bridge::Convert::to_std_string (
            Platform::String^ str) [static]
```

Converts a Platform::String object to a std::string object.

**Parameters**

| | |
|---|---|
| *str* | The Platform::String object to convert. |

**Returns**

The std::string object with the same text representation.

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

- conversions.h

## 8.8 interaxon::bridge::DspData Struct Reference

```
#include <bridge_dsp_data.h>
```

**Public Member Functions**

- DspData (std::string type_, std::vector< double > float_array_, std::vector< int64_t > int_array_, std::string version_)

**Public Attributes**

- std::string type
- std::vector< double > float_array
- std::vector< int64_t > int_array
- std::string version

### 8.8.1 Detailed Description

Data struct corresponds to a DSP message in the .muse file format. This message is very generic and allows you to save custom messages.

### 8.8.2 Constructor & Destructor Documentation

#### 8.8.2.1 DspData()

```
interaxon::bridge::DspData::DspData (
            std::string type_,
            std::vector< double > float_array_,
            std::vector< int64_t > int_array_,
            std::string version_) [inline]
```

### 8.8.3 Member Data Documentation

#### 8.8.3.1 float_array

```
std::vector<double> interaxon::bridge::DspData::float_array
```

array of floating point data

#### 8.8.3.2 int_array

```
std::vector<int64_t> interaxon::bridge::DspData::int_array
```

array of integer data

#### 8.8.3.3 type

```
std::string interaxon::bridge::DspData::type
```

Type of your message

#### 8.8.3.4 version

```
std::string interaxon::bridge::DspData::version
```

version of your message format

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

- bridge_dsp_data.h

## 8.9 interaxon::bridge::Error Struct Reference

```
#include <bridge_error.h>
```

**Public Member Functions**

- Error (ErrorType type_, int32_t code_, std::string info_)

**Public Attributes**

- ErrorType type
- int32_t code
- std::string info

### 8.9.1 Detailed Description

Represents an an error from an operation (synchronous or asynchronous). These will be forwarded to any Error↩ Listeners that are registered.

**See also**

MuseErrorListener

Muse::register_error_listener()

### 8.9.2 Constructor & Destructor Documentation

#### 8.9.2.1 Error()

```
interaxon::bridge::Error::Error (
            ErrorType type_,
            int32_t code_,
            std::string info_) [inline]
```

### 8.9.3 Member Data Documentation

#### 8.9.3.1 code

`int32_t interaxon::bridge::Error::code`

Machine-parseable error code

**Returns**

a machine parseable error code.

#### 8.9.3.2 info

`std::string interaxon::bridge::Error::info`

Human-readable description.

**Returns**

the english error message.

#### 8.9.3.3 type

`ErrorType interaxon::bridge::Error::type`

Type of error

**Returns**

the type of error that has occurred.

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

- bridge_error.h

## 8.10 interaxon::bridge::EventLoop Class Reference

`#include <bridge_event_loop.h>`

**Public Member Functions**

- virtual ∼EventLoop ()
- virtual void post (const std::shared_ptr< Action > &action)=0
- virtual void post_delayed (const std::shared_ptr< Action > &action, int64_t delay_milliseconds)=0
- virtual void cancel ()=0

### 8.10.1 Detailed Description

Wrapper around a platform-specific event loop.

### 8.10.2 Constructor & Destructor Documentation

#### 8.10.2.1 ∼EventLoop()

```
virtual interaxon::bridge::EventLoop::∼EventLoop ()  [inline], [virtual]
```

### 8.10.3 Member Function Documentation

#### 8.10.3.1 cancel()

```
virtual void interaxon::bridge::EventLoop::cancel ()  [pure virtual]
```

Cancel all pending actions.

#### 8.10.3.2 post()

```
virtual void interaxon::bridge::EventLoop::post (
            const std::shared_ptr< Action > & action)  [pure virtual]
```

Posts an action to the event loop for processing on the next pass.

**Parameters**

| | |
|---|---|
| *action* | The action to perform. |

#### 8.10.3.3 post_delayed()

```
virtual void interaxon::bridge::EventLoop::post_delayed (
            const std::shared_ptr< Action > & action,
            int64_t delay_milliseconds)  [pure virtual]
```

Posts an action to the event loop for processing after the specified delay in milliseconds has passed.

**Parameters**

| | |
|---|---|
| *action* | The action to perform. |
| *delay_milliseconds* | The time to wait before posting the action to the event loop for processing. |

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

- bridge_event_loop.h

## 8.11 interaxon::bridge::EventLoopFactory Class Reference

```
#include <event_loop_factory.h>
```

### Static Public Member Functions

- static std::shared_ptr< EventLoop > get_event_loop ()

### 8.11.1 Detailed Description

A factory for creating an EventLoop that can be used for processing Actions asynchronously on a secondary thread.

### 8.11.2 Member Function Documentation

#### 8.11.2.1 get_event_loop()

```
static std::shared_ptr< EventLoop > interaxon::bridge::EventLoopFactory::get_event_loop ()
[static]
```

Creates and returns an EventLoop.

**Returns**

EventLoop

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

- event_loop_factory.h

## 8.12 std::hash<::interaxon::bridge::Accelerometer > Struct Reference

```
#include <bridge_accelerometer.h>
```

### Public Member Functions

- size_t operator() (::interaxon::bridge::Accelerometer type) const

### 8.12.1 Member Function Documentation

#### 8.12.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::Accelerometer >::operator() (
             ::interaxon::bridge::Accelerometer type) const  [inline]
```

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

- bridge_accelerometer.h

## 8.13 std::hash< ::interaxon::bridge::AnnotationFormat > Struct Reference

```
#include <bridge_annotation_format.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::AnnotationFormat type) const

### 8.13.1 Member Function Documentation

#### 8.13.1.1 operator()()

```
size_t std::hash< ::interaxon::bridge::AnnotationFormat >::operator() (
            ::interaxon::bridge::AnnotationFormat type) const  [inline]
```

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

- bridge_annotation_format.h

## 8.14 std::hash< ::interaxon::bridge::Battery > Struct Reference

```
#include <bridge_battery.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::Battery type) const

### 8.14.1 Member Function Documentation

#### 8.14.1.1 operator()()

```
size_t std::hash< ::interaxon::bridge::Battery >::operator() (
            ::interaxon::bridge::Battery type) const  [inline]
```

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

- bridge_battery.h

## 8.15 std::hash< ::interaxon::bridge::ConnectionState > Struct Reference

```
#include <bridge_connection_state.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::ConnectionState type) const

### 8.15.1 Member Function Documentation

#### 8.15.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::ConnectionState >::operator() (
             ::interaxon::bridge::ConnectionState type) const  [inline]
```

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

- bridge_connection_state.h

## 8.16 std::hash< ::interaxon::bridge::DrlRef > Struct Reference

```
#include <bridge_drl_ref.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::DrlRef type) const

### 8.16.1 Member Function Documentation

#### 8.16.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::DrlRef >::operator() (
             ::interaxon::bridge::DrlRef type) const  [inline]
```

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

- bridge_drl_ref.h

## 8.17 std::hash< ::interaxon::bridge::Eeg > Struct Reference

```
#include <bridge_eeg.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::Eeg type) const

### 8.17.1 Member Function Documentation

#### 8.17.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::Eeg >::operator() (
            ::interaxon::bridge::Eeg type) const  [inline]
```

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

- bridge_eeg.h

## 8.18 std::hash<::interaxon::bridge::ErrorType > Struct Reference

```
#include <bridge_error_type.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::ErrorType type) const

### 8.18.1 Member Function Documentation

#### 8.18.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::ErrorType >::operator() (
            ::interaxon::bridge::ErrorType type) const  [inline]
```

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

- bridge_error_type.h

## 8.19 std::hash<::interaxon::bridge::Gyro > Struct Reference

```
#include <bridge_gyro.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::Gyro type) const

### 8.19.1 Member Function Documentation

#### 8.19.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::Gyro >::operator() (
            ::interaxon::bridge::Gyro type) const  [inline]
```

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

- bridge_gyro.h

## 8.20 std::hash<::interaxon::bridge::Magnetometer > Struct Reference

```
#include <bridge_magnetometer.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::Magnetometer type) const

### 8.20.1 Member Function Documentation

#### 8.20.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::Magnetometer >::operator() (
             ::interaxon::bridge::Magnetometer type) const [inline]
```

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

- bridge_magnetometer.h

## 8.21 std::hash<::interaxon::bridge::MessageType > Struct Reference

```
#include <bridge_message_type.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::MessageType type) const

### 8.21.1 Member Function Documentation

#### 8.21.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::MessageType >::operator() (
             ::interaxon::bridge::MessageType type) const [inline]
```

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

- bridge_message_type.h

## 8.22 std::hash<::interaxon::bridge::MuseDataPacketType > Struct Reference

```
#include <bridge_muse_data_packet_type.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::MuseDataPacketType type) const

### 8.22.1 Member Function Documentation

#### 8.22.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::MuseDataPacketType >::operator() (
             ::interaxon::bridge::MuseDataPacketType type) const  [inline]
```

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

- bridge_muse_data_packet_type.h

## 8.23 std::hash<::interaxon::bridge::MuseModel > Struct Reference

```
#include <bridge_muse_model.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::MuseModel type) const

### 8.23.1 Member Function Documentation

#### 8.23.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::MuseModel >::operator() (
             ::interaxon::bridge::MuseModel type) const  [inline]
```

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

- bridge_muse_model.h

## 8.24 std::hash<::interaxon::bridge::MusePreset > Struct Reference

```
#include <bridge_muse_preset.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::MusePreset type) const

### 8.24.1 Member Function Documentation

#### 8.24.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::MusePreset >::operator() (
            ::interaxon::bridge::MusePreset type) const  [inline]
```

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

- bridge_muse_preset.h

## 8.25 std::hash< ::interaxon::bridge::NotchFrequency > Struct Reference

```
#include <bridge_notch_frequency.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::NotchFrequency type) const

### 8.25.1 Member Function Documentation

#### 8.25.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::NotchFrequency >::operator() (
            ::interaxon::bridge::NotchFrequency type) const  [inline]
```

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

- bridge_notch_frequency.h

## 8.26 std::hash< ::interaxon::bridge::Optics > Struct Reference

```
#include <bridge_optics.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::Optics type) const

### 8.26.1 Member Function Documentation

#### 8.26.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::Optics >::operator() (
            ::interaxon::bridge::Optics type) const  [inline]
```

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

- bridge_optics.h

## 8.27 std::hash< ::interaxon::bridge::Ppg > Struct Reference

```
#include <bridge_ppg.h>
```

### Public Member Functions

- size_t operator() (::interaxon::bridge::Ppg type) const

### 8.27.1 Member Function Documentation

#### 8.27.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::Ppg >::operator() (
            ::interaxon::bridge::Ppg type) const [inline]
```

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

- bridge_ppg.h

## 8.28 std::hash< ::interaxon::bridge::Pressure > Struct Reference

```
#include <bridge_pressure.h>
```

### Public Member Functions

- size_t operator() (::interaxon::bridge::Pressure type) const

### 8.28.1 Member Function Documentation

#### 8.28.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::Pressure >::operator() (
            ::interaxon::bridge::Pressure type) const [inline]
```

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

- bridge_pressure.h

## 8.29 std::hash< ::interaxon::bridge::ReaderMusePlaybackSettings > Struct Reference

```
#include <bridge_reader_muse_playback_settings.h>
```

### Public Member Functions

- size_t operator() (::interaxon::bridge::ReaderMusePlaybackSettings type) const

## 8.29.1 Member Function Documentation

### 8.29.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::ReaderMusePlaybackSettings >::operator() (
             ::interaxon::bridge::ReaderMusePlaybackSettings type) const  [inline]
```

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

- bridge_reader_muse_playback_settings.h

## 8.30 std::hash<::interaxon::bridge::ResultLevel > Struct Reference

```
#include <bridge_result_level.h>
```

### Public Member Functions

- size_t operator() (::interaxon::bridge::ResultLevel type) const

## 8.30.1 Member Function Documentation

### 8.30.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::ResultLevel >::operator() (
             ::interaxon::bridge::ResultLevel type) const  [inline]
```

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

- bridge_result_level.h

## 8.31 std::hash<::interaxon::bridge::Severity > Struct Reference

```
#include <bridge_severity.h>
```

### Public Member Functions

- size_t operator() (::interaxon::bridge::Severity type) const

### 8.31.1 Member Function Documentation

#### 8.31.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::Severity >::operator() (
            ::interaxon::bridge::Severity type) const  [inline]
```

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

- bridge_severity.h

## 8.32 std::hash<::interaxon::bridge::TimestampMode > Struct Reference

```
#include <bridge_timestamp_mode.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::TimestampMode type) const

### 8.32.1 Member Function Documentation

#### 8.32.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::TimestampMode >::operator() (
            ::interaxon::bridge::TimestampMode type) const  [inline]
```

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

- bridge_timestamp_mode.h

## 8.33 std::hash<::interaxon::bridge::UltraViolet > Struct Reference

```
#include <bridge_ultra_violet.h>
```

**Public Member Functions**

- size_t operator() (::interaxon::bridge::UltraViolet type) const

### 8.33.1 Member Function Documentation

#### 8.33.1.1 operator()()

```
size_t std::hash<::interaxon::bridge::UltraViolet >::operator() (
            ::interaxon::bridge::UltraViolet type) const  [inline]
```

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

- bridge_ultra_violet.h

## 8.34 interaxon::bridge::LibmuseVersion Class Reference

```
#include <bridge_libmuse_version.h>
```

**Public Member Functions**

- virtual ∼LibmuseVersion ()

**Static Public Member Functions**

- static std::shared_ptr< ApiVersion > instance ()

### 8.34.1 Detailed Description

The version of the LibMuse library.

### 8.34.2 Constructor & Destructor Documentation

#### 8.34.2.1 ∼LibmuseVersion()

```
virtual interaxon::bridge::LibmuseVersion::∼LibmuseVersion ( ) [inline], [virtual]
```

### 8.34.3 Member Function Documentation

#### 8.34.3.1 instance()

```
static std::shared_ptr< ApiVersion > interaxon::bridge::LibmuseVersion::instance ( ) [static]
```

Returns the LibMuse API version.

**Returns**

the LibMuse API version.

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

- bridge_libmuse_version.h

## 8.35 interaxon::bridge::LogListener Class Reference

```
#include <bridge_log_listener.h>
```

**Public Member Functions**

- virtual ~LogListener ()
- virtual void receive_log (const LogPacket &log)=0

## 8.35.1 Detailed Description

User-supplied log message handler.

You can provide an instance of this class in order to integrate LibMuse logs into your application's logging system.

**See also**

LogManager

## 8.35.2 Constructor & Destructor Documentation

### 8.35.2.1 ~LogListener()

```
virtual interaxon::bridge::LogListener::~LogListener ()  [inline], [virtual]
```

## 8.35.3 Member Function Documentation

### 8.35.3.1 receive_log()

```
virtual void interaxon::bridge::LogListener::receive_log (
             const LogPacket & log)  [pure virtual]
```

Handler method for log messages. These are sent when interesting events happen. A simple handler may just print each message to stderr or stdout separated by newlines. It's also possible to implement arbitrary log policies within this handler.

**Warning**

receiveLog may be entered concurrently from multiple threads; if your listener implementation requires synchronization, it should provide its own.

Your listener should not make any LibMuse API calls; otherwise, unbounded recursion may occur, leading to resource exhaustion.

**Parameters**

| | |
|---|---|
| *log* | The log message. |

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

- bridge_log_listener.h

## 8.36 interaxon::bridge::LogManager Class Reference

```
#include <bridge_log_manager.h>
```

### Public Member Functions

- virtual ~LogManager ()
- virtual std::shared_ptr< LogListener > make_default_log_listener ()=0
- virtual void set_log_listener (const std::shared_ptr< LogListener > &listener)=0
- virtual void set_minimum_severity (Severity severity)=0
- virtual void write_log (Severity severity, bool raw, const std::string &tag, const std::string &message)=0
- virtual int64_t get_timestamp ()=0
- virtual double time_since (int64_t timestamp)=0

### Static Public Member Functions

- static std::shared_ptr< LogManager > instance ()

### 8.36.1 Detailed Description

Manages the listener for any logging events generated by LibMuse.

To disable logging:

```
set_log_listener(null);
```

To install a custom log listener for your application:

```
set_log_listener(your_log_listener);
```

To restore default logging to standard error (if you have previously changed the log listener):

```
set_log_listener(make_default_log_listener());
```

An instance of the default listener is installed unless another listener is supplied. The default log listener prints all messages to standard error as they show up, without any buffering, using the following formatting rules:

- Raw logs are printed directly without any formatting or newline.
- Other messages are formatted to pretty-print with the tag, timestamp, severity, and log message on a single line (prepending a newline if the last message printed was raw.)

For example, suppose that these packets showed up in order: {info, false, "A", 4, "thing A"}, {info, true, "FOO", 1, "+"}, {err, true, "BAR", 3, "-"}, {warn, false, "B", 2, "thing B"} where the second parameter idicates if this is a raw log message. The log output would be formatted roughly as such:

```
A [I 4.000] thing A
+-
B [W 2.000] thing B
```

In order to provide this formatting without buffering or losing messages, the default log listener uses a mutex, effectively making the logging subsystem single-threaded and IO-bound.

**See also**

LogListener

## 8.36.2 Constructor & Destructor Documentation

### 8.36.2.1 ∼LogManager()

```
virtual interaxon::bridge::LogManager::∼LogManager () [inline], [virtual]
```

## 8.36.3 Member Function Documentation

### 8.36.3.1 get_timestamp()

```
virtual int64_t interaxon::bridge::LogManager::get_timestamp () [pure virtual]
```

Return the current timestamp.

The returned value is in microseconds since some epoch – January 1 1970 on UNIX, but possibly other values on other systems.

**Returns**

The current timestamp.

### 8.36.3.2 instance()

```
static std::shared_ptr< LogManager > interaxon::bridge::LogManager::instance () [static]
```

Returns the shared log manager instance.

**Returns**

The shared log manager instance.

### 8.36.3.3 make_default_log_listener()

```
virtual std::shared_ptr< LogListener > interaxon::bridge::LogManager::make_default_log_↩
listener () [pure virtual]
```

Constructs and returns a default log listener.

**Returns**

an instance of the default log listener.

### 8.36.3.4 set_log_listener()

```
virtual void interaxon::bridge::LogManager::set_log_listener (
             const std::shared_ptr< LogListener > & listener) [pure virtual]
```

Sets the log listener.

There is only one log listener active at a time; hence this is just a plain setter rather than a register / unregister pair.

**Parameters**

| | |
|---|---|
| *listener* | the LogListener to use. |

### 8.36.3.5 set_minimum_severity()

```
virtual void interaxon::bridge::LogManager::set_minimum_severity (
             Severity severity)  [pure virtual]
```

Set the minimum log severity.

Messages at or above this severity will be passed to the configured log listener. Messages below this severity will be dropped.

**Parameters**

| | |
|---|---|
| *severity* | the minimum log severity to log. |

### 8.36.3.6 time_since()

```
virtual double interaxon::bridge::LogManager::time_since (
             int64_t timestamp)  [pure virtual]
```

Return the time elapsed in seconds since the passed timestamp.

**Parameters**

| | |
|---|---|
| *timestamp* | The prior timestamp. |

### 8.36.3.7 write_log()

```
virtual void interaxon::bridge::LogManager::write_log (
             Severity severity,
             bool raw,
             const std::string & tag,
             const std::string & message)  [pure virtual]
```

Write a log line.

This method sends messages to the configured log listener. It is exposed here primarily so that LibMuse can access it from its own platform-specific code, but there's nothing stopping you from using it and doing all your logging through LibMuse. Except please don't, because we may change the API at any time, and why would you?

The log's timestamp is the number of seconds since this log manager was created.

**Parameters**

| | |
|---|---|
| *severity* | The severity of this log message. |
| *raw* | `true` if this is a raw log message (no formatting), `false` otherwise |
| *tag* | The tag for this message. |
| *message* | The log message. |

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

- bridge_log_manager.h

## 8.37 interaxon::bridge::LogPacket Struct Reference

```
#include <bridge_log_packet.h>
```

**Public Member Functions**

- LogPacket (Severity severity_, bool raw_, std::string tag_, double timestamp_, std::string message_)

**Public Attributes**

- Severity severity
- bool raw
- std::string tag
- double timestamp
- std::string message

### 8.37.1 Detailed Description

A log message.

### 8.37.2 Constructor & Destructor Documentation

#### 8.37.2.1 LogPacket()

```
interaxon::bridge::LogPacket::LogPacket (
            Severity severity_,
            bool raw_,
            std::string tag_,
            double timestamp_,
            std::string message_)  [inline]
```

### 8.37.3 Member Data Documentation

#### 8.37.3.1 message

```
std::string interaxon::bridge::LogPacket::message
```

Content of this log message. Not newline-terminated; handlers should provide their own line-based formatting.

**Returns**

The log message content.

#### 8.37.3.2 raw

```
bool interaxon::bridge::LogPacket::raw
```

Denotes raw messages.

Raw messages are generally short (one or two character) messages that can be passed straight to the screen without any formatting or added newlines. (We use this to reduce the noise level on some of our verbose log mesages.)

**Returns**

`true` if this is a raw log message, `false` otherwise.

#### 8.37.3.3 severity

```
Severity interaxon::bridge::LogPacket::severity
```

Severity of this message.

**Returns**

The severity of this message.

#### 8.37.3.4 tag

```
std::string interaxon::bridge::LogPacket::tag
```

Human-readable, usually all-caps, tag for this message, e.g. "CONNECTOR" or "DEVICE".

**Returns**

The message tag.

#### 8.37.3.5 timestamp

```
double interaxon::bridge::LogPacket::timestamp
```

Timestamp for this log, in seconds.

**Returns**

The timestamp of this log message.

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

- bridge_log_packet.h

## 8.38 interaxon::bridge::Muse Class Reference

