PreSonus Plug-in Extensions

PreSonus Software Ltd.

Mar 09, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 Feature Overview

PreSonus Plug-in Extensions provide the following features:

- Support for *Sound Variations* (Studio One 5.2 and later)
- Support for *Instrument Extensions* (Studio One 5.2 and later)
- Query key switches from VST2 instrument plug-ins (Studio One 4.5 and later)
- Notifications for cloned plug-in instances in low-latency processing mode (Studio One 3.5.2 and later)
- Remote control channel volume, mute, solo, etc. from plug-in (Studio One 3.2.1 and later)
- Gain reduction reporting to the host application (Studio One 3 and later)
- High DPI scaling notifications for plug-in GUIs on Windows (Studio One 3 and later)
- Access context information like channel names, colors, etc. (Studio One 2.6 and later)
- Per-parameter commands for audio plug-ins related to automation and mapping to control surfaces
- Support for embedding plug-in views into the single-window environment of Studio One
- Support for sizable plug-in editors in VST2 using a similar mimic to VST3

1.2 DISCLAIMER

PreSonus Plug-In Extensions are host-specific extensions of existing proprietary technologies, provided to the community on an AS IS basis. They are not part of any official 3rd-party SDK and PreSonus is not affiliated with the owner of the underlying technology in any way.

CHAPTER

TWO

API REFERENCE

2.1 Context Information

2.1.1 struct Presonus::IContextInfoHandler

Overview

Notification interface for context information changes. *More...*

```
#include <ipslcontextinfo.h>

struct IContextInfoHandler: public FUnknown

    // methods

    virtual void notifyContextInfoChange() = 0;
;
```

Detailed Documentation

Notification interface for context information changes.

Implemented by the plug-in as extension of Steinberg::Vst::IEditController.

Methods

```
virtual void notifyContextInfoChange() = 0
```

Called by the host if context information has changed.

2.1.2 struct Presonus::IContextInfoHandler2

Overview

Replacement of *IContextInfoHandler* passing additional information about what changed on the host-side. *More...*

```
#include <ipslcontextinfo.h>

struct IContextInfoHandler2: public FUnknown

    // methods

    virtual void notifyContextInfoChange(Steinberg::FIDString id) = 0;
;
```

Detailed Documentation

Replacement of *IContextInfoHandler* passing additional information about what changed on the host-side.

This interface will be preferred if implemented by the plug-in. It is required to receive certain notifications like volume, pan, etc.

Methods

```
virtual void notifyContextInfoChange(Steinberg::FIDString id) = 0
```

Called by the host if context information has changed.

The identifier (id) is empty for the inital update.

2.1.3 struct Presonus::IContextInfoProvider

Overview

Callback interface to access context information from the host. *More...*

```
#include <ipslcontextinfo.h>

struct IContextInfoProvider: public FUnknown

    // methods

    virtual Steinberg::tresult getContextInfoValue(
        Steinberg::int32& value,
        Steinberg::FIDString id
        ) = 0;

    virtual Steinberg::tresult getContextInfoString(
```

```
        Steinberg::Vst::TChar* string,
        Steinberg::int32 maxCharCount,
        Steinberg::FIDString id
        ) = 0;
;

// direct descendants

struct IContextInfoProvider2;
```

Detailed Documentation

Callback interface to access context information from the host.

Implemented by the host as extension of Steinberg::Vst::IComponentHandler.

The host might not be able to report all available attributes at all times. Please check the return value of *getContextInfoValue()* and *getContextInfoString()*. It's not required to implement *IContextInfoHandler* on the plug-in side, but we recommend to do so. The host will then call notifyContextInfoChange() during initialization to inform the plug-in about the initial state of the available attributes.

Usage Example:

```
IComponentHandler* handler;
FUnknownPtr<IContextInfoProvider> contextInfoProvider (handler);

void PLUGIN_API MyEditController::notifyContextInfoChange ()

    int32 channelIndex = 0;
    contextInfoProvider->getContextInfoValue (channelIndex, ContextInfo::kIndex);

    TChar channelName[128] = 0;
    contextInfoProvider->getContextInfoString (channelName, 128, ContextInfo::kName);
```

Methods

```
virtual Steinberg::tresult getContextInfoValue(
    Steinberg::int32& value,
    Steinberg::FIDString id
    ) = 0
```

Get context information by identifier.

```
virtual Steinberg::tresult getContextInfoString(
    Steinberg::Vst::TChar* string,
    Steinberg::int32 maxCharCount,
    Steinberg::FIDString id
    ) = 0
```

Get context information by identifier.

2.1.4 struct Presonus::IContextInfoProvider2

Overview

Extension to *IContextInfoProvider* enabling the plug-in to modify host context information. *More...*

```
#include <ipslcontextinfo.h>

struct IContextInfoProvider2: public Presonus::IContextInfoProvider

    // methods

    virtual Steinberg::tresult getContextInfoValue(
        double& value,
        Steinberg::FIDString id
        ) = 0;

    virtual Steinberg::tresult setContextInfoValue(
        Steinberg::FIDString id,
        double value
        ) = 0;

    virtual Steinberg::tresult setContextInfoValue(
        Steinberg::FIDString id,
        Steinberg::int32 value
        ) = 0;

    virtual Steinberg::tresult setContextInfoString(
        Steinberg::FIDString id,
        Steinberg::Vst::TChar* string
        ) = 0;

    Steinberg::tresult getContextInfoValue(
        Steinberg::int32& value,
        Steinberg::FIDString id
        );
;

// direct descendants

struct IContextInfoProvider3;
```

Inherited Members

```
public:
    // methods

    virtual Steinberg::tresult getContextInfoValue(
        Steinberg::int32& value,
        Steinberg::FIDString id
        ) = 0;

    virtual Steinberg::tresult getContextInfoString(
        Steinberg::Vst::TChar* string,
        Steinberg::int32 maxCharCount,
        Steinberg::FIDString id
        ) = 0;
```

Detailed Documentation

Extension to *IContextInfoProvider* enabling the plug-in to modify host context information.

Values like volume or pan support both, numeric and string representation for get and set.

Methods

```
virtual Steinberg::tresult getContextInfoValue(
    double& value,
    Steinberg::FIDString id
    ) = 0
```

Get context information by identifier (floating-point).

```
virtual Steinberg::tresult setContextInfoValue(
    Steinberg::FIDString id,
    double value
    ) = 0
```

Set context information by identifier (floating-point).

```
virtual Steinberg::tresult setContextInfoValue(
    Steinberg::FIDString id,
    Steinberg::int32 value
    ) = 0
```

Set context information by identifier (integer).

```
virtual Steinberg::tresult setContextInfoString(
    Steinberg::FIDString id,
    Steinberg::Vst::TChar* string
    ) = 0
```

Set context information by identifier (string).

```
Steinberg::tresult getContextInfoValue(
    Steinberg::int32& value,
    Steinberg::FIDString id
    )
```

Get context information by identifier.

2.1.5 struct Presonus::IContextInfoProvider3

Overview

Extension to *IContextInfoProvider* and *IContextInfoProvider2*. *More...*

```
#include <ipslcontextinfo.h>

struct IContextInfoProvider3: public Presonus::IContextInfoProvider2

    // methods
```

```
    virtual Steinberg::tresult beginEditContextInfoValue(Steinberg::FIDString id) = 0;
    virtual Steinberg::tresult endEditContextInfoValue(Steinberg::FIDString id) = 0;
;
```

Inherited Members

```
public:
    // methods

    virtual Steinberg::tresult getContextInfoValue(
        Steinberg::int32& value,
        Steinberg::FIDString id
        ) = 0;

    virtual Steinberg::tresult getContextInfoString(
        Steinberg::Vst::TChar* string,
        Steinberg::int32 maxCharCount,
        Steinberg::FIDString id
        ) = 0;

    virtual Steinberg::tresult getContextInfoValue(
        double& value,
        Steinberg::FIDString id
        ) = 0;

    virtual Steinberg::tresult setContextInfoValue(
        Steinberg::FIDString id,
        double value
        ) = 0;

    virtual Steinberg::tresult setContextInfoValue(
        Steinberg::FIDString id,
        Steinberg::int32 value
        ) = 0;

    virtual Steinberg::tresult setContextInfoString(
        Steinberg::FIDString id,
        Steinberg::Vst::TChar* string
        ) = 0;

    Steinberg::tresult getContextInfoValue(
        Steinberg::int32& value,
        Steinberg::FIDString id
        );
```

Detailed Documentation

Extension to *IContextInfoProvider* and *IContextInfoProvider2*.

Enable the plug-in to signal begin and end of editing for context information values. Use IComponentHandler2::startGroupEdit() and IComponentHandler2::endGroupEdit() to signal group edits.

Methods

```
virtual Steinberg::tresult beginEditContextInfoValue(Steinberg::FIDString id) = 0
```

Begin edit of context info value,.

See also:

also IComponentHandler::beginEdit.

```
virtual Steinberg::tresult endEditContextInfoValue(Steinberg::FIDString id) = 0
```

End edit of context info value,.

