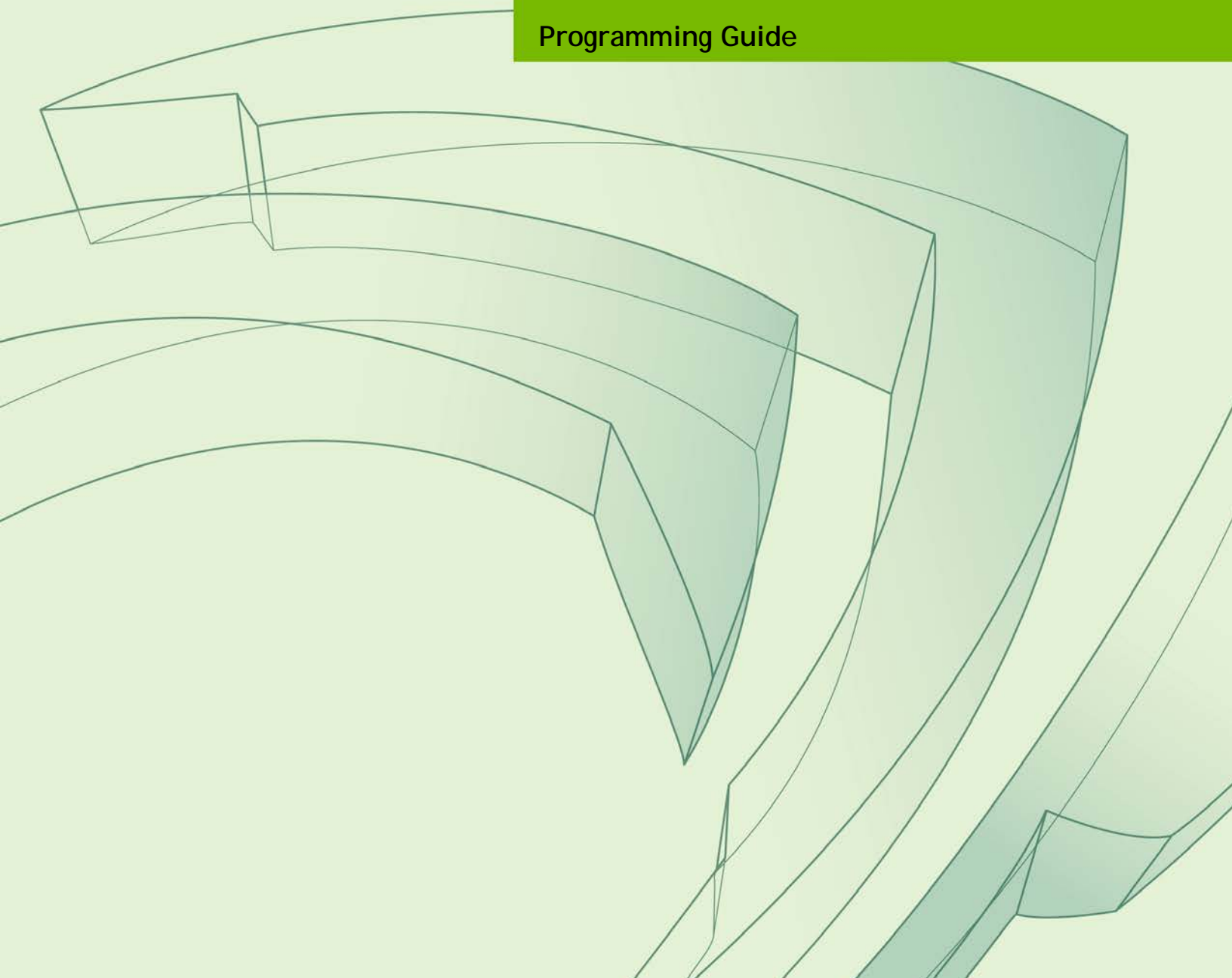




VRWORKS 360 VIDEO CALIBRATION

PG-08666-001_v01.1 | December 2017

Programming Guide



DOCUMENT CHANGE HISTORY

PG-08666-001_v01.1

Version	Date	Description of Change
01	2017-08-04	Initial release
01.1	2017-12-08	Release 1.1