```
#include <bridge_muse.h>
```

**Public Member Functions**

- virtual ∼Muse ()
- virtual void connect ()=0
- virtual void disconnect ()=0
- virtual void execute ()=0
- virtual void run_asynchronously ()=0
- virtual ConnectionState get_connection_state ()=0
- virtual std::string get_mac_address ()=0
- virtual std::string get_name ()=0
- virtual double get_rssi ()=0
- virtual MuseModel get_model ()=0
- virtual double get_last_discovered_time ()=0
- virtual void set_num_connect_tries (int32_t num_tries)=0
- virtual std::shared_ptr< MuseConfiguration > get_muse_configuration ()=0
- virtual std::shared_ptr< MuseVersion > get_muse_version ()=0
- virtual void register_connection_listener (const std::shared_ptr< MuseConnectionListener > &listener)=0
- virtual void unregister_connection_listener (const std::shared_ptr< MuseConnectionListener > &listener)=0
- virtual void register_data_listener (const std::shared_ptr< MuseDataListener > &listener, MuseDataPacketType type)=0
- virtual void unregister_data_listener (const std::shared_ptr< MuseDataListener > &listener, MuseDataPacketType type)=0
- virtual void register_error_listener (const std::shared_ptr< MuseErrorListener > &listener)=0
- virtual void unregister_error_listener (const std::shared_ptr< MuseErrorListener > &listener)=0
- virtual void unregister_all_listeners ()=0
- virtual void set_preset (MusePreset preset)=0
- virtual void enable_led_indicator (bool enable)=0
- virtual void enable_data_transmission (bool enable)=0
- virtual void set_notch_frequency (NotchFrequency new_frequency)=0
- virtual bool is_low_energy ()=0
- virtual bool is_paired ()=0
- virtual bool is_connectable ()=0
- virtual void set_license_data (const std::vector< uint8_t > &data)=0
- virtual void enable_exception (bool enable)=0
- virtual void set_property (const std::string &name, const std::string &value)=0

### 8.38.1 Detailed Description

Provides the client interface to a specific Muse Headband.

This class maps to a single Muse Headband, providing an API for receiving connection events and data packets, as well as the ability to inspect various properties of the headband it corresponds to. Users can initiate and halt a connection to the device, enable data transmission, find out the human-readable name of the headband, etc.

This class should never be constructed directly by a user; rather, references to it can be retrieved from MuseManager.

There are two main ways of interacting with a Muse:

1. calling run_asynchronously()

2. calling connect() on your own and then repeatedly calling execute()

If you call run_asynchronously() , you do not need to call connect() or execute() on your own. Everything is taken care of by LibMuse for the duration of the connection (i.e. until you receive a disconnected event).

If you call connect() on your own, you are responsible for calling execute() on your own as well. Execute performs small, non-blocking steps of work such as initiating a connection, streaming data or disconnecting. If execute() is not called, no work is performed. The Muse will not transition its connection state and you will not receive any data. If you are using this method of interacting with a Muse you need to continue calling execute() until you receive a DISCONNECTED event through a registered connection listener. It may take multiple execute calls to reach the DISCONNECTED after issuing the disconnect request. Failure to wait for the DISCONNECTED event can result in unanticipated consequences.

## 8.38.2 Constructor & Destructor Documentation

### 8.38.2.1 ∼Muse()

```
virtual interaxon::bridge::Muse::∼Muse ()  [inline], [virtual]
```

## 8.38.3 Member Function Documentation

### 8.38.3.1 connect()

```
virtual void interaxon::bridge::Muse::connect ()  [pure virtual]
```

Initiate a connection to a Muse headband.

This call is non-blocking and returns immediately. Since this returns before the connection is established, you should instead use a connection listener to implement any logic, such as updating a UI element, on successful connection. When the connection is successful you will receive a CONNECTED event. If the connection is unsuccessful you will receive a DISCONNECTED event.

**Threading:** method is thread-safe.

### 8.38.3.2 disconnect()

```
virtual void interaxon::bridge::Muse::disconnect ()  [pure virtual]
```

Disconnects your mobile device from Muse Headband.

If you are calling execute() on your own, remember to continue calling execute() until you receive the DISCONNECTED event.

**Threading:** method is thread-safe.

If you don't want to receive disconnection event, unregister listeners manually first.

### 8.38.3.3 enable_data_transmission()

```
virtual void interaxon::bridge::Muse::enable_data_transmission (
             bool enable)  [pure virtual]
```

Starts/stops data transmission (but keep-alive packets will be still sent). If you're using low-level interface (connect + execute), you should still call execute() to continue sending keep-alive messages. If you're using run_asynchronously() , then it will take care about sending keep-alive packets.
**Threading:** method is thread-safe. You can call it in the middle of an execute operation. If method is called before connection is established, the right setting will be passed to device during connection routine.

**Parameters**

| | |
|---|---|
| *enable* | `true` to start streaming data. `false` to pause the data stream. |

### 8.38.3.4 enable_exception()

```
virtual void interaxon::bridge::Muse::enable_exception (
            bool enable)  [pure virtual]
```

Enable/disable libmuse to rethrow any exceptions caught in your app code from the various listeners. For example, if there is a bug in your data listener code that results in an exception. Libmuse will catch that and it can rethrow that to cause your app to crash if enabled. Or if disabled it will swallow that exception and prevent the app from crashing. It is useful to enable this in development to help track down any bugs in your code more easily. It is best to disable this in production code to prevent your app from crashing. If this method is not called, the default is disabled.

**Parameters**

| | |
|---|---|
| *enable* | `true` to enable exception to be thrown. `false` to disable. |

### 8.38.3.5 enable_led_indicator()

```
virtual void interaxon::bridge::Muse::enable_led_indicator (
            bool enable)  [pure virtual]
```

Toggles the LED indicator state on Muse headbands supporting sleep.

**Warning**

The headband must be in the CONNECTED state before calling this method. **Threading:** method is thread-safe.

**Parameters**

| | |
|---|---|
| *enable* | `true` to turn indicator LEDs on. `false` to turn them off. |

### 8.38.3.6 execute()

```
virtual void interaxon::bridge::Muse::execute ()  [pure virtual]
```

Runs a single, non-blocking step of processing.

Reads data from Bluetooth if there is any; updates the connection state; sends packets to listeners; etc. This should be called relatively frequently: max 250ms, ideally 20ms.

If you are listening for computed values (e.g. band powers) rather than just raw EEG samples, then be aware that this method may trigger some amount of processing in the foreground before it returns. If this has undesirable effects on app performance, then you may want to move it to a dedicated thread.

Also be aware that all packets will be received on the same thread as execute() is run.

**Threading:** This method is NOT thread safe. You must synchronize the Muse object on your own if you wish to call it concurrently from more than one thread.

#### 8.38.3.7 get_connection_state()

```
virtual ConnectionState interaxon::bridge::Muse::get_connection_state ()  [pure virtual]
```

Returns current connection state.

**Threading:** method is thread-safe.

**Returns**

The current connection state.

#### 8.38.3.8 get_last_discovered_time()

```
virtual double interaxon::bridge::Muse::get_last_discovered_time ()  [pure virtual]
```

Returns the time at which this device was most recently discovered.

Muse devices broadcast service information every few seconds, and this is the last time LibMuse has heard this broadcast info.

Only implemented on low-energy Muses. Returns NaN if called on non-low-energy Muses.

The value is in microseconds since some common start time (e.g. epoch, device start time, or app start time) that is guaranteed not to change within a running app process. The value may be used e.g. to determine which of two devices was discovered more recently than the other. It should not be used as an absolute time, nor should the common start time be assumed to be the same between any two running app instances.

**Returns**

The time at which this device was most recently discovered.

#### 8.38.3.9 get_mac_address()

```
virtual std::string interaxon::bridge::Muse::get_mac_address ()  [pure virtual]
```

Returns Bluetooth MAC address of the Muse Headband.

**Threading:** method is thread-safe.

**Returns**

The MAC address of the Muse Headband.

### 8.38.3.10 get_model()

```
virtual MuseModel interaxon::bridge::Muse::get_model ()  [pure virtual]
```

Returns the model of the connected Muse Headband.

For Muse2 2018 ( MU_03 ) and later, the Muse must have reached the CONNECTED state before this method is called. Otherwise it will return MU_02.

This should return a value for every Muse currently available.

**Threading:** method is thread-safe.

**Returns**

The model of the Muse Headband.

### 8.38.3.11 get_muse_configuration()

```
virtual std::shared_ptr< MuseConfiguration > interaxon::bridge::Muse::get_muse_configuration
()  [pure virtual]
```

Returns struct which contains all information about Muse configuration.

**Threading:** method is thread-safe.

**Warning**

The Muse Configuration object is only populated during connection routine or after headband settings (like preset or notch frequency) are changed. If this is called before the Muse is connected, the configuration will be `null`.

**Returns**

The configuration information of this Muse or `null` if the configuration is unknown.

### 8.38.3.12 get_muse_version()

```
virtual std::shared_ptr< MuseVersion > interaxon::bridge::Muse::get_muse_version ()  [pure
virtual]
```

Returns all information about the version of the Muse.

**Threading:** method is thread-safe.

**Warning**

The Muse Version is populated during connection routine only. If this is called before the Muse is connected, the version will be `null`.

**Returns**

The version of this Muse or `null` if the version is unknown.

#### 8.38.3.13 get_name()

```
virtual std::string interaxon::bridge::Muse::get_name ()  [pure virtual]
```

Returns Bluetooth name of the Muse Headband.

**Threading:** method is thread-safe.

**Returns**

The name of the Muse Headband.

#### 8.38.3.14 get_rssi()

```
virtual double interaxon::bridge::Muse::get_rssi ()  [pure virtual]
```

Returns the RSSI of this device.

Only implemented on low-energy Muses. Returns NaN if called on non-low-energy Muses.

**Threading:** method is thread-safe.

**Returns**

The RSSI of a low-energy Muse or NaN if the Muse is not low-energy.

#### 8.38.3.15 is_connectable()

```
virtual bool interaxon::bridge::Muse::is_connectable ()  [pure virtual]
```

Returns true if the Muse is connectable, false otherwise. This will always return true for Muse 2014 ( MU_01 ). For Muse 2016 ( MU_02 ) or later headbands this will return true on Android and iOS. For Muse 2016 ( MU_02 ) or later headbands on Windows, this will return true if the headband is advertising it is connectable, false if it is not.

**Returns**

`true` if the Muse is connectable, `false` otherwise.

#### 8.38.3.16 is_low_energy()

```
virtual bool interaxon::bridge::Muse::is_low_energy ()  [pure virtual]
```

True if this device supports Bluetooth Low-Energy.

**Returns**

`true` if this Muse supports Bluetooth Low-Energy. `false` if it does not.

### 8.38.3.17 is_paired()

```
virtual bool interaxon::bridge::Muse::is_paired ()  [pure virtual]
```

Returns true if the Muse is paired with the OS, false otherwise. This will always return true for Muse 2014 ( MU_01 ). For Muse 2016 ( MU_02 ) or later headbands this will return false on Android and iOS. For Muse 2016 ( MU_02 ) or later headbands on Windows, this will return true if the headband is paired with the OS and false if it is not.

**Returns**

`true` if the Muse is paired with the OS, `false` otherwise.

### 8.38.3.18 register_connection_listener()

```
virtual void interaxon::bridge::Muse::register_connection_listener (
             const std::shared_ptr< MuseConnectionListener > & listener)  [pure virtual]
```

Registers a connection listener. The same listener cannot be registered twice. If the listener was already registered, then this method does nothing.

**Threading:** method is thread-safe.

**Parameters**

| | |
|---|---|
| *listener* | The listener to register. |

### 8.38.3.19 register_data_listener()

```
virtual void interaxon::bridge::Muse::register_data_listener (
             const std::shared_ptr< MuseDataListener > & listener,
             MuseDataPacketType type)  [pure virtual]
```

Registers a data listener. You can register the same listener to listen for different packet types - just call this method again. It's your responsibility to make sure that the listener handles all packet types correctly.

If this listener was already registered for this specific type then this method does nothing.

**Threading:** method is thread-safe.

**Parameters**

| | |
|---|---|
| *listener* | The listener to register. |
| *type* | The type of data packet the listener will receive. |

### 8.38.3.20 register_error_listener()

```
virtual void interaxon::bridge::Muse::register_error_listener (
             const std::shared_ptr< MuseErrorListener > & listener)  [pure virtual]
```

Registers an error listener.

**Threading:** method is thread-safe.

**Parameters**

|  |  |
|---|---|
| *listener* | The listener to register. |

#### 8.38.3.21 run_asynchronously()

```
virtual void interaxon::bridge::Muse::run_asynchronously ()  [pure virtual]
```

Connect to a headband out of the main logic flow.

This method handles all the connection logic for a single connection to a headband (i.e. starting with issuing a "connecting" event and ending with issuing a "disconnected" event). It may run in a separate thread or an event loop on the main thread; the details are platform-specific.

If this method is used, then all packets are delivered on the main thread. Windows is an exception. On Windows all packets are delivered on a dedicated thread other than the main thread.

The lifetime of effect of this method ends on disconnection; if you want to initiate a new connection to a headband that has disconnected, you should call run_asynchronously() on it again.

If this method is called multiple times within the lifetime of a single connection, any subsequent calls will do nothing other than print a log message.

Note that there is currently a race condition if run_asynchronously() is called

1. as a result of a DISCONNECTED packet, and
2. on a different thread from the one that delivered the notification.

This will be resolved in a future library release. For now, call run_asynchronously() on the same thread that sent the DISCONNECTED packet, or after some modest delay (say 1 second.)

**Threading:** This method is thread safe.

#### 8.38.3.22 set_license_data()

```
virtual void interaxon::bridge::Muse::set_license_data (
             const std::vector< uint8_t > & data)  [pure virtual]
```

Allows forwarding of license data from cloud.

**Parameters**

|  |  |
|---|---|
| *data* | The encrypted license blob. |

#### 8.38.3.23 set_notch_frequency()

```
virtual void interaxon::bridge::Muse::set_notch_frequency (
            NotchFrequency new_frequency)  [pure virtual]
```

Changes notch frequency (power line frequency).

**Threading:** method is thread-safe. You can call it in the middle of execute operation, but in this case be aware that this operation will interrupt data streaming to set new notch frequency. Data streaming will be restored after that. If method is called before a connection is established, the right setting will be passed to device during connection routine.

When combining a call setNotchFrequency with Muse::set_preset() setNotchFrequency can be called before or after the call to Muse::set_preset() The notch frequency will be applied to the preset after the preset is set.

**Warning**

This is only supported on Muse 2014 ( MU_01 ) headbands and then only with presets: MusePreset::PRESET_10 , MusePreset::PRESET_12 and MusePreset::PRESET_14 Under those presets, the only valid frequencies are: NotchFrequency::NOTCH_50HZ NotchFrequency::NOTCH_60HZ

Calling this with NotchFrequency::NOTCH_NONE will do nothing.

This does nothing on Muse 2016 ( MU_02 ) or later headbands and on Muse 2014 ( MU_01 ) headbands with presets MusePreset::PRESET_AB or MusePreset::PRESET_AD

**Parameters**

| | |
|---|---|
| *new_frequency* | The new notch frequency. |

#### 8.38.3.24 set_num_connect_tries()

```
virtual void interaxon::bridge::Muse::set_num_connect_tries (
            int32_t num_tries)  [pure virtual]
```

When LibMuse tries connect to a Muse, it will by default try only once before giving up. This is due to the fact that the user must allow the app permission to access the headband through a UI dialog. Setting this to a number larger than one will cause the UI dialog to pop up multiple times. This function only works for model MU-02 Muses, calling it on a MU-01 will do nothing. This function was added for internal testing purposes, and we do not anticipate that 3rd party clients of the LibMuse library will need to modify this value.

**Threading:** This method is thread safe.

**Parameters**

| | |
|---|---|
| *num_tries* | The number of times to try to connect before giving up. |

#### 8.38.3.25 set_preset()

```
virtual void interaxon::bridge::Muse::set_preset (
            MusePreset preset)  [pure virtual]
```

Changes Muse Headband settings. **Threading:** method is thread-safe. You can call it in the middle of execute operation, but in this case be aware that this operation will interrupt data streaming to set new preset. Data streaming will be restored after that. If method is called before connection is established, the right setting will be passed to device during connection routine.

**Parameters**

| | |
|---|---|
| *preset* | The new preset. |

### 8.38.3.26 set_property()

```
virtual void interaxon::bridge::Muse::set_property (
             const std::string & name,
             const std::string & value)  [pure virtual]
```

Factory use only

### 8.38.3.27 unregister_all_listeners()

```
virtual void interaxon::bridge::Muse::unregister_all_listeners ()  [pure virtual]
```

Unregisters all registered connection listeners and data listeners.

**Threading:** method is thread-safe.

### 8.38.3.28 unregister_connection_listener()

```
virtual void interaxon::bridge::Muse::unregister_connection_listener (
             const std::shared_ptr< MuseConnectionListener > & listener)  [pure virtual]
```

Unregisters connection listeners.

**Threading:** method is thread-safe.

**Parameters**

| | |
|---|---|
| *listener* | The listener to unregister. |

### 8.38.3.29 unregister_data_listener()

```
virtual void interaxon::bridge::Muse::unregister_data_listener (
             const std::shared_ptr< MuseDataListener > & listener,
             MuseDataPacketType type)  [pure virtual]
```

Unregisters a data listener that was registered before.

If the listener was not registered before, then this method does nothing.

**Threading:** method is thread-safe.

**Parameters**

| | |
|---|---|
| *listener* | The listener to unregister. |
| *type* | The type of data packet the listener will stop receiving. |

#### 8.38.3.30 unregister_error_listener()

```
virtual void interaxon::bridge::Muse::unregister_error_listener (
             const std::shared_ptr< MuseErrorListener > & listener)  [pure virtual]
```

Unregisters an info listener that was registered before.

If the listener was not registered before, then this method does nothing.

**Threading:** method is thread-safe.

**Parameters**

| | |
|---|---|
| *listener* | The listener to unregister. |

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

- bridge_muse.h

## 8.39 interaxon::bridge::MuseArtifactPacket Struct Reference

```
#include <bridge_muse_artifact_packet.h>
```

**Public Member Functions**

- MuseArtifactPacket (bool headband_on_, bool blink_, bool jaw_clench_, int64_t timestamp_)

**Public Attributes**

- bool headband_on
- bool blink
- bool jaw_clench
- int64_t timestamp

### 8.39.1 Detailed Description

Provides access to calculated artifacts.

### 8.39.2 Constructor & Destructor Documentation

#### 8.39.2.1 MuseArtifactPacket()

```
interaxon::bridge::MuseArtifactPacket::MuseArtifactPacket (
             bool headband_on_,
             bool blink_,
             bool jaw_clench_,
             int64_t timestamp_)  [inline]
```

## 8.39.3 Member Data Documentation

### 8.39.3.1 blink

```
bool interaxon::bridge::MuseArtifactPacket::blink
```

Eye blinking flag.

**Returns**

`true` if an eye blink was detected. `false` otherwise.

### 8.39.3.2 headband_on

```
bool interaxon::bridge::MuseArtifactPacket::headband_on
```

Flag which represents if the Muse Headband is on a person's head. Headband On detection is one of the first things muse elements algorithm does. If this flag is false you will not see other artifact or band power data.

**Returns**

`true` if the headband is on. `false` otherwise.

### 8.39.3.3 jaw_clench

```
bool interaxon::bridge::MuseArtifactPacket::jaw_clench
```

Jaw clench flag.

**Returns**

`true` if a jaw clench was detected. `false` otherwise.

### 8.39.3.4 timestamp

```
int64_t interaxon::bridge::MuseArtifactPacket::timestamp
```

Microseconds since epoch (usually Jan 1, 1970).

**Returns**

the timestamp of the packet expressed in microseconds since epoch.

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

- bridge_muse_artifact_packet.h

## 8.40 interaxon::bridge::MuseConfiguration Class Reference

```
#include <bridge_muse_configuration.h>
```

### Public Member Functions

- virtual ∼MuseConfiguration ()
- virtual MusePreset get_preset () const =0
- virtual std::string get_headband_name () const =0
- virtual std::string get_microcontroller_id () const =0
- virtual int32_t get_eeg_channel_count () const =0
- virtual int32_t get_afe_gain () const =0
- virtual int32_t get_downsample_rate () const =0
- virtual int32_t get_serout_mode () const =0
- virtual int32_t get_output_frequency () const =0
- virtual int32_t get_adc_frequency () const =0
- virtual bool get_notch_filter_enabled () const =0
- virtual NotchFrequency get_notch_filter () const =0
- virtual int32_t get_accelerometer_sample_frequency () const =0
- virtual bool get_battery_data_enabled () const =0
- virtual bool get_drl_ref_enabled () const =0
- virtual int32_t get_drl_ref_frequency () const =0
- virtual double get_battery_percent_remaining () const =0
- virtual std::string get_bluetooth_mac () const =0
- virtual std::string get_serial_number () const =0
- virtual std::string get_headset_serial_number () const =0
- virtual MuseModel get_model () const =0
- virtual std::string get_license_nonce () const =0
- virtual int32_t get_switch () const =0

### 8.40.1 Detailed Description

Encapsulates the configuration information of the Muse headband.

You must connect to the headband at least once to before this information is available. Once you have connected once the information will remain available, even after you disconnect.

### 8.40.2 Constructor & Destructor Documentation

#### 8.40.2.1 ∼MuseConfiguration()

```
virtual interaxon::bridge::MuseConfiguration::~MuseConfiguration ()  [inline], [virtual]
```

## 8.40.3 Member Function Documentation

### 8.40.3.1 get_accelerometer_sample_frequency()

```
virtual int32_t interaxon::bridge::MuseConfiguration::get_accelerometer_sample_frequency ()
const  [pure virtual]
```

Returns accelerometer sample frequency in Hz.

**Returns**

The frequency of accelerometer samples.

### 8.40.3.2 get_adc_frequency()

```
virtual int32_t interaxon::bridge::MuseConfiguration::get_adc_frequency () const  [pure virtual]
```

The sample frequency of EEG packets (ADC) in Hz. A value of -1 means "unknown".

**Returns**

The ADC frequency of the headband.

### 8.40.3.3 get_afe_gain()

