MilliSuono

Generated by Doxygen 1.15.0

Chapter 1

Directory Hierarchy

1.1 Directories

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Directory Documentation

5.1 include/core Directory Reference

Files

- file Node.hpp

  Defines the Node and Parameter structures for the MilliSuono system.
- file Port.hpp

  Defines the basic data structures for ports and events in the MilliSuono system.

5.2 include Directory Reference

Directories

- directory core

Chapter 6

Namespace Documentation

6.1 ms Namespace Reference

Classes

- struct Param

  Represents a named parameter of a Node.
- class Node

  Represents a processing unit in the MilliSuono graph.
- struct Event

  Represents a time-stamped event in the audio processing timeline.
- struct Port

  Represents an input or output port of a Node.

Typedefs

- using ControlValue = std::variant<float, int, bool, std::string>

  Represents the value carried by a control or event port.

Enumerations

- enum class PortType { Audio , Control , Event }

  Defines the possible types of ports in the MilliSuono system.

6.1.1 Typedef Documentation

6.1.1.1 ControlValue

```
using ms::ControlValue = std::variant<float, int, bool, std::string>
```

Represents the value carried by a control or event port.

This can be one of the following:

- float: for continuous parameters (e.g., gain, frequency)
- int: for discrete parameters or indices
- bool: for binary control signals (e.g., mute, toggle)
- std::string: for symbolic or textual data

6.1.2 Enumeration Type Documentation

6.1.2.1 PortType

```
enum class ms::PortType  [strong]
```

Defines the possible types of ports in the MilliSuono system.

- Audio: for audio signal connections
- Control: for control parameters (float, int, bool, string)
- Event: for time-stamped control or trigger events

Enumerator

Audio	
Control	
Event	

Chapter 7

Class Documentation

7.1 ms::Event Struct Reference

Represents a time-stamped event in the audio processing timeline.

`#include <Port.hpp>`

Public Member Functions

- Event (const std::string &type, const ControlValue &value, int sampleOffset)

  Constructs an Event object.

Public Attributes

- std::string type
- ControlValue value
- int sampleOffset

7.1.1 Detailed Description

Represents a time-stamped event in the audio processing timeline.

Events are typically generated by control sources (e.g., user interaction, automation, or MIDI input) and scheduled at a specific sample offset within a processing block.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Event()

```
ms::Event::Event (
            const std::string & type,
            const ControlValue & value,
            int sampleOffset)  [inline]
```

Constructs an Event object.

Parameters

type	The event type identifier.
value	The payload associated with the event.
sampleOffset	The sample index relative to the start of the processing block.

7.1.3 Member Data Documentation

7.1.3.1 sampleOffset

```
int ms::Event::sampleOffset
```

The sample offset within the current processing block at which the event occurs.

7.1.3.2 type

```
std::string ms::Event::type
```

Type or category of the event (e.g., "note_on", "param_change").

7.1.3.3 value

```
ControlValue ms::Event::value
```

The event payload, which can be any supported ControlValue type.

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

- include/core/Port.hpp

7.2 ms::Node Class Reference

Represents a processing unit in the MilliSuono graph.

```
#include <Node.hpp>
```

Public Member Functions

- Node (const std::string &id)

  Constructs a Node with a given identifier.
- virtual ~Node ()=default

  Virtual destructor for proper cleanup in derived classes.
- const std::string & getId () const

  Returns the unique identifier of the Node.
- const std::vector< Param > & getParams () const

  Returns the list of parameters associated with the Node (read-only).
- std::vector< Param > getParams ()

  Returns the list of parameters associated with the Node (mutable).
- const ControlValue * getParam (const std::string &name) const

  Retrieves a parameter value by name.
- void setParams (const std::vector< Param > &newParams)

  Sets the parameters of the Node.
- bool setParam (const std::string &name, const ControlValue &value)

  Sets a parameter value by name.
- float getFadeInDuration () const

  Gets the fade-in duration in milliseconds.
- void setFadeInDuration (float durationMs)