See also:

also IComponentHandler::endEdit.

```
// structs

struct Presonus::IContextInfoHandler;
struct Presonus::IContextInfoHandler2;
struct Presonus::IContextInfoProvider;
struct Presonus::IContextInfoProvider2;
struct Presonus::IContextInfoProvider3;
```

2.2 Edit Controller Extensions

2.2.1 enum Presonus::AutomationMode

Overview

Automation mode. *More...*

```
#include <ipsleditcontroller.h>

enum AutomationMode

    kAutomationNone  = 0,
    kAutomationOff,
    kAutomationRead,
    kAutomationTouch,
    kAutomationLatch,
    kAutomationWrite,
;
```

Detailed Documentation

Automation mode.

Used with *IEditControllerExtra*.

Enum Values

`kAutomationNone`

no automation data available

`kAutomationOff`

data available, but mode is set to off

`kAutomationRead`

data + read mode

`kAutomationTouch`

data + touch mode

`kAutomationLatch`

data + latch mode

`kAutomationWrite`

data + write mode

2.2.2 enum Presonus::ParamExtraFlags

Overview

Parameter extra flags. *More...*

```
#include <ipsleditcontroller.h>

enum ParamExtraFlags

    kParamFlagMicroEdit = 1<<0,
;
```

Detailed Documentation

Parameter extra flags.

Used with *IEditControllerExtra*.

Enum Values

```
kParamFlagMicroEdit
```

parameter should be displayed in host micro view

2.2.3 enum Presonus::SlaveMode

Overview

Slave mode. *More...*

```
#include <ipsleditcontroller.h>

enum SlaveMode

    kSlaveModeNormal,
    kSlaveModeLowLatencyClone,
;
```

Detailed Documentation

Slave mode.

Used with *ISlaveControllerHandler*.

Enum Values

```
kSlaveModeNormal
```

plug-in used in different location following given master

```
kSlaveModeLowLatencyClone
```

plug-in used as hidden slave for low latency processing following given master

2.2.4 struct Presonus::IEditControllerExtra

Overview

Extension to Steinberg::Vst::IEditController with additonal flags and notifications not available in the standard edit controller interface. *More...*

```
#include <ipslediteontroller.h>

struct IEditControllerExtra: public FUnknown

    // methods

    virtual Steinberg::int32 getParamExtraFlags(Steinberg::Vst::ParamID id) = 0;

    virtual Steinberg::tresult setParamAutomationMode(
        Steinberg::Vst::ParamID id,
        Steinberg::int32 automationMode
        ) = 0;
;
```

Detailed Documentation

Extension to Steinberg::Vst::IEditController with additonal flags and notifications not available in the standard edit controller interface.

Methods

```
virtual Steinberg::int32 getParamExtraFlags(Steinberg::Vst::ParamID id) = 0
```

Get extra flags for given parameter (see ParamExtraFlags).

```
virtual Steinberg::tresult setParamAutomationMode(
    Steinberg::Vst::ParamID id,
    Steinberg::int32 automationMode
    ) = 0
```

Set automation mode for given parameter (see AutomationMode).

2.2.5 struct Presonus::IGainReductionInfo

Overview

Interface to report gain reduction imposed to the audio signal by the plug-in to the host for display in the UI. *More...*

```
#include <ipslgainreduction.h>

struct IGainReductionInfo: public FUnknown
```

```
    // methods

    virtual double getGainReductionValueInDb() = 0;
;
```

Detailed Documentation

Interface to report gain reduction imposed to the audio signal by the plug-in to the host for display in the UI.

Implemented by the VST3 edit controller class.

Methods

```
virtual double getGainReductionValueInDb() = 0
```

Get current gain reduction for display.

The returned value in dB is either 0.0 (no reduction) or negative. The host calls this function periodically while the plug-in is active. The value is used AS IS for UI display purposes, without imposing additional ballistics or presentation latency compensation. Be sure to return zero if processing is bypassed internally. For multiple reduction stages, please report the sum in dB here.

2.2.6 struct Presonus::ISlaveControllerHandler

Overview

Extension to Steinberg::Vst::IEditController used to notify the plug-in about slave instances. *More...*

```
#include <ipsleditcontroller.h>

struct ISlaveControllerHandler: public FUnknown

    // methods

    virtual Steinberg::tresult addSlave(
        Steinberg::Vst::IEditController* slave,
        Steinberg::int32 slaveMode
        ) = 0;

    virtual Steinberg::tresult removeSlave(Steinberg::Vst::IEditController* slave) = 0;
;
```

Detailed Documentation

Extension to Steinberg::Vst::IEditController used to notify the plug-in about slave instances.

The host might decide to use "cloned" (slave) instances in various scenarios, e.g. to process audio paths with different latencies simultaneously or to synchronize grouped plug-in instances between multiple mixer channels - see SlaveMode. In this case multiple plug-in instances are active at the same time even though it looks like one to the user, i.e. only the editor of the master instance is visible and can be used to change parameters. The edit controller implementation has to synchronize parameter changes between instances that aren't visible to the host internally.

Methods

```
virtual Steinberg::tresult addSlave(
    Steinberg::Vst::IEditController* slave,
    Steinberg::int32 slaveMode
    ) = 0
```

Add slave edit controller.

Implementation must sync non-automatable parameters between this instance (master) and given slave instance internally, i.e. when the master (this) changes update all connected slaves.

```
virtual Steinberg::tresult removeSlave(Steinberg::Vst::IEditController* slave) = 0
```

Remove slave edit controller.

```
// enums

enum Presonus::AutomationMode;
enum Presonus::ParamExtraFlags;
enum Presonus::SlaveMode;

// structs

struct Presonus::IEditControllerExtra;
struct Presonus::IGainReductionInfo;
struct Presonus::ISlaveControllerHandler;
```

2.3 Host Commands

2.3.1 struct Presonus::CommandInfo

enum Presonus::CommandInfo::CommandFlags

Overview

```
#include <ipslhostcommands.h>

enum CommandFlags

    kCanExecute  = 1<<0,
    kIsSeparator = 1<<1,
    kIsChecked   = 1<<2,
;
```

Detailed Documentation

Enum Values

```
kCanExecute
```

used to display command enabled/disabled

```
kIsSeparator
```

not a command, it's a separator

```
kIsChecked
```

used to display command with a check mark

Overview

Describes a single command. *More...*

```
#include <ipslhostcommands.h>

struct CommandInfo

    // enums

    enum CommandFlags;

    // fields

    Steinberg::Vst::String128 title;
    Steinberg::int32 flags;
;
```

Detailed Documentation

Describes a single command.

Fields

```
Steinberg::Vst::String128 title
```

command title (possibly localized into active host language)

```
Steinberg::int32 flags
```

command flags

2.3.2 struct Presonus::ICommandList

Overview

Describes a list of commands. *More...*

```
#include <ipslhostcommands.h>

struct ICommandList: public FUnknown

    // methods

    virtual Steinberg::int32 getCommandCount() = 0;

    virtual Steinberg::tresult getCommandInfo(
        Steinberg::int32 index,
        CommandInfo& info
        ) = 0;

    virtual Steinberg::tresult executeCommand(Steinberg::int32 index) = 0;
;
```

Detailed Documentation

Describes a list of commands.

Methods

```
virtual Steinberg::int32 getCommandCount() = 0
```

Returns the number of commands.

```
virtual Steinberg::tresult getCommandInfo(
    Steinberg::int32 index,
    CommandInfo& info
    ) = 0
```

Get command information for a given index.

```
virtual Steinberg::tresult executeCommand(Steinberg::int32 index) = 0
```

Execute command at given index.

2.3.3 struct Presonus::IHostCommandHandler

Overview

Callback interface to access host-specific parameter commands to be integrated into a context menu inside the plug-in editor. *More...*

```
#include <ipslhostcommands.h>

struct IHostCommandHandler: public FUnknown

    // methods

    virtual ICommandList* createParamCommands(Steinberg::Vst::ParamID tag) = 0;

    virtual Steinberg::tresult popupCommandMenu(
        ICommandList* commandList,
        Steinberg::int32 xPos,
        Steinberg::int32 yPos,
        Steinberg::IPlugView* view = 0
        ) = 0;
;
```

Detailed Documentation

Callback interface to access host-specific parameter commands to be integrated into a context menu inside the plug-in editor.

Implemented as extension of Steinberg::Vst::IComponentHandler.

Please note that the intention of this set of interfaces is not to allow a generic menu implementation. This is the responsibility of a GUI toolkit. It basically provides a way to enumerate and execute commands anonymously, i.e. the plug-in does not have to know the exact sematics of the commands and the host does not break the consistency of the plug-in GUI.

Usage Example:

```
IComponentHandler* handler;
FUnknownPtr<IHostCommandHandler> commandHandler (handler);
if(commandHandler)
    if(ICommandList* commandList = commandHandler->createParamCommands (kMyParamId))

        FReleaser commandListReleaser (commandList);
        commandHandler->popupCommandMenu (commandList, xPos, yPos);
```

Methods

```
virtual ICommandList* createParamCommands(Steinberg::Vst::ParamID tag) = 0
```

Create list of currently available host commands for given parameter.

The command list has a short lifecycle, it is recreated whenever a context menu should appear. The returned pointer can be null, otherwise it has to be released.

```
virtual Steinberg::tresult popupCommandMenu(
    ICommandList* commandList,
    Steinberg::int32 xPos,
    Steinberg::int32 yPos,
    Steinberg::IPlugView* view = 0
    ) = 0
```

Helper to popup a command menu at given position.

