VRWORKS 360 HIGH-LEVEL VIDEO

PG-08677-001_v01.1 | December 2017

Programming Guide

DOCUMENT CHANGE HISTORY

PG-08677-001_v01.1

Version	Date	Description of Change
01	2017-08-07	Initial release
01.1	2017-12-08	Release 1.1

TABLE OF CONTENTS

Chapter 1. INTRODUCTION TO VRWORKS 360 HIGH-LEVEL VIDEO

The NVIDIA VRWorks™ 360 High-Level Video Stitching (NVStitch) SDK is a software package for creating 360-degree panoramic images or movies by stitching together multiple camera views provided by one of the many virtual reality (VR) rigs available on the market today. It provides a high-level interface to different components that enable the generation of these images and movies.

The SDK runs on recent models of GeForce and Quadro cards. Decoding and encoding are hardware accelerated, allowing real-time performance with high resolution input and output. The video processing part of the pipeline is CUDA-accelerated and optimized to use the low-level video SDK without loss of performance. Audio processing is executed on the CPU.

1.1 FEATURES OF THE SDK

The VRWorks 360 Video Stitching SDK provides these features:

- Uncompressed input and compressed input (H.264, MP4, or MOV)
- Compressed (H.264 or MP4) and uncompressed output
- GPU-accelerated pipeline with demonstrated live stitch capability
- Offline stitch selection
- Quality preset selection
- Audio stitch with optional stereo stretch
- Support for both pulse-code modulation (PCM) and Advanced Audio Coding (AAC) audio input formats
- Customizable calibration selection

1.2 FUNCTIONALITY OF THE SDK

In addition to providing a high-level interface to the low-level components, the VRWorks 360 Video Stitching SDK provides the following functionality:

- Video demultiplexing and multiplexing
- Video decoding and encoding
- Audio decoding and encoding

The functionality of the high-level SDK is used to enable the functionality of low-level components to be applied to compressed input and to produce compressed output.

Data flow in the SDK is illustrated in Figure 1.

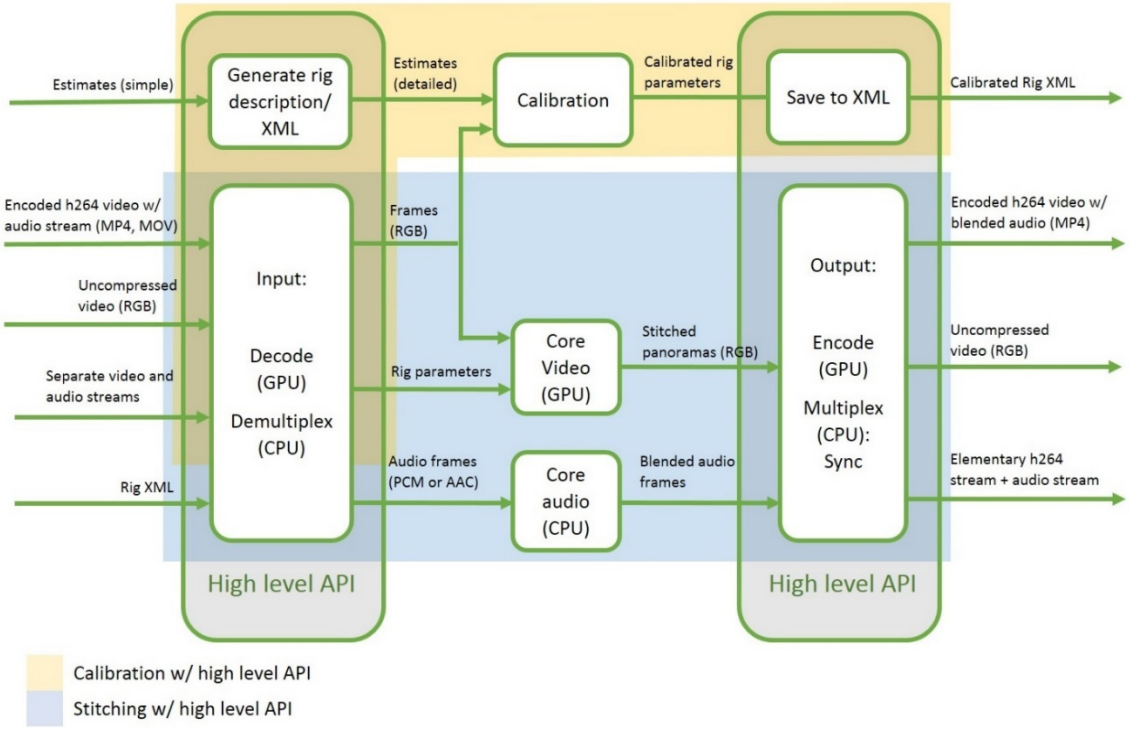

Figure 1 High-Level SDK Data Flow

1.3 INPUT TO AND OUTPUT FROM THE SDK

The SDK accepts as input up to 32 camera streams and creates a single stitched image or MP4 video file as output. Monoscopic and Stereoscopic output is available, depending on the number of cameras and the degree of overlap of the horizontal field of view in the source footage.

The following input and output image formats are supported:

- Uncompressed RGBX
- H264 frames
- MP4/MOV

The following input and output audio formats are supported:

- PCM
- AAC

Input can be passed through a GPU or system memory buffer. The MP4/MOV files are read directly by the SDK.

In addition to the footage from the cameras, the SDK requires a description of the input in XML format. This description includes the specifics of the input footage and the nature of the VR rig.

1.4 CALIBRATION SUPPORT

The SDK offers a calibration step in addition to the stitching pipeline to obtain more precise specifications of the content to be stitched. The calibration takes as input bitmap images from each camera and an optional generic rig specification. It provides an output XML file tuned to the source footage and the associated camera system.

1.5 LOW-LEVEL COMPONENTS IN THE HIGH-LEVEL SDK

The high-level SDK encompasses the following low-level components:

- NVCalib
- Low-Level Video SDK
- Low-Level Audio SDK

Because these components are not used directly by applications developed by using the high-level SDK, they are not described in this document. For details about the low-Level components in the high-level SDK, refer to the documentation that is provided separately for these components.

Chapter 2. GETTING STARTED VRWORKS 360 HIGH-LEVEL VIDEO

Use NVIDIA VRWorks 360 High-Level Video to calibrate a video rig, stitch video and blend audio.

2.1 CALIBRATING A VIDEO RIG

Figure 2 shows the execution flow for using VRWorks 360 High-Level Video to calibrate a video rig.

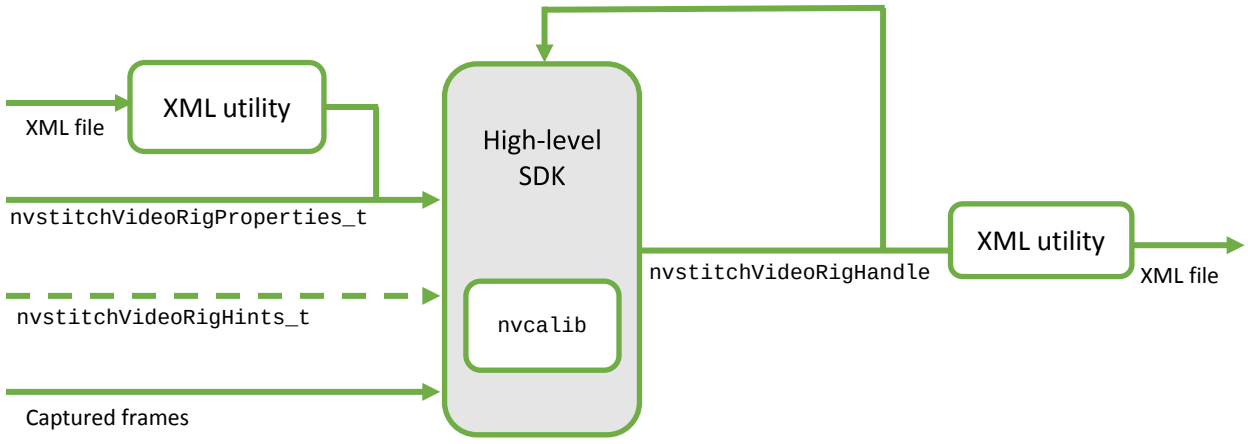

Figure 2 Calibration Execution Flow

All structures in this flow are defined in the header file `nvstitch_common_video.h`. All functions are declared in the header file `nvstitch.h`.

2.1.1 Specifying Calibration Estimates

As shown in Figure 2, you can specify calibration estimates in one of the following ways:

- Initialize a `rigProperties_t` structure from an XML file by using the utility functions provided with the SDK, and instantiate a video rig.
- Use a blank `rigProperties_t` structure to calibrate without initial estimates. The lack of input estimates can result in lower quality calibration in some cases.
- Generate estimates from simple hints by using the `nvstitchCreateVideoRigFromHints()` function. Only certain rig types are supported by this function.
- Manually initialize a `rigProperties_t` structure and instantiate a video rig.

2.1.2 Creating and Feeding a Calibration Instance

When creating a calibration instance, directly pass as a parameter the calibration estimates in the form of a video rig instance.