  Sets fade-in duration in milliseconds.
- void resetFadeIn ()

  Resets the fade-in effect to start from the beginning (useful when re-activating a node).
- const std::vector< Port > & getInputPorts () const

  Returns the list of input ports for the Node.
- const std::vector< Port > & getOutputPorts () const

  Returns the list of output ports for the Node.
- virtual void prepare (int sampleRate, int blockSize)

  Prepares the Node for processing. This function initializes the Node with the given sample rate and block size.
- virtual void process (const float *const *inputsm float **outputs, int nFrames)=0

  Processes audio data for the Node. This is a pure virtual function that must be implemented by subclasses.
- virtual void processControl (const std::unordered_map< std::string, ControlValue > &inputControls, std::unordered_map< std::string, ControlValue > &outputControls)

  Processes control data for the Node. Subclasses can override this to handle control data.
- virtual void processEvent (const std::unordered_map< std::string, Event > &inputEvents, std::unordered_map< std::string, Event > &outputEvents)

  Processes events for the Node. Subclasses can override this to handle events.

Protected Member Functions

- void applyFadeIn (float *buffer, int nFrames)

  Applies fade-in envelope to an audio buffer Subclasses should call this at the end of process() on their output buffers.
- void addInputPort (const std::string &name, PortType type)

  Adds an input port to the Node.
- void addOutputPort (const std::string &name, PortType type)

  Adds an output port to the Node.
- const float * getPhysicalInput (int channelIndex) const

  Get physical audio input from hardware For nodes that need direct hardware access (e.g., audio input nodes).

Protected Attributes

- std::vector< Port > inputPorts_
- std::vector< Port > outputPorts_
- int sampleRate_ = 44100
- int blockSize_ = 512

7.2.1 Detailed Description

Represents a processing unit in the MilliSuono graph.

A Node defines a functional unit (e.g. an oscillator, filter, or mixer) with a unique identifier and a set of configurable parameters. Nodes can be connected via Ports to form complex audio processing graphs.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Node()

```
ms::Node::Node (
            const std::string & id)  [inline]
```

Constructs a Node with a given identifier.

Parameters

id	The unique string identifier for the Node.

7.2.2.2 ~Node()

```
virtual ms::Node::~Node ()  [virtual], [default]
```

Virtual destructor for proper cleanup in derived classes.

7.2.3 Member Function Documentation

7.2.3.1 addInputPort()

```
void ms::Node::addInputPort (
            const std::string & name,
            PortType type)  [inline], [protected]
```

Adds an input port to the Node.

Parameters

name	The name of the input port.
type	The type of the input port.

7.2.3.2 addOutputPort()

```
void ms::Node::addOutputPort (
            const std::string & name,
            PortType type)  [inline], [protected]
```

Adds an output port to the Node.

Parameters

name	The name of the output port.
type	The type of the output port.

7.2.3.3 applyFadeIn()

```
void ms::Node::applyFadeIn (
            float * buffer,
            int nFrames)  [protected]
```

Applies fade-in envelope to an audio buffer Subclasses should call this at the end of process() on their output buffers.

Parameters

buffer	The audio buffer to apply fade-in to.
nFrames	The number of frames in the buffer.

7.2.3.4 getFadeInDuration()

```
float ms::Node::getFadeInDuration () const  [inline]
```

Gets the fade-in duration in milliseconds.

Returns

The fade-in duration in milliseconds.

7.2.3.5 getId()

```
const std::string & ms::Node::getId () const  [inline]
```

Returns the unique identifier of the Node.

Returns

The Node's identifier string.

7.2.3.6 getInputPorts()

```
const std::vector< Port > & ms::Node::getInputPorts () const  [inline]
```

Returns the list of input ports for the Node.

Returns

A const reference to the vector of input Ports.

7.2.3.7 getOutputPorts()

```
const std::vector< Port > & ms::Node::getOutputPorts () const  [inline]
```

Returns the list of output ports for the Node.

Returns

A const reference to the vector of output Ports.

