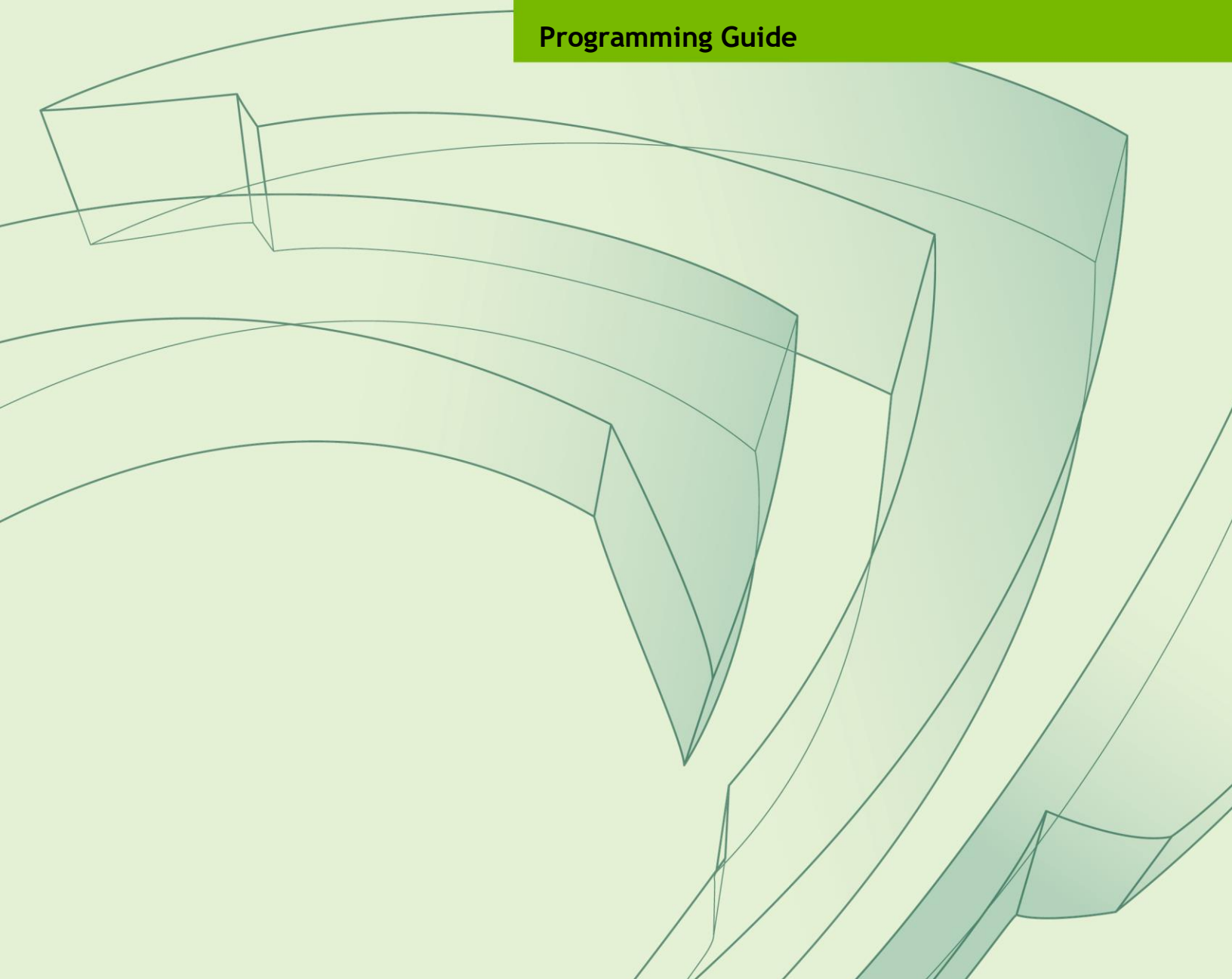
VRWORKS 360 LOW-LEVEL AUDIO

PG-08676-001_v01.1 | December 2017

Programming Guide



DOCUMENT CHANGE HISTORY

PG-08676-001_v01

Version	Date	Description of Change
01	2017-08-03	Initial release
01.1	2017-12-01	Release 1.1

TABLE OF CONTENTS

Chapter 1. INTRODUCTION TO VRWORKS 360 LOW-LEVEL AUDIO

The NVIDIA Sound Field (NVSF) API is a low-level audio processing API for combining several different audio inputs, such as from a microphone array, into a single sound field. NVSF is part of the NVIDIA VRWorks™ 360 Video SDK.