Coordinates are relative to view or in screen coordintes if view is null. Can be used for testing purpose, if the plug-in does not have its own context menu implementation or if it wants to use the look & feel of the host menu. This method is not supposed to support command lists implemented by the plug-in.

```
// structs

struct Presonus::CommandInfo;
struct Presonus::ICommandList;
struct Presonus::IHostCommandHandler;
```

2.4 Instrument Extensions

2.4.1 struct Presonus::IInstrumentController

enum Presonus::IInstrumentController::Feature

Overview

```
#include <ipslinstrumentcontroller.h>

enum Feature

    kReportKeyAssignment  = 1,
    kReportDrumInstrument = 2,
    kReportMiddleC        = 3,
    kModifyMiddleC        = 4,
;
```

Detailed Documentation

Enum Values

```
kReportKeyAssignment
```

getKeyAssignment is supported

```
kReportDrumInstrument
```

isDrumInstrument is supported

```
kReportMiddleC
```

middle C can be queried

```
kModifyMiddleC
```

middle C can be set via setMiddleCValue

Overview

Extension to Steinberg::Vst::IEditController. *More...*

```
#include <ipslinstrumentcontroller.h>

struct IInstrumentController: public FUnknown

    // enums

    enum Feature;

    // methods

    virtual Steinberg::TBool isInstrumentFeatureSupported(Steinberg::int32 which) = 0;
    virtual Steinberg::tresult setInstrumentObserver(IInstrumentObserver* observer) = 0;

    virtual Steinberg::tresult getKeyAssignment(
        IKeyAssignmentReceiver& result,
        Steinberg::int32 busIndex,
        Steinberg::int16 channel
        ) = 0;

    virtual Steinberg::TBool isDrumInstrument(
        Steinberg::int32 busIndex,
        Steinberg::int16 channel
        ) = 0;

    virtual Steinberg::tresult getMiddleCValue(int& pitch) = 0;
    virtual Steinberg::tresult setMiddleCValue(int pitch) = 0;
;

// direct descendants

class DefaultInstrumentController;
```

Detailed Documentation

Extension to Steinberg::Vst::IEditController.

Implemented by plug-in.

Methods

```
virtual Steinberg::TBool isInstrumentFeatureSupported(Steinberg::int32 which) = 0
```

Check if a certain instrument feature is supported.

The plug-in should store the pointer and report whenever data changes

```
virtual Steinberg::tresult setInstrumentObserver(IInstrumentObserver* observer) = 0
```

Receive callback interface to report instrument changes.

The plug-in should store the pointer and report whenever data changes

```
virtual Steinberg::tresult getKeyAssignment(
    IKeyAssignmentReceiver& result,
    Steinberg::int32 busIndex,
    Steinberg::int16 channel
    ) = 0
```

Get the current key assignment for a synth unit (busIndex + channel).

Returns:

kResultTrue on success

```
virtual Steinberg::TBool isDrumInstrument(
    Steinberg::int32 busIndex,
    Steinberg::int16 channel
    ) = 0
```

Suggest to use the drum editor for editing.

```
virtual Steinberg::tresult getMiddleCValue(int& pitch) = 0
```

Return the current middle C midi pitch.

Returns:

kResultTrue on success, kNotImplemented if not implemented, kResultFalse otherwise

```
virtual Steinberg::tresult setMiddleCValue(int pitch) = 0
```

Set middle C midi pitch.

Returns:

kResultTrue on success, kNotImplemented if not implemented, kResultFalse otherwise

2.4.2 struct Presonus::IInstrumentObserver

enum Presonus::IInstrumentObserver::ChangeMessage

Overview

Values used for IInstrumentObserver::onInstrumentChanged. *More...*

```
#include <ipslinstrumentcontroller.h>

enum ChangeMessage

    kKeyAssignmentChanged    = 1,
    kIsDrumInstrumentChanged = 2,
    kMiddleCChanged          = 3,
;
```

Detailed Documentation

Values used for IInstrumentObserver::onInstrumentChanged.

Enum Values

```
kKeyAssignmentChanged
```

Notify the host that key assignment info changed.

The host should call *IInstrumentController::getKeyAssignment* and discard any cached data.

```
kIsDrumInstrumentChanged
```

Notify the host that the instrument type changed.

The host should call *IInstrumentController::isDrumInstrument* in return.

```
kMiddleCChanged
```

Notify the host that the middle C settings changed.

The host should call *IInstrumentController::getMiddleCValue* in return.

Overview

Observer interface implemented by the host. *More...*

```
#include <ipslinstrumentcontroller.h>

struct IInstrumentObserver

    // enums

    enum ChangeMessage;

    // methods

    virtual void onInstrumentInfoChanged(
        Steinberg::int32 changeMessage,
        Steinberg::int32 busIndex = -1,
        Steinberg::int16 channel = -1
        ) = 0;
;
```

Detailed Documentation

Observer interface implemented by the host.

Notify the host that instrument data has changed.

Methods

```
virtual void onInstrumentInfoChanged(
    Steinberg::int32 changeMessage,
    Steinberg::int32 busIndex = -1,
    Steinberg::int16 channel = -1
    ) = 0
```

Notify the host that instrument data has changed.

This method must be called in the main thread. If the change value is only valid for a certain synth unit, busIndex and channel should be used to specify this.

2.4.3 struct Presonus::IKeyAssignmentReceiver

Overview

Callback interface for retrieving key assignemtns The host implements this to retrieve key assignments from the plug-in. *More...*

```
#include <ipslinstrumentcontroller.h>

struct IKeyAssignmentReceiver

    // methods

    virtual void addKeyAssignment(const KeyAssignment& info) = 0;
;
```

Detailed Documentation

Callback interface for retrieving key assignemtns The host implements this to retrieve key assignments from the plug-in.

Methods

```
virtual void addKeyAssignment(const KeyAssignment& info) = 0
```

Add info for keys/pitches (in display order).

2.4.4 struct Presonus::KeyAssignment

enum Presonus::KeyAssignment::Type

Overview

```
#include <ipslinstrumentcontroller.h>

enum Type

    kSustainable = 0,
    kOneShot     = 1,
    kFunction    = 2,
;
```

Detailed Documentation

Enum Values

`kSustainable`

sound starts with note on, ends with note off

`kOneShot`

sound starts with note on, note off does not end it, sound always plays for a defined time

`kFunction`

the pitch is assingned to a function (like a keyswitch)

Overview

Assignment of a single Key / Pitch / Note *More...*

```
#include <ipslinstrumentcontroller.h>

struct KeyAssignment

    // enums

    enum Type;

    // fields

    Steinberg::int16 midiPitch;
    Steinberg::int32 type;
    Steinberg::Vst::String128 title;
    Steinberg::Vst::ColorSpec color;

    // construction

    KeyAssignment(
        Steinberg::int16 pitch,
        Type _type
        );
;
```

Detailed Documentation

Assignment of a single Key / Pitch / Note

Fields

```
Steinberg::int16 midiPitch
```

the pitch

```
Steinberg::int32 type
```

assignment type

See also:

Type

```
Steinberg::Vst::String128 title
```

optional

```
Steinberg::Vst::ColorSpec color
```

optional

2.4.5 class Presonus::DefaultInstrumentController

Overview

Mix-in class implementing optional methods as default *More...*

```
#include <ipslinstrumentcontroller.h>

class DefaultInstrumentController: public Presonus::IInstrumentController

public:
    // methods

    virtual Steinberg::tresult getKeyAssignment(
        IKeyAssignmentReceiver& result,
        Steinberg::int32 busIndex,
        Steinberg::int16 channel
        );

    virtual Steinberg::TBool isDrumInstrument(
        Steinberg::int32 busIndex,
        Steinberg::int16 channel
        );

    virtual Steinberg::tresult getMiddleCValue(int& pitch);
    virtual Steinberg::tresult setMiddleCValue(int pitch);
;
```

Inherited Members

```
public:
    // enums

    enum Feature;

    // methods

    virtual Steinberg::TBool isInstrumentFeatureSupported(Steinberg::int32 which) = 0;
    virtual Steinberg::tresult setInstrumentObserver(IInstrumentObserver* observer) = 0;

    virtual Steinberg::tresult getKeyAssignment(
        IKeyAssignmentReceiver& result,
        Steinberg::int32 busIndex,
        Steinberg::int16 channel
        ) = 0;

    virtual Steinberg::TBool isDrumInstrument(
        Steinberg::int32 busIndex,
        Steinberg::int16 channel
        ) = 0;

    virtual Steinberg::tresult getMiddleCValue(int& pitch) = 0;
    virtual Steinberg::tresult setMiddleCValue(int pitch) = 0;
```

Detailed Documentation

Mix-in class implementing optional methods as default

Methods

```
virtual Steinberg::tresult getKeyAssignment(
    IKeyAssignmentReceiver& result,
    Steinberg::int32 busIndex,
    Steinberg::int16 channel
    )
```

Get the current key assignment for a synth unit (busIndex + channel).