```
virtual int32_t interaxon::bridge::MuseConfiguration::get_afe_gain () const  [pure virtual]
```

Gain to apply to incoming EEG samples.

**Returns**

The gain that should be applied to incoming EEG samples.

### 8.40.3.4 get_battery_data_enabled()

```
virtual bool interaxon::bridge::MuseConfiguration::get_battery_data_enabled () const  [pure
virtual]
```

Provides information about battery data transmission state (on/off).

**Returns**

`true` if the battery data transmission is enabled. `false` if it is not.

#### 8.40.3.5 get_battery_percent_remaining()

```
virtual double interaxon::bridge::MuseConfiguration::get_battery_percent_remaining () const
[pure virtual]
```

Returns battery charge remaining in percent.

**Returns**

The percentage of battery remaining.

#### 8.40.3.6 get_bluetooth_mac()

```
virtual std::string interaxon::bridge::MuseConfiguration::get_bluetooth_mac () const  [pure
virtual]
```

Bluetooth MAC address reported by device.

**Returns**

The MAC address of the headband.

#### 8.40.3.7 get_downsample_rate()

```
virtual int32_t interaxon::bridge::MuseConfiguration::get_downsample_rate () const  [pure
virtual]
```

EEG downsampling rate. A value of -1 means "unknown".

**Returns**

The EEG downsampling rate.

#### 8.40.3.8 get_drl_ref_enabled()

```
virtual bool interaxon::bridge::MuseConfiguration::get_drl_ref_enabled () const  [pure virtual]
```

Provides information about DRL/REF transmission state (on/off).

**Returns**

`true` if the DRL/REF data transmission is enabled. `false` if it is not.

#### 8.40.3.9 get_drl_ref_frequency()

```
virtual int32_t interaxon::bridge::MuseConfiguration::get_drl_ref_frequency () const [pure
virtual]
```

DRL/REF data output frequency in Hz. 0 if DRL/REF is not enabled.

**Returns**

The frequency at which DRL/REF packets are transmitted.

#### 8.40.3.10 get_eeg_channel_count()

```
virtual int32_t interaxon::bridge::MuseConfiguration::get_eeg_channel_count () const [pure
virtual]
```

Number of channels currently active.

**Returns**

The number of EEG channels currently active.

#### 8.40.3.11 get_headband_name()

```
virtual std::string interaxon::bridge::MuseConfiguration::get_headband_name () const [pure
virtual]
```

The name of the headband. This is only available on Muse 2016 ( MU_02 ) or later.

**Returns**

The name of the headband.

#### 8.40.3.12 get_headset_serial_number()

```
virtual std::string interaxon::bridge::MuseConfiguration::get_headset_serial_number () const
[pure virtual]
```

Headset serial number reported by device. Only applicable to Aster.

**Returns**

The headset serial number.

#### 8.40.3.13 get_license_nonce()

```
virtual std::string interaxon::bridge::MuseConfiguration::get_license_nonce () const [pure
virtual]
```

The licensing nonce

**Returns**

The licensing nonce.

#### 8.40.3.14 get_microcontroller_id()

```
virtual std::string interaxon::bridge::MuseConfiguration::get_microcontroller_id () const
[pure virtual]
```

Unique 96 bit ID identifying the microcontroller on the headband. This is only available on Muse 2016 ( MU_02 ) or later.

**Returns**

The id of the microcontroller.

#### 8.40.3.15 get_model()

```
virtual MuseModel interaxon::bridge::MuseConfiguration::get_model () const [pure virtual]
```

The headband model

**Returns**

The model of the headband.

#### 8.40.3.16 get_notch_filter()

```
virtual NotchFrequency interaxon::bridge::MuseConfiguration::get_notch_filter () const [pure
virtual]
```

Provides information about notch filter frequency (50 Hz/60 Hz).

**Returns**

The frequency of the notch filter.

#### 8.40.3.17 get_notch_filter_enabled()

```
virtual bool interaxon::bridge::MuseConfiguration::get_notch_filter_enabled () const [pure
virtual]
```

Provides information about notch filter state (on/off).

**Returns**

`true` if the notch filter is enabled. `false` if it is not.

#### 8.40.3.18 get_output_frequency()

```
virtual int32_t interaxon::bridge::MuseConfiguration::get_output_frequency () const [pure
virtual]
```

The output frequency of EEG packets in Hz.

**Returns**

The output frequency of EEG packets.

#### 8.40.3.19 get_preset()

```
virtual MusePreset interaxon::bridge::MuseConfiguration::get_preset () const [pure virtual]
```

Provides access to current preset.

#### 8.40.3.20 get_serial_number()

```
virtual std::string interaxon::bridge::MuseConfiguration::get_serial_number () const [pure
virtual]
```

Serial number reported by device.

**Returns**

The device serial number.

#### 8.40.3.21 get_serout_mode()

```
virtual int32_t interaxon::bridge::MuseConfiguration::get_serout_mode () const [pure virtual]
```

Serial output mode. The relates to the size of the EEG packets transmitted from the headband. A value of 2 indicates that the packet data is compressed (10 bits). A value of 3 indicates that the packet data is uncompressed (16 bits). A value of -1 means "unknown".

**Returns**

`2` if the data is compressed, `3` if the data is uncompressed or `-1` if this value is unknown.

#### 8.40.3.22 get_switch()

```
virtual int32_t interaxon::bridge::MuseConfiguration::get_switch () const  [pure virtual]
```

Factory use only

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

- bridge_muse_configuration.h

## 8.41 interaxon::bridge::MuseConnectionListener Class Reference

```
#include <bridge_muse_connection_listener.h>
```

**Public Member Functions**

- virtual ∼MuseConnectionListener ()
- virtual void receive_muse_connection_packet (const MuseConnectionPacket &packet, const std::shared_↩ ptr< Muse > &muse)=0

### 8.41.1 Detailed Description

Implement this interface to listen for changes to the current connection status (like connect/disconnect).

**Important:** DO NOT CALL Muse::connect() / Muse::disconnect() directly from a connection listener handler. Either create a new thread and run it from there or use Muse::run_asynchronously() which creates a new thread implicitly.

### 8.41.2 Constructor & Destructor Documentation

#### 8.41.2.1 ∼MuseConnectionListener()

```
virtual interaxon::bridge::MuseConnectionListener::∼MuseConnectionListener ()  [inline], [virtual]
```

### 8.41.3 Member Function Documentation

#### 8.41.3.1 receive_muse_connection_packet()

```
virtual void interaxon::bridge::MuseConnectionListener::receive_muse_connection_packet (
             const MuseConnectionPacket & packet,
             const std::shared_ptr< Muse > & muse)  [pure virtual]
```

Called from the Muse connection thread whenever there is a change in the current connection status of the Muse

**Warning**

It is important that you do not perform any computation intensive tasks in this callback. This would result in significant delays in all the listener callbacks from being called. You should delegate any intensive tasks to another thread or schedule it to run with a delay through handler/scheduler for the platform.

However, you can register/unregister listeners in this callback. All previously registered listeners would still receive callbacks for this current event. On subsequent events, the newly registered listeners will be called. For example, if you had 2 listeners 'A' and 'B' for this event. If, on the callback for listener A, listener A unregisters all listeners and registers a new listener 'C' and then in the callback for listener 'B', you unregister all listeners again and register a new listener 'D'. Then on the subsequent event callback, only listener D's callback will be invoked.

**Parameters**

| | |
|---|---|
| *packet* | The connection packet |
| *muse* | The Muse that sent the connection packet. |

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

- bridge_muse_connection_listener.h

## 8.42 interaxon::bridge::MuseConnectionPacket Struct Reference

```
#include <bridge_muse_connection_packet.h>
```

### Public Member Functions

- MuseConnectionPacket (ConnectionState previous_connection_state_, ConnectionState current_↩
connection_state_)

### Public Attributes

- ConnectionState previous_connection_state
- ConnectionState current_connection_state

### 8.42.1 Detailed Description

Packet provides information about the connection state.

### 8.42.2 Constructor & Destructor Documentation

#### 8.42.2.1 MuseConnectionPacket()

```
interaxon::bridge::MuseConnectionPacket::MuseConnectionPacket (
            ConnectionState previous_connection_state_,
            ConnectionState current_connection_state_)  [inline]
```

### 8.42.3 Member Data Documentation

#### 8.42.3.1 current_connection_state

```
ConnectionState interaxon::bridge::MuseConnectionPacket::current_connection_state
```

Provides access to the current connection status.

**Returns**

The current (new) connection state.

#### 8.42.3.2 previous_connection_state

```
ConnectionState interaxon::bridge::MuseConnectionPacket::previous_connection_state
```

Provides access to the previous connection status.

**Returns**

The prior connection state.

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

- bridge_muse_connection_packet.h

## 8.43 interaxon::bridge::MuseDataListener Class Reference

```
#include <bridge_muse_data_listener.h>
```

**Public Member Functions**

- virtual ∼MuseDataListener ()
- virtual void receive_muse_data_packet (const std::shared_ptr< MuseDataPacket > &packet, const std↩ ::shared_ptr< Muse > &muse)=0
- virtual void receive_muse_artifact_packet (const MuseArtifactPacket &packet, const std::shared_ptr< Muse > &muse)=0

### 8.43.1 Detailed Description

Implement this interface to receive data from the headband. The data may be either data packets or artifact packets. Check out MuseDataPacketType for information about all acceptable data packet types. All packets can be handled in one listener or you can specify a separate listener for every specific packet type.

A packet is only sent if a listener was registered to listen for this packet type.

### 8.43.2 Constructor & Destructor Documentation

#### 8.43.2.1 ∼MuseDataListener()

```
virtual interaxon::bridge::MuseDataListener::∼MuseDataListener ()  [inline], [virtual]
```

### 8.43.3 Member Function Documentation

#### 8.43.3.1 receive_muse_artifact_packet()

```
virtual void interaxon::bridge::MuseDataListener::receive_muse_artifact_packet (
            const MuseArtifactPacket & packet,
            const std::shared_ptr< Muse > & muse)  [pure virtual]
```

Handler method for Muse artifact packets. Packet is sent only if listener was registered to listen for such packets.

**Warning**

> It is important that you do not perform any computation intensive tasks in this callback. This would result in significant delays in all the listener callbacks from being called. You should delegate any intensive tasks to another thread or schedule it to run with a delay through handler/scheduler for the platform.

However, you can register/unregister listeners in this callback. All previously registered listeners would still receive callbacks for this current event. On subsequent events, the newly registered listeners will be called. For example, if you had 2 listeners 'A' and 'B' for this event. If, on the callback for listener A, listener A unregisters all listeners and registers a new listener 'C' and then in the callback for listener 'B', you unregister all listeners again and register a new listener 'D'. Then on the subsequent event callback, only listener D's callback will be invoked.

**Parameters**

| | |
|---|---|
| *packet* | The artifact packet |
| *muse* | The Muse that sent the artifact packet. |

#### 8.43.3.2 receive_muse_data_packet()

```
virtual void interaxon::bridge::MuseDataListener::receive_muse_data_packet (
            const std::shared_ptr< MuseDataPacket > & packet,
            const std::shared_ptr< Muse > & muse)  [pure virtual]
```

Handler method for Muse data packets

**Warning**

> It is important that you do not perform any computation intensive tasks in this callback. This would result in significant delays in all the listener callbacks from being called. You should delegate any intensive tasks to another thread or schedule it to run with a delay through handler/scheduler for the platform.

However, you can register/unregister listeners in this callback. All previously registered listeners would still receive callbacks for this current event. On subsequent events, the newly registered listeners will be called. For example, if you had 2 listeners 'A' and 'B' for this event. If, on the callback for listener A, listener A unregisters all listeners and registers a new listener 'C' and then in the callback for listener 'B', you unregister all listeners again and register a new listener 'D'. Then on the subsequent event callback, only listener D's callback will be invoked.

**Parameters**

| | |
|---|---|
| *packet* | The data packet |
| *muse* | The Muse that sent the data packet. |

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

- bridge_muse_data_listener.h

## 8.44 interaxon::bridge::MuseDataPacket Class Reference

```
#include <bridge_muse_data_packet.h>
```

#### Public Member Functions

- virtual ∼MuseDataPacket ()
- virtual MuseDataPacketType packet_type ()=0
- virtual int64_t timestamp ()=0
- virtual std::vector< double > values ()=0
- virtual int64_t values_size ()=0
- virtual double get_eeg_channel_value (Eeg channel_num)=0
- virtual double get_ppg_channel_value (Ppg channel_num)=0
- virtual double get_ppg_microamps (MuseModel model, double ppg_value)=0
- virtual double get_optics_channel_value (Optics channel_num)=0
- virtual double get_battery_value (Battery b)=0
- virtual double get_accelerometer_value (Accelerometer a)=0
- virtual double get_gyro_value (Gyro g)=0
- virtual double get_magnetometer_value (Magnetometer m)=0
- virtual double get_drl_ref_value (DrlRef drl)=0
- virtual double get_pressure_value (Pressure pressure)=0
- virtual double get_temperature_value ()=0
- virtual double get_uv_value (UltraViolet v)=0

#### Static Public Member Functions

- static std::shared_ptr< MuseDataPacket > make_uninitialized_packet (int64_t capacity)
- static std::shared_ptr< MuseDataPacket > make_packet (MuseDataPacketType type, int64_t timestamp, const std::vector< double > &values)

### 8.44.1 Detailed Description

Contains information received from the headband. Each packet can represent different data: eeg, accelerometer, quantization, etc. Take a look at MuseDataPacketType enum for all possible types

### 8.44.2 Constructor & Destructor Documentation

#### 8.44.2.1 ∼MuseDataPacket()

```
virtual interaxon::bridge::MuseDataPacket::∼MuseDataPacket ()  [inline], [virtual]
```

### 8.44.3 Member Function Documentation

#### 8.44.3.1 get_accelerometer_value()

```
virtual double interaxon::bridge::MuseDataPacket::get_accelerometer_value (
            Accelerometer a)  [pure virtual]
```

Get the Accelerometer value from the packet.

Calling this function does not perform additional allocations and is preferable to using values().

**Parameters**

| | |
|---|---|
| *a* | the Accelerometer value to retrieve (ie. Accelerometer::X ) |

**Returns**

the value requested.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | If this function is called on a packet type that is not type MuseDataPacketType::ACCELEROMETER , LibMuse will throw an exception. Use packet_type() to check the type before calling this function. |

### 8.44.3.2 get_battery_value()

```
virtual double interaxon::bridge::MuseDataPacket::get_battery_value (
            Battery b)  [pure virtual]
```

Get the Battery value from the packet.

Calling this function does not perform additional allocations and is preferable to using values().

**Parameters**

| | |
|---|---|
| *b* | the Battery value to retrieve (ie. Battery::MILLIVOLTS ) |

**Returns**

the value requested.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | If this function is called on a packet type that is not type MuseDataPacketType::BATTERY , LibMuse will throw an exception. Use packet_type() to check the type before calling this function. |

### 8.44.3.3 get_drl_ref_value()

```
virtual double interaxon::bridge::MuseDataPacket::get_drl_ref_value (
            DrlRef drl)  [pure virtual]
```

Get the DrlRef value from the packet.

Calling this function does not perform additional allocations and is preferable to using values().

**Parameters**

| | |
|---|---|
| *drl* | the DrlRef value to retrieve (ie. DrlRef::DRL ) |

**Returns**

the value requested.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | If this function is called on a packet type that is not type MuseDataPacketType::DRL_REF , LibMuse will throw an exception. Use packet_type() to check the type before calling this function. |

#### 8.44.3.4 get_eeg_channel_value()

```
virtual double interaxon::bridge::MuseDataPacket::get_eeg_channel_value (
            Eeg channel_num)  [pure virtual]
```

Get the raw EEG or EEG derived value from the packet. EEG derived value are data that is calculated based on the raw EEG values. Take a look at enum in MuseDataPacketType for values derived from EEG channel. Calling this function does not perform additional allocations and is preferable to using values().

**Parameters**

| | |
|---|---|
| *channel_num* | the Eeg channel to retrieve (ie. Eeg::EEG1) |

**Returns**

the value requested.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | If this function is called on a packet type that is not type MuseDataPacketType::EEG , LibMuse will throw an exception. Use packet_type() to check the type before calling this function. |

#### 8.44.3.5 get_gyro_value()

```
virtual double interaxon::bridge::MuseDataPacket::get_gyro_value (
            Gyro g)  [pure virtual]
```

Get the Gyro value from the packet.

Calling this function does not perform additional allocations and is preferable to using values().

**Parameters**

| | |
|---|---|
| *g* | the Gyro value to retrieve (ie. Gyro::X ) |

**Returns**

the value requested.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | If this function is called on a packet type that is not type MuseDataPacketType::GYRO , LibMuse will throw an exception. Use packet_type() to check the type before calling this function. |

### 8.44.3.6 get_magnetometer_value()

```
virtual double interaxon::bridge::MuseDataPacket::get_magnetometer_value (
            Magnetometer m)  [pure virtual]
```

Get the Magnetometer value from the packet.

Calling this function does not perform additional allocations and is preferable to using values().

**Parameters**

| | |
|---|---|
| *m* | the Magnetometer value to retrieve (ie. Magnetometer::X ) |

**Returns**

the value requested.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | If this function is called on a packet type that is not type MuseDataPacketType::MAGNETOMETER , LibMuse will throw an exception. Use packet_type() to check the type before calling this function. |

### 8.44.3.7 get_optics_channel_value()

```
virtual double interaxon::bridge::MuseDataPacket::get_optics_channel_value (
            Optics channel_num)  [pure virtual]
```

Get the optics value from the packet. Calling this function does not perform additional allocations and is preferable to using values().

**Parameters**

| | |
|---|---|
| *channel_num* | the OPTICS channel to retrieve (ie. Optics::NIR730_LO) |

**Returns**

the value requested.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | If this function is called on a packet type that is not type MuseDataPacketType::OPTICS , LibMuse will throw an exception. Use packet_type() to check the type before calling this function. |

### 8.44.3.8 get_ppg_channel_value()

```
virtual double interaxon::bridge::MuseDataPacket::get_ppg_channel_value (
            Ppg channel_num)  [pure virtual]
```

Get the PPG value from the packet. Calling this function does not perform additional allocations and is preferable to using values().

**Parameters**

| | |
|---|---|
| *channel_num* | the Ppg channel to retrieve (ie. Ppg::AMBIENT) |

**Returns**

the value requested.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | If this function is called on a packet type that is not type MuseDataPacketType::PPG , LibMuse will throw an exception. Use packet_type() to check the type before calling this function. |

### 8.44.3.9 get_ppg_microamps()

```
virtual double interaxon::bridge::MuseDataPacket::get_ppg_microamps (
            MuseModel model,
            double ppg_value)  [pure virtual]
```

Get PPG microamps from the raw PPG value.

**Parameters**

| | |
|---|---|
| *model* | the Muse model to which the conversion is applied. |
| *ppg_value* | the raw PPG channel value. |

**Returns**

the PPG value in microamps.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | |

### 8.44.3.10 get_pressure_value()

```
virtual double interaxon::bridge::MuseDataPacket::get_pressure_value (
            Pressure pressure)  [pure virtual]
```

Get the Pressure value from the packet.

Calling this function does not perform additional allocations and is preferable to using values().

**Parameters**

| | |
|---|---|
| *pressure* | the Pressure value to retrieve (ie. Pressure::AVERAGED ) |

**Returns**

the value requested.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | If this function is called on a packet type that is not type MuseDataPacketType::PRESSURE , LibMuse will throw an exception. Use packet_type() to check the type before calling this function. |

### 8.44.3.11 get_temperature_value()

```
virtual double interaxon::bridge::MuseDataPacket::get_temperature_value ()  [pure virtual]
```

Get the temperature value from the packet.

Calling this function does not perform additional allocations and is preferable to using values().

**Returns**

the value requested.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | If this function is called on a packet type that is not type MuseDataPacketType::TEMPERATURE , LibMuse will throw an exception. Use packet_type() to check the type before calling this function. |

### 8.44.3.12 get_uv_value()

```
virtual double interaxon::bridge::MuseDataPacket::get_uv_value (
             UltraViolet v)  [pure virtual]
```

Get the UltraViolet value from the packet.

Calling this function does not perform additional allocations and is preferable to using values().

**Parameters**

| | |
|---|---|
| *v* | the UltraViolet value to retrieve (ie. UltraViolet::UV_A ) |

**Returns**

the value requested.

**Exceptions**

| | |
|---|---|
| *SIGABRT* | If this function is called on a packet type that is not type MuseDataPacketType::ULTRA_VIOLET , LibMuse will throw an exception. Use packet_type() to check the type before calling this function. |

### 8.44.3.13 make_packet()

```
static std::shared_ptr< MuseDataPacket > interaxon::bridge::MuseDataPacket::make_packet (
             MuseDataPacketType type,
             int64_t timestamp,
             const std::vector< double > & values)  [static]
```

Create a new packet with the given contents.

**Parameters**

| | |
|---|---|
| *type* | the type of packet to create |
| *timestamp* | the timestamp of the packet |
| *values* | the data the packet contains. |

### 8.44.3.14 make_uninitialized_packet()

```
static std::shared_ptr< MuseDataPacket > interaxon::bridge::MuseDataPacket::make_uninitialized↩
_packet (
             int64_t capacity) [static]
```

Create a new packet with reserved capacity but unspecified contents.

**Parameters**

| | |
|---|---|
| *capacity* | the number of data entries to reserve. |

### 8.44.3.15 packet_type()

```
virtual MuseDataPacketType interaxon::bridge::MuseDataPacket::packet_type () [pure virtual]
```

Specifies what kind of values are stored in the packet.

**Returns**

the type of packet.

### 8.44.3.16 timestamp()

```
virtual int64_t interaxon::bridge::MuseDataPacket::timestamp () [pure virtual]
```

Microseconds since epoch (usually Jan 1, 1970).

**Returns**

the timestamp of the packet expressed in microseconds since epoch.

### 8.44.3.17 values()