NVSF provides options for performing these tasks:

- Combining any number of inputs into a single sound field
- Applying a stereo-spread effect to the combined audio
- Applying a gain to the combined audio

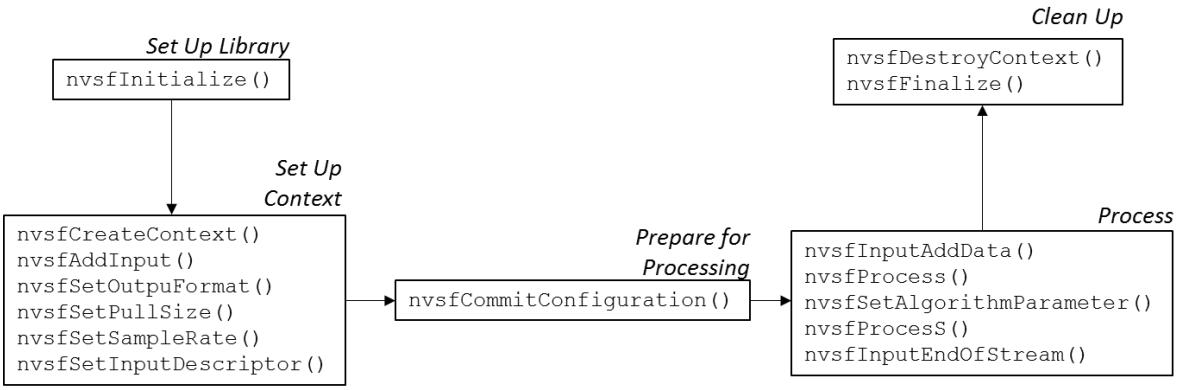
NVSF requires that all inputs have a uniform sample rate and that all inputs are in 32-bit floating point pulse code modulation (PCM) format. The client application is responsible for all conversions of sample rate and format.

NVSF operates on a push-pull model. Audio data from inputs is *pushed* into NVSF as the data becomes available. The client application can then request a *pull* operation to get a sound field. NVSF buffers the input and generates an output in response to a pull request only if sufficient data from each input has been presented.

Chapter 2. GETTING STARTED WITH THE VRWORKS 360 LOW-LEVEL AUDIO API

Use the NVIDIA Sound Field (NVSF) API to directly access low level audio sound field functions in your applications.

The following diagram shows how NVSF functions are organized by the phases of an application in which they are used.



For information about how to use these functions, see the following sections:

For detailed reference information about the API, see Chapter 3.

2.1 NVSF LIBRARY SETUP

To use NVSF, an application must first set up the library. To create the library, `nvsfInitialize()` must be called before any NVSF API functions can be called.

2.2 NVSF CONTEXT SETUP

After initialization, an application must call `nvsfCreateContext()` to create a processing context that holds all the state information for a sound field. Processing contexts may be created with a name. If an application creates a context with the same name, including the empty name, as an existing context, a handle to the existing context is returned and a reference count for that context is increased by 1. Creating a context with the name of an existing context is useful if the input and output components of the client application are structured so that the input and output components cannot easily communicate directly.

Before the NVSF context can be used for processing, inputs must be declared, parameters set, and the configuration committed. Inputs are specified to NVSF with the `nvsfAddInput()` function. Other functions, exist to set the sample rate, output format, and pull size. For details, see “NVSF Processing” on page 3.

2.3 PREPARATION FOR NVSF PROCESSING

After all inputs are specified and parameters are set, the client must call `nvsfCommitConfiguration()` to create the internal processing pipeline and ready NVSF for data processing. The configuration of a context cannot be changed after `nvsfCommitConfiguration()` has been called. After `nvsfCommitConfiguration()` has been called, attempting to add inputs or setting parameters other than the algorithm parameters is an error.

2.4 NVSF PROCESSING

During processing, the client application feeds audio data to NVSF by using `nvsfInputAddData()`. The client requests data from NVSF by using `nvsfProcess()`. When `nvsfProcess()` returns `NVSTITCH_SUCCESS`, it has generated *N* samples for each channel of the output, where *N* is the pull size set with `nvsfSetPullSize()`. If any of the inputs do not have enough data to generate *N* output samples, `nvsfProcess()` returns `NVSTITCH_AUDIO_OUT_OF_DATA`.