Returns:

kResultTrue on success

```
virtual Steinberg::TBool isDrumInstrument(
    Steinberg::int32 busIndex,
    Steinberg::int16 channel
    )
```

Suggest to use the drum editor for editing.

```
virtual Steinberg::tresult getMiddleCValue(int& pitch)
```

Return the current middle C midi pitch.

Returns:

kResultTrue on success, kNotImplemented if not implemented, kResultFalse otherwise

```
virtual Steinberg::tresult setMiddleCValue(int pitch)
```

Set middle C midi pitch.

Returns:

kResultTrue on success, kNotImplemented if not implemented, kResultFalse otherwise

2.4.6 Overview

The following interfaces allow to support extended features of a synth plug-in. *More...*

```
// structs

struct Presonus::IInstrumentController;
struct Presonus::IInstrumentObserver;
struct Presonus::IKeyAssignmentReceiver;
struct Presonus::KeyAssignment;

// classes

class Presonus::DefaultInstrumentController;
```

2.4.7 Detailed Documentation

The following interfaces allow to support extended features of a synth plug-in.

1. Report Key Assignment. Used in the host to display the function and name of key on a musical keyboard
2. Report which editor is used best with this instrument (Piano / Drum)
3. Report or set the "Middle C"

2.5 Sound Variations

2.5.1 struct Presonus::ISoundVariationController

Overview

Main interface for sound variation reporting. *More...*

```
#include <ipslsoundvariation.h>

struct ISoundVariationController: public FUnknown

    // methods

    virtual ISoundVariationInfo* getSoundVariationInfo(
        Steinberg::int32 busIndex,
        Steinberg::int16 channel
        ) = 0;

    virtual Steinberg::TBool isSoundVariationEventSupported() = 0;
    virtual Steinberg::TBool isDisableKeySwitchesSupported() = 0;
    virtual Steinberg::tresult disableKeySwitches(Steinberg::TBool state) = 0;
    virtual Steinberg::TBool areKeySwitchesDisabled() = 0;
;
```

Detailed Documentation

Main interface for sound variation reporting.

Extension to Steinberg::Vst::IEditController.

Methods

```
virtual ISoundVariationInfo* getSoundVariationInfo(
    Steinberg::int32 busIndex,
    Steinberg::int16 channel
    ) = 0
```

Get the variation info for the synth unit addressed by busIndex + channel.

```
virtual Steinberg::TBool isSoundVariationEventSupported() = 0
```

Return true if the plug-in can handle events of type *Vst3SoundVariationEvent* / *Vst2SoundVariationEvent*.

```
virtual Steinberg::TBool isDisableKeySwitchesSupported() = 0
```

Return true if the plug-in supports disableKeySwitches.

```
virtual Steinberg::tresult disableKeySwitches(Steinberg::TBool state) = 0
```

Enable a mode that ignores all activation sequence events and only handle *Vst3SoundVariationEvent* / *Vst2SoundVariationEvent*.

```
virtual Steinberg::TBool areKeySwitchesDisabled() = 0
```

Return true if the disableKeySwitches mode is active.

2.5.2 struct Presonus::ISoundVariationInfo

Overview

Report sound variations for a synth unit (busIndex + channel). *More...*

```
#include <ipslsoundvariation.h>

struct ISoundVariationInfo: public FUnknown

    // methods

    virtual Steinberg::tresult setVariationObserver(ISoundVariationObserver* handler) = 0;
    virtual Steinberg::tresult getVariationList(ISoundVariationList& list) = 0;
    virtual Steinberg::tresult getActiveVariation(VariationID& variation) = 0;
;
```

Detailed Documentation

Report sound variations for a synth unit (busIndex + channel).

Implemented by plug-in.

Methods

```
virtual Steinberg::tresult setVariationObserver(ISoundVariationObserver* handler) = 0
```

Set interface to report changes of sound variation list.

The plug-in should store the pointer and report the cases defined by *ISoundVariationObserver::ChangeMessage* to the host

```
virtual Steinberg::tresult getVariationList(ISoundVariationList& list) = 0
```

Get the current sound variations.

The plug-in should fill the given varition list.

Returns:

kResultTrue on success

```
virtual Steinberg::tresult getActiveVariation(VariationID& variation) = 0
```

Get the currently active sound variation.

Returns:

kResultTrue on success

2.5.3 struct Presonus::ISoundVariationList

Overview

Callback interface passed to *ISoundVariationInfo::getVariationList*. *More...*

```
#include <ipslsoundvariation.h>

struct ISoundVariationList

    // methods

    virtual void addVariation(const SoundVariationData& var) = 0;
    virtual void beginFolder(const SoundVariationFolderData& folderData) = 0;
    virtual void endFolder() = 0;
    virtual void setPresetInfo(const SoundVariationPresetInfo& info) = 0;
;
```

Detailed Documentation

Callback interface passed to *ISoundVariationInfo::getVariationList*.

The host implements this to retrieve the variations provided by the plug-in.

Methods

```
virtual void addVariation(const SoundVariationData& var) = 0
```

Append variation to the list.

This can be done on top-level or inside folders.

```
virtual void beginFolder(const SoundVariationFolderData& folderData) = 0
```

Optional.

Begin an new folder. All following sound variation will be added to the current folder.

```
virtual void endFolder() = 0
```

Each started folder needs to be closed again.

```
virtual void setPresetInfo(const SoundVariationPresetInfo& info) = 0
```

Tell the host which sound preset the reported variations belong to.

See also:

SoundVariationPresetInfo

2.5.4 struct Presonus::ISoundVariationObserver

enum Presonus::ISoundVariationObserver::ChangeMessage

Overview

Values used for *ISoundVariationObserver::onSoundVariationsChanged*. *More...*

```
#include <ipslsoundvariation.h>

enum ChangeMessage

    kPresetChanged          = 0,
    kVariationListModified  = 1,
    kActiveVariationChanged = 2,
;
```

Detailed Documentation

Values used for *ISoundVariationObserver::onSoundVariationsChanged*.

Enum Values

```
kPresetChanged
```

Notify the host that a new sound preset was loaded.

The host should call *ISoundVariationInfo::getVariationList* and discard any cached variation data.

```
kVariationListModified
```

Notify the host that the variation list of a loaded sound preset have been edited.

The host should call *ISoundVariationInfo::getVariationList* to retrieve modifications of the variation list. Cached variation data should be adjusted.

```
kActiveVariationChanged
```

Notify the host that the active variation has changed.

The host should call *ISoundVariationInfo::getActiveVariation* in return.

Overview

Observer interface implemented by the host. *More...*

```
#include <ipslsoundvariation.h>

struct ISoundVariationObserver

    // enums

    enum ChangeMessage;

    // methods

    virtual void onSoundVariationsChanged(Steinberg::int32 changeMessage) = 0;
;
```

Detailed Documentation

Observer interface implemented by the host.

Please note that this interface should not be used inside audio processing calls.

Methods

```
virtual void onSoundVariationsChanged(Steinberg::int32 changeMessage) = 0
```

Notify the host that the sound variation info has changed.

This method should be called in the main thread if possible.

Parameters:

changeMessage	is one value of enum ChangeMessage.

2.5.5 struct Presonus::ScoreSymbolList

Overview

Score Symbol List. *More...*

```
#include <ipslsoundvariation.h>

struct ScoreSymbolList

    // fields
```

```
    static const int kMaxItems = 4;
    int count;
    ScoreSymbolID symbols[kMaxItems];

    // methods

    static int getMaxItems();
    void clear();
    void addSymbol(ScoreSymbolID symbol);
;
```

Detailed Documentation

Score Symbol List.

Combination of notation symbols associated with a sound variation. Plug-in can suggest a unique score symbol comination that should trigger the variation.

2.5.6 struct Presonus::SoundActivationSequence

Overview

Sound Activation Sequence. *More...*

```
#include <ipslsoundvariation.h>

struct SoundActivationSequence

    // fields

    static const int kMaxItems = 8;
    int count;
    SoundActivationSequenceItem items[kMaxItems];

    // methods

    static int getMaxItems();
    void clear();
    void addItem(const SoundActivationSequenceItem& item);

    void addNote(
        const SoundActivationSequenceItem::Note& note,
        int type = SoundActivationSequenceItem::kNoteEvent
        );

    void addController(const SoundActivationSequenceItem::Controller& controller);
    void addProgramChange(CCValue value);
;
```

Detailed Documentation

Sound Activation Sequence.

List of events that activate a sound variation in the plug-in

Methods

```
void clear()
```

Remove all.

```
void addItem(const SoundActivationSequenceItem& item)
```

Add sound activation sequence item

```
void addNote(
    const SoundActivationSequenceItem::Note& note,
    int type = SoundActivationSequenceItem::kNoteEvent
    )
```

Add a note event (.

See also:

SoundActivationSequenceItem)

```
void addController(const SoundActivationSequenceItem::Controller& controller)
```

Add a controller event.

A VST3 plug-in needs to support IMidiMapping interface and will receive MIDI-CC as parameter changes

```
void addProgramChange(CCValue value)
```

Add a program change event.