After the instance has been created, use the `nvstitchFeedCalibrationInput()` function to individually feed the instance with each frame in a set of frames captured by the cameras in the rig.

2.1.3 Performing the Calibration

After all input frames have been fed to the calibration instance, call `nvstitchCalibrate()` to perform the calibration. The result of this function call is a newly calibrated video rig instance. This instance can be used directly as input to the stitching pipeline, saved as an XML file, or both used directly and saved. If you want to refine the results with additional frames, you can also use the calibrated rig as input for the next iteration of the calibration process.

2.2 STITCHING AUDIO AND VIDEO

Figure 3 shows the execution flow for using VRWorks 360 High-Level Video to stitch audio and video.

For detailed reference information about the functions and data types in this API, see Chapter 3.

Structures describing video and audio rigs are defined in the following header files:

- Video rig structures are defined in `nvstitch_common_video.h`.

- Audio rig structures are defined in `nvstitch_common_audio.h`.

Other structures and all functions in this flow are declared in the header file `nvstitch.h`.

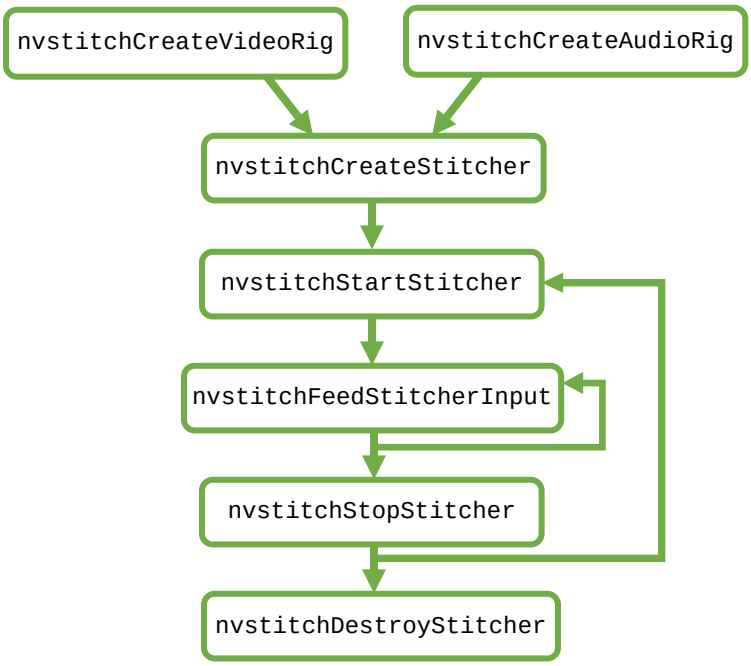

Figure 3 Stitching Execution Flow

2.2.1 Creating Audio and Video Rig Instances

The first step in setting up a stitching pipeline is to create a video rig instance, which describes geometrical properties of every camera in the rig and the rig itself.

As described in "Specifying Calibration Estimates" on page 5, video rig instance can be created in several ways. To achieve best quality, calibrate the initial rig estimate as described in that section.

After the video rig has been calibrated, video rig properties can be reused to stitch different feeds from a given camera.

If you want to include blended audio in the output, create an audio rig instance. An audio rig instance includes all relevant information on the audio sources. Use one of the utility functions in the SDK to initialize `nvstitchAudioRigProperties_t` objects from an XML file. Then call the function `nvstitchCreateAudioRigInstance()` to create the audio rig instance.

2.2.2 Configuring a Stitcher Instance

The next step in setting up a stitching pipeline is to set the `nvstitchStitcherProperties_t` structure. This structure contains all parameters needed to properly configure the stitcher, including input format and settings such as quality presets. The SDK includes a utility function that populates `nvstitchStitcherProperties_t` structures from an XML file.

2.2.3 Creating a Stitcher Instance

After the rigs and the structure containing the stitcher properties are initialized, call the `nvstitchCreateStitcher()` function to create a stitcher instance. The function returns a handle to the instance through an output parameter. If an error occurs in the initialization, an error code is returned, and the handle parameter is not set.

In systems where multiple GPUs are available, GPU selection is automatic and depends whether the stitching is stereoscopic or monoscopic:

- For stereoscopic stitching, all compatible GPUs are used to spread the computational load and decrease execution time.
- For monoscopic stitching, which is less computationally intensive, only the most suitable GPU is used.

This behavior is different than the low-level video SDK, in which the GPUs are selected manually.

2.2.4 Starting and Feeding a Stitcher Instance

When the application is ready to start feeding input video and audio data, call the `nvstitchStartStitcher()` function to set the stitcher into the running state, in which it can accept input data.

When the stitcher is running, feed the stitcher instance with input either as individual frames for buffer input, or as files for MP4 input. Stitching is asynchronous.

To read the output, use one of the following mechanisms:

- Use a callback that is executed after each frame of buffer output. The callback is executed in a separate thread.
- Access the output buffers directly by calling these functions:
  - `nvstitchGetOutputBuffer()`
  - `nvstitchGetOutputDevice()`
  - `nvstitchoGetInputStream()`

2.2.5 Stopping and Destroying a Stitcher Instance

After the application has fed all input to the stitcher, call `nvstitchStopStitcher()`. This function blocks until all pending stitching is completed.

A single stitcher instance can be started and stopped multiple times to stitch separate input feeds from the same camera.

After the last input set is fed into the stitcher and the instance is stopped, destroy the instance by calling the `nvstitchDestroyStitcher()` function. This function can also be used at any time to interrupt the stitching process. However, valid output is not guaranteed if the instance is destroyed before the stitcher is stopped.

2.2.6 Destroying Audio and Video Rig Instances

Destroy video and audio rig instances when they are no longer needed.

- To destroy a video rig instance, call the `nvstitchDestroyVideoRigInstance()` function.
- To destroy an audio rig instance, call the `nvstitchDestroyAudioRigInstance()` function.

Chapter 3. VRWORKS 360 HIGH-LEVEL VIDEO API REFERENCE

3.1 COMMON AUDIO ENUMERATIONS

These enumerations are defined in the header file `nvstitch_common_audio.h`.

3.1.1 nvstitchAudioDataSource

This enumeration indicates whether the audio data that is being fed to a stitcher is read from a file or presented to the stitcher in memory buffers.

Enumerator	Description
`NVSTITCH_AUDIO_DATA_SOURCE_FILE = 0`	The audio data is read from a file specified by its filename.
`NVSTITCH_AUDIO_DATA_SOURCE_BUFFER`	The audio data is presented from memory buffers.

3.1.2 nvstitchAudioFormat

This enumeration indicates the encoding type or data format of an audio stream.

Enumerator	Description
`NVSTITCH_AUDIO_FORMAT_PCM16LE`	Little-endian 16-bit signed PCM.
`NVSTITCH_AUDIO_FORMAT_PCM24LE`	Little-endian 24-bit signed PCM.
`NVSTITCH_AUDIO_FORMAT_PCM32LE`	Little-endian 32-bit signed PCM.
`NVSTITCH_AUDIO_FORMAT_PCM32FLOAT`	Single-precision (32-bit) little-endian floating-point PCM.
`NVSTITCH_AUDIO_FORMAT_AAC`	AAC stream with ADTS headers.

Enumerator	Description
NVSTITCH_AUDIO_FORMAT_AAC_LC_RAW	RAW AAC stream.

3.1.3 nvstitchAudioInputType

This enumeration indicates the type of an audio source.

Enumerator	Description
NVSTITCH_AUDIO_INPUT_TYPE_OMNI = 0	Each channel of this input is treated as independent and having no directional information.
NVSTITCH_AUDIO_INPUT_TYPE_SHOTGUN_MIC	Each channel of this input is treated as independent and having identical directionality with a shotgun microphone-style pickup pattern.
NVSTITCH_AUDIO_INPUT_TYPE_CARDIOID_MIC	Each channel of this input is treated as independent and having identical directionality with a cardioid microphone-style pickup pattern.
NVSTITCH_AUDIO_INPUT_TYPE_SUPERCARDIOID_MIC	Each channel of this input is treated as independent and having identical directionality with a shotgun supercardioid-style pickup pattern.

3.1.4 nvstitchAudioOutputType

This enumeration designates the type of output sound field to be created.

Enumerator	Description
NVSTITCH_AUDIO_OUTPUT_NONE = 0	No audio blending (no audio output).
NVSTITCH_AUDIO_OUTPUT_STEREO_MIXDOWN	Mixes sources with stereo spreading effect.

3.2 COMMON VIDEO ENUMERATIONS

These enumerations are defined in the header file `nvstitch_common_video.h`.

3.2.1 nvstitchCameraLayout

This enumeration defines the possible roles of a camera in the stereo stitching pipeline.

Enumerator	Description
NVSTITCH_CAMERA_LAYOUT_EQUATORIAL = 0	Equatorial ring camera (default).

Enumerator	Description
NVSTITCH_CAMERA_LAYOUT_GENERAL	Other cameras, currently not used for stereo stitching.