When an input stream has ended, the client application should call `nvsfInputEndOfStream()` to indicate that no more data will come from this stream. If other streams are still active, the stream that has ended behaves as if it is being fed silence. After all streams have ended and all of the data that was pushed to NVSF through `nvsfInputAddData()` has been consumed, calls to `nvsfProcess()` return the `NVSTITCH_AUDIO_ALL_STREAMS_ENDED` error code.

At any time during operation, including after `nvsfCommitConfiguration()` has been called, the client application can call `nvsfSetAlgorithmParameter()` to adjust parameters to the algorithm that generates output audio from the inputs. Changes to parameters take effect the next time nvsfProcess() is called.

2.5 NVSF CLEANUP

After the client application has finished using a context, it must call `nvsfDestroyContext()` to destroy the context. If the context was created more than once, a reference counter is decreased by 1. The context is deallocated and destroyed only when the reference count reaches zero.

After the application has finished using NVSF, it should call `nvsfFinalize()` to clean up any remaining API state. Any calls to NVSF API functions after `nvsfFinalize()` has been called are errors.

2.6 NVSF HEADER, LIBRARY, AND DLL FILES

The following table lists the header, library, and dynamic link library (DLL) files required for building an application with NVSF. The table also gives lists folder in the SDK where each file is located.

File Type	File Name	Folder
Header file to include in the application	`nvsf.h`	`/nvstitch/include/`
Library file to link to at compilation time	`nvsf.lib`	`/nvstitch/lib/`
DLL file to add to the application or system path	`nvsf.dll`	`/nvstitch/binary/`

Chapter 3. VRWORKS 360 LOW-LEVEL AUDIO API REFERENCE

3.1 DATA TYPES

3.1.1 Default Values

The default values defined in the subsections that follow are used if no other values are specified.

3.1.1.1 NVSF_DEFAULT_OUTPUT_FORMAT

```
const uint32_t NVSF_DEFAULT_OUTPUT_FORMAT = NVSF_OUTPUT_STEREO_MIXDOWN;
```

3.1.1.2 NVSF_DEFAULT_OUTPUT_GAIN

```
const float    NVSF_DEFAULT_OUTPUT_GAIN = 1.0f;
```

3.1.1.3 NVSF_DEFAULT_PULL_SIZE

```
const uint32_t NVSF_DEFAULT_PULL_SIZE = 1024;
```

3.1.1.4 NVSF_DEFAULT_SAMPLE_RATE

```
const uint32_t NVSF_DEFAULT_SAMPLE_RATE = 48000;
```

3.1.1.5 NVSF_DEFAULT_STEREO_SPREAD_COEFFICIENT

```
const float NVSF_DEFAULT_STEREO_SPREAD_COEFFICIENT = 0.45f;
```

3.1.2 Opaque Types

The opaque types defined in the subsections that follow are for handles in a user application.

3.1.2.1 nvsf_t

```
typedef void* nvsf_t;
```

3.1.2.2 nvsfInput_t

```
typedef uint64_t nvsfInput_t;
```

3.1.2.3 nvsfOutput_t

```
typedef void* nvsfOutput_t;
```

3.2 ENUMERATIONS

3.2.1 nvsfAlgorithmParameter_t

This enumeration defines the parameters for the sound field creation algorithm. Some parameters may or may not be applicable, depending on the selected output format.

Enumerator	Description
NVSF_STEREO_SPREAD_MIX_COEFFICIENT	Controls the amount of the stereo spread effect that is applied in NVSF_OUTPUT_STEREO_MIXDOWN. This parameter value is a 32-bit float in the range [0.0, 1.0]. A value of 0.0 disables the effect and the left and right output channels will be identical. A value of 1.0 turns the effect up to its maximum effect. The default value NVSF_DEFAULT_STEREO_SPREAD_COEFFICIENT is used if this parameter is not set.
NVSF_OUTPUT_GAIN	Sets a uniform gain to be applied to all channels of the output. This parameter is a 32-bit float in the range [0.0 - Inf). It has the effect of multiplying each output sample by the provided gain. The default value of NVSF_DEFAULT_OUTPUT_GAIN is used if this parameter is not set.