2.5.7 struct Presonus::SoundActivationSequenceItem

enum Presonus::SoundActivationSequenceItem::ItemTypes

Overview

```
#include <ipslsoundvariation.h>

enum ItemTypes

    kNoteEvent     = 0,
```

```
    kNoteOnEvent   = 1,
    kNoteOffEvent  = 2,
    kControlEvent  = 3,
    kProgramChange = 4,
;
```

Detailed Documentation

Enum Values

```
kNoteEvent
```

Note event: Host should send a note-on followed by note off.

The note off is either ignored in the plug-in or it terminates momentary sound variations. is *Note*

```
kNoteOnEvent
```

Single note-on event.

is *Note*

```
kNoteOffEvent
```

Single note-off event.

is *Note*

```
kControlEvent
```

Controller event.

is *Controller*

```
kProgramChange
```

Program change.

is *ProgramChange*

struct Presonus::SoundActivationSequenceItem::Controller

```
#include <ipslsoundvariation.h>

struct Controller

    // fields

    CCNumber number;
    CCValue value;

    // construction

    Controller(
        CCNumber n,
```

```
        CCValue v
        );
;
```

struct Presonus::SoundActivationSequenceItem::Note

```
#include <ipslsoundvariation.h>

struct Note

    // fields

    Pitch pitch;
    float velocity;

    // construction

    Note(
        Pitch p = 0,
        float v = 1.f
        );
;
```

struct Presonus::SoundActivationSequenceItem::ProgramChange

```
#include <ipslsoundvariation.h>

struct ProgramChange

    // fields

    CCValue value;

    // construction

    ProgramChange(CCValue v);
;
```

Overview

Sound Activation Sequence Item. *More...*

```
#include <ipslsoundvariation.h>

struct SoundActivationSequenceItem

    // enums

    enum ItemTypes;

    // structs

    struct Controller;
    struct Note;
    struct ProgramChange;
```

```
    // fields

    int type;
    Note note;
    Controller controller;
    ProgramChange programChange;
    Steinberg::int32 placeHoder[16];
;
```

Detailed Documentation

Sound Activation Sequence Item.

One item of a sound variation activation sequence

2.5.8 struct Presonus::SoundVariationData

enum Presonus::SoundVariationData::Flags

Overview

```
#include <ipslsoundvariation.h>

enum Flags

    kIsMomentary = 1<<0,
    kIsDefault   = 1<<1,
;
```

Detailed Documentation

Enum Values

```
kIsMomentary
```

Previous sound variation is re-enabled when this variation terminates.

Activation sequence can only be a note (with length) - and / or kTerminateTypeId must be supported

```
kIsDefault
```

Main or default variation which is active on loading the sound preset.

No more than one variation should have this flag.

Overview

Sound Variation. *More...*

```
#include <ipslsoundvariation.h>

struct SoundVariationData

    // enums

    enum Flags;

    // fields

    VariationID identifier;
    Steinberg::Vst::String128 title;
    SoundActivationSequence activationSequence;
    Steinberg::Vst::ColorSpec color;
    Pitch triggerPitch;
    ScoreSymbolList scoreSymbols;
    Steinberg::int32 flags;

    // construction

    SoundVariationData(VariationID id = -1);
;
```

Detailed Documentation

Sound Variation.

Used with *ISoundVariationList::addVariation* to report a single variation

Fields

```
VariationID identifier
```

Each variation needs an id which must be unique in the reported list.

This ID is used to address variations. It can be implemented as a simple counter as long as it meets the following rules

- The id must be the same each time the sound variations are queried.
- The id must be the same each time a sound is loaded.
- If the synth gui allows to move or edit a variation, the ids of all reported variations must not change.

The id will be stored with trigger events of host documents. They are also used to transmit sound variation activation requests via *Vst3SoundVariationEvent* / *Vst2SoundVariationEvent*

```
Steinberg::Vst::String128 title
```

Displayed name of the variation.

```
SoundActivationSequence activationSequence
```

Specify how the variation is activated.

Alternatively the plug-in can support Sound Variation Events.

```
Steinberg::Vst::ColorSpec color
```

optional - set to 0 if not provided

```
Pitch triggerPitch
```

optional - synth can suggest a default key switch, set to -1 if not provided

```
ScoreSymbolList scoreSymbols
```

optional -

See also:

ScoreSymbolList

```
Steinberg::int32 flags
```

optional -

See also:

Flags

2.5.9 struct Presonus::SoundVariationFolderData

enum Presonus::SoundVariationFolderData::Flags

Overview

```
#include <ipslsoundvariation.h>

enum Flags

    kAddTitleToVariations = 1<<0,
;
```

Detailed Documentation

Enum Values

```
kAddTitleToVariations
```

Prepend the title of this folder to display variation names.

Overview

Sound Variation Folder. *More...*

```
#include <ipslsoundvariation.h>

struct SoundVariationFolderData

    // enums

    enum Flags;

    // fields

    Steinberg::Vst::String128 title;
    Steinberg::Vst::ColorSpec color;
    Steinberg::int32 flags;
;
```

Detailed Documentation

Sound Variation Folder.

Used with ISoundVariationList::begin folder to report nested variations

Fields

```
Steinberg::Vst::String128 title
```

Displayed name of the folder.

```
Steinberg::Vst::ColorSpec color
```

Color - opional (set to 0 if not available)

```
Steinberg::int32 flags
```

optional -

See also:

Flags

2.5.10 struct Presonus::SoundVariationPresetInfo

Overview

Info about the loaded sound preset. *More...*

```
#include <ipslsoundvariation.h>

struct SoundVariationPresetInfo

    // fields

    Steinberg::Vst::String128 name;
    Steinberg::Vst::String128 path;
;
```

Detailed Documentation

Info about the loaded sound preset.

The host uses this to

- display the preset name together with the sound variation list.
- automatically store/restore additional data with the sound variations

Fields

```
Steinberg::Vst::String128 name
```

Displayed name of the preset.

```
Steinberg::Vst::String128 path
```

Internal qualifier to resolve name clashes (optional).

Not intended to be displayed, but should be valid for file systems so the host can use it to create folders and organize sound variation map presets.

2.5.11 struct Presonus::SymbolID

Overview

Symbol definitions. *More...*

```
#include <ipslsoundvariation.h>

struct SymbolID

    // fields

    static const ScoreSymbolID kStaccato = 'stac';
    static const ScoreSymbolID kStaccatissimo = 'stis';
    static const ScoreSymbolID kTenuto = 'tenu';
    static const ScoreSymbolID kAccent = 'acce';
    static const ScoreSymbolID kStrongAccent = 'marc';
    static const ScoreSymbolID kForceFP = 'fpno';
    static const ScoreSymbolID kForceFFP = 'ffpn';
    static const ScoreSymbolID kForceFZ = 'fzdo';
    static const ScoreSymbolID kForceFFZ = 'ffzo';
    static const ScoreSymbolID kForceSF = 'sfdo';
    static const ScoreSymbolID kForceSFF = 'sffo';
    static const ScoreSymbolID kForceSFZ = 'sfzo';
    static const ScoreSymbolID kForceSFFZ = 'sffz';
    static const ScoreSymbolID kForceSFP = 'sfpo';
    static const ScoreSymbolID kForceSFFP = 'sffp';
    static const ScoreSymbolID kMezzoStaccato = 'mzsc';
    static const ScoreSymbolID kAccentTenuto = 'actn';
    static const ScoreSymbolID kAccentStaccato = 'acst';
    static const ScoreSymbolID kAccentStaccatissimo = 'acso';
    static const ScoreSymbolID kStrongAccentTenuto = 'mrtn';
    static const ScoreSymbolID kStrongAccentStaccato = 'mrst';
    static const ScoreSymbolID kStrongAccentStaccatissimo = 'mrso';
    static const ScoreSymbolID kTremolo1 = 'trm1';
    static const ScoreSymbolID kTremolo2 = 'trm2';
    static const ScoreSymbolID kTremolo3 = 'trm3';
    static const ScoreSymbolID kIntervalTremolo1 = 'itr1';
    static const ScoreSymbolID kIntervalTremolo2 = 'itr2';
    static const ScoreSymbolID kIntervalTremolo3 = 'itr3';
    static const ScoreSymbolID kArpeggioNormal = 'arpN';
    static const ScoreSymbolID kArpeggioUp = 'arpU';
    static const ScoreSymbolID kArpeggioDown = 'arpD';
    static const ScoreSymbolID kGlissando = 'glss';
    static const ScoreSymbolID kPortamento = 'port';
    static const ScoreSymbolID kSlur = 'slur';
    static const ScoreSymbolID kTrill = 'trll';
    static const ScoreSymbolID kTrillSharp = 'trlS';
    static const ScoreSymbolID kTrillNatural = 'trlN';
    static const ScoreSymbolID kTrillFlat = 'trlF';
    static const ScoreSymbolID kVibrato = 'vibr';
    static const ScoreSymbolID kWideVibrato = 'wvib';
    static const ScoreSymbolID kCircle = 'circ';
    static const ScoreSymbolID kPlus = 'plus';
    static const ScoreSymbolID kLaissezVibrer = 'lvib';
    static const ScoreSymbolID kConSordino = 'sord';
    static const ScoreSymbolID kSenzaSordino = 'ssor';
    static const ScoreSymbolID kArco = 'arco';
    static const ScoreSymbolID kPizzicato = 'pizz';
    static const ScoreSymbolID kColLegno = 'legn';
    static const ScoreSymbolID kSulPonticello = 'pont';
    static const ScoreSymbolID kSulTasto = 'tast';
    static const ScoreSymbolID kBehindBridge = 'bhnd';
    static const ScoreSymbolID kDownBow = 'dnbw';
```

```
    static const ScoreSymbolID kUpBow = 'upbw';
    static const ScoreSymbolID kBartokPizzicato = 'snap';
    static const ScoreSymbolID kPedalDown = 'pddn';
    static const ScoreSymbolID kPedalUp = 'pdup';
    static const ScoreSymbolID kHammerOn = 'hmon';
    static const ScoreSymbolID kPullOff = 'plof';
    static const ScoreSymbolID kGuitarTap = 'gtap';
;
```

Detailed Documentation

Symbol definitions.