3.2.2 nvstitchCameraSetPropertyFlag

This enumeration denotes the camera properties that were set by the user. The `set_flags` field in `nvstitchCameraProperties_t` contains a set of `nvstitchCameraSetPropertyFlag` values.

All properties except camera translation must be set to use the camera for stitching.

Enumerator	Description
CAMERA_PRINCIPAL_POINT_SET = 1 << 0	
CAMERA_DISTORTION_TYPE_SET = 1 << 1	
CAMERA_DISTORTION_COEFFICIENTS_SET = 1 << 2	
CAMERA_FISHEYE_RADIUS_SET = 1 << 3	
CAMERA_ROTATION_SET = 1 << 4	
CAMERA_TRANSLATION_SET = 1 << 5	
CAMERA_ALL_SET = (1 << 6) - 1	

3.2.3 nvstitchDistortionType

This enumeration defines lens distortion models, which determine how distortion coefficients in `nvstitchCameraProperties_t` are interpreted.

Enumerator	Description
NVSTITCH_DISTORTION_TYPE_FISHEYE = 0	Fisheye lens (f θ) with 4 radial distortion coefficients: $f\theta\ (1 + k_0\theta^2 + k_1\theta^4 + k_2\theta^6 + k_3\theta^8)$.
NVSTITCH_DISTORTION_TYPE_BROWN	Brown perspective (f tan θ) model with 3 radial distortion coefficients (k_0, k_1, k_4) and 2 tangential coefficients (k_2, k_3).

3.2.4 nvstitchEye

This enumeration defines the corresponding eye for the output stereoscopic and monoscopic panoramas.

Enumerator	Descriptor
NVSTITCH_EYE_LEFT = 0	Left eye stereo panorama.
NVSTITCH_EYE_RIGHT	Right eye stereo panorama.
NVSTITCH_EYE_MONO	Monoscopic panorama.

3.2.5 nvstitchStitcherQuality

This enumeration defines presets for the quality of video stitching, which specify the trade-off between quality and performance.

Enumerator	Description
`NVSTITCH_STITCHER_QUALITY_HIGH = 0`	High quality-low performance, which is applicable to offline stitching.
`NVSTITCH_STITCHER_QUALITY_MEDIUM`	Balanced quality-performance, which is applicable to real-time stitching on high-end systems.
`NVSTITCH_STITCHER_QUALITY_LOW`	Lower quality-high performance, which is applicable to real-time stitching with a wider range of systems.

3.2.6 nvstitchPanoramaProjectionType

This enumeration is used in `nvstitchStitcherProperties_t` and `nvssVideoStitcherProperties_t` to specify the output panorama format.

Enumerator	Description
`NVSTITCH_PANORAMA_PROJECTION_EQUIRECTANGULAR = 0`	Equirectangular panorama in a 2:1 ratio (for example, 3840×1920). Stereoscopic output consists of two vertically stacked panoramas.

3.2.7 nvstitchStitcherPipelineType

This enumeration is used in `nvstitchVideoPipelineOptions_t` and `nvssVideoStitcherProperties_t` to specify the stitching algorithm.

Enumerator	Description
`NVSTITCH_STITCHER_PIPELINE_NONE = 0`	No stitching (no image output).
`NVSTITCH_STITCHER_PIPELINE_MONO`	Monoscopic stitching.
`NVSTITCH_STITCHER_PIPELINE_STEREO`	Stereoscopic stitching.

3.3 GENERALLY APPLICABLE ENUMERATIONS

These enumerations are defined in the header file `nvstitch.h`.

3.3.1 nvstitchEncoderSettingType

This enumeration is used in `nvstitchEncodeSettings_t` to define encoder settings.

Enumerator	Description
`NVSTITCH_ENCODER_AVG_BITRATE = 0`	A setting that specifies the average bitrate of the output video, in bits per second.
`NVSTITCH_ENCODER_MAX_BITRATE`	A setting that specifies maximum allowed bitrate of the output video, in bits per second.

3.3.2 nvstitchMediaForm

This enumeration is used in `nvstitchStitcherProperties_t` and `nvstitchPayload_t` to define the forms (containers) in which media can be fed into a stitcher.

Enumerator	Description
`NVSTITCH_MEDIA_FORM_NONE = 0`	Empty payload (use with `nvstitchGetStitcherOutputBuffer`)
`NVSTITCH_MEDIA_FORM_HOST_BUFFER`	System memory buffer.
`NVSTITCH_MEDIA_FORM_DEVICE_BUFFER`	GPU memory buffer.
`NVSTITCH_MEDIA_FORM_FILE`	A file on a local disk drive.

3.3.3 nvstitchMediaFormat

This enumeration is used in `nvstitchStitcherProperties_t` to define supported image formats.

Enumerator	Description
`NVSTITCH_MEDIA_FORMAT_RGBA8UI = 0`	32-bit RGB format with 8-bit channels.
`NVSTITCH_MEDIA_FORMAT_YUV420`	Planar YUV420, not yet supported.
`NVSTITCH_MEDIA_FORMAT_H264_BITSTREAM`	A stream of H264 frames.
`NVSTITCH_MEDIA_FORMAT_MP4`	MP4 file format (subset H.264+AAC).

3.3.4 nvstitchVideoRigType

This enumeration is used in `nvstitchVideoRigHints_t` to define the hint types.

Enumerator	Description
NVSTITCH_RIG_TYPE_RING = 0	Homogeneous rig consisting of an equatorial ring of cameras.

3.4 COMMON AUDIO STRUCTURES

These structures are defined in the header file nvstitch_common_audio.h.

3.4.1 nvstitchAudioConfiguration_t

```
typedef struct
{
    uint32_t            version;
    nvstitchAudioFormat format;
    uint32_t            channels;
    uint32_t            sampleRate;
    uint32_t            bitRate;
    uint32_t            samplesPerFrame;
}
nvstitchAudioConfiguration_t;
```

3.4.1.1 Members

version

Type: uint32_t

Set to NVSTITCH_VERSION.

format

Type: nvstitchAudioFormat

The data format of the audio input or output.

channels

Type: uint32_t

The number of audio channels.

sampleRate

Type: uint32_t

The sampling rate of the audio.

`bitRate`

Type: `uint32_t`

The bit rate of the audio input.

`samplesPerFrame`

Type: `uint32_t`

The number of samples per audio frame.

3.4.1.2 Remarks

This structure contains information describing the audio data format for a single input or output.

3.4.2 nvstitchAudioPayload_t

```
typedef struct
{
    union
    {
        struct {void* ptr; uint64_t timestamp; uint32_t size; } buffer;
        struct {char* name;}                                     file;
    } payload;
}
```

3.4.2.1 Members

`buffer`

Type: `struct {void* ptr; uint64_t timestamp; uint32_t size; }`

Fill when `nvstitchAudioDataSource` is `NVSTITCH_AUDIO_DATA_SOURCE_BUFFER`.

- `ptr` is a pointer to a buffer containing audio data
- `timestamp` is reserved for future use. Timestamps are not propagated in this release.
- `size` is the size in bytes of the data stored in the buffer passed through ptr.

`file`

Type: `struct {char* name;}`

Fill when `nvstitchAudioDataSource` is `NVSTITCH_AUDIO_DATA_SOURCE_FILE`.

`name` is a C-style string that contains the filename of the file containing audio data.

3.4.2.2 Remarks

This structure defines an audio payload. An audio payload contains audio data in buffers or the name of a file from which data can be read.

3.4.3 nvstitchAudioRigProperties_t

```
typedef struct
{
    uint32_t                          version;
    uint32_t                          num_sources;
    nvstitchAudioSourceProperties_t* sources;
}
nvstitchAudioRigProperties_t;
```

3.4.3.1 Members

`version`

Type: `uint32_t`

Set to `NVSTITCH_VERSION`.

`num_sources`

Type: `uint32_t`

The number of audio sources in the rig.

`sources`

Type: `nvstitchAudioSourceProperties_t*`

An array of length `num_sources` of `nvstitchAudioSourceProperties_t` properties, one for each audio source.

3.4.3.2 Remarks

This structure defines the properties of an audio rig, which is a set of audio sources.

3.4.4 nvstitchAudioSourceProperties_t

```
typedef struct
{
    uint32_t               version;
    nvstitchAudioInputType input_type;
    nvstitchPose_t         pose;
}
nvstitchAudioSourceProperties_t;
```

3.4.4.1 Members

`version`

Type: `uint32_t`

Set to `NVSTITCH_VERSION`.

`input_type`

Type: `nvstitchAudioInputType`

The type of the input.

`pose`

Type: `nvstitchPose_t`

The pose, location, and translation of the audio source or capture device.

3.4.4.2 Remarks

This structure contains information about an audio source used in creating the sound field or capture device recording it.

3.5 COMMON STRUCTURES

These structures are defined in the header file `nvstitch_common.h`.