3.2.2 nvsfOutputType_t

This enumeration defines all supported output formats. The output format selected controls how the NVIDIA Sound Field (NVSF) API processes the data and how many output channels are generated.

Enumerator	Description
`NVSF_OUTPUT_STEREO_MIXDOWN`	Applies a stereo spatialization effect to all inputs and then mixes all inputs together evenly. The stereo spatialization effect can be controlled by calling `nvsfSetAlgorithmParameter()` with the argument `NVSF_STEREO_SPREAD_MIX_COEFFICIENT`. There are two output channels: •`channel0` is left •`channel1` is right

3.3 STRUCTURES

3.3.1 nvsfInputDescriptor_t

```
typedef struct NVSF_INPUT_DESCRIPTOR_struct
{
    uint32_t                numChannels;
    nvstitchAudioInputType  type;
    nvstitchPose_t          pose;
} nvsfInputDescriptor_t;
```

3.3.1.1 Members

`numChannels`

Type: `uint32_t`

The number of channels in this input.

`type`

Type: `nvstitchAudioInputType`

The type of input, for example, a microphone.

`Pose`

Type: `nvstitchPose_t`

Information about the location and orientation of the input.

3.3.1.2 Remarks

This structure contains information about a single input.

3.4 FUNCTIONS

3.4.1 nvsfAddInput

```
nvstitchResult nvsfAddInput(
    nvsf_t                        context,
    nvsfInput_t*                  pInputHandle,
    const nvsfInputDescriptor_t* pDescriptor
);
```

3.4.1.1 Parameters

`Context`

Type: nvsf_t

A valid NVSF context handle.

`pInputHandle [out]`

Type: `nvsfInput_t*`

A pointer to an input handle that contains the handle to the input created by this call when the function returns.

`pDescriptor`

Type: `const nvsfInputDescriptor_t*`

A pointer to an `nvsfInputDescriptor_t` structure populated with values describing the properties of the input being added to NVSF.

3.4.1.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_STATE` if NVSF is not initialized or the context is invalid
- `NVSTITCH_AUDIO_CONFIGURATION_COMMITTED` if NVSF properties cannot be changed because the configuration has already been committed
- `NVSTITCH_ERROR_NULL_POINTER` if `pInputHandle` or `pDescriptor` are `NULL`

3.4.1.3 Remarks

This function adds a new input to the specified NVSF context. This function must be called before `nvsfCommitConfiguration()` is called.

3.4.2 nvsfCommitConfiguration

```
nvstitchResult nvsfCommitConfiguration(
    nvsf_t context
);
```

3.4.2.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

3.4.2.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_STATE` if NVSF is not initialized or the context is invalid
- `NVSTITCH_ERROR_NOT_IMPLEMENTED` if the output format is not supported
- `NVSTITCH_ERROR_GENERAL` if an error occurs during the setup of internal processing components

3.4.2.3 Remarks

This function commits the NVSF configuration and prepares it for processing.

3.4.3 nvsfCreateContext

```
nvstitchResult nvsfCreateContext(
    nvsf_t*      pContext,
    const char   name[],
    const size_t nameLength
);
```

3.4.3.1 Parameters

`pContext [out]`

Type: `nvsf_t`

A pointer to an NVSF context handle. When the function returns, this parameter contains the handle to the NVSF context.

`name`

Type: `const char`

A C-style string name for the context.

`nameLength`

Type: `const size_t`

The number of characters in the name string.

3.4.3.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_STATE` if NVSF is not initialized
- `NVSTITCH_ERROR_NULL_POINTER` if `pContext` is `NULL`

3.4.3.3 Remarks

This function creates a context with the specified name. If a context with this name exists, a handle to the existing context is returned and the reference count of that context is increased by 1.

3.4.4 nvsfDestroyContext

```
nvstitchResult nvsfDestroyContext(
    nvsf_t context
);
```

3.4.4.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

3.4.4.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_STATE` if NVSF is not initialized
- `NVSTITCH_BAD_PARAMETER` if the specified input is not part of this context

3.4.4.3 Remarks

This function decreases by 1 the reference count of the context with the specified name. If the reference count has reached zero, this function also destroys the context.

3.4.5 nvsfFinalize

```
nvstitchResult nvsfFinalize();
```

3.4.5.1 Parameters

None.