Fields

```
static const ScoreSymbolID kMezzoStaccato = 'mzsc'
```

kStaccato | kTenuto

```
static const ScoreSymbolID kAccentTenuto = 'actn'
```

kAccent | kTenuto

```
static const ScoreSymbolID kAccentStaccato = 'acst'
```

kAccent | kStaccato,

```
static const ScoreSymbolID kAccentStaccatissimo = 'acso'
```

kAccent | kStaccatissimo

```
static const ScoreSymbolID kStrongAccentTenuto = 'mrtn'
```

kStrongAccent | kTenuto

```
static const ScoreSymbolID kStrongAccentStaccato = 'mrst'
```

kStrongAccent | kStaccato

```
static const ScoreSymbolID kStrongAccentStaccatissimo = 'mrso'
```

kStrongAccent | kStaccatissimo

```
static const ScoreSymbolID kTremolo1 = 'trm1'
```

tremolo one slash (note repeated)

```
static const ScoreSymbolID kTremolo2 = 'trm2'
```

tremolo 2 slashes

```
static const ScoreSymbolID kTremolo3 = 'trm3'
```

tremolo 3 slashes

```
static const ScoreSymbolID kIntervalTremolo1 = 'itr1'
```

tremolo with next note / fingered tremolo

```
static const ScoreSymbolID kSlur = 'slur'
```

legato

```
static const ScoreSymbolID kCircle = 'circ'
```

o meaning depends on instrument: open / harmonic (flageolett)

```
static const ScoreSymbolID kPlus = 'plus'
```

- (closed)

```
static const ScoreSymbolID kLaissezVibrer = 'lvib'
```

l.v.

```
static const ScoreSymbolID kConSordino = 'sord'
```

Con Sordino https://en.wikipedia.org/wiki/Mute_(music)

```
static const ScoreSymbolID kSenzaSordino = 'ssor'
```

revert kConSordino

```
static const ScoreSymbolID kColLegno = 'legn'
```

“mit dem Holz” - reverted bow

```
static const ScoreSymbolID kSulPonticello = 'pont'
```

“am steg”

```
static const ScoreSymbolID kSulTasto = 'tast'
```

flautando “nahe am Griffbrett”

```
static const ScoreSymbolID kBartokPizzicato = 'snap'
```

circle with line crossing the top

2.5.12 struct Presonus::Vst2SoundVariationEvent

Overview

VST2 Sound Activation Event. *More...*

```
#include <ipslsoundvariation.h>

struct Vst2SoundVariationEvent

    // fields

    static const Steinberg::int32 kTypeId = 'PSVE';
```

```
    static const Steinberg::int32 kTerminateTypeId = 'PSVT';
    Steinberg::int32 type;
    Steinberg::int32 byteSize;
    Steinberg::int32 deltaFrames;
    Steinberg::int32 flags;
    Steinberg::int32 channel;
    VariationID variationId;
;
```

Detailed Documentation

VST2 Sound Activation Event.

Event sent to a synth in order to activate a sound variation. See also VstEvent

Fields

```
static const Steinberg::int32 kTypeId = 'PSVE'
```

Activate sound variation.

```
static const Steinberg::int32 kTerminateTypeId = 'PSVT'
```

Terminate momentary sound variation.

```
Steinberg::int32 type
```

Vst2SoundVariationEvent::kTypeId or *Vst2SoundVariationEvent::kTerminateTypeId*.

```
Steinberg::int32 byteSize
```

sizeof(Vst2SoundVariationEvent) - 2 * sizeof(VstInt32);

```
Steinberg::int32 deltaFrames
```

sample frames related to the current block start sample position

```
Steinberg::int32 flags
```

See also:

VstMidiEventFlags

```
Steinberg::int32 channel
```

MIDI channel.

```
VariationID variationId
```

id of variation to be activated

2.5.13 struct Presonus::Vst3SoundVariationEvent

Overview

Event sent to a synth in order to activate a sound variation. *More...*

```
#include <ipslsoundvariation.h>

struct Vst3SoundVariationEvent

    // fields

    static const Steinberg::int16 kTypeId = 'VE';
    static const Steinberg::int16 kTerminateTypeId = 'VT';
    Steinberg::int32 busIndex;
    Steinberg::int32 sampleOffset;
    Steinberg::Vst::TQuarterNotes ppqPosition;
    Steinberg::uint16 flags;
    Steinberg::uint16 type;
    Steinberg::int32 channel;
    VariationID variationId;
;
```

Detailed Documentation

Event sent to a synth in order to activate a sound variation.

Events of this type are sent only if *ISoundVariationController::isSoundVariationEventSupported* returns true. A Steinberg::Vst::Event which has the type SoundVariationEvent::kTypeId must be reinterpret-casted to Presonus::SoundVariationEvent and the according sound variation should be activated.

see also Steinberg::Vst::Event

Fields

```
static const Steinberg::int16 kTypeId = 'VE'
```

Activate sound variation.

```
static const Steinberg::int16 kTerminateTypeId = 'VT'
```

Terminate momentary sound variation.

```
Steinberg::int32 busIndex
```

event bus index

```
Steinberg::int32 sampleOffset
```

sample frames related to the current block start sample position

```
Steinberg::Vst::TQuarterNotes ppqPosition
```

position in project

```
Steinberg::uint16 flags
```

combination of EventFlags - and kTerminateMomentary

```
Steinberg::uint16 type
```

Vst3SoundVariationEvent::kType.

```
Steinberg::int32 channel
```

channel index in event bus (like NoteOnEvent)

```
VariationID variationId
```

id of variation to be activated

2.5.14 Overview

The following interfaces allow to report available sound variations of a synth. *More...*

```
// structs

struct Presonus::ISoundVariationController;
struct Presonus::ISoundVariationInfo;
struct Presonus::ISoundVariationList;
struct Presonus::ISoundVariationObserver;
struct Presonus::ScoreSymbolList;
struct Presonus::SoundActivationSequence;
struct Presonus::SoundActivationSequenceItem;
struct Presonus::SoundVariationData;
struct Presonus::SoundVariationFolderData;
struct Presonus::SoundVariationPresetInfo;
struct Presonus::SymbolID;
struct Presonus::Vst2SoundVariationEvent;
struct Presonus::Vst3SoundVariationEvent;

// global variables

static const int Presonus::kGetSoundVariationController = 'GSVC';
```

2.5.15 Detailed Documentation

The following interfaces allow to report available sound variations of a synth.

Sound Variations are alterations of a loaded sound - also known as articulations. Usually they are realized as sample layers and often enabled with a key switch note. The main purpose of this interface extension is to

- Support complex event sequences that enable variations.
- Report variation names, colors and how variations are organized in folders

- Report which is the active variation
- Report the name of the sound preset the current variations belong to

Overview:

- The main entry point for this feature is *ISoundVariationController* provided by the plug-in.
- For each unit the plug-in provides a *ISoundVariationInfo* (addressed via bus and channel index).
- The host will can query variations via *ISoundVariationInfo::getVariationList*.
- The plug-in must report changes to the list with *ISoundVariationObserver::onSoundVariationsChanged*
- The host will activate variations either by sending the reported sound activation sequence or using a special Event (optional - if the plug-in supports this)

Global Variables

```
static const int Presonus::kGetSoundVariationController = 'GSVC'
```

VST2 opcode passed to AEffectDispatcherProc to get *ISoundVariationController* instance.

Host call looks like this:

```
if(effect->dispatch (effect, effCanDo, 0, 0, (void*)Presonus::PlugCanDos::canDoGetSoundVariationController) ==

    Presonus::ISoundVariationController* soundVariationController = 0;
    effect->dispatcher (effect, effVendorSpecific, Presonus::kVendorID, Presonus::kGetSoundVariationController,
    // ...
```

Plug-In side looks like this:

```
VstIntPtr MySynth::dispatcher (VstInt32 opcode, VstInt32 index, VstIntPtr value, void* ptr, float opt)

    if(opcode == effVendorSpecific && index == Presonus::kVendorID && value == Presonus::kGetSoundVariationCont

        Presonus::ISoundVariationController** result = (Presonus::ISoundVariationController**)ptr;
        if(soundVariationController == 0)
            soundVariationController = new SoundVariationController;

         *result = soundVariationController;
        return 1;

    return AudioEffectX::dispatcher (opcode, index, value, ptr, opt);
```

2.6 VST 2 Extensions

2.6.1 namespace Presonus::PlugCanDos

Overview

Identifiers to be passed to VST2's canDo() method. *More...*

```
namespace PlugCanDos

// global variables

static const char* canDoGetSoundVariationController = "getSoundVariationController";
static const char* canDoViewResize = "supportsViewResize";
static const char* canDoViewEmbedding = "supportsViewEmbedding";
static const char* canDoViewDpiScaling = "supportsViewDpiScaling";
static const char* canDoViewSystemDpiScaling = "supportsViewSystemDpiScaling";
static const char* canDoGainReductionInfo = "supportsGainReductionInfo";
static const char* canDoSlaveEffects = "supportsSlaveEffects";
static const char* canDoMidiKeySwitchInfo = "supportsMidiKeySwitchInfo";
static const char* canDoMPENotifications = "supportsMPENotifications";

 // namespace PlugCanDos
```

Detailed Documentation

Identifiers to be passed to VST2's canDo() method.