3.5.1 nvstitchPose_t

```
typedef struct
{
    float rotation[3*3];
    float translation[3];
}
nvstitchPose_t;
```

3.5.1.1 Members

`rotation[3*3]`

Type: `float`

A 3×3 row major rotation matrix. The rotation implements the following transformation:

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} r_0 & r_1 & r_2 \\ r_3 & r_4 & r_5 \\ r_6 & r_7 & r_8 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix}$$

`translation[3]`

Type: `float`

The position in centimeters with respect to the center of the rig.

3.5.1.2 Remarks

This structure is used to describe a position in 3D space in terms of its rotation and translation.

The right-handed (Y-up) coordinate system is used, where the **x** axis points to the right, the **y** axis points up, and the **z** axis comes inward.

3.6 COMMON VIDEO STRUCTURES

These enumerations are defined in the header file `nvstitch_common_video.h`.

3.6.1 nvstitchCameraMappingMatrix_t

```
typedef struct
{
    struct { uint32_t x, y; } map_offset;
    struct { uint32_t x, y; } map_size;
    struct { uint32_t x, y; } input_size;
    nvstitchCameraLayout      layout;
    float*                    matrix;
}
nvstitchCameraMappingMatrix_t;
```

3.6.1.1 Members

`map_offset`

Type: `struct { uint32_t x, y; }`

The location of the top-left corner of the projection map in the output 360 panorama.

`map_size`

Type: `struct { uint32_t x, y; }`

The size of the projection map in pixels.

`input_size`

Type: `struct { uint32_t x, y; }`

The size of the camera input in pixels.

`layout`

Type: `nvstitchCameraLayout`

Camera type, based on its orientation.

`matrix`

Type: `float*`

The projection map, containing an array of pairs of floating point coordinates for each pixel in the equirectangular region bounded by `map_offset` and `map_size`. Each pair indicates the location in the source image that is to be sampled for the corresponding location in the equirectangular panorama. Negative numbers are used to indicate out-of-bounds.

3.6.1.2 Remarks

This structure describes a camera projection map.

3.6.2 nvstitchCameraMapping_t

```
typedef struct
{
    uint32_t                        version;
    uint32_t                        num_cameras;
    struct { uint32_t x, y; }       reference_resolution;
    float                           rig_diameter;
    nvstitchCameraMappingMatrix_t*  matrices;
}
nvstitchCameraMapping_t;
```

3.6.2.1 Members

`version`

Type: `uint32_t`

The version of this structure.

`num_cameras`

Type: `uint32_t`

The number of cameras in the rig.

`reference_resolution`

Type: `struct { uint32_t x, y; }`

The resolution of the equirectangular panorama for which the projection maps are defined.

`rig_diameter`

Type: `float`

Estimated rig diameter in centimeters.

`matrices`

Type: `nvstitchCameraMappingMatrix_t*`

An array of projection maps, which must contain `num_cameras` elements.

3.6.2.2 Remarks

This structure contains all the projection maps for a given rig.

3.6.3 nvstitchCameraProperties_t

```
typedef struct
{
    uint32_t                  version;
    nvstitchCameraLayout      camera_layout;
    struct { uint32_t x, y; } image_size;
    struct { float x, y; }    principal_point;
    float                     focal_length;
    nvstitchDistortionType    distortion_type;
    float                     distortion_coefficients[5];
    float                     fisheye_radius;
    nvstitchPose_t            extrinsics;
    uint32_t                  set_flags;
}
nvstitchCameraProperties_t;
```

3.6.3.1 Members

`version`

Type: `uint32_t`

The version of this structure.

`camera_layout`

Type: `nvstitchCameraLayout`

The camera type based on its orientation.

`image_size`

Type: `struct { uint32_t x, y; }`

The number of pixels horizontally and vertically in the camera image.

`principal_point`

Type: `struct { float x, y; }`

The location of the principal point, which is also known as the center of projection, with respect to the upper-left corner. Note that the Y axis goes downward in the image.

`focal_length`

Type: `float`

The focal length of the lens in pixels per radian. It specifies the angular pixel density at the center of projection and is a measure of the spatial resolution of the lens.

The focal length of an ideal equidistant fisheye lens can be computed from the ratio of the pixel-to-angular distance between two points on the image. For example, if ϕ is the angle (field-of-view) in radians between the leftmost and rightmost pixels in an image of width *W*, the focal length is computed as follows:

$$f = \frac{W - 1}{\varphi}$$

`distortion_type`

Type: `nvstitchDistortionType`

The distortion model of the lens, which is one of the models defined in “nvstitchDistortionType” on page 11.

`distortion_coefficients`

Type: `float`

Distortion coefficients of the lens, which depend on the distortion model of the lens as explained in “nvstitchDistortionType” on page 11.

`fisheye_radius`

Type: `float`

The radius in pixels of the circle used for clipping fisheye images. A radius of 0 implies that no circular clipping is to be performed.

`extrinsics`

Type: `nvstitchPose_t`

The position of the camera, in 3D space in terms of its rotation and translation. For details, see “nvstitchPose_t” on page 17.

`set_flags`

Type: `uint32_t`

Indicates which fields in the structure are set by the user. The value is a combination of `nvstitchCameraSetPropertyFlag` enumerators. See “nvstitchCameraSetPropertyFlag” on page 11 for details of these enumerators.

3.6.3.2 Remarks

This structure contains camera information, including the intrinsics and extrinsics of the camera.

3.6.4 nvstitchImageBuffer_t

```
typedef struct nvstitchImageBuffer_st
{
    void*    dev_ptr;
    silent   pitch;
    size     row_bytes;
    size_t   width;
    size_t   height;
}
nvstitchImageBuffer_t;
```

3.6.4.1 Members

`dev_ptr`

Type: `void*`

The pointer to the device buffer.

`pitch`

Type: `size_t`

The pitch in bytes. The pitch is the stride between pixels vertically. The pitch may include padding between successive rows.

`row_bytes`

Type: `size_t`

The width in bytes, less than or equal to the magnitude of the pitch. This is equal to the width in pixels times the size of each pixel.

`width`

Type: `size_t`

The number of pixels in the buffer horizontally.

`height`

Type: `size_t`

The number of pixels in the buffer vertically.

3.6.4.2 Remarks

This structure defines a pitch-linear image buffer in device memory.

3.6.5 nvstitchVideoRigProperties_t

```
typedef struct
{
    uint32_t                    version;
    float                       rig_diameter;
    float                       rotation[3*3];
    uint32_t                    num_cameras;
    nvstitchCameraProperties_t* cameras;
}
nvstitchVideoRigProperties_t;
```

3.6.5.1 Members

`version`

Type: `uint32_t`

The version of this structure.

`rig_diameter`

Type: `float`

The diameter of the camera ring in cm.

`rotation`

Type: `float`

Not yet supported.

`num_cameras`

Type: `uint32_t`

The number of cameras in the rig.

`cameras`

Type: `nvstitchCameraProperties_t*`

An array of camera properties, which must contain `num_cameras` elements.

3.6.5.2 Remarks

This structure is used in `nvstitchCreateVideoRigInstance` and `nvstitchGetVideoRigProperties` to define camera rig information, including the position of the rig and properties of each camera in the rig.

3.7 GENERALLY APPLICABLE STRUCTURES

These structures are defined in the header file `nvstitch.h`.

3.7.1 nvstitchAudioRigHandle

```
typedef struct _nvstitchAudioRigInstance_t* nvstitchAudioRigHandle;
```

3.7.1.1 Remarks

This type defines the handle of an audio rig instance. It is used to reference the audio rig after the rig is created.

3.7.2 nvstitchCalibrationInstanceHandle

```
typedef struct _nvstitchCalibrationInstance_t*
nvstitchCalibrationInstanceHandle;
```

3.7.2.1 Remarks

This type defines the handle of a calibration instance. It is used to reference the calibration instance after the instance is created.

3.7.3 nvstitchStitcherHandle

```
typedef struct _nvstitchStitcherHandle_t* nvstitchStitcherHandle;
```

3.7.3.1 Remarks

This type defines the handle of a stitcher instance. It is used to reference the stitcher instance after the instance is created.

3.7.4 nvstitchVideoRigHandle

```
typedef struct _nvstitchVideoRigInstance_t* nvstitchVideoRigHandle;
```

3.7.4.1 Remarks

This type defines the handle of a video rig instance. It is used to reference the video rig after the rig is created.

3.7.5 nvstitchCalibrationProperties_t

```
typedef struct
{
    uint32_t               version;
    uint32_t               frame_count;
    nvstitchVideoRigHandle rig_estimate;
    nvstitchMediaFormat    input_format;
    nvstitchMediaForm      input_form;
}
nvstitchCalibrationProperties_t;
```

3.7.5.1 Members

`version`

Type: `uint32_t`

The version of the structure.

`frame_count`

Type: `uint32_t`

The number of frames that will be used for calibration.

`rig_estimate`

Type: `nvstitchVideoRigHandle`

The handle to the estimated parameters of the video rig.

`input_format`

Type: `nvstitchMediaFormat`

The format (representation) of the input images that will be used for calibration.

`input_form`

Type: `nvstitchMediaForm`

The form (container) of the input images that will be used for calibration.

3.7.5.2 Remarks

This structure is used in used in `nvstitchCreateCalibrationInstance` to specify calibration parameters.