```
virtual std::vector< double > interaxon::bridge::MuseDataPacket::values () [pure virtual]
```

Raw packet data as an array. The size of this array and the meaning of its elements depend on the packet type. See the documentation for MuseDataPacketType for details.

Note that this method causes new memory to be allocated for an array of boxed double values. If this impacts your application's performance, you should use get_xxx_value() methods instead.

**Deprecated** Use MuseDataPacket::get_accelerometer_value() , MuseDataPacket::get_battery_value() , MuseDataPacket::get_drl_ref_value() , MuseDataPacket::get_eeg_channel_value() , MuseDataPacket::get_ppg_channe , MuseDataPacket::get_gyro_value() instead.

#### 8.44.3.18 values_size()

```
virtual int64_t interaxon::bridge::MuseDataPacket::values_size ()  [pure virtual]
```

Get the number of values in this packet.

**Returns**

the number of data values in this packet.

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

- bridge_muse_data_packet.h

## 8.45 interaxon::bridge::MuseErrorListener Class Reference

```
#include <bridge_muse_error_listener.h>
```

**Public Member Functions**

- virtual ∼MuseErrorListener ()
- virtual void receive_error (const Error &error, const std::shared_ptr< Muse > &muse)=0

### 8.45.1 Detailed Description

Listens for Muse Error packets.

### 8.45.2 Constructor & Destructor Documentation

#### 8.45.2.1 ∼MuseErrorListener()

```
virtual interaxon::bridge::MuseErrorListener::∼MuseErrorListener ()  [inline], [virtual]
```

### 8.45.3 Member Function Documentation

#### 8.45.3.1 receive_error()

```
virtual void interaxon::bridge::MuseErrorListener::receive_error (
             const Error & error,
             const std::shared_ptr< Muse > & muse)  [pure virtual]
```

Handler method for Muse error packets. Packet is sent iff muse receives an exception.

Generally if an error occurred, LogListener::receive_log() will be called with a LogPacket of "error" severity. It is therefore not usually necessary or advisable to print log messages for errors received by this method. Rather, this handler should take actions like alerting users or deciding to end a session.

**Warning**

It is important that you do not perform any computation intensive tasks in this callback. This would result in significant delays in all the listener callbacks from being called. You should delegate any intensive tasks to another thread or schedule it to run with a delay through handler/scheduler for the platform.

However, you can register/unregister listeners in this callback. All previously registered listeners would still receive callbacks for this current event. On subsequent events, the newly registered listeners will be called. For example, if you had 2 listeners 'A' and 'B' for this event. If, on the callback for listener A, listener A unregisters all listeners and registers a new listener 'C' and then in the callback for listener 'B', you unregister all listeners again and register a new listener 'D'. Then on the subsequent event callback, only listener D's callback will be invoked.

**Parameters**

| | |
|---|---|
| *error* | The error encountered. |
| *muse* | The Muse that generated the error. |

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

- bridge_muse_error_listener.h

## 8.46 interaxon::bridge::MuseFile Class Reference

```
#include <bridge_muse_file.h>
```

**Public Member Functions**

- virtual ∼MuseFile ()
- virtual bool open (bool for_writing)=0
- virtual bool write (const std::vector< uint8_t > &buffer)=0
- virtual std::vector< uint8_t > read (int32_t length)=0
- virtual bool close (bool for_writing)=0

### 8.46.1 Detailed Description

A File interface. Normally you will not need to use this class directly as you can get a MuseFileReader or MuseFileWriter using MuseFileFactory .
If you provide your own implementation of this class, you should provide a way to specify the path to the file you want to access in a platform specific way.

**Threading:** MuseFile is not thread safe by default, but MuseFileWriter provides the necessary synchronization for its usage. If you are using MuseFile outside of that context, make sure the calls to the MuseFile::open() , MuseFile::write() and MuseFile::close() functions are thread-safe. Specifically, make sure you do not call these methods on the main UI thread. Please use an async task instead in order to avoid crashes.

### 8.46.2 Constructor & Destructor Documentation

#### 8.46.2.1 ∼MuseFile()

```
virtual interaxon::bridge::MuseFile::∼MuseFile ()  [inline], [virtual]
```

### 8.46.3 Member Function Documentation

#### 8.46.3.1 close()

```
virtual bool interaxon::bridge::MuseFile::close (
            bool for_writing)  [pure virtual]
```

Closes opened file.
This method is called when MuseFileWriter or MuseFileReader is destroyed, or when close is called on either class.

**Parameters**

|  |  |
|---|---|
| *for_writing* | `true` if the file should be closed for writing. `false` if for reading. |

**Returns**

`true` if file is closed properly, `false` otherwise.

### 8.46.3.2 open()

```
virtual bool interaxon::bridge::MuseFile::open (
            bool for_writing)  [pure virtual]
```

Opens/creates the file. You can open same file for both writing and reading if you really (!) have to do so. Provide synchronization in this case. This method is called automatically when MuseFileWriter or MuseFileReader is created.

**Parameters**

|  |  |
|---|---|
| *for_writing* | `true` if the file should be opened for writing. `false` if for reading. |

**Returns**

`true` when file is opened successfully, `false` otherwise.

### 8.46.3.3 read()

```
virtual std::vector< uint8_t > interaxon::bridge::MuseFile::read (
            int32_t length)  [pure virtual]
```

Reads "length" bytes from file starting from current pointer position.

**Parameters**

|  |  |
|---|---|
| *length* | The number of bytes to read. |

**Returns**

A byte array containing the data read. An empty buffer is returned if the file is empty or if the end of file was reached.

### 8.46.3.4 write()

```
virtual bool interaxon::bridge::MuseFile::write (
            const std::vector< uint8_t > & buffer)  [pure virtual]
```

Writes buffer to file.
This method is called when you flush MuseFileWriter

**Parameters**

| | |
|---|---|
| *buffer* | The data to write. |

**Returns**

`true` when buffer is written to successfully, `false` otherwise.

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

- bridge_muse_file.h

## 8.47 interaxon::bridge::MuseFileFactory Class Reference

```
#include <muse_file_factory.h>
```

### Static Public Member Functions

- static std::shared_ptr< MuseFileWriter > get_muse_file_writer (const std::string &file_path)
- static std::shared_ptr< MuseFileReader > get_muse_file_reader (const std::string &file_path)
- static std::shared_ptr< MuseFile > get_muse_file (const std::string &file_path)

### 8.47.1 Detailed Description

Creates MuseFileWriter, MuseFileReader and MuseFile objects.

### 8.47.2 Member Function Documentation

#### 8.47.2.1 get_muse_file()

```
static std::shared_ptr< MuseFile > interaxon::bridge::MuseFileFactory::get_muse_file (
            const std::string & file_path)  [static]
```

Creates and returns MuseFile object, which uses Interaxon's implementation.

**Parameters**

| | |
|---|---|
| *file_path* | The absolute path of the file. |

**Returns**

MuseFile

#### 8.47.2.2 get_muse_file_reader()

```
static std::shared_ptr< MuseFileReader > interaxon::bridge::MuseFileFactory::get_muse_file_↩
reader (
            const std::string & file_path)  [static]
```

Creates and returns MuseFileReader object based on provided path. Interaxon's MuseFile implementation is used in this case.

**Parameters**

| | |
|---|---|
| *file_path* | The absolute path of the file to read. |

**Returns**

MuseFileReader

### 8.47.2.3 get_muse_file_writer()

```
static std::shared_ptr< MuseFileWriter > interaxon::bridge::MuseFileFactory::get_muse_file_↩
writer (
            const std::string & file_path)  [static]
```

Creates and returns MuseFileWriter object based on provided path. Interaxon's MuseFile implementation is used in this case.

Note that upon creation of MuseFileWriter, an Annotation is automatically written out to the file. The annotation contains the app's name and version and libmuse version. If app's name and version can not be determined, they will be empty strings.

**Parameters**

| | |
|---|---|
| *file_path* | The absolute path of the file to write. |

**Returns**

MuseFileWriter

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

- muse_file_factory.h

## 8.48 interaxon::bridge::MuseFileReader Class Reference

```
#include <bridge_muse_file_reader.h>
```

### Public Member Functions

- virtual ~MuseFileReader ()
- virtual bool open ()=0
- virtual bool close ()=0
- virtual Result goto_next_message ()=0
- virtual MessageType get_message_type ()=0
- virtual int32_t get_message_id ()=0
- virtual int64_t get_message_timestamp ()=0
- virtual AnnotationData get_annotation ()=0
- virtual std::shared_ptr< MuseConfiguration > get_configuration ()=0
- virtual std::shared_ptr< MuseVersion > get_version ()=0
- virtual ComputingDeviceConfiguration get_computing_device_configuration ()=0
- virtual DspData get_dsp ()=0
- virtual std::shared_ptr< MuseDataPacket > get_data_packet ()=0
- virtual MuseArtifactPacket get_artifact_packet ()=0

**Static Public Member Functions**

- static std::shared_ptr< MuseFileReader > get_file_reader (const std::shared_ptr< MuseFile > &file)

### 8.48.1 Detailed Description

Reads a .muse file formatted according to the Muse protobuf schema.

MuseFileReader is very similar to ReaderMuse as both classes allow you to parse a file and generate data packets from it. In most instances, ReaderMuse is the better choice at it abstracts the details of the parsing of the file and allows you to handle the data the same way that you would if you were connected to an actual Muse . Use MuseFileReader if you want finer control of the parsing of the file.

When the file reader is created it always starts reading from the beginning of the file to the end.

**Threading:** It is NOT thread safe, you have to provide your own synchronization mechanism if you plan to calls methods of this class from different threads. Make sure you do not call these methods from the main UI thread. Please use an async task instead to avoid crashes.

### 8.48.2 Constructor & Destructor Documentation

#### 8.48.2.1 ~MuseFileReader()

```
virtual interaxon::bridge::MuseFileReader::~MuseFileReader ()  [inline], [virtual]
```

### 8.48.3 Member Function Documentation

#### 8.48.3.1 close()

```
virtual bool interaxon::bridge::MuseFileReader::close ()  [pure virtual]
```

Closes the file. Calls MuseFile::close() You don't have to call close explicitly unless you want to close file immediately. close is called automatically, when the file reader object is destroyed.

#### 8.48.3.2 get_annotation()

```
virtual AnnotationData interaxon::bridge::MuseFileReader::get_annotation ()  [pure virtual]
```

Returns annotation data at the current position in the file.

**Returns**

The annotation data at the current position in the file.

**Exceptions**

| | |
|---|---|
| *IncorrectMessageType* | If current message type is not MessageType::ANNOTATION |

#### 8.48.3.3 get_artifact_packet()

```
virtual MuseArtifactPacket interaxon::bridge::MuseFileReader::get_artifact_packet ()  [pure
virtual]
```

Returns the muse artifact packet data at the current position in the file.

**Returns**

The muse artifact packet at the current position in the file.

**Exceptions**

| | |
|---|---|
| *IncorrectMessageType* | If current message type is not MessageType::ARTIFACT , |

#### 8.48.3.4 get_computing_device_configuration()

```
virtual ComputingDeviceConfiguration interaxon::bridge::MuseFileReader::get_computing_device↩
_configuration ()  [pure virtual]
```

Returns computing device configuration data at the current position in the file.

**Returns**

The device configuration data at the current position in the file.

**Exceptions**

| | |
|---|---|
| *IncorrectMessageType* | If current message type is not MessageType::COMPUTING_DEVICE |

#### 8.48.3.5 get_configuration()

```
virtual std::shared_ptr< MuseConfiguration > interaxon::bridge::MuseFileReader::get_configuration
()  [pure virtual]
```

Returns muse configuration data at the current position in the file.

**Returns**

The configuration data at the current position in the file.

**Exceptions**

| | |
|---|---|
| *IncorrectMessageType* | If current message type is not MessageType::CONFIGURATION |

#### 8.48.3.6 get_data_packet()

```
virtual std::shared_ptr< MuseDataPacket > interaxon::bridge::MuseFileReader::get_data_packet
()  [pure virtual]
```

Returns the muse data packet data at the current position in the file. Use this method to get EEG, Accelerometer, Battery and Quantization packets.

**Returns**

The muse data packet data at the current position in the file.

**Exceptions**

| | |
|---|---|
| *IncorrectMessageType* | If current message type is not one of: MessageType::ACCELEROMETER , MessageType::ACC_DROPPED , MessageType::BATTERY , MessageType::EEG , MessageType::EEG_DROPPED , MessageType::GYRO , MessageType::MUSE_ELEMENTS or MessageType::QUANTIZATION |

#### 8.48.3.7 get_dsp()

```
virtual DspData interaxon::bridge::MuseFileReader::get_dsp ()  [pure virtual]
```

Returns the dsp data at the current position in the file.

**Returns**

The dsp data at the current position in the file.

**Exceptions**

| | |
|---|---|
| *IncorrectMessageType* | If current message type is not MessageType::DSP |

#### 8.48.3.8 get_file_reader()

```
static std::shared_ptr< MuseFileReader > interaxon::bridge::MuseFileReader::get_file_reader (
            const std::shared_ptr< MuseFile > & file)  [static]
```

Returns an instance of file reader. Automatically opens a file.

**Returns**

a MuseFileReader instance.

#### 8.48.3.9 get_message_id()

```
virtual int32_t interaxon::bridge::MuseFileReader::get_message_id ()  [pure virtual]
```

Returns the id of the message at the current position in the file or `-1` if the id isn't found in the protobuf specification.

**Returns**

The id of the message at the current position in the file or `-1` if the id isn't found in the protobuf specification.

#### 8.48.3.10 get_message_timestamp()

```
virtual int64_t interaxon::bridge::MuseFileReader::get_message_timestamp ()  [pure virtual]
```

Returns the timestamp of the message at the current position in the file.

**Returns**

The timestamp of the message.

#### 8.48.3.11 get_message_type()

```
virtual MessageType interaxon::bridge::MuseFileReader::get_message_type ()  [pure virtual]
```

Returns the type of message at the current position in the file.

**Returns**

The type of message at the current position in the file.

#### 8.48.3.12 get_version()

```
virtual std::shared_ptr< MuseVersion > interaxon::bridge::MuseFileReader::get_version ()
[pure virtual]
```

Returns muse version data at the current position in the file.

**Returns**

The version data at the current position in the file.

**Exceptions**

| | |
|---|---|
| *IncorrectMessageType* | If current message type is not MessageType::VERSION |

#### 8.48.3.13 goto_next_message()

```
virtual Result interaxon::bridge::MuseFileReader::goto_next_message ()  [pure virtual]
```

Reads the next message in the protobuf stream.

**Returns**

The Result object with the status of the read operation.

#### 8.48.3.14 open()

```
virtual bool interaxon::bridge::MuseFileReader::open ()  [pure virtual]
```

Opens an existing file Use this method if you explicitly closed file and want to open it again. Calls MuseFile::open()

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

- bridge_muse_file_reader.h

## 8.49 interaxon::bridge::MuseFileWriter Class Reference

```
#include <bridge_muse_file_writer.h>
```

### Public Member Functions

- virtual ∼MuseFileWriter ()
- virtual bool open ()=0
- virtual bool is_open ()=0
- virtual bool close ()=0
- virtual void discard_buffered_packets ()=0
- virtual bool flush ()=0
- virtual int32_t get_bufferred_messages_count ()=0
- virtual int32_t get_buffered_messages_size ()=0
- virtual int64_t get_total_bytes_written () const =0
- virtual void add_artifact_packet (int32_t id, const MuseArtifactPacket &packet)=0
- virtual void add_data_packet (int32_t id, const std::shared_ptr< MuseDataPacket > &packet)=0
- virtual void add_annotation_string (int32_t id, const std::string &annotation)=0
- virtual void add_annotation (int32_t id, const AnnotationData &annotation)=0
- virtual void add_configuration (int32_t id, const std::shared_ptr< MuseConfiguration > &configuration)=0
- virtual void add_version (int32_t id, const std::shared_ptr< MuseVersion > &version)=0
- virtual void add_computing_device_configuration (int32_t id, const ComputingDeviceConfiguration &configuration)=0
- virtual void add_dsp (int32_t id, const DspData &dsp)=0
- virtual void set_timestamp_mode (TimestampMode mode)=0
- virtual void set_timestamp (int64_t timestamp)=0

### Static Public Member Functions

- static std::shared_ptr< MuseFileWriter > get_file_writer (const std::shared_ptr< MuseFile > &file)

### 8.49.1 Detailed Description

This class manages saving Muse packets and other data into a file, which can later be read and replayed by MusePlayer. Google Protobuf is used to store the information. For better control, data is not written to the file immediately, but stored in a buffer instead. The buffer then can be flushed to a file or discarded.

Note that upon creation of a MuseFileWriter , an Annotation and the ComputingDeviceConfiguration are automatically written out to the file. The annotation contains the app's name and version and libmuse version. If app's name and version can not be determined, they will be empty strings. The ComputingDeviceConfiguration contains hardware information for the computing device. If any information fields can not be determined, they will be empty strings.

**Threading:** It is thread safe, so you can call write()/get()/flush()/discard() operations from different threads. However, make sure you do not call these methods from the main UI thread. Please use an async task instead to avoid crashes.

**Warning**

> If you don't call flush() method, messages will be accumulated in memory and eventually you will get memory overflow error. It's library client responsibility to call flush() or discardBufferedPackets() and to clean the memory. You can use helper methods to get how many messages are currently stored in the buffer and how much memory is used.

### 8.49.2 Constructor & Destructor Documentation

#### 8.49.2.1 ∼MuseFileWriter()

```
virtual interaxon::bridge::MuseFileWriter::∼MuseFileWriter ()  [inline], [virtual]
```

### 8.49.3 Member Function Documentation

#### 8.49.3.1 add_annotation()

```
virtual void interaxon::bridge::MuseFileWriter::add_annotation (
            int32_t id,
            const AnnotationData & annotation)  [pure virtual]
```

Adds annotation data structure to the buffer. Similar to add_annotation_string() , but this method allows you to extra fields. Empty fields in annotation struct won't be added to protobuf. If annotation.data field is empty, method returns immediately.

**Parameters**

| | |
|---|---|
| *id* | The id of the device generating the annotation. This is arbitrary and is used to differentiate data from multiple devices in a single file. |
| *annotation* | The annotation data to add. |

#### 8.49.3.2 add_annotation_string()

```
virtual void interaxon::bridge::MuseFileWriter::add_annotation_string (
            int32_t id,
            const std::string & annotation)  [pure virtual]
```

Adds an annotation string to the buffer. It may be useful if you want to log specific events. This is a simplified version of add_annotation() Use this method if you want to add one string to protobuf. Use add_annotation() for more advanced options. This function does nothing if annotation is empty.

**Parameters**

| | |
|---|---|
| *id* | The id of the device generating the packet. This is arbitrary and is used to differentiate data from multiple devices in a single file. |
| *annotation* | The annotation string to add. |

### 8.49.3.3 add_artifact_packet()

```
virtual void interaxon::bridge::MuseFileWriter::add_artifact_packet (
             int32_t id,
             const MuseArtifactPacket & packet)  [pure virtual]
```

Adds MuseArtifactPacket to the buffer.

**Parameters**

| | |
|---|---|
| *id* | The id of the device generating the packet. This is arbitrary and is used to differentiate data from multiple devices in a single file. |
| *packet* | The artifact packet to add. |

### 8.49.3.4 add_computing_device_configuration()

```
virtual void interaxon::bridge::MuseFileWriter::add_computing_device_configuration (
             int32_t id,
             const ComputingDeviceConfiguration & configuration)  [pure virtual]
```

Adds information about the running device to the buffer.

**Parameters**

| | |
|---|---|
| *id* | The id of the device saving the configuration. This is arbitrary and is used to differentiate data from multiple devices in a single file. |
| *configuration* | The device configuration data to add. |

### 8.49.3.5 add_configuration()

```
virtual void interaxon::bridge::MuseFileWriter::add_configuration (
             int32_t id,
             const std::shared_ptr< MuseConfiguration > & configuration)  [pure virtual]
```

Adds MuseConfiguration to the buffer.

**Parameters**

| | |
|---|---|
| *id* | The id of the device saving the configuration. This is arbitrary and is used to differentiate data from multiple devices in a single file. |
| *configuration* | The configuration data to add. |

#### 8.49.3.6 add_data_packet()

```
virtual void interaxon::bridge::MuseFileWriter::add_data_packet (
            int32_t id,
            const std::shared_ptr< MuseDataPacket > & packet)  [pure virtual]
```

Adds MuseDataPacket to the buffer. All current packets are supported.

**Parameters**

| | |
|---|---|
| *id* | The id of the device generating the packet. This is arbitrary and is used to differentiate data from multiple devices in a single file. |
| *packet* | The data packet to add. |

### 8.49.3.7 add_dsp()

```
virtual void interaxon::bridge::MuseFileWriter::add_dsp (
            int32_t id,
            const DspData & dsp)  [pure virtual]
```

With this method you can save your custom data.

**Parameters**

| | |
|---|---|
| *id* | The id of the device saving the data. This is arbitrary and is used to differentiate data from multiple devices in a single file. |
| *dsp* | The data to store. |

### 8.49.3.8 add_version()

```
virtual void interaxon::bridge::MuseFileWriter::add_version (
            int32_t id,
            const std::shared_ptr< MuseVersion > & version)  [pure virtual]
```

Adds MuseVersion to the buffer.

**Parameters**

| | |
|---|---|
| *id* | The id of the device saving the version. This is arbitrary and is used to differentiate data from multiple devices in a single file. |
| *version* | The version data to add. |

### 8.49.3.9 close()

```
virtual bool interaxon::bridge::MuseFileWriter::close ()  [pure virtual]
```

Closes the file. Calls MuseFile::close()

You don't have to call close explicitly unless you want to close file immediately. close is called automatically, when the file writer object is destroyed.

**Returns**

`false` if the file could not be closed for any reason. `true` otherwise.

#### 8.49.3.10 discard_buffered_packets()

```
virtual void interaxon::bridge::MuseFileWriter::discard_buffered_packets ()  [pure virtual]
```

Removes all saved messages from the memory.

#### 8.49.3.11 flush()