Global Variables

```
static const char* canDoGetSoundVariationController = "getSoundVariationController"
```

Check if plug-in can provide a *ISoundVariationController*.

```
static const char* canDoViewResize = "supportsViewResize"
```

Check if view can be resized by the host.

```
static const char* canDoViewEmbedding = "supportsViewEmbedding"
```

Check if view can be embedded by the host.

```
static const char* canDoViewDpiScaling = "supportsViewDpiScaling"
```

Check if view scaling for high-DPI is supported by the plug-in.

```
static const char* canDoViewSystemDpiScaling = "supportsViewSystemDpiScaling"
```

Check if system DPI scaling should be turned on for the plug-in.

```
static const char* canDoGainReductionInfo = "supportsGainReductionInfo"
```

Check if gain reduction reporting is supported by the plug-in.

```
static const char* canDoSlaveEffects = "supportsSlaveEffects"
```

Check if slave effects are supported by plug-in.

```
static const char* canDoMidiKeySwitchInfo = "supportsMidiKeySwitchInfo"
```

Check if plug-in can export MIDI key switch information.

```
static const char* canDoMPENotifications = "supportsMPENotifications"
```

Check if plug-in supports MPE notifications.

2.6.2 enum Presonus::Opcodes

Overview

Opcodes. *More...*

```
#include <pslvst2extensions.h>

enum Opcodes

    kVendorID                          = 'PreS',
    kEffEditCheckSizeConstraints       = 'AeCc',
    kEffEditSetRect                    = 'AeSr',
    kEffEditSetEmbedded                = 'AeEm',
    kEffEditSetContentScaleFactor      = 'AeCs',
    kEffGetGainReductionValueInDb      = 'GRdB',
    kEffAddSlave                       = 'AdSl',
    kEffRemoveSlave                    = 'RmSl',
    kEffGetMidiKeyAssignment           = 'MKAs',
    kMidiKeyAssignmentIsKeySwitch      = 1<<0,
    kMidiKeyAssignmentIsActiveKeySwitch = 1<<1,
    kMidiKeyAssignmentIsAssigned       = 1<<2,
    kEffGetMPEEnabled                  = 'gMPE',
    kHostMPEEnabledChangeNotification  = 'cMPE',
;
```

Detailed Documentation

Opcodes.

Enum Values

```
kVendorID
```

PreSonus vendor ID - distinguishes our calls from other VST2 extensions.

Pass this vendor ID as “index” (aka “lArg1”) parameter for vendor specific calls.

```
kEffEditCheckSizeConstraints
```

The host can suggest a new editor size, and the plug-in can modify the suggested size to a suitable value if it cannot resize to the given values.

The ptrArg is a ERect* to the input/output rect. This differs from the ERect** used by effEditGetRect, because here the rect is owned by the host, not the plug-in. The result is 0 on failure, 1 on success.

```
kEffEditSetRect
```

The host can set a new size after negotiating the size via the above kEffEditCheckSizeConstraints, triggering the actual resizing.

The ptrArg is a ERect* to the input/output rect. This differs from the ERect** used by effEditGetRect, because here the rect is owned by the host, not the plug-in. The result is 0 on failure, 1 on success.

```
kEffEditSetEmbedded
```

When the view is embedded, it may need to adjust its UI, e.g.

by suppressing its built-in resizing facility because this is then controlled by the host. The ptrArg is a VstInt32*, pointing to 0 to disable or to 1 to enable embedding. Per default, embedding is disabled until the host calls this to indicate otherwise.

```
kEffEditSetContentScaleFactor
```

Inform the view about the current content scaling factor.

The factor is passed in the opt argument. For more details, please check the documentation of *Presonus::IPlugInViewScaling*.

```
kEffGetGainReductionValueInDb
```

Get current gain reduction for display.

The ptrArg is a float* to be set to the dB value. For more details, please check the documentation of *Presonus::IGainReductionInfo*.

```
kEffAddSlave
```

Add slave effect.

The ptrArg is a pointer to the slave AEffect, the ‘opt’ float transmits the mode (see enum SlaveMode). For more details, please check the documentation of *Presonus::ISlaveControllerHandler*.

```
kEffRemoveSlave
```

Remove slave effect.

The ptrArg is a pointer to the slave AEffect. For more details, please check the documentation of *Presonus::ISlaveControllerHandler*.

```
kEffGetMidiKeyAssignment
```

Get MIDI key assignment.

The ptrArg is a pointer to a MidiKeyName structure as defined in the VST2 SDK. The opt argument is the MIDI channel.

```
kMidiKeyAssignmentIsKeySwitch
```

key is used as key switch

```
kMidiKeyAssignmentIsActiveKeySwitch
```

key switch is currently active

```
kMidiKeyAssignmentIsAssigned
```

key is assigned (no sound otherwise)

```
kEffGetMPEEnabled
```

Called by host to check if plug-in has MPE mode currently enabled (return value is 1).

```
kHostMPEEnabledChangeNotification
```

Notify host that plug-in has enabled or disabled MPE mode.

Passed as 'value' argument to audioMasterVendorSpecific. Host in turn queries new mode via kEffGetMPEEnabled and returns 1 on success or 0 if not called from main thread.

2.6.3 Overview

Vendor-specific opcodes a VST2 plug-in can implement to add non-standard features like embedding its views as subview into the host, resizing from the host, high-DPI scaling, etc. *More...*

```
// namespaces

namespace Presonus::PlugCanDos;

// enums

enum Presonus::Opcodes;
```

2.6.4 Detailed Documentation

Vendor-specific opcodes a VST2 plug-in can implement to add non-standard features like embedding its views as subview into the host, resizing from the host, high-DPI scaling, etc.

- Embedding corresponds to the *Presonus::IPlugInViewEmbedding* VST3 extended interface.
- Resizing works like VST3's checkSizeConstraint() and onSize() methods, VST3's canResize() is defined via canDoViewResize.
- For "DPI-aware" host applications on the Windows platform a similar mimic to the *Presonus::IPlugInViewScaling* VST3 extended interface is defined here.
- Gain reduction reporting corresponds to the *Presonus::IGainReductionInfo* VST3 interface.
- Slave effect handling corresponds to the *Presonus::ISlaveControllerHandler* VST3 interface.

2.7 View Extensions

2.7.1 struct Presonus::IPlugInViewEmbedding

Overview

Support for plug-in view embedding, to be implemented by the VST3 controller class. *More...*

```
#include <ipslviewembedding.h>

struct IPlugInViewEmbedding: public FUnknown

    // methods

    virtual Steinberg::TBool isViewEmbeddingSupported() = 0;

    virtual Steinberg::tresult setViewIsEmbedded(
        Steinberg::IPlugView* view,
        Steinberg::TBool embedded
        ) = 0;
;
```

Detailed Documentation

Support for plug-in view embedding, to be implemented by the VST3 controller class.

Methods

```
virtual Steinberg::TBool isViewEmbeddingSupported() = 0
```

Check if view embedding is supported.

```
virtual Steinberg::tresult setViewIsEmbedded(
    Steinberg::IPlugView* view,
    Steinberg::TBool embedded
    ) = 0
```

Inform plug-in that its view will be embedded.

2.7.2 struct Presonus::IPlugInViewScaling

Overview

Support for plug-in view content scaling, to be implemented by the VST3 IPlugView class. *More...*

```
#include <ipslviewscaling.h>

struct IPlugInViewScaling: public FUnknown

    // methods

    virtual Steinberg::tresult setContentScaleFactor(float factor) = 0;
;
```

Detailed Documentation

Support for plug-in view content scaling, to be implemented by the VST3 IPlugView class.

(Please note that there is now an ABI-compatible interface in the VST3 SDK v3.6.6 named Steinberg::IPlugViewContentScaleSupport).

On Windows, if a process is "DPI-aware" and the system DPI setting is different from the default value of 96 DPI, the application is responsible to scale the contents of its windows accordingly, including child windows provided by 3rd party plug-ins.

This interface is used by the host to inform the plug-in about the current scaling factor. The scaling factor is used to convert user space coordinates aka DIPs (device-independent pixels) to physical pixels on screen.

The plug-in has to be prepared to deal with the following scaling factors:

96 DPI = 100% scaling (factor = 1.0) 120 DPI = 125% scaling (factor = 1.25) 144 DPI = 150% scaling (factor = 1.5) 192 DPI = 200% scaling (factor = 2.0)

On Windows 8.1 or later DPI settings are per monitor. The scaling factor for a window can change when it is moved between screens.