TABLE OF CONTENTS

Chapter 1. Introduction to 360 Video Calibration	1
1.1 Features of the VRWorks 360 Camera Calibration SDK	1
1.2 Structure of the VRWorks 360 Camera Calibration SDK	2
Chapter 2. Getting Started with 360 Video Calibration	3
2.1 Before You Begin	3
2.1.1 Camera Rig Guidelines	3
2.1.2 Input Image Guidelines	4
2.1.3 Input Camera Parameters	4
2.2 Create Instance	6
2.3 Set Camera Properties	6
2.4 Set Images	7
2.5 Calibrate	7
2.6 Get Camera Properties	7
2.7 Optional: Get Calibration Accuracy	8
Chapter 3. 360 Video Calibration API Reference	9
3.1 Enumerations	9
3.1.1 nvcalibCameraLensModelType	9
3.1.2 nvcalibCameraProperties	9
3.1.3 nvcalibDataType	11
3.1.4 nvcalibExtrinsicsAxes	12
3.1.5 nvcalibInputImageFormat	12
3.1.6 nvcalibOption	13
3.1.7 nvcalibOptionOptimizeParams	13
3.1.8 nvOrientationOption	14
3.1.9 nvcalibResultParameter	15
3.2 Common Types	15
3.2.1 nvcalibInstance	15
3.3 Functions	15
3.3.1 nvcalibCalibrate	15
3.3.2 nvcalibCreateInstance	16
3.3.3 nvcalibDestroyInstance	17
3.3.4 nvcalibGetCameraProperty	17
3.3.5 nvcalibGetLastErrorString	19
3.3.6 nvcalibGetResultParameter	19
3.3.7 nvcalibGetRigProperties	21
3.3.8 nvcalibSetCameraProperty	22
3.3.9 nvcalibSetImages	23
3.3.10 nvcalibSetOption	24
3.3.11 nvcalibSetRigProperties	25
3.4 Return Codes	26

Chapter 1.

INTRODUCTION TO 360 VIDEO CALIBRATION

In computer vision, most applications that process images from cameras require the mathematical relationship between points in the 3D world and points in the 2D image space. This relationship is represented by the parameters of the camera model, also called the intrinsics and the extrinsics.