```
virtual bool interaxon::bridge::MuseFileWriter::flush ()  [pure virtual]
```

Flashes saved messages to the disk. Calls MuseFile::write()

**Returns**

`false` if can not write to file for any reason. `true` otherwise.

#### 8.49.3.12 get_buffered_messages_size()

```
virtual int32_t interaxon::bridge::MuseFileWriter::get_buffered_messages_size ()  [pure virtual]
```

Returns the size of saved messages in bytes.

**Returns**

the size of the saved messages in bytes.

#### 8.49.3.13 get_bufferred_messages_count()

```
virtual int32_t interaxon::bridge::MuseFileWriter::get_bufferred_messages_count ()  [pure
virtual]
```

Returns number of saved messages

**Returns**

the number of saved messages.

#### 8.49.3.14 get_file_writer()

```
static std::shared_ptr< MuseFileWriter > interaxon::bridge::MuseFileWriter::get_file_writer (
             const std::shared_ptr< MuseFile > & file)  [static]
```

Returns an instance of file writer. Automatically opens a file.

**Returns**

an instance of the file writer.

### 8.49.3.15 get_total_bytes_written()

```
virtual int64_t interaxon::bridge::MuseFileWriter::get_total_bytes_written () const  [pure
virtual]
```

Returns the total number of bytes written to the file.

**Returns**

the total number of bytes written to the file.

### 8.49.3.16 is_open()

```
virtual bool interaxon::bridge::MuseFileWriter::is_open ()  [pure virtual]
```

Checks if the file writer is open.

**Returns**

`true` when file writer is open, `false` otherwise.

### 8.49.3.17 open()

```
virtual bool interaxon::bridge::MuseFileWriter::open ()  [pure virtual]
```

Opens a file if it exists or creates a new one. If file already exists, all new messages will be appended at the end. Use this method if you explicitly closed file and want to open it again. Calls MuseFile::open()

**Returns**

`false` if the file could not be opened for any reason. `true` otherwise.

### 8.49.3.18 set_timestamp()

```
virtual void interaxon::bridge::MuseFileWriter::set_timestamp (
            int64_t timestamp)  [pure virtual]
```

Set the timestamp for the next message to be written.

This only does anything in explicit timestamp mode. In explicit mode, the timestamp set by set_timestamp applies to the next message and all subsequent messages. If set_timestamp is not called, then the timestamp is 0.

In all other modes, calling set_timestamp causes an assert failure – the app crashes in debug mode, and it is a noop in release mode.

The timestamp is in microseconds since some epoch (usually Jan 1, 1970).

**Parameters**

| | |
|---|---|
| *timestamp* | The time to use for the timestamp in microseconds. |

#### 8.49.3.19 set_timestamp_mode()

```
virtual void interaxon::bridge::MuseFileWriter::set_timestamp_mode (
            TimestampMode mode)  [pure virtual]
```

Set the timestamp mode.

By default, the mode is TimestampMode::LEGACY

**Parameters**

| | |
|---|---|
| *mode* | The timestamp mode to set. |

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

- bridge_muse_file_writer.h

## 8.50 interaxon::bridge::MuseListener Class Reference

```
#include <bridge_muse_listener.h>
```

**Public Member Functions**

- virtual ∼MuseListener ()
- virtual void muse_list_changed ()=0

### 8.50.1 Detailed Description

Implement this interface to receive callbacks whenever a Muse headband is discovered after calling MuseManager::start_listening()

### 8.50.2 Constructor & Destructor Documentation

#### 8.50.2.1 ∼MuseListener()

```
virtual interaxon::bridge::MuseListener::∼MuseListener ()  [inline], [virtual]
```

### 8.50.3 Member Function Documentation

#### 8.50.3.1 muse_list_changed()

```
virtual void interaxon::bridge::MuseListener::muse_list_changed ()  [pure virtual]
```

Called when the list of Muses detected by MuseManager has changed.

- You will receive 1 callback for each headband discovered and added to the list. For Muse 2014 ( MU_01 ) and Muse 2016 ( MU_02 ) or later headbands, there will be a callback for each headband paired with the system even if the headband is powered off.
- You will receive 1 callback for each removal operation. If more than 1 headband is removed as the result of a single removal operation, you will only receive 1 callback instead of 1 callback for each headband.

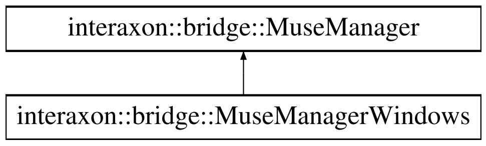
The documentation for this class was generated from the following file:

- bridge_muse_listener.h

## 8.51 interaxon::bridge::MuseManager Class Reference

```
#include <bridge_muse_manager.h>
```

Inheritance diagram for interaxon::bridge::MuseManager:


interaxon::bridge::MuseManager
interaxon::bridge::MuseManagerWindows

### Public Member Functions

- virtual ∼MuseManager ()
- virtual std::vector< std::shared_ptr< Muse > > get_muses ()=0
- virtual void start_listening ()=0
- virtual void stop_listening ()=0
- virtual void set_muse_listener (const std::shared_ptr< MuseListener > &listener)=0
- virtual AdvertisingStats get_advertising_stats (const std::shared_ptr< Muse > &m)=0
- virtual void reset_advertising_stats ()=0
- virtual void remove_from_list_after (int64_t time)=0

### Static Public Attributes

- static constexpr int64_t DEFAULT_REMOVE_FROM_LIST_AFTER = 30

### 8.51.1 Detailed Description

Discovers the available Muse headbands that this device can connect to.

To connect to a headband, first call MuseManager::set_muse_listener() to receive callbacks when a headband is discovered. Then call MuseManager::start_listening() When MuseManager detects a headband you will receive a MuseListener::muse_list_changed() callback. You can then call MuseManager::get_muses() followed by the appropriate function on the Muse to connect to the headband.

**See also**

Muse

MuseListener

### 8.51.2 Constructor & Destructor Documentation

#### 8.51.2.1 ∼MuseManager()

```
virtual interaxon::bridge::MuseManager::∼MuseManager () [inline], [virtual]
```

### 8.51.3 Member Function Documentation

#### 8.51.3.1 get_advertising_stats()

```
virtual AdvertisingStats interaxon::bridge::MuseManager::get_advertising_stats (
            const std::shared_ptr< Muse > & m) [pure virtual]
```

Returns information about the advertising packets seen by LibMuse. This is intended for internal usage at Interaxon and is currently only implemented on Android for Muse 2016 ( MU_02 ) or later. This function does nothing on iOS or Windows.

#### 8.51.3.2 get_muses()

```
virtual std::vector< std::shared_ptr< Muse > > interaxon::bridge::MuseManager::get_muses ()
[pure virtual]
```

Returns all currently available Muse headbands.

The returned list is sorted lexicographically by Muse name.

Note that until you start listening, this list will be empty.

**Returns**

The list of available muses.

#### 8.51.3.3 remove_from_list_after()

```
virtual void interaxon::bridge::MuseManager::remove_from_list_after (
            int64_t time) [pure virtual]
```

MuseManager will automatically remove a Muse 2016 ( MU_02 ) or later headband from the list of Muses if it has not received some type of communication from the headband within this time period.

By default this is MuseManager::DEFAULT_REMOVE_FROM_LIST_AFTER To have Muse 2016 ( MU_02 ) or later headbands remain in the list until the next call to MuseManager::start_listening() set the time to 0.

**Parameters**

| | |
|---|---|
| *time* | Specified in seconds, the time to wait before removing a headband from the list. If set to 0, headbands will not be removed automatically. |

### 8.51.3.4 reset_advertising_stats()

```
virtual void interaxon::bridge::MuseManager::reset_advertising_stats ()  [pure virtual]
```

Erases all previously stored information about advertising packets. This is intended for internal usage at Interaxon and is currently only implemented Muse 2016 ( MU_02 ) or later. This function does nothing on iOS.

### 8.51.3.5 set_muse_listener()

```
virtual void interaxon::bridge::MuseManager::set_muse_listener (
             const std::shared_ptr< MuseListener > & listener)  [pure virtual]
```

Set the listener to be notified on changes to the list of Muses.

**Parameters**

| | |
|---|---|
| *listener* | The listener that will receive the callback when a Muse is discovered. |

### 8.51.3.6 start_listening()

```
virtual void interaxon::bridge::MuseManager::start_listening ()  [pure virtual]
```

Start listening for any possible Muse devices.

As Muses are discovered, they are added to the list returned by MuseManager::get_muses()

Listening is a fairly heavyweight operation, so it should be disabled when it is not needed, e.g. after a device has been found. Specifically, you *must* call stop_listening() before you try to connect to a Muse you have previously discovered.

### 8.51.3.7 stop_listening()

```
virtual void interaxon::bridge::MuseManager::stop_listening ()  [pure virtual]
```

Stop listening for Muse devices.

Stops receiving advertising packets or modifying the list of Muses returned by MuseManager::get_muses() If this is called while not scanning, it is a no-op.

### 8.51.4 Member Data Documentation

#### 8.51.4.1 DEFAULT_REMOVE_FROM_LIST_AFTER

```
int64_t interaxon::bridge::MuseManager::DEFAULT_REMOVE_FROM_LIST_AFTER = 30  [static], [constexpr]
```

The default time in seconds after which a headband will be removed from the list of muses if MuseManager has had no contact with it.

This has a value of 30 seconds.

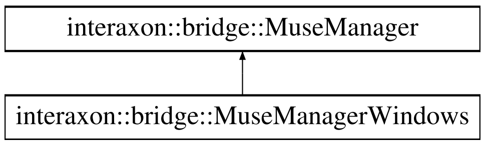
The documentation for this class was generated from the following file:

- bridge_muse_manager.h

## 8.52 interaxon::bridge::MuseManagerWindows Class Reference

```
#include <muse_manager_windows.h>
```

Inheritance diagram for interaxon::bridge::MuseManagerWindows:


interaxon::bridge::MuseManager
interaxon::bridge::MuseManagerWindows

#### Public Member Functions

- virtual void set_recorder_info (const std::string &name, const std::string &version)=0

### Public Member Functions inherited from interaxon::bridge::MuseManager

- virtual ~MuseManager ()
- virtual std::vector< std::shared_ptr< Muse > > get_muses ()=0
- virtual void start_listening ()=0
- virtual void stop_listening ()=0
- virtual void set_muse_listener (const std::shared_ptr< MuseListener > &listener)=0
- virtual AdvertisingStats get_advertising_stats (const std::shared_ptr< Muse > &m)=0
- virtual void reset_advertising_stats ()=0
- virtual void remove_from_list_after (int64_t time)=0

#### Static Public Member Functions

- static std::shared_ptr< MuseManagerWindows > get_instance ()

**Additional Inherited Members**

### Static Public Attributes inherited from interaxon::bridge::MuseManager

- static constexpr int64_t DEFAULT_REMOVE_FROM_LIST_AFTER = 30

## 8.52.1 Detailed Description

Extends MuseManager to provide Windows specific functionality.

MuseManager provides access to all Muse headbanads connected to this Windows device.

## 8.52.2 Member Function Documentation

### 8.52.2.1 get_instance()

```
static std::shared_ptr< MuseManagerWindows > interaxon::bridge::MuseManagerWindows::get_↩
instance ( ) [static]
```

Returns the shared instance of the MuseManager for Windows.

**Returns**

the shared instance of the MuseManager for Windows.

### 8.52.2.2 set_recorder_info()

```
virtual void interaxon::bridge::MuseManagerWindows::set_recorder_info (
            const std::string & name,
            const std::string & version) [pure virtual]
```

Sets the recorder information for writing to Muse files

The app name and version to be recorded as an annotation

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

- muse_manager_windows.h

## 8.53 interaxon::bridge::MuseVersion Class Reference

```
#include <bridge_muse_version.h>
```

#### Public Member Functions

- virtual ∼MuseVersion ()
- virtual std::string get_running_state () const =0
- virtual std::string get_hardware_version () const =0
- virtual std::string get_bsp_version () const =0
- virtual std::string get_firmware_version () const =0
- virtual std::string get_bootloader_version () const =0
- virtual std::string get_firmware_build_number () const =0
- virtual std::string get_firmware_type () const =0
- virtual int32_t get_protocol_version () const =0
- virtual std::string get_ble_firmware_version () const =0

#### Static Public Member Functions

- static std::shared_ptr< MuseVersion > make_default_version ()
- static std::shared_ptr< MuseVersion > make_version (const std::string &json)

## 8.53.1 Detailed Description

Provides access to Muse firmware and hardware versions.

You must connect to the headband at least once to before this information is available. Once you have connected once the information will remain available, even after you disconnect.

## 8.53.2 Constructor & Destructor Documentation

### 8.53.2.1 ∼MuseVersion()

```
virtual interaxon::bridge::MuseVersion::∼MuseVersion ()  [inline], [virtual]
```

## 8.53.3 Member Function Documentation

### 8.53.3.1 get_ble_firmware_version()

```
virtual std::string interaxon::bridge::MuseVersion::get_ble_firmware_version () const  [pure
virtual]
```

Provides access to the Bluetooth firmware version.

**Returns**

The BLE firmware version.

#### 8.53.3.2 get_bootloader_version()

```
virtual std::string interaxon::bridge::MuseVersion::get_bootloader_version () const  [pure
virtual]
```

Provides access to Muse bootloader version.

**Returns**

The bootloader version.

#### 8.53.3.3 get_bsp_version()

```
virtual std::string interaxon::bridge::MuseVersion::get_bsp_version () const  [pure virtual]
```

BSP (board support package) version. This is only available on Muse 2016 ( MU_02 ) or later.

**Returns**

The BSP version.

#### 8.53.3.4 get_firmware_build_number()

```
virtual std::string interaxon::bridge::MuseVersion::get_firmware_build_number () const  [pure
virtual]
```

Provides access to Muse firmware build number. This is only available on Muse 2014 ( MU_01 ).

**Returns**

The firmware build number.

#### 8.53.3.5 get_firmware_type()

```
virtual std::string interaxon::bridge::MuseVersion::get_firmware_type () const  [pure virtual]
```

Type of firmware. One of `consumer`, `research` or `test`.

**Returns**

The type of firmware.

#### 8.53.3.6 get_firmware_version()

```
virtual std::string interaxon::bridge::MuseVersion::get_firmware_version () const  [pure virtual]
```

Provides access to the firmware version.

**Returns**

The firmware version.

### 8.53.3.7 get_hardware_version()

```
virtual std::string interaxon::bridge::MuseVersion::get_hardware_version () const  [pure virtual]
```

Provides access to hardware version.

**Returns**

The hardware version.

### 8.53.3.8 get_protocol_version()

```
virtual int32_t interaxon::bridge::MuseVersion::get_protocol_version () const  [pure virtual]
```

Provides access to Muse communication protocol version.

**Returns**

The communication protocol version.

### 8.53.3.9 get_running_state()

```
virtual std::string interaxon::bridge::MuseVersion::get_running_state () const  [pure virtual]
```

Provides access to the running state.

- For Muse 2014 ( MU_01 ) this is one of `app`, `bootloader` or `test`.
- For Muse 2016 ( MU_02 ) or later this is one of `headband` or `bootloader`.

  **Returns**

  The current running state of the headband.

### 8.53.3.10 make_default_version()

```
static std::shared_ptr< MuseVersion > interaxon::bridge::MuseVersion::make_default_version ()
[static]
```

Internal use only. Create a default version.

### 8.53.3.11 make_version()

```
static std::shared_ptr< MuseVersion > interaxon::bridge::MuseVersion::make_version (
            const std::string & json)  [static]
```

Internal use only. Create version from JSON string on Muse 2016 ( MU_02 ) or later.

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

- bridge_muse_version.h

## 8.54 interaxon::bridge::ReaderListener Class Reference

```
#include <bridge_reader_listener.h>
```

**Public Member Functions**

- virtual ∼ReaderListener ()
- virtual void receive_annotation (const AnnotationData &annotation)=0
- virtual void receive_version (const std::shared_ptr< MuseVersion > &version)=0
- virtual void receive_configuration (const std::shared_ptr< MuseConfiguration > &configuration)=0
- virtual void receive_computing_device_configuration (const ComputingDeviceConfiguration &computing_↩ device_configuration)=0

### 8.54.1 Detailed Description

Implement this interface to receive annotations and configuration/version info from a ReaderMuse

### 8.54.2 Constructor & Destructor Documentation

#### 8.54.2.1 ∼ReaderListener()

```
virtual interaxon::bridge::ReaderListener::∼ReaderListener ()  [inline], [virtual]
```

### 8.54.3 Member Function Documentation

#### 8.54.3.1 receive_annotation()

```
virtual void interaxon::bridge::ReaderListener::receive_annotation (
            const AnnotationData & annotation)  [pure virtual]
```

Called once for each annotation in the file.

**Parameters**

| | |
|---|---|
| *annotation* | The annotation data that was read. |

#### 8.54.3.2 receive_computing_device_configuration()

```
virtual void interaxon::bridge::ReaderListener::receive_computing_device_configuration (
            const ComputingDeviceConfiguration & computing_device_configuration)  [pure virtual]
```

Called each time a computing device packet is encountered.

**Parameters**

| | |
|---|---|
| *computing_device_configuration* | The device configuration that was read. |

### 8.54.3.3 receive_configuration()

```
virtual void interaxon::bridge::ReaderListener::receive_configuration (
            const std::shared_ptr< MuseConfiguration > & configuration)  [pure virtual]
```

Called each time a configuration packet is encountered.

**Parameters**

| | |
|---|---|
| *configuration* | The configuration data that was read. |

### 8.54.3.4 receive_version()

```
virtual void interaxon::bridge::ReaderListener::receive_version (
            const std::shared_ptr< MuseVersion > & version)  [pure virtual]
```

Called each time a version packet is encountered.

**Parameters**

| | |
|---|---|
| *version* | The version data that was read. |

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

- bridge_reader_listener.h

## 8.55 interaxon::bridge::ReaderMuse Class Reference

```
#include <bridge_reader_muse.h>
```

### Public Member Functions

- virtual ~ReaderMuse ()
- virtual void run ()=0
- virtual void run_in_real_timespan ()=0
- virtual int64_t current_time ()=0
- virtual void playback ()=0
- virtual void stop_playback ()=0
- virtual void set_playback_settings (ReaderMusePlaybackSettings settings)=0
- virtual ReaderMusePlaybackSettings get_playback_settings () const =0
- virtual void set_reader_listener (const std::shared_ptr< ReaderListener > &listener)=0
- virtual void set_playback_listener (const std::shared_ptr< ReaderPlaybackListener > &listener)=0
- virtual std::shared_ptr< Muse > as_muse ()=0

### 8.55.1 Detailed Description

A Muse associated with a MuseFileReader

This interface can be used to play back packets from a .muse file so that they can be handled the same way that you would handle data from an actual Muse . Details of the parsing are abstracted and you will simply receive data packets with the data that is read.

When you use this class, register MuseDataListener for the data packets you would like to receive by calling ReaderMuse::as_muse() then Muse::register_data_listener()

**See also**

ReaderMuseBuilder
ReaderMusePlaybackSettings
ReaderPlaybackListener

### 8.55.2 Constructor & Destructor Documentation

#### 8.55.2.1 ∼ReaderMuse()

```
virtual interaxon::bridge::ReaderMuse::∼ReaderMuse ()  [inline], [virtual]
```

### 8.55.3 Member Function Documentation

#### 8.55.3.1 as_muse()

```
virtual std::shared_ptr< Muse > interaxon::bridge::ReaderMuse::as_muse ()  [pure virtual]
```

Downcast to Muse

Note that most of the API of the returned Muse is stubbed and will cause assert failures if called in debug mode. The methods that are implemented are:

- isLowEnergy / is_low_energy
- enableDataTransmission / enable_data_transmission
- register∗
- unregister∗

Note that the only listener that will ever receive packets from a ReaderMuse is MuseDataListener

#### 8.55.3.2 current_time()

```
virtual int64_t interaxon::bridge::ReaderMuse::current_time ()  [pure virtual]
```

Corresponds to the timestamp field for the last packet read.

This does not change within a given listener's body.

**Returns**

The timestamp of the last packet read.

#### 8.55.3.3 get_playback_settings()

```
virtual ReaderMusePlaybackSettings interaxon::bridge::ReaderMuse::get_playback_settings ()
const  [pure virtual]
```

Returns the settings that will be used when playing back a file with this ReaderMuse

**Returns**

The settings that will be used to play back the file.

#### 8.55.3.4 playback()

```
virtual void interaxon::bridge::ReaderMuse::playback ()  [pure virtual]
```

Play back the MuseFile and call each listener for each packet as appropriate.

The speed of the playback is controlled by the playback settings. Simulated playback settings require the ReaderMuse to be constructed with an EventLoop to simulate the time between packets. Calling playback without an EventLoop will not playback the file and will log a warning.

This only works once; the file is not rewound on subsequent calls. If you want to play a file again, create another ReaderMuse

**See also**

ReaderMusePlaybackSettings

#### 8.55.3.5 run()

```
virtual void interaxon::bridge::ReaderMuse::run ()  [pure virtual]
```

Run through all packets in the file, calling each listener for each packet as appropriate.

This only works once; the file is not rewound on subsequent calls. If you want to play a file again, create another ReaderMuse

**Deprecated** Use playback() with the playback setting ReaderMusePlaybackSettings::AS_FAST_AS_POSSIBLE_WITH_SAVED_TIM
to replicate this behaviour.

#### 8.55.3.6 run_in_real_timespan()

```
virtual void interaxon::bridge::ReaderMuse::run_in_real_timespan ()  [pure virtual]
```

Run through all packets in the file, calling each listener for each packet as appropriate in 1:1 timespan

This only works once; the file is not rewound on subsequent calls. If you want to play a file again, create another ReaderMuse

**Deprecated** Use playback() with the playback setting ReaderMusePlaybackSettings::SIMULATED_WITH_SAVED_TIMESTAMP to replicate this behaviour.

#### 8.55.3.7 set_playback_listener()