Methods

```
virtual Steinberg::tresult setContentScaleFactor(float factor) = 0
```

Inform the view about the current content scaling factor.

The scaling factor can change if the window is moved between screens.

2.7.3 struct Presonus::IPlugInViewSystemScalingSupport

Overview

The plug-in view supports system DPI scaling on Windows, i.e. *More...*

```
#include <ipslviewscaling.h>

struct IPlugInViewSystemScalingSupport: public FUnknown

;
```

Detailed Documentation

The plug-in view supports system DPI scaling on Windows, i.e.

it is prepared that the host can change the DPI_AWARENESS_CONTEXT of the main thread.

```
// structs

struct Presonus::IPlugInViewEmbedding;
struct Presonus::IPlugInViewScaling;
struct Presonus::IPlugInViewSystemScalingSupport;
```

Further Reading:

Context Information
Edit Controller Extensions
Host Commands
Instrument Extensions
Sound Variations
VST 2 Extensions
View Extensions

Reference and Index:

2.8 Global Namespace

2.8.1 namespace Presonus

namespace Presonus::ContextInfo

enum Presonus::ContextInfo::ChannelIndexMode

Overview

Channel index mode. *More...*

```
#include <ipslcontextinfo.h>

enum ChannelIndexMode

    kFlatIndex    = 0,
    kPerTypeIndex,
;
```

Detailed Documentation

Channel index mode.

Enum Values

```
kFlatIndex
```

channel indices are contiguous (example: track 1, track 2, bus 3, bus 4)

```
kPerTypeIndex
```

channel indices restarts at zero for each type (example: track 1, track 2, bus 1, bus 2)

enum Presonus::ContextInfo::ChannelType

Overview

Channel types. *More...*

```
#include <ipslcontextinfo.h>

enum ChannelType

    kTrack = 0,
    kBus,
    kFX,
    kSynth,
    kIn,
    kOut,
;
```

Detailed Documentation

Channel types.

Enum Values

```
kTrack
```

audio track

```
kBus
```

audio bus

```
kFX
```

FX channel.

```
kSynth
```

output of virtual instrument

```
kIn
```

input from audio driver

```
kOut
```

output to audio driver (main or sub-out)

Overview

```
namespace ContextInfo

// enums

enum ChannelIndexMode;
enum ChannelType;

// global variables

const Steinberg::FIDString kID = "id";
const Steinberg::FIDString kName = "name";
const Steinberg::FIDString kType = "type";
const Steinberg::FIDString kMain = "main";
const Steinberg::FIDString kIndex = "index";
const Steinberg::FIDString kColor = "color";
const Steinberg::FIDString kVisibility = "visibility";
const Steinberg::FIDString kSelected = "selected";
const Steinberg::FIDString kMultiSelect = "multiselect";
const Steinberg::FIDString kFocused = "focused";
const Steinberg::FIDString kRegionName = "regionName";
const Steinberg::FIDString kRegionSelected = "regionSelected";
const Steinberg::FIDString kVolume = "volume";
const Steinberg::FIDString kMaxVolume = "maxVolume";
const Steinberg::FIDString kPan = "pan";
const Steinberg::FIDString kMute = "mute";
const Steinberg::FIDString kSolo = "solo";
const Steinberg::FIDString kSendCount = "sendcount";
const Steinberg::FIDString kSendLevel = "sendlevel";
const Steinberg::FIDString kMaxSendLevel = "maxSendlevel";
const Steinberg::FIDString kActiveDocumentID = "activeDocumentID";
const Steinberg::FIDString kDocumentID = "documentID";
const Steinberg::FIDString kDocumentName = "documentName";
const Steinberg::FIDString kDocumentFolder = "documentFolder";
const Steinberg::FIDString kAudioFolder = "audioFolder";
const Steinberg::FIDString kIndexMode = "indexMode";

 // namespace ContextInfo
```

Detailed Documentation

Global Variables

```
const Steinberg::FIDString kID = "id"
```

(R) channel identifier, use to compare identity (string)

```
const Steinberg::FIDString kName = "name"
```

(R/W) channel name, can be displayed to the user (string)

```
const Steinberg::FIDString kType = "type"
```

(R) channel type (int32, see ChannelType enumeration)

```
const Steinberg::FIDString kMain = "main"
```

(R) channel is main output (int32, 0: false, 1: true)

```
const Steinberg::FIDString kIndex = "index"
```

(R) channel index (int32, starts at zero)

```
const Steinberg::FIDString kColor = "color"
```

(R/W) channel color (int32: RGBA, starts with red value in lowest byte)

```
const Steinberg::FIDString kVisibility = "visibility"
```

(R) channel visibility (int32, 0: false, 1: true)

```
const Steinberg::FIDString kSelected = "selected"
```

(R/W) selection state, channel is selected exlusively and scrolled into view on write (int32, 0: false, 1: true)

```
const Steinberg::FIDString kMultiSelect = "multiselect"
```

(W) select channel without unselecting others (int32, 0: false, 1: true)

```
const Steinberg::FIDString kFocused = "focused"
```

(R) focus for user input when multiple channels are selected (int32, 0: false, 1: true)

```
const Steinberg::FIDString kRegionName = "regionName"
```

(R) name of region/event for region/event-based effects (string)

```
const Steinberg::FIDString kRegionSelected = "regionSelected"
```

(R) selection state of region/event for region/event-based effects (int32, 0: false, 1: true)

```
const Steinberg::FIDString kVolume = "volume"
```

(R/W) volume factor [float, 0. = -oo dB, 1. = 0dB, etc.], also available as string

```
const Steinberg::FIDString kMaxVolume = "maxVolume"
```

(R) maximum volume factor [float, 1. = 0dB], also available as string

```
const Steinberg::FIDString kPan = "pan"
```

(R/W) stereo panning [float, < 0.5 = (L), 0.5 = (C), > 0.5 = (R)], also available as string

```
const Steinberg::FIDString kMute = "mute"
```

(R/W) mute (int32, 0: false, 1: true)

```
const Steinberg::FIDString kSolo = "solo"
```

(R/W) solo (int32, 0: false, 1: true)

```
const Steinberg::FIDString kSendCount = "sendcount"
```

(R) send count [int]

```
const Steinberg::FIDString kSendLevel = "sendlevel"
```

(R/W) send level factor, index is appended to id (e.g. “sendlevel0” for first), also available as string

```
const Steinberg::FIDString kMaxSendLevel = "maxSendlevel"
```

(R) maximum send level factor, also available as string

```
const Steinberg::FIDString kActiveDocumentID = "activeDocumentID"
```

(R) active document identifier, use to get identity of the active document (string)

```
const Steinberg::FIDString kDocumentID = "documentID"
```

(R) document identifier, use to compare identity (string)

```
const Steinberg::FIDString kDocumentName = "documentName"
```

(R) document name, can be displayed to user (string)

```
const Steinberg::FIDString kDocumentFolder = "documentFolder"
```

(R) document folder (string)

```
const Steinberg::FIDString kAudioFolder = "audioFolder"
```

(R) folder for audio files (string)

```
const Steinberg::FIDString kIndexMode = "indexMode"
```

(R) channel index mode (default is flat, see ChannelIndexMode enumeration)

```
namespace Presonus

// namespaces

namespace Presonus::ContextInfo;
namespace Presonus::PlugCanDos;

// typedefs

typedef Steinberg::int16 Pitch;
typedef Steinberg::int16 CCNumber;
typedef Steinberg::int16 CCValue;
typedef Steinberg::int32 VariationID;
typedef Steinberg::uint32 ScoreSymbolID;

// enums

enum AutomationMode;
enum Opcodes;
enum ParamExtraFlags;
enum SlaveMode;

// structs

struct CommandInfo;
struct ICommandList;
struct IContextInfoHandler;
struct IContextInfoHandler2;
struct IContextInfoProvider;
struct IContextInfoProvider2;
```

```
struct IContextInfoProvider3;
struct IEditControllerExtra;
struct IGainReductionInfo;
struct IHostCommandHandler;
struct IInstrumentController;
struct IInstrumentObserver;
struct IKeyAssignmentReceiver;
struct IPlugInViewEmbedding;
struct IPlugInViewScaling;
struct IPlugInViewSystemScalingSupport;
struct ISlaveControllerHandler;
struct ISoundVariationController;
struct ISoundVariationInfo;
struct ISoundVariationList;
struct ISoundVariationObserver;
struct KeyAssignment;
struct ScoreSymbolList;
struct SoundActivationSequence;
struct SoundActivationSequenceItem;
struct SoundVariationData;
struct SoundVariationFolderData;
struct SoundVariationPresetInfo;
struct SymbolID;
struct Vst2SoundVariationEvent;
struct Vst3SoundVariationEvent;

// classes

class DefaultInstrumentController;

// global variables

static const int kGetSoundVariationController = 'GSVC';

 // namespace Presonus
```

```
// namespaces

namespace Presonus;
    namespace Presonus::ContextInfo;
    namespace Presonus::PlugCanDos;
```

Global Namespace

genindex