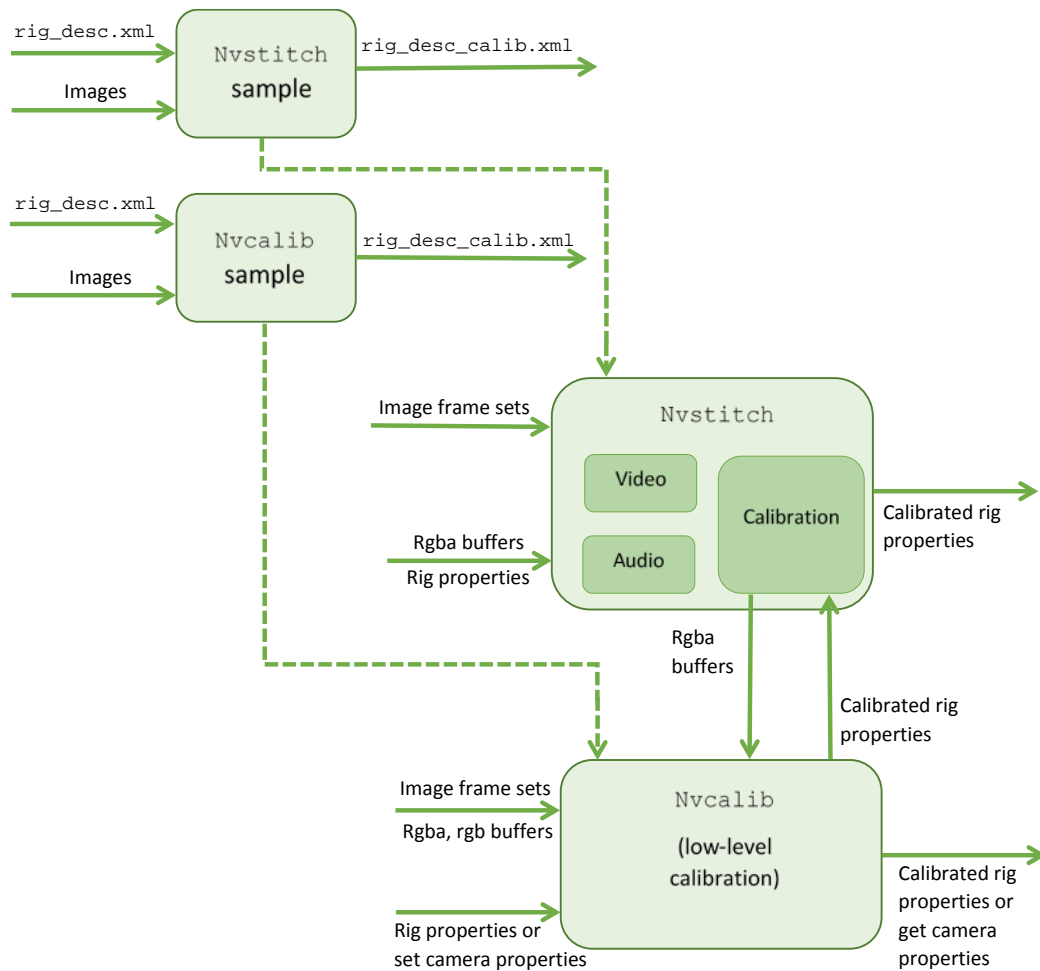
- ▶ The intrinsics represent the mapping between the camera coordinate system and the pixel coordinate system. They comprise focal length, principal point, and distortion coefficients.
- ▶ The extrinsics represent the orientation of the camera relative to the world. They comprise rotation and translation.

Camera calibration is the process of deriving and tuning these intrinsics and extrinsics for a camera. The NVIDIA VRWorks™ 360 Camera calibration SDK allows developers to calibrate camera systems to achieve this goal.

1.1 FEATURES OF THE VRWORKS 360 CAMERA CALIBRATION SDK

- ▶ Can be customized to any camera system and any number of iterations
- ▶ Allows calibration with partial/no parameter estimates
- ▶ Outputs a cross-correlation based quality metric
- ▶ Accepts simple input XML format
- ▶ Provides setter and getter functions for individual parameters in the API.
- ▶ Calibrates for fish-eye and perspective lenses.
- ▶ Supports up to five distortion coefficients with checks for nonmonotonic curves

1.2 STRUCTURE OF THE VRWORKS 360 CAMERA CALIBRATION SDK



Chapter 2.

GETTING STARTED WITH 360 VIDEO CALIBRATION

Before performing a calibration, obtain a set of images from each camera and the input estimates for camera properties. You can then use the `nvcalib` API to perform calibration as explained in this chapter.

For detailed reference information about the `nvcalib` API, see Chapter 3. The functions and data types in this API are defined in the header file `calibration_api.h`.

2.1 BEFORE YOU BEGIN

For best video stitching and camera calibration results, follow the guidelines in this section for camera rig setup and input images. Also ensure that you have values or estimates for the camera input parameters that you'll need to complete the calibration.

The `nvcalib_sample` application provides command-line arguments for specifying the file format of the input images and the input camera parameters.

2.1.1 Camera Rig Guidelines

For best video stitching, following these guidelines for setting up your camera rig:

- ▶ Place the cameras in the rig close together.
- ▶ Maintain good overlap between adjacent cameras.
- ▶ Use wider field of view (FOV) cameras for better overlap.
- ▶ Use higher resolution cameras for better feature extraction.

- Use many cameras for better 360-degree coverage and overlap.



Note: All cameras must overlap with at-least one other camera and no groups of cameras can be isolated.

Heterogeneous rigs (rigs with different cameras) are supported. Input camera parameter estimates are required for heterogeneous rigs.