```
virtual void interaxon::bridge::ReaderMuse::set_playback_listener (
             const std::shared_ptr< ReaderPlaybackListener > & listener)  [pure virtual]
```

Set a listener to receive events related to playback. For example when playback has finished successfully or was interrupted due to error.

**Parameters**

| | |
|---|---|
| *listener* | The playback listener to receive callbacks. |

#### 8.55.3.8 set_playback_settings()

```
virtual void interaxon::bridge::ReaderMuse::set_playback_settings (
             ReaderMusePlaybackSettings settings)  [pure virtual]
```

Sets the settings to use when playing back a file with this ReaderMuse

**Parameters**

| | |
|---|---|
| *settings* | The settings to use for playback. |

#### 8.55.3.9 set_reader_listener()

```
virtual void interaxon::bridge::ReaderMuse::set_reader_listener (
             const std::shared_ptr< ReaderListener > & listener)  [pure virtual]
```

Set a listener to receive annotations, version, and configuration packets.

**Parameters**

| | |
|---|---|
| *listener* | The listener for annotation, version and configuration packets. |

#### 8.55.3.10 stop_playback()

```
virtual void interaxon::bridge::ReaderMuse::stop_playback ()  [pure virtual]
```

Stop playback of the MuseFile

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

- bridge_reader_muse.h

## 8.56 interaxon::bridge::ReaderMuseBuilder Class Reference

```
#include <bridge_reader_muse_builder.h>
```

**Public Member Functions**

- virtual ∼ReaderMuseBuilder ()
- virtual std::shared_ptr< ReaderMuseBuilder > with_packet_types (const std::unordered_set< MuseDataPacketType > &types)=0
- virtual std::shared_ptr< ReaderMuseBuilder > skip_packet_types (const std::unordered_set< MuseDataPacketType > &types)=0
- virtual std::shared_ptr< ReaderMuseBuilder > with_model (MuseModel model)=0
- virtual std::shared_ptr< ReaderMuseBuilder > with_playback_settings (ReaderMusePlaybackSettings settings)=0
- virtual std::shared_ptr< ReaderMuseBuilder > with_event_loop (const std::shared_ptr< EventLoop > &loop)=0
- virtual std::shared_ptr< ReaderMuse > build (const std::shared_ptr< MuseFileReader > &reader)=0
- virtual std::shared_ptr< ReaderMuse > build_with_async (const std::shared_ptr< MuseFileReader > &reader, const std::shared_ptr< EventLoop > &async_loop)=0

**Static Public Member Functions**

- static std::shared_ptr< ReaderMuseBuilder > get ()

### 8.56.1 Detailed Description

Builds a ReaderMuse that plays back the packets in the given MuseFileReader It exposes a way to set the configuration with chainable method calls, e.g.:

```
auto reader_muse = ReaderMuseBuilder::get()
    ->with_packet_types(my_packet_types)
    ->with_model(my_model)
    ->build(my_reader);
```

### 8.56.2 Constructor & Destructor Documentation

#### 8.56.2.1 ∼ReaderMuseBuilder()

```
virtual interaxon::bridge::ReaderMuseBuilder::∼ReaderMuseBuilder ()  [inline], [virtual]
```

### 8.56.3 Member Function Documentation

#### 8.56.3.1 build()

```
virtual std::shared_ptr< ReaderMuse > interaxon::bridge::ReaderMuseBuilder::build (
              const std::shared_ptr< MuseFileReader > & reader)  [pure virtual]
```

Construct a ReaderMuse

**Parameters**

| | |
|---|---|
| *reader* | The MuseFileReader to use to read the file. |

**Returns**

A ReaderMuse configured with the parameters that were set.

### 8.56.3.2 build_with_async()

```
virtual std::shared_ptr< ReaderMuse > interaxon::bridge::ReaderMuseBuilder::build_with_async (
            const std::shared_ptr< MuseFileReader > & reader,
            const std::shared_ptr< EventLoop > & async_loop)  [pure virtual]
```

Construct a ReaderMuse

**Parameters**

| | |
|---|---|
| *reader* | The MuseFileReader to use to read the file. |
| *async_loop* | The EventLoop to use for the call to Muse::run_asynchronously() |

**Returns**

A ReaderMuse configured with the parameters that were set.

**Deprecated** Set the EventLoop with ReaderMuseBuilder::with_event_loop() and then call ReaderMuseBuilder::build() instead.

### 8.56.3.3 get()

```
static std::shared_ptr< ReaderMuseBuilder > interaxon::bridge::ReaderMuseBuilder::get ()
[static]
```

Returns a reference to a ReaderMuseBuilder

**Returns**

A reference to a ReaderMuseBuilder

### 8.56.3.4 skip_packet_types()

```
virtual std::shared_ptr< ReaderMuseBuilder > interaxon::bridge::ReaderMuseBuilder::skip_↩
packet_types (
            const std::unordered_set< MuseDataPacketType > & types)  [pure virtual]
```

The set of packet types to explicitly ignore in the file.

The default is the empty set.

The eventual set used is computed by the last call to with_packet_types() minus the last call to skip_packet_types()

**Parameters**

| | |
|---|---|
| *types* | The set of packet types to skip. |

**Returns**

A reference to the same ReaderMuseBuilder

### 8.56.3.5 with_event_loop()

```
virtual std::shared_ptr< ReaderMuseBuilder > interaxon::bridge::ReaderMuseBuilder::with_←
event_loop (
            const std::shared_ptr< EventLoop > & loop)  [pure virtual]
```

The EventLoop to use to handle simulated playback.

The default is a null pointer (no event loop).

**Parameters**

| | |
|---|---|
| *loop* | The EventLoop to use. |

**Returns**

A reference to the same ReaderMuseBuilder

### 8.56.3.6 with_model()

```
virtual std::shared_ptr< ReaderMuseBuilder > interaxon::bridge::ReaderMuseBuilder::with_model
(
            MuseModel model)  [pure virtual]
```

The model that this Muse should say it is.

The default is Muse 2014 ( MU_01 ).

**Parameters**

| | |
|---|---|
| *model* | The model to use. |

**Returns**

A reference to the same ReaderMuseBuilder

### 8.56.3.7 with_packet_types()

```
virtual std::shared_ptr< ReaderMuseBuilder > interaxon::bridge::ReaderMuseBuilder::with_←
packet_types (
            const std::unordered_set< MuseDataPacketType > & types)  [pure virtual]
```

The set of packet types to pass through from the file.

The default set contains the following all packet types in MuseDataPacketType

The eventual set used is computed by the last call to with_packet_types() minus the last call to skip_packet_types()

**Parameters**

|  |  |
|---|---|
| *types* | The set of packet types to read. |

**Returns**

A reference to the same ReaderMuseBuilder

### 8.56.3.8 with_playback_settings()

```
virtual std::shared_ptr< ReaderMuseBuilder > interaxon::bridge::ReaderMuseBuilder::with_↩
playback_settings (
            ReaderMusePlaybackSettings settings) [pure virtual]
```

The playback settings to use with playing back the file.

The default is ReaderMusePlaybackSettings::AS_FAST_AS_POSSIBLE_WITH_SAVED_TIMESTAMP

**Parameters**

|  |  |
|---|---|
| *settings* | The playback settings to use. |

**Returns**

A reference to the same ReaderMuseBuilder

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

- bridge_reader_muse_builder.h

## 8.57 interaxon::bridge::ReaderPlaybackListener Class Reference

```
#include <bridge_reader_playback_listener.h>
```

**Public Member Functions**

- virtual ∼ReaderPlaybackListener ()
- virtual void receive_playback_done ()=0
- virtual void receive_playback_interrupted ()=0

### 8.57.1 Detailed Description

Listens for ReaderMuse events related to the playback of the file.

### 8.57.2 Constructor & Destructor Documentation

#### 8.57.2.1 ∼ReaderPlaybackListener()

```
virtual interaxon::bridge::ReaderPlaybackListener::∼ReaderPlaybackListener ()  [inline], [virtual]
```

### 8.57.3 Member Function Documentation

#### 8.57.3.1 receive_playback_done()

```
virtual void interaxon::bridge::ReaderPlaybackListener::receive_playback_done ()  [pure virtual]
```

Called when playback of muse file has finished (i.e., reach EOF).

#### 8.57.3.2 receive_playback_interrupted()

```
virtual void interaxon::bridge::ReaderPlaybackListener::receive_playback_interrupted ()  [pure
virtual]
```

Called when playback was interrupted/error occurred when reading the file.

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

- bridge_reader_playback_listener.h

## 8.58 interaxon::bridge::Result Struct Reference

```
#include <bridge_result.h>
```

#### Public Member Functions

- Result (ResultLevel level_, std::string type_, int32_t code_, std::string info_)

#### Public Attributes

- ResultLevel level
- std::string type
- int32_t code
- std::string info

### 8.58.1 Detailed Description

Represents the result of a synchronous operation.

Results can be at different levels: eg, success, info, warning, error

### 8.58.2 Constructor & Destructor Documentation

#### 8.58.2.1 Result()

```
interaxon::bridge::Result::Result (
            ResultLevel level_,
            std::string type_,
            int32_t code_,
            std::string info_ ) [inline]
```

### 8.58.3 Member Data Documentation

#### 8.58.3.1 code

```
int32_t interaxon::bridge::Result::code
```

A machine-parseable error code

**Returns**

0 if the result is a success, non-zero otherwise.

#### 8.58.3.2 info

```
std::string interaxon::bridge::Result::info
```

Human-readable description

**Returns**

A verbose description of the result (error message or status)

#### 8.58.3.3 level

```
ResultLevel interaxon::bridge::Result::level
```

The level of the result.

**Returns**

The ResultLevel

#### 8.58.3.4 type

```
std::string interaxon::bridge::Result::type
```

The type of operation that this result applies to as a string.

**Returns**

The type of this result.

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

- bridge_result.h

## 8.59 interaxon::bridge::Stringify Class Reference

```
#include <bridge_stringify.h>
```

**Public Member Functions**

- virtual ~Stringify ()
- virtual std::string packet_type (MuseDataPacketType type)=0
- virtual std::string connection_state (ConnectionState state)=0

**Static Public Member Functions**

- static std::shared_ptr< Stringify > instance ()

### 8.59.1 Detailed Description

Converts some LibMuse constants to strings.

This is primarily used for LibMuse internal logging purposes. Don't use this to generate UI elements; the results are not guaranteed to be human-friendly, nor localized. You may, however, use this for your own logging or debugging purposes.

### 8.59.2 Constructor & Destructor Documentation

#### 8.59.2.1 ~Stringify()

```
virtual interaxon::bridge::Stringify::~Stringify ()  [inline], [virtual]
```

### 8.59.3 Member Function Documentation

#### 8.59.3.1 connection_state()

```
virtual std::string interaxon::bridge::Stringify::connection_state (
            ConnectionState state)  [pure virtual]
```

String connection state.

**Parameters**

| | |
|---|---|
| *state* | The connection state. |

**Returns**

The string representation of the connection state.

#### 8.59.3.2 instance()

```
static std::shared_ptr< Stringify > interaxon::bridge::Stringify::instance ()  [static]
```

Returns the singleton instance.

**Returns**

the singleton instance.

#### 8.59.3.3 packet_type()

```
virtual std::string interaxon::bridge::Stringify::packet_type (
            MuseDataPacketType type)  [pure virtual]
```

String packet type.

Only the primitive packet types (eeg, accelerometer, gyro, battery, drl_ref) are human-readable. The rest show up as "type_<n>" where n is the integer corresponding to the constant.

**Parameters**

| | |
|---|---|
| *type* | The packet type. |

**Returns**

The string representation of the packet type.

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

- bridge_stringify.h

# Chapter 9

# File Documentation

## 9.1 mainpage.dox File Reference

## 9.2 bridge_accelerometer.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::Accelerometer >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::Accelerometer : int {
  interaxon::bridge::X ,
  interaxon::bridge::Y ,
  interaxon::bridge::Z }

## 9.3 bridge_accelerometer.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class Accelerometer : int {
    X,
    Y,
    Z,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::Accelerometer> {
    size_t operator()(::interaxon::bridge::Accelerometer type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.4 bridge_action.h File Reference

### Classes

- class interaxon::bridge::Action

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.5 bridge_action.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from async.djinni

#pragma once

namespace interaxon { namespace bridge {

class Action {
public:
    virtual ~Action() {}

    virtual void run() = 0;
};

} }  // namespace interaxon::bridge
```

## 9.6 bridge_advertising_stats.h File Reference

### Classes

- struct interaxon::bridge::AdvertisingStats

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.7 bridge_advertising_stats.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from sdk_bridge.djinni

#pragma once

#include <cstdint>
#include <utility>

namespace interaxon { namespace bridge {

struct AdvertisingStats final {
    int32_t numAdvertisingPackets;
    double avgAdvertisingInterval;
    double sigmaAdvertisingInterval;
    double maxAdvertisingInterval;
    bool isLost;
    bool hasBadMac;

    AdvertisingStats(int32_t numAdvertisingPackets_,
                     double avgAdvertisingInterval_,
                     double sigmaAdvertisingInterval_,
                     double maxAdvertisingInterval_,
                     bool isLost_,
                     bool hasBadMac_)
    : numAdvertisingPackets(std::move(numAdvertisingPackets_))
    , avgAdvertisingInterval(std::move(avgAdvertisingInterval_))
    , sigmaAdvertisingInterval(std::move(sigmaAdvertisingInterval_))
    , maxAdvertisingInterval(std::move(maxAdvertisingInterval_))
    , isLost(std::move(isLost_))
    , hasBadMac(std::move(hasBadMac_))
    {}
};

} }  // namespace interaxon::bridge
```

## 9.8 bridge_annotation_data.h File Reference

### Classes

- struct interaxon::bridge::AnnotationData

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.9 bridge_annotation_data.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from muse_file.djinni

#pragma once

#include "api/bridge_annotation_format.h"
#include <string>
#include <utility>

namespace interaxon { namespace bridge {

struct AnnotationData final {
    std::string data;
    AnnotationFormat format;
    std::string event_type;
    std::string event_id;
    std::string parent_id;

    AnnotationData(std::string data_,
                   AnnotationFormat format_,
                   std::string event_type_,
                   std::string event_id_,
                   std::string parent_id_)
    : data(std::move(data_))
    , format(std::move(format_))
    , event_type(std::move(event_type_))
    , event_id(std::move(event_id_))
    , parent_id(std::move(parent_id_))
    {}
};

} }  // namespace interaxon::bridge
```

## 9.10 bridge_annotation_format.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::AnnotationFormat >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::AnnotationFormat : int {
  interaxon::bridge::PLAIN_STRING ,
  interaxon::bridge::JSON ,
  interaxon::bridge::OSC }

## 9.11 bridge_annotation_format.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from muse_file.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class AnnotationFormat : int {
    PLAIN_STRING,
    JSON,
    OSC,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::AnnotationFormat> {
    size_t operator()(::interaxon::bridge::AnnotationFormat type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.12 bridge_api_version.h File Reference

### Classes

- class interaxon::bridge::ApiVersion

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.13 bridge_api_version.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from libmuse_version.djinni

#pragma once

#include <cstdint>
#include <string>

namespace interaxon { namespace bridge {

class ApiVersion {
public:
    virtual ~ApiVersion() {}

    virtual int64_t get_monotonic() = 0;

    virtual int64_t get_major() = 0;

    virtual int64_t get_minor() = 0;

    virtual int64_t get_patch() = 0;

    virtual int64_t get_api() = 0;

    virtual std::string get_string() = 0;
};

} }  // namespace interaxon::bridge
```

## 9.14 bridge_battery.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::Battery >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::Battery : int {
  interaxon::bridge::CHARGE_PERCENTAGE_REMAINING ,
  interaxon::bridge::MILLIVOLTS ,
  interaxon::bridge::TEMPERATURE_CELSIUS ,
  interaxon::bridge::AVERAGE_CURRENT ,
  interaxon::bridge::TIME_TO_EMPTY ,
  interaxon::bridge::TIME_TO_FULL ,
  interaxon::bridge::BATTERY_CAPACITY ,
  interaxon::bridge::REMAINING_CAPACITY ,
  interaxon::bridge::BATTERY_AGE ,
  interaxon::bridge::TOTAL_CYCLES }

## 9.15 bridge_battery.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class Battery : int {
    CHARGE_PERCENTAGE_REMAINING,
    MILLIVOLTS,
    TEMPERATURE_CELSIUS,
    AVERAGE_CURRENT,
    TIME_TO_EMPTY,
    TIME_TO_FULL,
    BATTERY_CAPACITY,
    REMAINING_CAPACITY,
    BATTERY_AGE,
    TOTAL_CYCLES,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::Battery> {
    size_t operator()(::interaxon::bridge::Battery type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.16 bridge_computing_device_configuration.h File Reference

### Classes

- struct interaxon::bridge::ComputingDeviceConfiguration

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.17 bridge_computing_device_configuration.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from muse_file.djinni

#pragma once

#include <cstdint>
#include <string>
#include <utility>

namespace interaxon { namespace bridge {

struct ComputingDeviceConfiguration final {
    std::string os_type;
    std::string os_version;
    std::string hardware_model_name;
    std::string hardware_model_id;
    std::string processor_name;
    std::string processor_speed;
    int32_t number_of_processors;
    std::string memory_size;
    std::string bluetooth_version;
    std::string time_zone;
    int32_t time_zone_offset_seconds;

    ComputingDeviceConfiguration(std::string os_type_,
                                 std::string os_version_,
                                 std::string hardware_model_name_,
                                 std::string hardware_model_id_,
                                 std::string processor_name_,
                                 std::string processor_speed_,
                                 int32_t number_of_processors_,
                                 std::string memory_size_,
                                 std::string bluetooth_version_,
                                 std::string time_zone_,
                                 int32_t time_zone_offset_seconds_)
    : os_type(std::move(os_type_))
    , os_version(std::move(os_version_))
    , hardware_model_name(std::move(hardware_model_name_))
    , hardware_model_id(std::move(hardware_model_id_))
    , processor_name(std::move(processor_name_))
    , processor_speed(std::move(processor_speed_))
    , number_of_processors(std::move(number_of_processors_))
    , memory_size(std::move(memory_size_))
    , bluetooth_version(std::move(bluetooth_version_))
    , time_zone(std::move(time_zone_))
    , time_zone_offset_seconds(std::move(time_zone_offset_seconds_))
    {}
};

} }  // namespace interaxon::bridge
```

## 9.18 bridge_connection_state.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::ConnectionState >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

    *STL namespace.*

### Enumerations

- enum class interaxon::bridge::ConnectionState : int {
  interaxon::bridge::UNKNOWN ,
  interaxon::bridge::CONNECTED ,
  interaxon::bridge::CONNECTING ,
  interaxon::bridge::DISCONNECTED ,
  interaxon::bridge::NEEDS_UPDATE ,
  interaxon::bridge::NEEDS_LICENSE }

## 9.19 bridge_connection_state.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from sdk_bridge.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class ConnectionState : int {
    UNKNOWN,
    CONNECTED,
    CONNECTING,
    DISCONNECTED,
    NEEDS_UPDATE,
    NEEDS_LICENSE,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::ConnectionState> {
    size_t operator()(::interaxon::bridge::ConnectionState type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.20 bridge_drl_ref.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::DrlRef >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::DrlRef : int {
  interaxon::bridge::DRL ,
  interaxon::bridge::REF }

## 9.21 bridge_drl_ref.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class DrlRef : int {
    DRL,
    REF,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::DrlRef> {
    size_t operator()(::interaxon::bridge::DrlRef type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.22 bridge_dsp_data.h File Reference

### Classes

- struct interaxon::bridge::DspData

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.23 bridge_dsp_data.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from muse_file.djinni

#pragma once

#include <cstdint>
#include <string>
#include <utility>
#include <vector>

namespace interaxon { namespace bridge {

struct DspData final {
    std::string type;
    std::vector<double> float_array;
    std::vector<int64_t> int_array;
    std::string version;

    DspData(std::string type_,
            std::vector<double> float_array_,
            std::vector<int64_t> int_array_,
            std::string version_)
    : type(std::move(type_))
    , float_array(std::move(float_array_))
    , int_array(std::move(int_array_))
    , version(std::move(version_))
    {}
};

} }  // namespace interaxon::bridge
```

## 9.24 bridge_eeg.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::Eeg >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::Eeg : int {
  interaxon::bridge::EEG1 ,
  interaxon::bridge::EEG2 ,
  interaxon::bridge::EEG3 ,
  interaxon::bridge::EEG4 ,
  interaxon::bridge::AUX_LEFT ,
  interaxon::bridge::AUX_RIGHT ,
  interaxon::bridge::AUX1 ,
  interaxon::bridge::AUX2 ,
  interaxon::bridge::AUX3 ,
  interaxon::bridge::AUX4 }

## 9.25 bridge_eeg.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class Eeg : int {
    EEG1,
    EEG2,
    EEG3,
    EEG4,
    AUX_LEFT,
    AUX_RIGHT,
    AUX1,
    AUX2,
    AUX3,
    AUX4,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::Eeg> {
    size_t operator()(::interaxon::bridge::Eeg type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.26 bridge_error.h File Reference

### Classes

- struct interaxon::bridge::Error

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.27 bridge_error.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from validation.djinni

#pragma once

#include "api/bridge_error_type.h"
#include <cstdint>
#include <string>
#include <utility>

namespace interaxon { namespace bridge {

struct Error final {
    ErrorType type;
    int32_t code;
    std::string info;

    Error(ErrorType type_,
          int32_t code_,
          std::string info_)
    : type(std::move(type_))
    , code(std::move(code_))
    , info(std::move(info_))
    {}
};

} }  // namespace interaxon::bridge
```

## 9.28 bridge_error_type.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::ErrorType >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::ErrorType : int {
  interaxon::bridge::FAILURE ,
  interaxon::bridge::TIMEOUT ,
  interaxon::bridge::OVERLOADED ,
  interaxon::bridge::UNIMPLEMENTED }

## 9.29 bridge_error_type.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from validation.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class ErrorType : int {
    FAILURE,
    TIMEOUT,
    OVERLOADED,
    UNIMPLEMENTED,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::ErrorType> {
    size_t operator()(::interaxon::bridge::ErrorType type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.30 bridge_event_loop.h File Reference