7.2.3.8 getParam()

```
const ControlValue * ms::Node::getParam (
            const std::string & name) const  [inline]
```

Retrieves a parameter value by name.

Parameters

name	The name of the parameter to retrieve.

Returns

A pointer to the ControlValue if found, nullptr otherwise.

7.2.3.9 getParams() [1/2]

```
std::vector< Param > ms::Node::getParams ()  [inline]
```

Returns the list of parameters associated with the Node (mutable).

Returns

A const reference to the vector of Params.

7.2.3.10 getParams() [2/2]

```
const std::vector< Param > & ms::Node::getParams () const  [inline]
```

Returns the list of parameters associated with the Node (read-only).

Returns

A const reference to the vector of Params.

7.2.3.11 getPhysicalInput()

```
const float * ms::Node::getPhysicalInput (
            int channelIndex) const  [protected]
```

Get physical audio input from hardware For nodes that need direct hardware access (e.g., audio input nodes).

Parameters

channelIndex	The Physical channel index to read from

Returns

Pointer to the float buffer of the physical input channel or nullptr

7.2.3.12 prepare()

```
virtual void ms::Node::prepare (
            int sampleRate,
            int blockSize)  [inline], [virtual]
```

Prepares the Node for processing. This function initializes the Node with the given sample rate and block size.

Parameters

sampleRate	The sample rate in Hz.
blockSize	The block size in samples.

7.2.3.13 process()

```
virtual void ms::Node::process (
            const float *const *inputsm float ** outputs,
            int nFrames)  [pure virtual]
```

Processes audio data for the Node. This is a pure virtual function that must be implemented by subclasses.

Parameters

inputs	An array of input audio buffers.
outputs	An array of output audio buffers.
nFrames	The number of frames to process.

7.2.3.14 processControl()

```
virtual void ms::Node::processControl (
            const std::unordered_map< std::string, ControlValue > & inputControls,
            std::unordered_map< std::string, ControlValue > & outputControls)  [inline],
[virtual]
```

Processes control data for the Node. Subclasses can override this to handle control data.

Parameters

inputControls	A map of input control values.
outputControls	A map to store output control values.

7.2.3.15 processEvent()

```
virtual void ms::Node::processEvent (
            const std::unordered_map< std::string, Event > & inputEvents,
            std::unordered_map< std::string, Event > & outputEvents)  [inline], [virtual]
```

Processes events for the Node. Subclasses can override this to handle events.

Parameters

inputEvents	A map of input events.
outputEvents	A map to store output events.

7.2.3.16 resetFadeIn()

```
void ms::Node::resetFadeIn ()  [inline]
```

Resets the fade-in effect to start from the beginning (useful when re-activating a node).

7.2.3.17 setFadeInDuration()

```
void ms::Node::setFadeInDuration (
            float durationMs)  [inline]
```

Sets fade-in duration in milliseconds.

Parameters

durationMs	The fade-in duration in milliseconds (0 = disabled, default = 50ms).

7.2.3.18 setParam()

```
bool ms::Node::setParam (
            const std::string & name,
            const ControlValue & value)  [inline]
```

Sets a parameter value by name.

Parameters

name	The name of the parameter to set.
value	The new value to assign to the parameter.

Returns

True if the parameter was found and set, false otherwise.

7.2.3.19 setParams()

```
void ms::Node::setParams (
            const std::vector< Param > & newParams)  [inline]
```

Sets the parameters of the Node.

Parameters

newParams	A vector of Params to set for the Node.

7.2.4 Member Data Documentation

7.2.4.1 blockSize_

```
int ms::Node::blockSize_ = 512  [protected]
```

The block size for processing audio samples.

7.2.4.2 inputPorts_

```
std::vector<Port> ms::Node::inputPorts_  [protected]
```

The list of input ports for the Node.

7.2.4.3 outputPorts_

```
std::vector<Port> ms::Node::outputPorts_  [protected]
```

The list of output ports for the Node.

7.2.4.4 sampleRate_

```
int ms::Node::sampleRate_ = 44100  [protected]
```

The sample rate at which the Node operates.