Rigs that have a large translation (greater than 30 cm) between cameras are not supported.

2.1.2 Input Image Guidelines

For best camera calibration results, follow these guidelines:

- Provide input images captured at the same time from each camera, or captured from a scene with no movement.
- Although one calibration image per camera is sufficient, try to provide more frames for better calibration.
- Capture calibration images of scenes with objects far away.

The `nvcalib_sample` application supports PNG, JPEG, and BMP image formats.

The following `nvcalib_sample` command-line argument specifies the file format of the input images:

```
--ext {png | jpg | bmp}
```

2.1.3 Input Camera Parameters

Input camera parameter estimates are optional for most scenes and rigs. However, for best performance, provide input parameter estimates if possible.



Note: Some rigs and scenes require input parameter estimates. For example, for heterogenous rigs, you **must** provide input parameter estimates.

2.1.3.1 Required Parameters

Lens Type

The lens type of the camera, which is either wide angle (fisheye) or normal angle (brown).

The following `nvcalib_sample` command-line argument specifies the lens type:

```
--lens {fisheye | brown}
```

2.1.3.2 Optional Parameters

Focal Length

The distance in pixels between the optical center and the sensor plane.

The focal length is computed automatically during calibration. If calibration fails to compute the focal length, provide either an estimate of the focal length or the field of view of the cameras.

The focal length is typically found in camera manufacturers' data sheets, in mm. To convert the value in mm to pixels, use the following formula:

$$\text{Focal length in pixels} = (\text{focal length in mm} \div \text{camera sensor width in mm}) \times \text{image width in pixels}$$

The following `nvcalib_sample` command-line argument specifies the focal length:

```
--f1 focal-length-in-pixels
```

Field of View

The horizontal and vertical angle covered by the camera. Provide the field of view if focal length is not known.

The field of view is typically available in camera manufacturers' data sheets as HFOV (horizontal field of view) and VFOV (vertical field of view).

The following `nvcalib_sample` command-line arguments specify the HFOV and VFOV:

```
--hfov horizontal-field-of-view
--vfov vertical-field-of-view
```

Fisheye Radius

The radius of the valid pixels for fisheye images. Specify this parameter only for fisheye-lens cameras.

The fisheye radius is detected automatically from the image. Issues such as lens flare and dark or low light scenes can impair the ability to detect the fisheye radius.

The following `nvcalib_sample` command-line argument specifies the fisheye radius:

```
--radius radius-in-pixels
```

Principal Point

The optical center of the cameras, which is typically the center of the image. This parameter is optional, and an estimate is not required.

Rotation

The orientation of each camera in yaw, pitch, roll angles, or rotation matrix.

This parameter is optional, and an estimate is typically not required.

Translation

The translation in cm along the X, Y, and Z direction from the center of the rig.

This parameter is optional. Calibration currently does not support rigs with translation greater than 20-30 cm.



Note: Translation calibration is highly dependent on the scene and may not converge for all scenes.

The following `nvcalib_sample` command-line argument specifies the translation:

```
--trans translation-in-cm
```

2.2 CREATE INSTANCE

Create an `nvcalibInstance` by calling the `nvcalibCreateInstance()` function, passing in the following parameters:

- ▶ `framesCount` as the number of image sets that will be fed for calibration
- ▶ `camerasCount` as the number of cameras in the rig

The call to `nvcalibCreateInstance()` provides a handle to the calibration instance.

2.3 SET CAMERA PROPERTIES

Set all required camera properties before calling the function to perform the calibration.

You can set properties for a rig and individual cameras in a rig or set individual properties of a camera:

- ▶ To set properties for a rig and individual cameras in a rig, call the `nvcalibSetRigProperties()` function.
The structures `nvstitchVideoRigProperties_t` and `nvstitchCameraProperties_t` are defined in the header file `nvstitch_common_video.h`.
- ▶ To set individual properties of a camera, call the `nvcalibSetCameraProperty()` function.

If you do not supply initial estimates for camera properties, the calibration SDK attempts to determine the properties solely from the input images. In many instances, the images will contain sufficient information for the calibration to succeed.