3.4.5.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success

3.4.5.3 Remarks

This function finalizes the NVSF API. No other functions may be called after this function has been called.

3.4.6 nvsfGetInputDescriptor

```
nvstitchResult nvsfGetInputDescriptor(
    nvsf_t                  context,
    nvsfInput_t             inputHandle,
    nvsfInputDescriptor_t * pDescriptor
);
```

3.4.6.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

`inputHandle`

Type: `nvsfInput_t`

A valid input handle.

`pDescriptor [out]`

Type: `nvsfInputDescriptor_t*`

A pointer to an `nvsfInputDescriptor_t` structure that this function will fill with the information NVSF has about the specified input.

3.4.6.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_STATE` if NVSF is not initialized or the context is invalid
- `NVSTITCH_ERROR_BAD_PARAMETER` if the requested input is not part of this context

3.4.6.3 Remarks

This function returns the input descriptor of an existing input.

3.4.7 nvsfGetOutputFormat

```
nvstitchResult nvsfGetOutputFormat(
    nvsf_t                   context,
    nvstitchAudioOutputType* pType
);
```

3.4.7.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

`pType [out]`

Type: `nvstitchAudioOutputType*`

A pointer to an `nvstitchAudioOutputType` that this function will fill with the output type set for the specified context.

3.4.7.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_STATE` if NVSF is not initialized or the context is invalid
- `NVSTITCH_ERROR_NULL_POINTER` if `pType` is `NULL`

3.4.7.3 Remarks

Returns the output format this NVSF context is set to generate

3.4.8 nvsfGetPullSize

```
nvstitchResult nvsfGetPullSize(
    nvsf_t    context,
    uint32_t* pNumSamples
);
```

3.4.8.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

`pNumSamples [out]`

Type: `uint32_t`

A pointer to a 32-bit unsigned integer that this function will fill. The integer is the number of samples returned by a successful call to `nsfProcess()` in the same context.

3.4.8.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_STATE` if NVSF is not initialized or the context is invalid
- `NVSTITCH_ERROR_NULL_POINTER` if `pNumSamples` is `NULL`.

3.4.8.3 Remarks

This function returns the number of samples that each call to `nvsfProcess()` will generate.

3.4.9 nvsfGetSampleRate

```
nvstitchResult nvsfGetSampleRate(
    nvsf_t    context,
    uint32_t* pSampleRate
);
```

3.4.9.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

`pSampleRate [out]`

Type: `uint32_t*`

A pointer to a 32-bit unsigned integer that this function will fill with the sample rate of the specified NVSF context.

3.4.9.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_STATE` if NVSF is not initialized or the context is invalid
- `NVSTITCH_ERROR_NULL_POINTER` if `pSampleRate` is `NULL`

3.4.9.3 Remarks

This function returns the universal sample rate for all inputs and outputs of NVSF.

3.4.10 nvsfGetVersion

```
nvstitchResult nvsfGetVersion(
    uint32_t* pVersion
);
```

3.4.10.1 Parameters

`pVersion [out]`

Type: `uint32_t*`

A pointer to the returned version number.

3.4.10.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER` if `pVersion` is `NULL`

3.4.10.3 Remarks

This function returns the version number of the NVSF API as *MMNN*, where *MM* is the major version number and *NN* is the minor version number.

3.4.11 nvsfInitialize

```
nvstitchResult nvsfInitialize();
```

3.4.11.1 Parameters

None.

3.4.11.2 Return Value

Returns `NVSTITCH_SUCCESS` on success.

3.4.11.3 Remarks

This function initializes the NVSF API. It must be called before any other functions are called.

3.4.12 nvsfInputAddData

```
nvstitchResult nvsfInputAddData(
....nvsf_t      context,
    nvsfInput_t inputIdentifier,
    float**     pData,
    uint32_t    numSamples,
    uint64_t    timestamp
);
```

3.4.12.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

`inputIdentifier`

Type: `nvsfInput_t`

The input handle for which the samples are to be pushed.

`pData`

Type: `float**`

An array of *N* pointers, where *N* is the number of channels in the specified input. The pointers in `pData` point to contiguous (not-interleaved) arrays of `numSamples` floats for each channel.

`numSamples`

Type: `uint32_t`

The number of samples to be pushed.