3.7.6 nvstitchEncodeSettings_t

```
typedef struct
{
    nvstitchEncoderSettingType enc_setting_type;
    union
    {
        float bitrate;
    } settings;
}
nvstitchEncodeSettings_t;
```

3.7.6.1 Members

`enc_setting_type`

Type: `nvstitchEncoderSettingType`

The type of the setting specified in the structure.

`bitrate`

Type: `float`

The target bit rate of the output video.

3.7.6.2 Remarks

This structure is used in `nvstitchSetStitcherEncoderSetting` to define output encoder settings.

3.7.7 nvstitchPayload_t

```
typedef struct
{
    nvstitchMediaForm payload_type;
    struct { uint32_t x, y; } image_size;
    union
    {
```

```
        struct
        {
            void*    ptr;
            uint32_t pitch;
            int64_t  timestamp;
        } buffer;
        struct
        {
            void*    ptr;
            uint32_t size;
            int64_t  timestamp;
        } frame;
        struct
        {
            const char* name;
            uint32_t    fps_num, fps_den;
            float       dt;
        } file;
    } payload;
}
nvstitchPayload_t;
```

3.7.7.1 Members

`payload_type`

Type: `nvstitchMediaForm`

The media form described in the `payload` union.

`image_size`

Type: `struct { uint32_t x, y; }`

The image size in pixels.

`buffer.ptr`

Type: `void*`

A pointer to the buffer containing the payload, which may be in system memory or GPU memory.

`buffer.pitch`

Type: `uint32_t`

The buffer pitch in bytes. The pitch is the stride between pixels vertically.

`buffer.timestamp`

Type: `int64_t`

Timestamp of the payload, which is used to synchronize different input streams.

`frame.ptr`

Type: `void*`

A pointer to the buffer containing the payload, which may be in system memory or GPU memory.

`frame.size`

Type: `uint32_t`

The size of the buffer in bytes.

`frame.timestamp`

Type: `int64_t`

Timestamp of the payload, which is used to synchronize different input streams.

`file.name`

Type: `char*`

File name

`file.fps_num, file.fps_den`

Type: `int64_t`

The video frame rate, specified as a rational number, that is, the ratio of an integral numerator to a denominator, for example, 30000/1001.

`file.dt`

Type: `float`

Number of seconds to trim from the start of the video.

3.7.7.2 Remarks

This structure defines the properties of different input sources or output destinations in a media payload.

3.7.8 nvstitchStitcherProperties_t

```
typedef struct
{
    uint32_t                        version;
    nvstitchVideoRigHandle          video_rig;
    nvstitchAudioRigHandle          audio_rig;
```

```
    nvstitchVideoPipelineOptions_t* video_pipeline_options;

    nvstitchMediaFormat             input_format;
    nvstitchMediaForm               input_form;

    uint32_t                        num_audio_inputs;
    nvstitchAudioDataSource         *audio_input_form;
    nvstitchAudioConfiguration_t    **audio_input_configurations;

    nvstitchPanoramaProjectionType  output_projection;
    nvstitchMediaFormat             output_format;
    uint32_t                        num_output_payloads;
    nvstitchPayload_t*              output_payloads;

    nvstitchAudioOutputType         audio_output_blend;
    nvstitchAudioDataSource         audio_output_form;
    nvstitchAudioConfiguration_t    *audio_output_format;

    float                            audio_output_gain;
    float audio_sound_field_parameters[NUM_SOUND_FIELD_PARAMETERS];
}
nvstitchStitcherProperties_t;
```

3.7.8.1 Members

`version`

Type: `uint32_t`

The version of the structure.

`video_rig`

Type: `nvstitchVideoRigHandle`

The camera rig to be used for video input.

`audio_rig`

Type: `nvstitchAudioRigHandle`

The set of microphones to be used for audio input.

`video_pipeline_options`

Type: `nvstitchVideoPipelineOptions_t*`

Video stitching pipeline options.

`input_format`

Type: `nvstitchMediaFormat`

The format (representation) of the input images.

`input_form`

Type: `nvstitchMediaForm`

The form (container) in which the input images are fed to the stitcher, specifying whether the images come from a buffer or a file.

`num_audio_inputs`

Type: `uint32_t`

The number of audio inputs, which must be the same as the number of inputs in the rig.

`*audio_input_form`

Type: `nvstitchAudioDataSource`

An array of source *types (forms)*, specifying whether each audio source is a buffer or a file.

`**audio_input_configurations`

Type: `nvstitchAudioConfiguration_t`

An array of source *formats*, one per input. Set the `nvstitchAudioConfiguration_t*` for an input to `NULL` to indicate that the audio data will come from an MP4 file. Audio data for that input will be provided by the demultiplexer within the API.

`output_projection`

Type: `nvstitchPanoramaProjectionType`

The projection type of the output 360 panorama.

`output_format`

Type: `nvstitchMediaFormat`

The format of the output panorama.

`num_output_payloads`

Type: `uint32_t`

The number of output panorama descriptors. This number is always one for mono stitching, but can be either one or two for stereo stitching.

`output_payloads`

Type: `nvstitchPayload_t*`

The output panorama descriptor.

- If a single output is used for stereo, the output is in top-bottom format.
- If separate output buffers are used for left and right panoramas, the left eye panorama is saved in the first buffer.

`audio_output_blend`

Type: `nvstitchAudioOutputType`

The audio blend type.

`audio_output_form`

Type: `nvstitchAudioDataSource`

The form of the audio output, specifying whether the audio should be output to a buffer or a file.

`*audio_output_format`

Type: `nvstitchAudioConfiguration_t`

The format (representation) of the audio output.

`audio_output_gain`

Type: `float`

A linear gain applied to all output channels. Valid range is 0.0 and higher.

`audio_sound_field_parameters`

Type: `float`

Sound field encoding parameters whose meaning depends on `audio_output_blend`.

For `NVSTITCH_AUDIO_OUTPUT_STEREO_MIXDOWN`, the zeroth index of the array indicates the stereo spread coefficient. The valid range is 0.0, which turns the spread effect off, to 1.0, which specifies the maximum spread effect.

3.7.8.2 Remarks

This structure defines the properties used to create a stitcher instance.

3.7.9 nvstitchVideoPipelineOptions_t

```
typedef struct
{
    nvstitchStitcherPipelineType pipeline_type;
    nvstitchStitcherQuality      stitch_quality;
    union
    {
        struct
        {
            float feather_width;
        } mono;
        struct
        {
            float ipd;
            int min_dist;
        } stereo;
    } options;
}
nvstitchVideoPipelineOptions_t;
```

3.7.9.1 Members

`pipeline_type`

Type: `nvstitchStitcherPipelineType`

The type of stitching (mono or stereo).

`stitch_quality`

Type: `nvstitchStitcherQuality`

The stitching quality preset.

`mono.feather_width`

Type: `float`

The width of the blend region used in multi-resolution blending.

`stereo.ipd`

Type: `float`

The target interpupillary distance in the output pair of panoramas.

`stereo.min_dist`

Type: `int`

The minimum required distance of objects from the camera rig in cm.

3.7.9.2 Remarks

This structure is used in `nvstitchCreateStitcher` to specify video stitching options.

3.7.10 nvstitchVideoRigHints_t

```
typedef struct
{
    nvstitchVideoRigType rig_type;
    union
    {
        struct
        {
            uint32_t                  num_cameras;
            float                     rig_diameter;
            struct { uint32_t x, y; } image_size;
            struct { float x, y; }    fov;
            float                     roll;
        } ring;
    } hints;
}
nvstitchVideoRigHints_t;
```

3.7.10.1 Members

`rig_type`

Type: `nvstitchVideoRigType`

The type of the rig hints in the `hints` union. Set to 0.

`ring.num_cameras`

Type: `uint32_t`

The number of cameras in the rig.

`ring.rig_diameter`

Type: `float`

The estimated diameter of the rig, in centimeters.

`ring.image_size`

Type: `struct { uint32_t x, y; }`

The number of pixels horizontally and vertically in all of camera images. This parameter assumes that all cameras are the same model.

`ring.fov`

Type: `struct { float x, y; }`

The estimated fields of view horizontally and vertically, through the principal point, in radians. If both angles are zero, the value is ignored. The focal length estimate is computed from these angles.

`ring.roll`

Type: `float`

The angle in radians through which the camera images must be rotated to be oriented correctly. For example:

- If the images are upright, the angle is 0.
- If the images must be rotated 90° clockwise, the angle is +π/2.
- If the images must be rotated 90° counterclockwise, the angle is -π/2.

3.7.10.2 Remarks

This structure is used in `nvstitchCreateVideoRigFromHints` to generate video rigs from a set of hints. Currently, only rigs with cameras distributed along the equator are supported.

3.8 FUNCTIONS

3.8.1 nvstitchCalibrate

```
nvstitchResult NVSTITCHAPI nvstitchCalibrate(
    nvstitchCalibrationInstanceHandle calibration_instance,
    nvstitchVideoRigHandle* calibrated_rig
);
```

3.8.1.1 Parameters

`calibration_instance [in]`

Type: `nvstitchCalibrationInstanceHandle`

The handle of the calibration instance.