### Classes

- class interaxon::bridge::EventLoop

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.31 bridge_event_loop.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from async.djinni

#pragma once

#include <cstdint>
#include <memory>

namespace interaxon { namespace bridge {

class Action;

class EventLoop {
public:
    virtual ~EventLoop() {}

    virtual void post(const std::shared_ptr<Action> & action) = 0;

    virtual void post_delayed(const std::shared_ptr<Action> & action, int64_t delay_milliseconds) = 0;

    virtual void cancel() = 0;
};

} }  // namespace interaxon::bridge
```

## 9.32 bridge_gyro.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::Gyro >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::Gyro : int {
  interaxon::bridge::X ,
  interaxon::bridge::Y ,
  interaxon::bridge::Z }

## 9.33 bridge_gyro.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

```

```
enum class Gyro : int {
    X,
    Y,
    Z,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::Gyro> {
    size_t operator()(::interaxon::bridge::Gyro type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.34 bridge_libmuse_version.h File Reference

### Classes

- class interaxon::bridge::LibmuseVersion

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.35 bridge_libmuse_version.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from libmuse_version.djinni

#pragma once

#include <memory>

namespace interaxon { namespace bridge {

class ApiVersion;

class LibmuseVersion {
public:
    virtual ~LibmuseVersion() {}

    static std::shared_ptr<ApiVersion> instance();
};

} }  // namespace interaxon::bridge
```

## 9.36 bridge_log_listener.h File Reference

### Classes

- class interaxon::bridge::LogListener

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.37 bridge_log_listener.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from logging.djinni

#pragma once

namespace interaxon { namespace bridge {

struct LogPacket;

class LogListener {
public:
    virtual ~LogListener() {}

    virtual void receive_log(const LogPacket & log) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.38 bridge_log_manager.h File Reference

### Classes

- class interaxon::bridge::LogManager

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.39 bridge_log_manager.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from logging.djinni

#pragma once

#include <cstdint>
#include <memory>
#include <string>

namespace interaxon { namespace bridge {

class LogListener;
enum class Severity;

class LogManager {
public:
    virtual ~LogManager() {}

    static std::shared_ptr<LogManager> instance();

    virtual std::shared_ptr<LogListener> make_default_log_listener() = 0;

    virtual void set_log_listener(const std::shared_ptr<LogListener> & listener) = 0;

    virtual void set_minimum_severity(Severity severity) = 0;

    virtual void write_log(Severity severity, bool raw, const std::string & tag, const std::string &
message) = 0;

    virtual int64_t get_timestamp() = 0;

    virtual double time_since(int64_t timestamp) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.40 bridge_log_packet.h File Reference

### Classes

- struct interaxon::bridge::LogPacket

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.41 bridge_log_packet.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from logging.djinni

#pragma once

#include "api/bridge_severity.h"
#include <string>
#include <utility>

namespace interaxon { namespace bridge {

struct LogPacket final {
    Severity severity;
    bool raw;
    std::string tag;
    double timestamp;
    std::string message;

    LogPacket(Severity severity_,
              bool raw_,
              std::string tag_,
              double timestamp_,
              std::string message_)
    : severity(std::move(severity_))
    , raw(std::move(raw_))
    , tag(std::move(tag_))
    , timestamp(std::move(timestamp_))
    , message(std::move(message_))
    {}
};

} }  // namespace interaxon::bridge
```

## 9.42 bridge_magnetometer.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::Magnetometer >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::Magnetometer : int {
  interaxon::bridge::X ,
  interaxon::bridge::Y ,
  interaxon::bridge::Z }

## 9.43 bridge_magnetometer.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class Magnetometer : int {
    X,
    Y,
    Z,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::Magnetometer> {
    size_t operator()(::interaxon::bridge::Magnetometer type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.44 bridge_message_type.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::MessageType >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::MessageType : int {
  interaxon::bridge::EEG ,
  interaxon::bridge::QUANTIZATION ,
  interaxon::bridge::ACCELEROMETER ,
  interaxon::bridge::BATTERY ,
  interaxon::bridge::VERSION ,
  interaxon::bridge::CONFIGURATION ,
  interaxon::bridge::ANNOTATION ,
  interaxon::bridge::HISTOGRAM ,
  interaxon::bridge::ALG_VALUE ,
  interaxon::bridge::DSP ,
  interaxon::bridge::COMPUTING_DEVICE ,
  interaxon::bridge::EEG_DROPPED ,
  interaxon::bridge::ACC_DROPPED ,
  interaxon::bridge::CALM_APP ,
  interaxon::bridge::CALM_ALG ,
  interaxon::bridge::MUSE_ELEMENTS ,
  interaxon::bridge::GYRO ,
  interaxon::bridge::ARTIFACT ,
  interaxon::bridge::PRESSURE ,
  interaxon::bridge::TEMPERATURE ,
  interaxon::bridge::ULTRA_VIOLET ,
  interaxon::bridge::MAGNETOMETER ,
  interaxon::bridge::PPG ,
  interaxon::bridge::THERMISTOR ,
  interaxon::bridge::OPTICS ,
  interaxon::bridge::ALGORITHM }

## 9.45 bridge_message_type.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from muse_file.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class MessageType : int {
    EEG,
    QUANTIZATION,
    ACCELEROMETER,
    BATTERY,
    VERSION,
    CONFIGURATION,
    ANNOTATION,
    HISTOGRAM,
    ALG_VALUE,
    DSP,
    COMPUTING_DEVICE,
    EEG_DROPPED,
    ACC_DROPPED,
    CALM_APP,
    CALM_ALG,
    MUSE_ELEMENTS,
    GYRO,
    ARTIFACT,
    PRESSURE,
    TEMPERATURE,
    ULTRA_VIOLET,
    MAGNETOMETER,
    PPG,
    THERMISTOR,
    OPTICS,
    ALGORITHM,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::MessageType> {
    size_t operator()(::interaxon::bridge::MessageType type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.46 bridge_muse.h File Reference

### Classes

- class interaxon::bridge::Muse

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.47 bridge_muse.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from sdk_bridge.djinni

#pragma once

#include <cstdint>
#include <memory>
#include <string>
#include <vector>

namespace interaxon { namespace bridge {

class MuseConfiguration;
class MuseConnectionListener;
class MuseDataListener;
class MuseErrorListener;
class MuseVersion;
enum class ConnectionState;
enum class MuseDataPacketType;
enum class MuseModel;
enum class MusePreset;
enum class NotchFrequency;

class Muse {
public:
    virtual ~Muse() {}

    virtual void connect() = 0;

    virtual void disconnect() = 0;

    virtual void execute() = 0;

    virtual void run_asynchronously() = 0;

    virtual ConnectionState get_connection_state() = 0;

    virtual std::string get_mac_address() = 0;

    virtual std::string get_name() = 0;

    virtual double get_rssi() = 0;

    virtual MuseModel get_model() = 0;

    virtual double get_last_discovered_time() = 0;

    virtual void set_num_connect_tries(int32_t num_tries) = 0;

    virtual std::shared_ptr<MuseConfiguration> get_muse_configuration() = 0;

    virtual std::shared_ptr<MuseVersion> get_muse_version() = 0;

    virtual void register_connection_listener(const std::shared_ptr<MuseConnectionListener> &
listener) = 0;

    virtual void unregister_connection_listener(const std::shared_ptr<MuseConnectionListener> &
listener) = 0;

    virtual void register_data_listener(const std::shared_ptr<MuseDataListener> & listener,
MuseDataPacketType type) = 0;

    virtual void unregister_data_listener(const std::shared_ptr<MuseDataListener> & listener,
MuseDataPacketType type) = 0;

    virtual void register_error_listener(const std::shared_ptr<MuseErrorListener> & listener) = 0;

    virtual void unregister_error_listener(const std::shared_ptr<MuseErrorListener> & listener) = 0;

    virtual void unregister_all_listeners() = 0;

    virtual void set_preset(MusePreset preset) = 0;

    virtual void enable_led_indicator(bool enable) = 0;

    virtual void enable_data_transmission(bool enable) = 0;

    virtual void set_notch_frequency(NotchFrequency new_frequency) = 0;

    virtual bool is_low_energy() = 0;

    virtual bool is_paired() = 0;

    virtual bool is_connectable() = 0;

    virtual void set_license_data(const std::vector<uint8_t> & data) = 0;

    virtual void enable_exception(bool enable) = 0;

    virtual void set_property(const std::string & name, const std::string & value) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.48 bridge_muse_artifact_packet.h File Reference

### Classes

- struct interaxon::bridge::MuseArtifactPacket

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.49 bridge_muse_artifact_packet.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <cstdint>
#include <utility>

namespace interaxon { namespace bridge {

struct MuseArtifactPacket final {
    bool headband_on;
    bool blink;
    bool jaw_clench;
    int64_t timestamp;

    MuseArtifactPacket(bool headband_on_,
                       bool blink_,
                       bool jaw_clench_,
                       int64_t timestamp_)
    : headband_on(std::move(headband_on_))
    , blink(std::move(blink_))
    , jaw_clench(std::move(jaw_clench_))
    , timestamp(std::move(timestamp_))
    {}
};

} }  // namespace interaxon::bridge
```

## 9.50 bridge_muse_configuration.h File Reference

### Classes

- class interaxon::bridge::MuseConfiguration

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.51 bridge_muse_configuration.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from museinfo.djinni

#pragma once

#include <cstdint>
#include <string>

namespace interaxon { namespace bridge {

enum class MuseModel;
enum class MusePreset;
enum class NotchFrequency;

class MuseConfiguration {
public:
    virtual ~MuseConfiguration() {}

    virtual MusePreset get_preset() const = 0;

    virtual std::string get_headband_name() const = 0;
    virtual std::string get_microcontroller_id() const = 0;

    virtual int32_t get_eeg_channel_count() const = 0;

    virtual int32_t get_afe_gain() const = 0;

    virtual int32_t get_downsample_rate() const = 0;

    virtual int32_t get_serout_mode() const = 0;

    virtual int32_t get_output_frequency() const = 0;

    virtual int32_t get_adc_frequency() const = 0;

    virtual bool get_notch_filter_enabled() const = 0;

    virtual NotchFrequency get_notch_filter() const = 0;

    virtual int32_t get_accelerometer_sample_frequency() const = 0;

    virtual bool get_battery_data_enabled() const = 0;

    virtual bool get_drl_ref_enabled() const = 0;

    virtual int32_t get_drl_ref_frequency() const = 0;

    virtual double get_battery_percent_remaining() const = 0;

    virtual std::string get_bluetooth_mac() const = 0;

    virtual std::string get_serial_number() const = 0;

    virtual std::string get_headset_serial_number() const = 0;

    virtual MuseModel get_model() const = 0;

    virtual std::string get_license_nonce() const = 0;

    virtual int32_t get_switch() const = 0;
};

} }  // namespace interaxon::bridge
```

## 9.52 bridge_muse_connection_listener.h File Reference

### Classes

- class interaxon::bridge::MuseConnectionListener

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.53 bridge_muse_connection_listener.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from listeners.djinni

#pragma once

#include <memory>

namespace interaxon { namespace bridge {

class Muse;
struct MuseConnectionPacket;
class MuseConnectionListener {
public:
    virtual ~MuseConnectionListener() {}

    virtual void receive_muse_connection_packet(const MuseConnectionPacket & packet, const
std::shared_ptr<Muse> & muse) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.54 bridge_muse_connection_packet.h File Reference

### Classes

- struct interaxon::bridge::MuseConnectionPacket

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.55 bridge_muse_connection_packet.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include "api/bridge_connection_state.h"
#include <utility>

namespace interaxon { namespace bridge {

struct MuseConnectionPacket final {
    ConnectionState previous_connection_state;
    ConnectionState current_connection_state;

    MuseConnectionPacket(ConnectionState previous_connection_state_,
                         ConnectionState current_connection_state_)
    : previous_connection_state(std::move(previous_connection_state_))
    , current_connection_state(std::move(current_connection_state_))
    {}
};

} }  // namespace interaxon::bridge
```

## 9.56 bridge_muse_data_listener.h File Reference

### Classes

- class interaxon::bridge::MuseDataListener

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.57 bridge_muse_data_listener.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from listeners.djinni

#pragma once

#include <memory>

namespace interaxon { namespace bridge {

class Muse;
class MuseDataPacket;
struct MuseArtifactPacket;

class MuseDataListener {
public:
    virtual ~MuseDataListener() {}

    virtual void receive_muse_data_packet(const std::shared_ptr<MuseDataPacket> & packet, const
std::shared_ptr<Muse> & muse) = 0;

    virtual void receive_muse_artifact_packet(const MuseArtifactPacket & packet, const
std::shared_ptr<Muse> & muse) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.58 bridge_muse_data_packet.h File Reference

### Classes

- class interaxon::bridge::MuseDataPacket

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.59 bridge_muse_data_packet.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <cstdint>
#include <memory>
#include <vector>

namespace interaxon { namespace bridge {

enum class Accelerometer;
enum class Battery;
enum class DrlRef;
enum class Eeg;
enum class Gyro;
enum class Magnetometer;
enum class MuseDataPacketType;
enum class MuseModel;
enum class Optics;
enum class Ppg;
enum class Pressure;
enum class UltraViolet;

class MuseDataPacket {
public:
    virtual ~MuseDataPacket() {}

    static std::shared_ptr<MuseDataPacket> make_uninitialized_packet(int64_t capacity);

    static std::shared_ptr<MuseDataPacket> make_packet(MuseDataPacketType type, int64_t timestamp,
const std::vector<double> & values);

    virtual MuseDataPacketType packet_type() = 0;

    virtual int64_t timestamp() = 0;

    virtual std::vector<double> values() = 0;

    virtual int64_t values_size() = 0;

    virtual double get_eeg_channel_value(Eeg channel_num) = 0;

    virtual double get_ppg_channel_value(Ppg channel_num) = 0;

    virtual double get_ppg_microamps(MuseModel model, double ppg_value) = 0;

    virtual double get_optics_channel_value(Optics channel_num) = 0;

    virtual double get_battery_value(Battery b) = 0;

    virtual double get_accelerometer_value(Accelerometer a) = 0;

    virtual double get_gyro_value(Gyro g) = 0;

    virtual double get_magnetometer_value(Magnetometer m) = 0;

    virtual double get_drl_ref_value(DrlRef drl) = 0;

    virtual double get_pressure_value(Pressure pressure) = 0;

    virtual double get_temperature_value() = 0;

    virtual double get_uv_value(UltraViolet v) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.60 bridge_muse_data_packet_type.h File Reference

### Classes

- struct std::hash<::interaxon::bridge::MuseDataPacketType >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::MuseDataPacketType : int {
  interaxon::bridge::ACCELEROMETER ,
  interaxon::bridge::GYRO ,
  interaxon::bridge::EEG ,
  interaxon::bridge::DROPPED_ACCELEROMETER ,
  interaxon::bridge::DROPPED_EEG ,
  interaxon::bridge::QUANTIZATION ,
  interaxon::bridge::BATTERY ,

interaxon::bridge::DRL_REF ,
interaxon::bridge::ALPHA_ABSOLUTE ,
interaxon::bridge::BETA_ABSOLUTE ,
interaxon::bridge::DELTA_ABSOLUTE ,
interaxon::bridge::THETA_ABSOLUTE ,
interaxon::bridge::GAMMA_ABSOLUTE ,
interaxon::bridge::ALPHA_RELATIVE ,
interaxon::bridge::BETA_RELATIVE ,
interaxon::bridge::DELTA_RELATIVE ,
interaxon::bridge::THETA_RELATIVE ,
interaxon::bridge::GAMMA_RELATIVE ,
interaxon::bridge::ALPHA_SCORE ,
interaxon::bridge::BETA_SCORE ,
interaxon::bridge::DELTA_SCORE ,
interaxon::bridge::THETA_SCORE ,
interaxon::bridge::GAMMA_SCORE ,
interaxon::bridge::IS_GOOD ,
interaxon::bridge::HSI ,
interaxon::bridge::HSI_PRECISION ,
interaxon::bridge::ARTIFACTS ,
interaxon::bridge::MAGNETOMETER ,
interaxon::bridge::PRESSURE ,
interaxon::bridge::TEMPERATURE ,
interaxon::bridge::ULTRA_VIOLET ,
interaxon::bridge::NOTCH_FILTERED_EEG ,
interaxon::bridge::VARIANCE_EEG ,
interaxon::bridge::VARIANCE_NOTCH_FILTERED_EEG ,
interaxon::bridge::PPG ,
interaxon::bridge::IS_PPG_GOOD ,
interaxon::bridge::IS_HEART_GOOD ,
interaxon::bridge::THERMISTOR ,
interaxon::bridge::IS_THERMISTOR_GOOD ,
interaxon::bridge::AVG_BODY_TEMPERATURE ,
interaxon::bridge::CLOUD_COMPUTED ,
interaxon::bridge::OPTICS ,
interaxon::bridge::TOTAL }

## 9.61 bridge_muse_data_packet_type.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class MuseDataPacketType : int {
    ACCELEROMETER,
    GYRO,
    EEG,
    DROPPED_ACCELEROMETER,
    DROPPED_EEG,
    QUANTIZATION,
    BATTERY,
    DRL_REF,
    ALPHA_ABSOLUTE,
    BETA_ABSOLUTE,
    DELTA_ABSOLUTE,
    THETA_ABSOLUTE,
    GAMMA_ABSOLUTE,
    ALPHA_RELATIVE,
    BETA_RELATIVE,
    DELTA_RELATIVE,
    THETA_RELATIVE,
    GAMMA_RELATIVE,
    ALPHA_SCORE,
    BETA_SCORE,
    DELTA_SCORE,
    THETA_SCORE,
    GAMMA_SCORE,
    IS_GOOD,
    HSI,
    HSI_PRECISION,
    ARTIFACTS,
    MAGNETOMETER,
    PRESSURE,
    TEMPERATURE,
    ULTRA_VIOLET,
    NOTCH_FILTERED_EEG,
    VARIANCE_EEG,
    VARIANCE_NOTCH_FILTERED_EEG,
    PPG,
    IS_PPG_GOOD,
    IS_HEART_GOOD,
    THERMISTOR,
    IS_THERMISTOR_GOOD,
    AVG_BODY_TEMPERATURE,
    CLOUD_COMPUTED,
    OPTICS,
    TOTAL,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::MuseDataPacketType> {
    size_t operator()(::interaxon::bridge::MuseDataPacketType type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.62 bridge_muse_error_listener.h File Reference

### Classes

- class interaxon::bridge::MuseErrorListener

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.63 bridge_muse_error_listener.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from listeners.djinni

#pragma once

#include <memory>

namespace interaxon { namespace bridge {

class Muse;
struct Error;

class MuseErrorListener {
public:
    virtual ~MuseErrorListener() {}

    virtual void receive_error(const Error & error, const std::shared_ptr<Muse> & muse) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.64 bridge_muse_file.h File Reference

### Classes

- class interaxon::bridge::MuseFile

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.65 bridge_muse_file.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from muse_file.djinni

#pragma once

#include <cstdint>
#include <vector>

namespace interaxon { namespace bridge {

class MuseFile {
public:
    virtual ~MuseFile() {}

    virtual bool open(bool for_writing) = 0;

    virtual bool write(const std::vector<uint8_t> & buffer) = 0;

    virtual std::vector<uint8_t> read(int32_t length) = 0;

    virtual bool close(bool for_writing) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.66 bridge_muse_file_reader.h File Reference

### Classes

- class interaxon::bridge::MuseFileReader

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.67 bridge_muse_file_reader.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from muse_file.djinni

#pragma once

#include <cstdint>
#include <memory>

namespace interaxon { namespace bridge {

class MuseConfiguration;
class MuseDataPacket;
class MuseFile;
class MuseVersion;
enum class MessageType;
struct AnnotationData;
struct ComputingDeviceConfiguration;
struct DspData;
struct MuseArtifactPacket;
struct Result;

class MuseFileReader {
public:
    virtual ~MuseFileReader() {}

    static std::shared_ptr<MuseFileReader> get_file_reader(const std::shared_ptr<MuseFile> & file);

    virtual bool open() = 0;

    virtual bool close() = 0;

    virtual Result goto_next_message() = 0;

    virtual MessageType get_message_type() = 0;

    virtual int32_t get_message_id() = 0;

    virtual int64_t get_message_timestamp() = 0;

    virtual AnnotationData get_annotation() = 0;

    virtual std::shared_ptr<MuseConfiguration> get_configuration() = 0;

    virtual std::shared_ptr<MuseVersion> get_version() = 0;

    virtual ComputingDeviceConfiguration get_computing_device_configuration() = 0;

    virtual DspData get_dsp() = 0;

    virtual std::shared_ptr<MuseDataPacket> get_data_packet() = 0;

    virtual MuseArtifactPacket get_artifact_packet() = 0;
};

} }  // namespace interaxon::bridge
```

## 9.68 bridge_muse_file_writer.h File Reference

### Classes

- class interaxon::bridge::MuseFileWriter

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.69 bridge_muse_file_writer.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from muse_file.djinni

#pragma once

#include <cstdint>
#include <memory>
#include <string>

namespace interaxon { namespace bridge {

class MuseConfiguration;
class MuseDataPacket;
class MuseFile;
class MuseVersion;
enum class TimestampMode;
struct AnnotationData;
struct ComputingDeviceConfiguration;
struct DspData;
struct MuseArtifactPacket;

class MuseFileWriter {
public:
    virtual ~MuseFileWriter() {}

    static std::shared_ptr<MuseFileWriter> get_file_writer(const std::shared_ptr<MuseFile> & file);

    virtual bool open() = 0;

    virtual bool is_open() = 0;

    virtual bool close() = 0;

    virtual void discard_buffered_packets() = 0;

    virtual bool flush() = 0;

    virtual int32_t get_bufferred_messages_count() = 0;

    virtual int32_t get_buffered_messages_size() = 0;

    virtual int64_t get_total_bytes_written() const = 0;

    virtual void add_artifact_packet(int32_t id, const MuseArtifactPacket & packet) = 0;

    virtual void add_data_packet(int32_t id, const std::shared_ptr<MuseDataPacket> & packet) = 0;

    virtual void add_annotation_string(int32_t id, const std::string & annotation) = 0;

    virtual void add_annotation(int32_t id, const AnnotationData & annotation) = 0;

    virtual void add_configuration(int32_t id, const std::shared_ptr<MuseConfiguration> &
configuration) = 0;

    virtual void add_version(int32_t id, const std::shared_ptr<MuseVersion> & version) = 0;

    virtual void add_computing_device_configuration(int32_t id, const ComputingDeviceConfiguration &
configuration) = 0;

    virtual void add_dsp(int32_t id, const DspData & dsp) = 0;

    virtual void set_timestamp_mode(TimestampMode mode) = 0;

    virtual void set_timestamp(int64_t timestamp) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.70 bridge_muse_listener.h File Reference

### Classes

- class interaxon::bridge::MuseListener

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.71 bridge_muse_listener.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from sdk_bridge.djinni

#pragma once

namespace interaxon { namespace bridge {

class MuseListener {
public:
    virtual ~MuseListener() {}

    virtual void muse_list_changed() = 0;
};

} }  // namespace interaxon::bridge
```