`timestamp`

Type: `uint64_t`

Reserved for future use. Timestamps are not propagated by NVSF in this version of NVSF.

3.4.12.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_AUDIO_CONFIGURATION_NOT_COMMITTED` if the call to this function is illegal because the configuration has not been committed
- `NVSTITCH_ERROR_BAD_STATE` if the requested input is not part of this context, NVSF is not initialized, or the context is invalid
- `NVSTITCH_ERROR_NULL_POINTER` if `pData` is `NULL`

3.4.12.3 Remarks

This function pushes the specified number of samples for all channels of a specified input into NVSF.

3.4.13 nvsfInputEndOfStream

```
nvstitchResult nvsfInputEndOfStream(
    nvsf_t      context,
    nvsfInput_t inputIdentifier
);
```

3.4.13.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

`inputIdentifier`

Type: `nvsfInput_t`

The input handle of the stream that has ended.

3.4.13.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_STATE` if the specified input is not part of this context, the context is invalid, or NVSF is not initialized

3.4.13.3 Remarks

This function indicates that no more data will come from this stream. After this function is called, the input behaves as though silence (represented by zeros) has been pushed into the buffer until all streams end and all data pushed into all streams has been consumed.

3.4.14 nvsfProcess

```
nvstitchResult nvsfProcess(
    nvsf_t context,
    float ** pData,
    uint64_t* pTimeStamp
);
```

3.4.14.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

`pData`

Type: `float **`

An array of *N* pointers, where *N* is the number of channels as determined by the output format. Each pointer must point to a contiguous (non-interleaved) array where NVSF will put the results.

`pTimeStamp`

Type: `uint64_t*`

Reserved for future use. Timestamps are not propagated by NVSF in this version of NVSF.

3.4.14.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_AUDIO_CONFIGURATION_NOT_COMMITTED` if the call to this function is illegal because the configuration has not been committed
- `NVSTITCH_AUDIO_OUT_OF_DATA` if there is insufficient data from at least one input to generate an output
- `NVSTITCH_AUDIO_ALL_STREAMS_ENDED` if all streams have ended and all buffered data has been consumed
- `NVSTITCH_ERROR_BAD_STATE` if the internal processing structures encountered an unexpected state, the context is invalid, or NVSF is not initialized

3.4.14.3 Remarks

This function requests one set of output buffers. If the call to this function succeeds, the output is stored in the arrays specified by `pData` If NVSF cannot generate the output because of insufficient data, the contents of the arrays specified by `pData` are invalid and the function returns `NVSTITCH_AUDIO_OUT_OF_DATA` or `NVSTITCH_AUDIO_ALL_STREAMS_ENDED`.

3.4.15 nvsfSetAlgorithmParameter

```
nvstitchResult nvsfSetAlgorithmParameter(
    nvsf_t context,
    nvsfAlgorithmParameter_t parameter,
    void* value
);
```

3.4.15.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

`parameter`

Type: `nvsfAlgorithmParameter_t`

The enumeration for the algorithm parameter to be set.

`value`

Type: `void*`. The type of the value depends on the algorithm parameter value as shown in the following table.

Parameter	Type of `value`	Valid Range
`NVSF_STEREO_SPREAD_MIX_COEFFICIENT`	`float`	[0.0 - 1.0]
`NVSF_OUTPUT_GAIN`	`float`	[0.0 - Inf)

3.4.15.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_PARAMETER` if any of the following conditions applies:
  - `parameter` does not specify a recognized algorithm parameter.
  - The value passed for the parameter is outside its valid range.
  - NVSF is not initialized
  - The context is invalid
- `NVSTITCH_ERROR_NULL_POINTER` if value is `NULL`

3.4.15.3 Remarks

This function changes the value of the specified algorithm parameter to the contents of the memory value points to.

3.4.16 nvsfSetInputDescriptor

```
nvstitchResult nvsfSetInputDescriptor(
    nvsf_t                  context,
    nvsfInput_t             inputHandle,
    nvsfInputDescriptor_t* pDescriptor
);
```

3.4.16.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

`inputHandle`

Type: `nvsfInput_t`

The input handle for the selected input.

`pDescriptor`

Type: `nvsfInputDescriptor_t*`

A pointer to an `nvstInputDescriptor_t` structure whose contents will replace the current information for the specified input within the specified NVSF context.