`calibrated_rig [out]`

Type: `nvstitchVideoRigHandle*`

A pointer to the output video rig handle, which is the result of the calibration process.

3.8.1.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_CALIB_MISSING_INPUT_IMAGE`
- `NVSTITCH_CALIB_BAD_INPUT_IMAGE`

3.8.1.3 Remarks

This function runs calibration and produces a calibrated rig. All input images must be fed in before this function is called.

3.8.2 nvstitchCreateAudioRigInstance

```
nvstitchResult NVSTITCHAPI nvstitchCreateAudioRigInstance(
    const nvstitchAudioRigProperties_t* properties,
    nvstitchAudioRigHandle*             audio_rig_instance,
    nvstitchResult                      *perSourceStatus
);
```

3.8.2.1 Parameters

`properties [in]`

Type: `nvstitchAudioRigProperties_t*`

A pointer to the set of properties describing the rig, including the properties of each audio source.

`audio_rig_instance [out]`

Type: `nvstitchAudioRigHandle*`

A pointer to the handle of the newly created rig

`perSourceStatus`

Type: `nvstitchResult*`

Not used. Pass in a null pointer.

3.8.2.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success

- NVSTITCH_ERROR_NULL_POINTER
- NVSTITCH_ERROR_OUT_OF_SYSTEM_MEMORY
- NVSTITCH_ERROR_INVALID_VERSION

3.8.2.3 Remarks

This function creates an audio rig instance from the input properties and returns a handle to the newly created rig. The handle is used in all function calls that take an audio rig as an input parameter

3.8.3 nvstitchCreateCalibrationInstance

```
nvstitchResult NVSTITCHAPI nvstitchCreateCalibrationInstance(
    const nvstitchCalibrationProperties_t properties,
    nvstitchCalibrationInstanceHandle*    calibration_instance
);
```

3.8.3.1 Parameters

`properties [in]`

Type: `nvstitchCalibrationProperties_t`

A pointer to the set of calibration properties.

`calibration_instance [out]`

Type: `nvstitchCalibrationInstanceHandle*`

A pointer to the handle of the newly created calibration instance.

3.8.3.2 Return Value

Returns one of the following values:

- NVSTITCH_SUCCESS on success
- NVSTITCH_ERROR_BAD_PARAMETER
- NVSTITCH_ERROR_NULL_POINTER
- NVSTITCH_ERROR_INVALID_VERSION
- NVSTITCH_CALIB_BAD_CAMERA_PROPERTY_VALUE

3.8.3.3 Remarks

This function creates a calibration instance from the input properties and returns a handle to the newly created instance. The handle is used as a parameter in later function calls.

3.8.4 nvstitchCreateStitcher

```
nvstitchResult NVSTITCHAPI nvstitchCreateStitcher(
    const nvstitchStitcherProperties_t* properties,
    nvstitchStitcherHandle*             stitcher
);
```

3.8.4.1 Parameters

`properties [in]`

Type: `nvstitchStitcherProperties_t*`

A pointer to the set of stitcher properties, including the audio and video rigs.

`stitcher [out]`

Type: `nvstitchStitcherHandle*`

A pointer to the handle of the newly created stitcher instance.

3.8.4.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_INVALID_VERSION`
- `NVSTITCH_ERROR_NOT_IMPLEMENTED`
- `NVSTITCH_ERROR_OUT_OF_SYSTEM_MEMORY`

3.8.4.3 Remarks

This function creates a stitcher instance from the input properties and returns a handle to the newly created instance. The handle is used as a parameter in later function calls.

3.8.5 nvstitchCreateStitcherUsingMaps

```
nvstitchResult NVSTITCHAPI nvstitchCreateStitcherUsingMaps(
    const nvstitchStitcherProperties_t* properties,
    const nvstitchCameraMapping_t*      cam_mapping,
    nvstitchStitcherHandle*             stitcher
);
```

3.8.5.1 Parameters

`properties [in]`

Type: `nvstitchStitcherProperties_t*`

A pointer to the set of stitcher properties, including the audio and video rigs. Camera intrinsics and extrinsics are not required when projection maps are used.

`cam_mapping [in]`

Type: `nvstitchCameraMapping_t*`

A pointer to the structure containing the projection maps.

`stitcher [out]`

Type: `nvstitchStitcherHandle*`

A pointer to the handle of the newly created stitcher instance.

3.8.5.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_INVALID_VERSION`
- `NVSTITCH_ERROR_OUT_OF_SYSTEM_MEMORY`

3.8.5.3 Remarks

This function creates a stitcher instance from the specified input properties and camera projection maps. A handle to the newly created instance is returned through the `stitcher` parameter. The handle is used as a parameter in later function calls.

3.8.6 nvstitchCreateVideoRigFromHints

```
nvstitchResult NVSTITCHAPI nvstitchCreateVideoRigFromHints(
    const nvstitchVideoRigHints_t* hints,
    nvstitchVideoRigHandle*        rig
);
```

3.8.6.1 Parameters

`hints [in]`

Type: `nvstitchVideoRigHints_t*`

A pointer to the input video rig hints that contain the estimates to be used for creating the rig.

`Rig [out]`

Type: `nvstitchVideoRigHandle*`

A pointer to the handle of the newly created rig.

3.8.6.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_CALIB_BAD_CAMERA_PROPERTY_VALUE`

3.8.6.3 Remarks

This function creates a video rig instance from the estimates in the specified video rig hints. The rig generated this way is intended only to provide a good starting point for calibration and should not be used for stitching.

The distortion coefficients are set to 0, and the principal point coincides with the center of the images.

The fisheye circle is set for the maximum round fisheye, which is typically conservative, and can be fed into the calibrator. However, better results can be achieved by measuring the circle in an image editing program to eliminate undesirable pixels, and editing the default results.

3.8.7 nvstitchCreateVideoRigInstance

```
nvstitchResult NVSTITCHAPI nvstitchCreateVideoRigInstance(
    const nvstitchVideoRigProperties_t* properties,
    nvstitchVideoRigHandle*             video_rig_instance
);
```

3.8.7.1 Parameters

`properties [in]`

Type: `nvstitchVideoRigProperties_t*`

A pointer to the set of properties describing the rig, including the intrinsics and extrinsics of each camera

`video_rig_instance [out]`

Type: `nvstitchVideoRigHandle*`

A pointer to the handle of the newly created rig.

3.8.7.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_INVALID_VERSION`

- `NVSTITCH_ERROR_OUT_OF_SYSTEM_MEMORY`
- `NVSTITCH_ERROR_NULL_POINTER`

3.8.7.3 Remarks

This function creates a video rig instance from the input properties and returns a handle to the newly created rig. The handle is used in all function calls that take a video rig as an input parameter.

3.8.8 nvstitchDestroyAudioRigInstance

```
nvstitchResult NVSTITCHAPI nvstitchDestroyAudioRigInstance(
    nvstitchAudioRigHandle audio_rig_instance
);
```

3.8.8.1 Parameters

`audio_rig_instance [in]`

Type: `nvstitchAudioRigHandle`

The handle of the audio rig to be destroyed.

3.8.8.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`

3.8.8.3 Remarks

This function destroys an audio rig instance. The input parameter `nvstitchAudioRigHandle` is not modified, but it becomes invalid after this function is called.

3.8.9 nvstitchDestroyCalibrationInstance

```
nvstitchResult NVSTITCHAPI nvstitchDestroyCalibrationInstance(
    nvstitchCalibrationInstanceHandle calibration_instance
);
```

3.8.9.1 Parameters

`calibration_instance [in]`

Type: `nvstitchCalibrationInstanceHandle`

The handle of the calibration instance to be destroyed.

3.8.9.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`

3.8.9.3 Remarks

This function destroys a calibration instance. The input parameter `nvstitchCalibrationInstanceHandle` is not modified, but it becomes invalid after this function is called.

3.8.10 nvstitchDestroyStitcher

```
nvstitchResult NVSTITCHAPI nvstitchDestroyStitcher(
    nvstitchStitcherHandle stitcher
);
```

3.8.10.1 Parameters

`stitcher [in]`

Type: `nvstitchStitcherHandle`

The handle of the stitcher instance to be destroyed.

3.8.10.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`

3.8.10.3 Remarks

This function destroys a stitcher instance. The input parameter `nvstitchStitcherHandle` is not modified, but it becomes invalid after this function is called.

3.8.11 nvstitchDestroyVideoRigInstance

```
nvstitchResult NVSTITCHAPI nvstitchDestroyVideoRigInstance(
    nvstitchVideoRigHandle video_rig_instance
);
```

3.8.11.1 Parameters

`video_rig_instance [in]`

Type: `nvstitchVideoRigHandle`

The handle of the video rig to be destroyed.

3.8.11.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`

3.8.11.3 Remarks

This function destroys a video rig instance. The input parameter `nvstitchVideoRigHandle` is not modified, but it becomes invalid after this function is called.