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

- include/core/Node.hpp

7.3 ms::Param Struct Reference

Represents a named parameter of a Node.

```
#include <Node.hpp>
```

Public Member Functions

- Param (const std::string &paramName, const ControlValue &paramValue)
  Constructs a Param with a given name and value.

Public Attributes

- std::string name
- ControlValue value

7.3.1 Detailed Description

Represents a named parameter of a Node.

A parameter stores a name and a corresponding ControlValue. Parameters can represent any configurable property such as gain, frequency, or mode.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 Param()

```
ms::Param::Param (
            const std::string & paramName,
            const ControlValue & paramValue)  [inline]
```

Constructs a Param with a given name and value.

Parameters

paramName	The name of the parameter.
paramValue	The value of the parameter.

7.3.3 Member Data Documentation

7.3.3.1 name

```
std::string ms::Param::name
```

The unique name identifying the parameter.

7.3.3.2 value

```
ControlValue ms::Param::value
```

The current value of the parameter.

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

- include/core/Node.hpp

7.4 ms::Port Struct Reference

Represents an input or output port of a Node.

```
#include <Port.hpp>
```

Public Member Functions

- Port (const std::string &name, PortType type)
  Constructs a Port object.

Public Attributes

- std::string name
- PortType type

7.4.1 Detailed Description

Represents an input or output port of a Node.

Ports define the interface through which nodes exchange audio, control, or event data in the MilliSuono engine.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Port()

```
ms::Port::Port (
            const std::string & name,
            PortType type)  [inline]
```

Constructs a Port object.

Parameters

name	The name identifying the port.
type	The port type (Audio, Control, or Event).

7.4.3 Member Data Documentation

7.4.3.1 name

```
std::string ms::Port::name
```

The unique name of the port within a node.

7.4.3.2 type

```
PortType ms::Port::type
```

The type of the port (Audio, Control, or Event).

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

- include/core/Port.hpp

Chapter 8

File Documentation

8.1 include/core/Node.hpp File Reference

Defines the Node and Parameter structures for the MilliSuono system.

```
#include "Port.hpp"
#include <string>
#include <unordered_map>
#include <vector>
```

Classes

- struct ms::Param

  Represents a named parameter of a Node.
- class ms::Node

  Represents a processing unit in the MilliSuono graph.

Namespaces

- namespace ms

8.1.1 Detailed Description

Defines the Node and Parameter structures for the MilliSuono system.

It provides the core data strctures for representing processing units and their configurable parameters within the MilliSuono framework.

8.2 Node.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "Port.hpp"
00003 #include <string>
00004 #include <unordered_map>
00005 #include <vector>
00006
00014
00015 namespace ms {
00016
00024 struct Param {
00026   std::string name;
00027
00029   ControlValue value;
00030
00036   Param(const std::string &paramName, const ControlValue &paramValue)
00037       : name(paramName), value(paramValue) {}
00038 };
00039
00047 class Node {
00048 public:
00053   Node(const std::string &id) : id_(id) {}
00054
00058   virtual ~Node() = default;
00059
00064   const std::string &getId() const { return id_; }
00065
00070   const std::vector<Param> &getParams() const { return params_; }
00071
00076   std::vector<Param> getParams() { return params_; }
00077
00083   const ControlValue *getParam(const std::string &name) const {
00084     for (const auto &param : params_) {
00085       if (param.name == name) {
00086         return &param.value;
00087       }
00088     }
00089     return nullptr;
00090   }
00091
00096   void setParams(const std::vector<Param> &newParams) { params_ = newParams; }
00097
00104   bool setParam(const std::string &name, const ControlValue &value) {
00105     for (auto &param : params_) {
00106       if (param.name == name) {
00107         param.value = value;
00108         return true;
00109       }
00110     }
00111     return false;
00112   }
00113
00118   float getFadeInDuration() const { return fadeInDurationMs_; }
00119
00125   void setFadeInDuration(float durationMs) {
00126     fadeInDurationMs_ = durationMs;
00127     updateFadeInSamples();
00128   }
00129
00134   void resetFadeIn() {
00135     currentFadeInSample_ = 0;
00136     fadeInActive_ = (fadeInDurationMs_ > 0.0f);
00137   }
00138
00143   const std::vector<Port> &getInputPorts() const { return inputPorts_; }
00144
00149   const std::vector<Port> &getOutputPorts() const { return outputPorts_; }
00150
00158   virtual void prepare(int sampleRate, int blockSize) {
00159     sampleRate_ = sampleRate;
00160     blockSize_ = blockSize;
00161     updateFadeInSamples();
00162     currentFadeInSample_ = 0;
00163     fadeInActive_ = (fadeInDurationMs_ > 0.0f);
00164   }
00165
00173   virtual void process(const float *const *inputsm float **outputs, int nFrames) = 0;
00174
00181   virtual void processControl(
00182       const std::unordered_map<std::string, ControlValue> &inputControls,
00183       std::unordered_map<std::string, ControlValue> &outputControls) {}
00190   virtual void processEvent(
00191       const std::unordered_map<std::string, Event> &inputEvents,
```