3.4.16.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_STATE` if NVSF is not initialized or the context is invalid
- `NVSTITCH_AUDIO_CONFIGURATION_COMMITTED` if NVSF properties cannot be changed because the configuration has already been committed
- `NVSTITCH_ERROR_BAD_PARAMETER` if the requested input is not part of this context

3.4.16.3 Remarks

This function changes the input descriptor of an existing input. This function must be called before `nvsfCommitCofiguration()` is called.

3.4.17 nvsfSetOutputFormat

```
nvstitchResult nvsfSetOutputFormat(
    nvsf_t                  context,
    nvstitchAudioOutputType type
);
```

3.4.17.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

`type`

Type: `nvstitchAudioOutputType`

An enumerated output type. This value selects the type of processing that NVSF performs.

3.4.17.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success

- NVSTITCH_ERROR_BAD_STATE if NVSF is not initialized or the context is invalid
- NVSTITCH_AUDIO_CONFIGURATION_COMMITTED if NVSF properties cannot be changed because configuration has already been committed

3.4.17.3 Remarks

This function sets the output format that will be generated by this NVSF context. This function must be called before nvsfCommitConfiguration() is called.

3.4.18 nvsfSetPullSize

```
nvstitchResult nvsfSetPullSize(
    nvsf_t   context,
    uint32_t numSamples
);
```

3.4.18.1 Parameters

context

Type: nvsf_t

A valid NVSF context handle.

numSamples

Type: uint32_t

The number of samples that will be generated by a successful call to nvsfProcess().

numSamples must be greater than or equal to 32.

3.4.18.2 Return Value

Returns one of the following values:

- NVSTITCH_SUCCESS on success
- NVSTITCH_ERROR_BAD_STATE if NVSF is not initialized or the context is invalid
- NVSTITCH_AUDIO_CONFIGURATION_COMMITTED if NVSF properties cannot be changed because configuration has already been committed
- NVSTITCH_ERROR_BAD_PARAMETER if numSamples is outside the valid range

3.4.18.3 Remarks

This function sets the number of samples that each call to nvsfProcess() will generate. This function must be called before nvsfCommitConfiguration() is called.

3.4.19 nvsfSetSampleRate

```
nvstitchResult nvsfSetSampleRate(
    nvsf_t   context,
    uint32_t sampleRate
);
```

3.4.19.1 Parameters

`context`

Type: `nvsf_t`

A valid NVSF context handle.

`sampleRate`

Type: `uint32_t`

The sample rate of input and output audio buffers.

`sampleRate` must be in the range [32000, 192000].

3.4.19.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_STATE` if NVSF is not initialized or the context is invalid
- `NVSTITCH_AUDIO_CONFIGURATION_COMMITTED` if NVSF properties cannot be changed because the configuration has already been committed
- `NVSTITCH_ERROR_BAD_PARAMETER` if `sampleRate` is outside the valid range

3.4.19.3 Remarks

This function sets the universal sample rate for all inputs and outputs of NVSF. This function must be called before `nvsfCommitConfiguration()` is called.

Notice

The information provided in this specification is believed to be accurate and reliable as of the date provided. However, NVIDIA Corporation (“NVIDIA”) does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This publication supersedes and replaces all other specifications for the product that may have been previously supplied.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and other changes to this specification, at any time and/or to discontinue any product or service without notice. Customer should obtain the latest relevant specification before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer. NVIDIA hereby expressly objects to applying any customer general terms and conditions with regard to the purchase of the NVIDIA product referenced in this specification.

NVIDIA products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on these specifications will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this specification. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this specification, or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this specification. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA. Reproduction of information in this specification is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

ROVI Compliance Statement

NVIDIA Products that support Rovi Corporation’s Revision 7.1.L1 Anti-Copy Process (ACP) encoding technology can only be sold or distributed to buyers with a valid and existing authorization from ROVI to purchase and incorporate the device into buyer’s products.

This device is protected by U.S. patent numbers 6,516,132; 5,583,936; 6,836,549; 7,050,698; and 7,492,896 and other intellectual property rights. The use of ROVI Corporation's copy protection technology in the device must be authorized by ROVI Corporation and is intended for home and other limited pay-per-view uses only, unless otherwise authorized in writing by ROVI Corporation. Reverse engineering or disassembly is prohibited.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2017 NVIDIA Corporation. All rights reserved.