3.8.12 nvstitchFeedCalibrationInput

```
nvstitchResult NVSTITCHAPI nvstitchFeedCalibrationInput(
    uint32_t                          packet_index,
    nvstitchCalibrationInstanceHandle calibration_instance,
    uint32_t                          source_index,
    const nvstitchPayload_t*          payload
);
```

3.8.12.1 Parameters

`packet_index [in]`

Type: `uint32_t`

The frame number of the input being fed.

`calibration_instance [in]`

Type: `nvstitchCalibrationInstanceHandle`

The handle of the calibration instance.

`source_index [in]`

Type: `uint32_t`

The index of the camera in the video rig used.

`payload [in]`

Type: `nvstitchPayload_t*`

A pointer to the payload structure describing the input.

3.8.12.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_INDEX`
- `NVSTITCH_ERROR_BAD_PARAMETER`

3.8.12.3 Remarks

This function feeds input from one camera's calibration instance. When payload is a buffer, each call feeds a single frame. In contrast, video files are fed in a single call.

3.8.13 nvstitchFeedStitcherAudio

```
nvstitchResult NVSTITCHAPI nvstitchFeedStitcherAudio(
    uint32_t                      packet_index,
    nvstitchStitcherHandle        stitcher,
    uint32_t                      source_index,
    const nvstitchAudioPayload_t* payload
);
```

3.8.13.1 Parameters

`packet_index [in]`

Type: `uint32_t`

Currently unused. Set to zero.

`stitcher [in]`

Type: `nvstitchStitcherHandle`

The handle of the stitcher instance.

`source_index [in]`

Type: `uint32_t`

The index of the input audio source.

`payload [in]`

Type: `nvstitchAudioPayload_t*`

A pointer to the input payload.

3.8.13.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_NOT_ALLOWED`
- `NVSTITCH_ERROR_BAD_STATE`

3.8.13.3 Remarks

This function feeds input audio data from a single sound source into the stitcher. This data can be a short buffer or a full input file depending on the specified payload. For details about the format of the payload, see “nvstitchAudioPayload_t” on page 15.

3.8.14 nvstitchFeedStitcherAudioVideo

```
nvstitchResult NVSTITCHAPI nvstitchFeedStitcherAudioVideo(
    uint32_t                    packet_index,
    const nvstitchStitcherHandle stitcher,
    uint32_t                    source_index,
    uint32_t                    audio_source_index,
    const nvstitchPayload_t*    payload,
    bool                        allow_dropping_frames
);
```

3.8.14.1 Parameters

`packet_index [in]`

Type: `uint32_t`

Currently unused. Packet index of the input payload.

`stitcher [in]`

Type: `nvstitchStitcherHandle`

The handle of the stitcher instance.

`source_index [in]`

Type: `uint32_t`

The index of the input's source camera.

`audio_source_index [in]`

Type: `uint32_t`

The index of the input's audio source.

`payload [in]`

Type: `nvstitchPayload_t*`

A pointer to the input payload.

`allow_dropping_frames [in]`

Type: `bool`

A Boolean value that sets behavior of the pipeline on overflow:

- `True`: The pipeline drops input frames when the input buffer is full.
- `False`: The pipeline blocks calling thread.

3.8.14.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_NOT_ALLOWED`
- `NVSTITCH_ERROR_BAD_STATE`

3.8.14.3 Remarks

This function feeds combined (multiplexed) input from a single camera and single sound source. The only type of input that this function supports is an MP4 file containing both video and audio.

3.8.15 nvstitchFeedStitcherVideo

```
nvstitchResult NVSTITCHAPI nvstitchFeedStitcherVideo(
    uint32_t                packet_index,
    nvstitchStitcherHandle  stitcher,
    uint32_t                source_index,
    const nvstitchPayload_t* payload,
    bool                    allow_dropping_frames
);
```

3.8.15.1 Parameters

`packet_index [in]`

Type: `uint32_t`

The packet index of the input payload. This value will be propagated through the pipeline and returned with the matching output payload.

`stitcher [in]`

Type: `nvstitchStitcherHandle`

The handle of the stitcher instance.

`source_index [in]`

Type: `uint32_t`

The index of the input's source camera.

`payload [in]`

Type: `nvstitchPayload_t*`

A pointer to the input payload.

`allow_dropping_frames [in]`

Type: `bool`

A Boolean value that sets behavior of the pipeline on overflow:

- `True`: The pipeline drops input frames when the input buffer is full.
- `False`: The pipeline blocks calling thread.

3.8.15.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_NOT_ALLOWED`
- `NVSTITCH_ERROR_BAD_STATE`

3.8.15.3 Remarks

This function feeds input from a single camera into the stitcher. The input is a single frame when the input is from a buffer or a full video when the input is from an MP4 file.

3.8.16 nvstitchGetAudioRigMicProperties

```
nvstitchResult NVSTITCHAPI nvstitchGetAudioRigMicProperties(
    const nvstitchAudioRigHandle     audio_rig_instance,
    const uint32_t                   index,
    nvstitchAudioSourceProperties_t* audio_source
);
```

3.8.16.1 Parameters

`audio_rig_instance [in]`

Type: `nvstitchAudioRigHandle`

The handle of the audio rig that contains the audio source.

`index [in]`

Type: `uint32_t`

The audio source index in the audio rig.

`audio_source [out]`

Type: `nvstitchAudioSourceProperties_t*`

A pointer to the destination structure that is to contain the audio source properties.

3.8.16.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_PARAMETER`

3.8.16.3 Remarks

This function copies the properties of the audio source with the specified index into the location referenced by the output parameter `audio_source`. Audio source indexes are based on the array of sources in the `nvstitchAudioRigProperties_t` structure.

3.8.17 nvstitchGetErrorString

```
const char* NVSTITCHAPI nvstitchGetErrorString(
    nvstitchResult error
);
```

3.8.17.1 Parameters

`error [in]`

Type: `nvstitchResult`

An `nvstitchResult` value for which the string description will be returned.

3.8.17.2 Return Value

Returns an array of null-terminated characters describing the specified error code.

3.8.17.3 Remarks

This function returns a string describing the specified `NVSTITCH_` error code.

3.8.18 nvstitchGetStitcherOutputBuffer

```
nvstitchResult NVSTITCHAPI nvstitchGetStitcherOutputBuffer(
    const nvstitchStitcherHandle stitcher,
    nvstitchEye                  eye,
    nvstitchImageBuffer_t*       buffer
);
```

3.8.18.1 Parameters

`stitcher [in]`

Type: `nvstitchStitcherHandle`

The handle of the stitcher instance.

`eye [in]`

Type: `nvstitchEye`

The stereoscopic eye index.

`buffer [out]`

Type: `nvstitchImageBuffer_t*`

A pointer to the output buffer structure.

3.8.18.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_BAD_STATE`

3.8.18.3 Remarks

This function copies information about stitcher output buffers into the location pointed to by the parameter `buffer`. For a stereoscopic stitching pipeline, left and right eye buffers are copied separately.

3.8.19 nvstitchGetStitcherOutputDevice

```
nvstitchResult NVSTITCHAPI nvstitchGetStitcherOutputDevice(
    const nvstitchStitcherHandle stitcher,
    nvstitchEye                  eye,
    int*                         device
);
```

3.8.19.1 Parameters

`stitcher [in]`

Type: `nvstitchStitcherHandle`

The handle of the stitcher instance.

`eye [in]`

Type: `nvstitchEye`

The index of the input's camera.

`device [out]`

Type: `int*`

A pointer to the output device.

3.8.19.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_BAD_STATE`

3.8.19.3 Remarks

This function returns the device on which the output buffer is stored.

Return value is a pointer to the device index, as associated in CUDA device enumeration.

3.8.20 nvstitchGetStitcherOutputStream

```
nvstitchResult NVSTITCHAPI nvstitchGetStitcherOutputStream(
    const nvstitchStitcherHandle stitcher,
    nvstitchEye                  eye,
    struct CUstream_st**         stream
);
```

3.8.20.1 Parameters

`stitcher [in]`

Type: `nvstitchStitcherHandle`

The handle of the stitcher instance.

`eye [in]`

Type: `nvstitchEye`

The index of the input's camera.

`stream [out]`

Type: `CUstream_st**`

A pointer to the output CUDA stream.

3.8.20.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_BAD_STATE`

3.8.20.3 Remarks

This function returns the CUDA stream which should be used when reading the output buffer to ensure correct synchronization. Alternatively, the default stream can be used when reading the output buffer, but that can lead to performance loss.

3.8.21 nvstitchGetVideoRigCameraProperties

```
nvstitchResult NVSTITCHAPI nvstitchGetVideoRigCameraProperties(
    const nvstitchVideoRigHandle video_rig_instance,
    const uint32_t               index,
    nvstitchCameraProperties_t*  camera
);
```

3.8.21.1 Parameters

`video_rig_instance [in]`

Type: `nvstitchVideoRigHandle`

The handle of the video rig that contains the camera.

`index [in]`

Type: `uint32_t`

The camera index in the video rig.

`camera [out]`

Type: `nvstitchCameraProperties_t*`

A pointer to the destination structure that is to contain a copy of the camera properties.