```
00192       std::unordered_map<std::string, Event> &outputEvents) {}
00193
00194 protected:
00201   void applyFadeIn(float *buffer, int nFrames);
00202
00208   void addInputPort(const std::string &name, PortType type) {
00209     inputPorts_.push_back(Port(name, type));
00210   }
00211
00217   void addOutputPort(const std::string &name, PortType type) {
00218     outputPorts_.push_back(Port(name, type));
00219   }
00220
00228   const float *getPhysicalInput(int channelIndex) const;
00229
00231   std::vector<Port> inputPorts_;
00232
00234   std::vector<Port> outputPorts_;
00235
00237   int sampleRate_ = 44100;
00239   int blockSize_ = 512;
00240
00241 private:
00243   const std::string id_;
00244
00246   std::vector<Param> params_;
00247
00249   float fadeInDurationMs_ = 50.0f;
00251   int fadeInSamples_ = 0;
00253   int currentFadeInSample_ = 0;
00255   bool fadeInActive_ = false;
00256
00261   void updateFadeInSamples() {
00262     fadeInSamples_ = static_cast<int>((fadeInDurationMs_ / 1000.0f) *
00263                                       static_cast<float>(sampleRate_));
00264   }
00265 };
00266
00267 } // namespace ms
```

8.3 include/core/Port.hpp File Reference

Defines the basic data structures for ports and events in the MilliSuono system.

```
#include <string>
#include <variant>
```

Classes

- struct ms::Event

  Represents a time-stamped event in the audio processing timeline.
- struct ms::Port

  Represents an input or output port of a Node.

Namespaces

- namespace ms

Typedefs

- using ms::ControlValue = std::variant<float, int, bool, std::string>

  Represents the value carried by a control or event port.

Enumerations

- enum class ms::PortType { ms::Audio , ms::Control , ms::Event }

  Defines the possible types of ports in the MilliSuono system.

8.3.1 Detailed Description

Defines the basic data structures for ports and events in the MilliSuono system.

This file declares the fundamental types used for representing audio, control, and event connections within the MilliSuono framework.

8.4 Port.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <string>
00003 #include <variant>
00004
00013
00014 namespace ms {
00015
00023 enum class PortType { Audio, Control, Event };
00024
00034 using ControlValue = std::variant<float, int, bool, std::string>;
00035
00043 struct Event {
00045   std::string type;
00046
00048   ControlValue value;
00049
00052   int sampleOffset;
00053
00061   Event(const std::string &type, const ControlValue &value, int sampleOffset)
00062       : type(type), value(value), sampleOffset(sampleOffset) {}
00063 };
00064
00071 struct Port {
00073   std::string name;
00074
00076   PortType type;
00077
00083   Port(const std::string &name, PortType type) : name(name), type(type) {}
00084 };
00085
00086 } // namespace ms
```

Index