## 9.72 bridge_muse_manager.h File Reference

### Classes

- class interaxon::bridge::MuseManager

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.73 bridge_muse_manager.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from sdk_bridge.djinni

#pragma once

#include <cstdint>
#include <memory>
#include <vector>

namespace interaxon { namespace bridge {

class Muse;
class MuseListener;
struct AdvertisingStats;

class MuseManager {
public:
    virtual ~MuseManager() {}

    static constexpr int64_t DEFAULT_REMOVE_FROM_LIST_AFTER = 30;

    virtual std::vector<std::shared_ptr<Muse» get_muses() = 0;

    virtual void start_listening() = 0;

    virtual void stop_listening() = 0;

    virtual void set_muse_listener(const std::shared_ptr<MuseListener> & listener) = 0;

    virtual AdvertisingStats get_advertising_stats(const std::shared_ptr<Muse> & m) = 0;

    virtual void reset_advertising_stats() = 0;

    virtual void remove_from_list_after(int64_t time) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.74 bridge_muse_model.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::MuseModel >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::MuseModel : int {
  interaxon::bridge::MU_01 ,
  interaxon::bridge::MU_02 ,
  interaxon::bridge::MU_03 ,
  interaxon::bridge::MU_04 ,
  interaxon::bridge::MU_05 ,
  interaxon::bridge::MU_06 ,
  interaxon::bridge::MS_03 }

## 9.75 bridge_muse_model.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from museinfo.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class MuseModel : int {
    MU_01,
    MU_02,
    MU_03,
    MU_04,
    MU_05,
    MU_06,
    MS_03,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::MuseModel> {
    size_t operator()(::interaxon::bridge::MuseModel type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.76 bridge_muse_preset.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::MusePreset >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::MusePreset : int {
  interaxon::bridge::PRESET_10 ,
  interaxon::bridge::PRESET_12 ,
  interaxon::bridge::PRESET_14 ,
  interaxon::bridge::PRESET_20 ,
  interaxon::bridge::PRESET_21 ,
  interaxon::bridge::PRESET_22 ,
  interaxon::bridge::PRESET_23 ,
  interaxon::bridge::PRESET_AB ,
  interaxon::bridge::PRESET_AD ,
  interaxon::bridge::PRESET_31 ,
  interaxon::bridge::PRESET_32 ,
  interaxon::bridge::PRESET_50 ,
  interaxon::bridge::PRESET_51 ,
  interaxon::bridge::PRESET_52 ,
  interaxon::bridge::PRESET_53 ,
  interaxon::bridge::PRESET_54 ,
  interaxon::bridge::PRESET_55 ,
  interaxon::bridge::PRESET_60 ,
  interaxon::bridge::PRESET_61 ,
  interaxon::bridge::PRESET_63 ,
  interaxon::bridge::PRESET_1021 ,
  interaxon::bridge::PRESET_1022 ,
  interaxon::bridge::PRESET_1023 ,
  interaxon::bridge::PRESET_1024 ,
  interaxon::bridge::PRESET_1025 ,
  interaxon::bridge::PRESET_1026 ,
  interaxon::bridge::PRESET_1027 ,
  interaxon::bridge::PRESET_1028 ,

interaxon::bridge::PRESET_1029 ,
interaxon::bridge::PRESET_102A ,
interaxon::bridge::PRESET_1031 ,
interaxon::bridge::PRESET_1032 ,
interaxon::bridge::PRESET_1033 ,
interaxon::bridge::PRESET_1034 ,
interaxon::bridge::PRESET_1035 ,
interaxon::bridge::PRESET_1036 ,
interaxon::bridge::PRESET_1041 ,
interaxon::bridge::PRESET_1042 ,
interaxon::bridge::PRESET_1043 ,
interaxon::bridge::PRESET_1044 ,
interaxon::bridge::PRESET_1045 ,
interaxon::bridge::PRESET_1046 }

## 9.77 bridge_muse_preset.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from museinfo.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class MusePreset : int {
    PRESET_10,
    PRESET_12,
    PRESET_14,
    PRESET_20,
    PRESET_21,
    PRESET_22,
    PRESET_23,
    PRESET_AB,
    PRESET_AD,
    PRESET_31,
    PRESET_32,
    PRESET_50,
    PRESET_51,
    PRESET_52,
    PRESET_53,
    PRESET_54,
    PRESET_55,
    PRESET_60,
    PRESET_61,
    PRESET_63,
    PRESET_1021,
    PRESET_1022,
    PRESET_1023,
    PRESET_1024,
    PRESET_1025,
    PRESET_1026,
    PRESET_1027,
    PRESET_1028,
    PRESET_1029,
    PRESET_102A,
    PRESET_1031,
    PRESET_1032,
    PRESET_1033,
    PRESET_1034,
    PRESET_1035,
    PRESET_1036,
    PRESET_1041,
    PRESET_1042,
    PRESET_1043,
    PRESET_1044,
    PRESET_1045,
    PRESET_1046,
};

} }  // namespace interaxon::bridge

```

```
namespace std {

template <>
struct hash<::interaxon::bridge::MusePreset> {
    size_t operator()(::interaxon::bridge::MusePreset type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.78 bridge_muse_version.h File Reference

### Classes

- class interaxon::bridge::MuseVersion

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.79 bridge_muse_version.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from museinfo.djinni

#pragma once

#include <cstdint>
#include <memory>
#include <string>

namespace interaxon { namespace bridge {

class MuseVersion {
public:
    virtual ~MuseVersion() {}

    static std::shared_ptr<MuseVersion> make_default_version();

    static std::shared_ptr<MuseVersion> make_version(const std::string & json);

    virtual std::string get_running_state() const = 0;

    virtual std::string get_hardware_version() const = 0;

    virtual std::string get_bsp_version() const = 0;

    virtual std::string get_firmware_version() const = 0;

    virtual std::string get_bootloader_version() const = 0;

    virtual std::string get_firmware_build_number() const = 0;

    virtual std::string get_firmware_type() const = 0;

    virtual int32_t get_protocol_version() const = 0;

    virtual std::string get_ble_firmware_version() const = 0;
};

} }  // namespace interaxon::bridge
```

## 9.80 bridge_notch_frequency.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::NotchFrequency >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::NotchFrequency : int {
  interaxon::bridge::NOTCH_NONE ,
  interaxon::bridge::NOTCH_50HZ ,
  interaxon::bridge::NOTCH_60HZ }

## 9.81 bridge_notch_frequency.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from museinfo.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class NotchFrequency : int {
    NOTCH_NONE,
    NOTCH_50HZ,
    NOTCH_60HZ,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::NotchFrequency> {
    size_t operator()(::interaxon::bridge::NotchFrequency type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.82 bridge_optics.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::Optics >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::Optics : int {
  interaxon::bridge::OPTICS1 ,
  interaxon::bridge::OPTICS2 ,
  interaxon::bridge::OPTICS3 ,
  interaxon::bridge::OPTICS4 ,
  interaxon::bridge::OPTICS5 ,
  interaxon::bridge::OPTICS6 ,
  interaxon::bridge::OPTICS7 ,
  interaxon::bridge::OPTICS8 ,
  interaxon::bridge::OPTICS9 ,
  interaxon::bridge::OPTICS10 ,
  interaxon::bridge::OPTICS11 ,
  interaxon::bridge::OPTICS12 ,
  interaxon::bridge::OPTICS13 ,
  interaxon::bridge::OPTICS14 ,
  interaxon::bridge::OPTICS15 ,
  interaxon::bridge::OPTICS16 }

## 9.83 bridge_optics.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class Optics : int {
    OPTICS1,
    OPTICS2,
    OPTICS3,
    OPTICS4,
    OPTICS5,
    OPTICS6,
    OPTICS7,
    OPTICS8,
    OPTICS9,
    OPTICS10,
    OPTICS11,
    OPTICS12,
    OPTICS13,
    OPTICS14,
    OPTICS15,
    OPTICS16,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::Optics> {
    size_t operator()(::interaxon::bridge::Optics type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.84 bridge_ppg.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::Ppg >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::Ppg : int {
  interaxon::bridge::AMBIENT ,
  interaxon::bridge::IR ,
  interaxon::bridge::RED }

## 9.85 bridge_ppg.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class Ppg : int {
    AMBIENT,
    IR,
    RED,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::Ppg> {
    size_t operator()(::interaxon::bridge::Ppg type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.86 bridge_pressure.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::Pressure >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::Pressure : int {
  interaxon::bridge::RAW ,
  interaxon::bridge::AVERAGED }

## 9.87 bridge_pressure.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class Pressure : int {
    RAW,
    AVERAGED,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::Pressure> {
    size_t operator()(::interaxon::bridge::Pressure type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.88 bridge_reader_listener.h File Reference

### Classes

- class interaxon::bridge::ReaderListener

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.89 bridge_reader_listener.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from listeners.djinni

#pragma once

#include <memory>

namespace interaxon { namespace bridge {

class MuseConfiguration;
class MuseVersion;
struct AnnotationData;
struct ComputingDeviceConfiguration;

class ReaderListener {
public:
    virtual ~ReaderListener() {}

    virtual void receive_annotation(const AnnotationData & annotation) = 0;

    virtual void receive_version(const std::shared_ptr<MuseVersion> & version) = 0;

    virtual void receive_configuration(const std::shared_ptr<MuseConfiguration> & configuration) = 0;

    virtual void receive_computing_device_configuration(const ComputingDeviceConfiguration & computing_device_configuration) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.90 bridge_reader_muse.h File Reference

### Classes

- class interaxon::bridge::ReaderMuse

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.91 bridge_reader_muse.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from sdk_bridge.djinni

#pragma once

#include <cstdint>
#include <memory>

namespace interaxon { namespace bridge {

class Muse;
class ReaderListener;
class ReaderPlaybackListener;
enum class ReaderMusePlaybackSettings;

class ReaderMuse {
public:
    virtual ~ReaderMuse() {}

    virtual void run() = 0;
    virtual void run_in_real_timespan() = 0;

    virtual int64_t current_time() = 0;

    virtual void playback() = 0;

    virtual void stop_playback() = 0;

    virtual void set_playback_settings(ReaderMusePlaybackSettings settings) = 0;

    virtual ReaderMusePlaybackSettings get_playback_settings() const = 0;

    virtual void set_reader_listener(const std::shared_ptr<ReaderListener> & listener) = 0;

    virtual void set_playback_listener(const std::shared_ptr<ReaderPlaybackListener> & listener) = 0;

    virtual std::shared_ptr<Muse> as_muse() = 0;
};

} }  // namespace interaxon::bridge
```

## 9.92 bridge_reader_muse_builder.h File Reference

### Classes

- class interaxon::bridge::ReaderMuseBuilder

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.93 bridge_reader_muse_builder.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from sdk_bridge.djinni

#pragma once

#include <memory>
#include <unordered_set>

namespace interaxon { namespace bridge {

class EventLoop;
class MuseFileReader;
class ReaderMuse;
enum class MuseDataPacketType;
enum class MuseModel;
enum class ReaderMusePlaybackSettings;

class ReaderMuseBuilder {
public:
    virtual ~ReaderMuseBuilder() {}

    static std::shared_ptr<ReaderMuseBuilder> get();

    virtual std::shared_ptr<ReaderMuseBuilder> with_packet_types(const
std::unordered_set<MuseDataPacketType> & types) = 0;

    virtual std::shared_ptr<ReaderMuseBuilder> skip_packet_types(const
std::unordered_set<MuseDataPacketType> & types) = 0;

    virtual std::shared_ptr<ReaderMuseBuilder> with_model(MuseModel model) = 0;

    virtual std::shared_ptr<ReaderMuseBuilder> with_playback_settings(ReaderMusePlaybackSettings
settings) = 0;
```

```
    virtual std::shared_ptr<ReaderMuseBuilder> with_event_loop(const std::shared_ptr<EventLoop> &
loop) = 0;

    virtual std::shared_ptr<ReaderMuse> build(const std::shared_ptr<MuseFileReader> & reader) = 0;

    virtual std::shared_ptr<ReaderMuse> build_with_async(const std::shared_ptr<MuseFileReader> &
reader, const std::shared_ptr<EventLoop> & async_loop) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.94 bridge_reader_muse_playback_settings.h File Reference

### Classes

- struct std::hash<::interaxon::bridge::ReaderMusePlaybackSettings >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::ReaderMusePlaybackSettings : int {
  interaxon::bridge::AS_FAST_AS_POSSIBLE_WITH_SAVED_TIMESTAMP ,
  interaxon::bridge::SIMULATED_WITH_SAVED_TIMESTAMP ,
  interaxon::bridge::SIMULATED_WITH_SYSTEM_CLOCK_TIMESTAMP }

## 9.95 bridge_reader_muse_playback_settings.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from sdk_bridge.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class ReaderMusePlaybackSettings : int {
    AS_FAST_AS_POSSIBLE_WITH_SAVED_TIMESTAMP,
    SIMULATED_WITH_SAVED_TIMESTAMP,
    SIMULATED_WITH_SYSTEM_CLOCK_TIMESTAMP,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::ReaderMusePlaybackSettings> {
    size_t operator()(::interaxon::bridge::ReaderMusePlaybackSettings type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.96 bridge_reader_playback_listener.h File Reference

### Classes

- class interaxon::bridge::ReaderPlaybackListener

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.97 bridge_reader_playback_listener.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from listeners.djinni

#pragma once

namespace interaxon { namespace bridge {

class ReaderPlaybackListener {
public:
    virtual ~ReaderPlaybackListener() {}

    virtual void receive_playback_done() = 0;

    virtual void receive_playback_interrupted() = 0;
};

} }  // namespace interaxon::bridge
```

## 9.98 bridge_result.h File Reference

### Classes

- struct interaxon::bridge::Result

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.99 bridge_result.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from validation.djinni

#pragma once

#include "api/bridge_result_level.h"
#include <cstdint>
#include <string>
#include <utility>

namespace interaxon { namespace bridge {

struct Result final {
    ResultLevel level;
    std::string type;
    int32_t code;
    std::string info;

    Result(ResultLevel level_,
           std::string type_,
           int32_t code_,
           std::string info_)
    : level(std::move(level_))
    , type(std::move(type_))
    , code(std::move(code_))
    , info(std::move(info_))
    {}
};

} }  // namespace interaxon::bridge
```

## 9.100 bridge_result_level.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::ResultLevel >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::ResultLevel : int {
  interaxon::bridge::R_NONE ,
  interaxon::bridge::R_SUCCESS ,
  interaxon::bridge::R_INFO ,
  interaxon::bridge::R_WARN ,
  interaxon::bridge::R_ERROR ,
  interaxon::bridge::R_DEBUG }

## 9.101 bridge_result_level.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from validation.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class ResultLevel : int {
    R_NONE,
    R_SUCCESS,
    R_INFO,
    R_WARN,
    R_ERROR,
    R_DEBUG,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::ResultLevel> {
    size_t operator()(::interaxon::bridge::ResultLevel type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.102 bridge_severity.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::Severity >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::Severity : int {
  interaxon::bridge::SEV_VERBOSE ,
  interaxon::bridge::SEV_INFO ,
  interaxon::bridge::SEV_WARN ,
  interaxon::bridge::SEV_ERROR ,
  interaxon::bridge::SEV_FATAL ,
  interaxon::bridge::SEV_DEBUG ,
  interaxon::bridge::TOTAL }

## 9.103 bridge_severity.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from logging.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class Severity : int {
    SEV_VERBOSE,
    SEV_INFO,
    SEV_WARN,
    SEV_ERROR,
    SEV_FATAL,
    SEV_DEBUG,
    TOTAL,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::Severity> {
    size_t operator()(::interaxon::bridge::Severity type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.104 bridge_stringify.h File Reference

### Classes

- class interaxon::bridge::Stringify

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.105 bridge_stringify.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <memory>
#include <string>

namespace interaxon { namespace bridge {

enum class ConnectionState;
enum class MuseDataPacketType;

class Stringify {
public:
    virtual ~Stringify() {}

    static std::shared_ptr<Stringify> instance();

    virtual std::string packet_type(MuseDataPacketType type) = 0;

    virtual std::string connection_state(ConnectionState state) = 0;
};

} }  // namespace interaxon::bridge
```

## 9.106 bridge_timestamp_mode.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::TimestampMode >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::TimestampMode : int {
  interaxon::bridge::LEGACY ,
  interaxon::bridge::CURRENT ,
  interaxon::bridge::EXPLICIT }

## 9.107 bridge_timestamp_mode.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from muse_file.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class TimestampMode : int {
    LEGACY,
    CURRENT,
    EXPLICIT,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::TimestampMode> {
    size_t operator()(::interaxon::bridge::TimestampMode type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.108 bridge_ultra_violet.h File Reference

### Classes

- struct std::hash< ::interaxon::bridge::UltraViolet >

### Namespaces

- namespace interaxon
- namespace interaxon::bridge
- namespace std

  *STL namespace.*

### Enumerations

- enum class interaxon::bridge::UltraViolet : int {
  interaxon::bridge::UV_INDEX ,
  interaxon::bridge::UV_A ,
  interaxon::bridge::UV_B }

## 9.109 bridge_ultra_violet.h

Go to the documentation of this file.

```
// AUTOGENERATED FILE - DO NOT MODIFY!
// This file generated by Djinni from packets.djinni

#pragma once

#include <functional>

namespace interaxon { namespace bridge {

enum class UltraViolet : int {
    UV_INDEX,
    UV_A,
    UV_B,
};

} }  // namespace interaxon::bridge

namespace std {

template <>
struct hash<::interaxon::bridge::UltraViolet> {
    size_t operator()(::interaxon::bridge::UltraViolet type) const {
        return std::hash<int>()(static_cast<int>(type));
    }
};

}  // namespace std
```

## 9.110 event_loop_factory.h File Reference

### Classes

- class interaxon::bridge::EventLoopFactory

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.111 event_loop_factory.h

Go to the documentation of this file.

```
// Copyright 2016 InteraXon Inc.
#pragma once

#include <memory>

namespace interaxon {
namespace bridge {

class EventLoop;

class EventLoopFactory {

public:
    static std::shared_ptr<EventLoop> get_event_loop();

};

}  // namespace bridge
} // namespace interaxon
```

## 9.112 computing_device_configuration_factory.h File Reference

### Classes

- class interaxon::bridge::ComputingDeviceConfigurationFactory

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.113 computing_device_configuration_factory.h

Go to the documentation of this file.

```
// Copyright 2017 Interaxon Inc.
#pragma once

#include <memory>
#include "api/bridge_computing_device_configuration.h"

namespace interaxon {
    namespace bridge {

        class ComputingDeviceConfigurationFactory {
        public:

            static std::shared_ptr<ComputingDeviceConfigurationFactory> get_instance();

            ComputingDeviceConfigurationFactory(const ComputingDeviceConfigurationFactory& rhs) =
delete;
            ComputingDeviceConfigurationFactory& operator=(const ComputingDeviceConfigurationFactory&
rhs) = delete;

            ComputingDeviceConfiguration get_computing_device_configuration();

        private:
            ComputingDeviceConfigurationFactory();
        };

    } // namespace bridge
} // namespace interaxon
```

## 9.114 conversions.h File Reference

### Classes

- class interaxon::bridge::Convert

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.115 conversions.h

Go to the documentation of this file.

```
// Copyright 2016 Interaxon Inc.
#pragma once

#include <string>

namespace interaxon {
    namespace bridge {

        class Convert {
        public:
            static Platform::String^ to_platform_string(const std::string &str);

            static std::string to_std_string(Platform::String^ str);

        };

    }
}
```

## 9.116 muse_file_factory.h File Reference

### Classes

- class interaxon::bridge::MuseFileFactory

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.117 muse_file_factory.h

Go to the documentation of this file.

```
// Copyright 2016 Interaxon, Inc.
#pragma once

#include <memory>
#include <string>

namespace interaxon {
    namespace bridge {

        class MuseFile;
        class MuseFileWriter;
        class MuseFileReader;

        class MuseFileFactory {
        public:
            static std::shared_ptr<MuseFileWriter> get_muse_file_writer(const std::string& file_path);

            static std::shared_ptr<MuseFileReader> get_muse_file_reader(const std::string& file_path);


            static std::shared_ptr<MuseFile> get_muse_file(const std::string& file_path);

        };

    } // namespace bridge
} // namespace interaxon
```

## 9.118 muse_manager_windows.h File Reference

### Classes

- class interaxon::bridge::MuseManagerWindows

### Namespaces

- namespace interaxon
- namespace interaxon::bridge

## 9.119 muse_manager_windows.h

Go to the documentation of this file.

```
// Copyright 2016 Interaxon, Inc.
#pragma once

#include <memory>
#include "api/bridge_muse_manager.h"

namespace interaxon { namespace bridge {

class MuseManagerWindows: public MuseManager {

public:

    static std::shared_ptr<MuseManagerWindows> get_instance();

    virtual void set_recorder_info(const std::string& name, const std::string& version) = 0;
};

}
}
```