3.8.21.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_INDEX`

3.8.21.3 Remarks

This function copies the properties of the camera with the specified index into the location referenced by the output parameter `camera`. Camera indexes are based on the array of cameras in the `nvstitchVideoRigProperties_t` structure.

3.8.22 nvstitchGetVideoRigProperties

```
nvstitchResult NVSTITCHAPI nvstitchGetVideoRigProperties(
    const nvstitchVideoRigHandle  video_rig_instance,
    nvstitchVideoRigProperties_t* video_rig_properties
);
```

3.8.22.1 Parameters

`video_rig_instance [in]`

Type: `nvstitchVideoRigHandle`

The handle of the video rig.

`video_rig_properties [out]`

Type: `nvstitchVideoRigProperties_t*`

A pointer to the destination structure that is to contain a copy of the video rig properties.

3.8.22.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`

3.8.22.3 Remarks

This function copies the properties of the specified video rig, including the properties of all cameras in the rig, into the location referenced by the output parameter `video_rig_properties`.

3.8.23 nvstitchSetStitcherEncoderSetting

```
nvstitchResult NVSTITCHAPI nvstitchSetStitcherEncoderSetting(
    nvstitchStitcherHandle   stitcher,
    nvstitchEncodeSettings_t encode_settings
);
```

3.8.23.1 Parameters

`stitcher [in]`

Type: `nvstitchStitcherHandle`

The handle of the stitcher instance.

`encode_settings [in]`

Type: `nvstitchEncodeSettings_t`

A structure specifying the encoder setting for the stitcher.

3.8.23.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_NOT_ALLOWED`

3.8.23.3 Remarks

This function sets one of the supported encoder settings for a stitcher. This function can only be called before the stitcher is put into running state.

3.8.24 nvstitchStartStitcher

```
nvstitchResult NVSTITCHAPI nvstitchStartStitcher(
    nvstitchStitcherHandle   stitcher,
    nvstitchStitcherOnOutput output_callback,
    void*                    app_data
);
```

3.8.24.1 Parameters

`stitcher [in]`

Type: `nvstitchStitcherHandle`

The handle of the stitcher instance.

`output_callback [in]`

Type: `nvstitchStitcherOnOutput`

A pointer to the stitcher callback function.

`app_data [in]`

Type: `void*`

A pointer to user-defined data, which will be accessible in the callback.

3.8.24.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_BAD_STATE`
- `NVSTITCH_ERROR_BAD_PARAMETER`
- `NVSTITCH_ERROR_GENERAL`

3.8.24.3 Remarks

This function sets the callback function and puts the stitcher into running state. The stitcher can accept calls from functions that feed input to the stitcher only after this function is called.

The following functions feed input to a stitcher:

- `nvstitchFeedStitcherAudio`
- `nvstitchFeedStitcherAudioVideo`
- `nvstitchFeedStitcherVideo`

3.8.25 nvstitchStopStitcher

```
NVSTITCHAPI nvstitchStopStitcher(
    nvstitchStitcherHandle stitcher
);
```

3.8.25.1 Parameters

`stitcher [in]`

Type: `nvstitchStitcherHandle`

The handle of the stitcher instance.

3.8.25.2 Return Value

Returns one of the following values:

- `NVSTITCH_SUCCESS` on success
- `NVSTITCH_ERROR_NULL_POINTER`
- `NVSTITCH_ERROR_NOT_ALLOWED`

3.8.25.3 Remarks

This function puts the stitcher into a state where it no longer accepts input feed. This function blocks until all remaining frames in the pipeline are stitched.

A stitcher instance must be started again to stitch additional feed after it has been stopped.

3.9 CALLBACK FUNCTION TYPES

3.9.1 nvstitchStitcherOnOutput

```
typedef void (NVSTITCHCALLBACK *nvstitchStitcherOnOutput)(
    uint32_t                      packet_index,
    const nvstitchPayload_t*      out_payload,
    const nvstitchAudioPayload_t* out_audio_payload,
    void*                         app_data
);
```

3.9.1.1 Members

`packet_index [in]`

Type: `uint32_t`

The packet index of the output payload, matching the index of the source input frames.

`out_payload [in]`

Type: `nvstitchPayload_t*`

A pointer to the payload describing the stitched video or image output.

`out_audio_payload [in]`

Type: `nvstitchAudioPayload_t*`

A pointer to the payload describing the stitched audio output.

`app_data[in]`

Type: `void*`

A pointer to user-defined data.

3.9.1.2 Remarks

This data type defines the stitcher callback. Callbacks are executed when a stitcher completes a frame.

3.10 RETURN CODES

In addition to the common and general NVAPI return status codes, the VRWorks 360 High-Level Video interfaces may return the following more specific values:

Return Code	Description
`NVSTITCH_SUCCESS`	Successful execution.
`NVSTITCH_ERROR_GENERAL`	Generic error code, without a specific meaning. Use `nvstitchGetLastErrorString()` to get additional information on the error.
`NVSTITCH_ERROR_BAD_STATE`	Denotes an invalid internal state. The instance that is returning this error is not expected to recover and should be destroyed.
`NVSTITCH_ERROR_BAD_PARAMETER`	One or more of the input parameters is incorrect. Use `nvstitchGetLastErrorString()` to get additional information about the error.

Return Code	Description
NVSTITCH_ERROR_BAD_INDEX	An input index parameter has an out-of-bounds value.
NVSTITCH_ERROR_INVALID_VERSION	The version field in a structure is set to an invalid value. This error may occur because the version field is not initialized properly, or because the SDK version does not support the API version that is currently used.
NVSTITCH_ERROR_OUT_OF_SYSTEM_MEMORY	A system memory allocation has failed.
NVSTITCH_ERROR_OUT_OF_VIDEO_MEMORY	A GPU memory allocation has failed. In the stitching pipeline, this error can often be avoided by lowering the output resolution.
NVSTITCH_ERROR_UNSUPPORTED_CONFIG	The specified audio encoding configuration is unsupported.
NVSTITCH_ERROR_INVALID_DEVICE	The GPU does not meet the minimum requirements.
NVSTITCH_ERROR_NOT_IMPLEMENTED	The specified feature is not yet implemented.
NVSTITCH_ERROR_MISSING_FILE	An input file is missing or invalid.
NVSTITCH_ERROR_NULL_POINTER	A null pointer is passed as a required input parameter.
NVSTITCH_ERROR_NOT_ALLOWED	The function call is not applicable in the current state. For example, input can be fed into the stitcher only while it is in the running state, that is, after `nvstitchStartStitcher()` has been called.
NVSTITCH_ERROR_ENCODER_INIT_FAILED	This error is returned if the encoder initialization fails for any reason. Use `nvstitchGetLastErrorString()` to get additional information about the error.
NVSTITCH_VIDEO_UNSUPPORTED_RIG	Rig properties do not satisfy the requirements for stereo stitching, or the rig type is not supported. The only supported rig type is a horizontal ring of cameras.
NVSTITCH_CALIB_FAILED_CONVERGENCE	Calibration has failed to converge with given input images and settings.
NVSTITCH_CALIB_BAD_INPUT_IMAGE	The input image size width and height provided to `nvstitchFeedCalibrationInput()` do not match the width and height properties of the camera.
NVSTITCH_CALIB_MISSING_INPUT_IMAGE	One or more input images is missing. In the call to `nvstitchFeedCalibrationInput()`, provide all images for each `packet_index < nvstitchCalibrationProperties_t.frame_count` and `source_index < number of cameras`.
NVSTITCH_CALIB_BAD_CAMERA_PROPERTY_VALUE	One or more camera properties for the cameras are incorrect or invalid. Use `nvstitchGetLastErrorString()` to get the camera number and the property.
NVSTITCH_CALIB_INSUFFICIENT_FEATURES	Not enough features are present in one or more input images to perform calibration. Provide input images with more objects in the scene.
NVSTITCH_AUDIO_UNSUPPORTED_CODEC	The audio compression algorithm of the file or buffer data is not supported.

Return Code	Description
NVSTITCH_AUDIO_BUFFER_FULL	Submission of audio data failed because internal buffers do not have enough available space.
NVSTITCH_AUDIO_CONFIGURATION_COMMITTED	A configuration call was made after the low-level audio processing configuration was committed.
NVSTITCH_AUDIO_CONFIGURATION_NOT_COMMITTED	A processing call was made but the low-level audio processing configuration had not been committed.
NVSTITCH_AUDIO_OUT_OF_DATA	A frame of audio could not be generated because at least one input does not have enough data to generate an output frame.
NVSTITCH_AUDIO_INSUFFICIENT_OUTPUT_BUFFER	The size of audio output buffer is insufficient to encode an output frame in the requested output format.
NVSTITCH_AUDIO_ALL_STREAMS_ENDED	A frame of audio could not be generated because all input streams have stopped providing data and all buffered data has been used.
NVSTITCH_AUDIO_CONFIGURATION_MISMATCH	The sample rate or number of channels specified for an AAC file or buffer input did not match the information contained in the AAC.

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.