2.4 SET IMAGES

Provide input images as RGBA, RGB, BGR, BGRA buffers by calling the `nvcalibSetImages()` function. Call this function with the set of images `framesCount` number of times.

Each call to this function provides pointer to a single set of image buffers from all cameras.

2.5 CALIBRATE

After setting the input camera properties and images, call the `nvcalibCalibrate()` function to perform the calibration. This function checks if all camera parameters and input images have been provided and, if so, proceeds with the calibration. The call to `nvcalibCalibrate()` takes several seconds to several minutes depending on the number of cameras.

2.6 GET CAMERA PROPERTIES

After successful calibration, get the calibrated camera properties. The camera properties obtained can then be used for stitching.

You can get properties for a rig and individual cameras in a rig or set individual properties of a camera:

- ▶ To get properties for a rig and individual cameras in a rig, call the `nvcalibGetRigProperties()` function. Get the properties in the structure `nvstitchVideoRigProperties_t`.
The structures `nvstitchVideoRigProperties_t` and `nvstitchCameraProperties_t` are defined in the header file `nvstitch_common_video.h`.
- ▶ To get individual properties of a camera, call the `nvcalibGetCameraProperty()` function.

2.7 OPTIONAL: GET CALIBRATION ACCURACY

Get the accuracy of the results of camera calibration by calling the `nvcalibGetResultParameter()` function.

Chapter 3.

360 VIDEO CALIBRATION API REFERENCE

3.1 ENUMERATIONS

3.1.1 `nvcalibCameraLensModelType`

This enumeration defines the model type of the camera lens. It is used in `nvcalibSetCameraProperty()` and `nvcalibGetCameraProperty()`.

Enumerator	Description
<code>NVCALIB_LENS_MODEL_BROWN = 0</code>	Brown model, suitable for normal-angle perspective lenses. 5 coefficients.
<code>NVCALIB_LENS_MODEL_FISHEYE</code>	Fisheye $f \cdot \theta$ model with coefficients, suitable for wide-angle camera lenses. 4 coefficients.
<code>NVCALIB_LENS_MODEL_FISHEYE_EQUIDISTANT</code>	Fisheye with $f \cdot \theta$ fixed projection (no optimization on distortion). 0 coefficients.
<code>NVCALIB_LENS_MODEL_FISHEYE_STEREOGRAPHIC</code>	Fisheye with $2f \cdot \tan(\theta/2)$ fixed projection (no optimization on distortion). 0 coefficients.
<code>NVCALIB_LENS_MODEL_FISHEYE_EQUISOLID</code>	Fisheye with $2f \cdot \sin(\theta/2)$ fixed projection (no optimization on distortion). 0 coefficients.
<code>NVCALIB_LENS_MODEL_FISHEYE_ORTHOGRAPHIC</code>	Fisheye with $f \cdot \sin(\theta)$ fixed projection (no optimization on distortion). 0 coefficients.

3.1.2 `nvcalibCameraProperties`

This enumeration defines the properties of a camera.

Use the extrinsics properties as follows:

- For rotation, you must specify one of the following sets of properties:
 - `NVCALIB_CAM_PROP_EXTR_ROT_YPR`

- NVCALIB_CAM_PROP_EXTR_AXES and NVCALIB_CAM_PROP_EXTR_ROT_MATRIX
- For translation, you can optionally specify NVCALIB_CAM_PROP_EXTR_AXES and NVCALIB_CAM_PROP_EXTR_TRANSLATION.

Enumerator	Description
NVCALIB_CAM_PROP_WIDTH = 0	Type: NVCALIB_DATATYPE_UINT32 Camera width. Required.
NVCALIB_CAM_PROP_HEIGHT	Type: NVCALIB_DATATYPE_UINT32 Camera height. Required.
NVCALIB_CAM_PROP_INPUT_IMAGE_FORMAT	Type: nvcalibInputImageFormat, NVCALIB_DATATYPE_UINT32 The format of the input image data. Required.
NVCALIB_CAM_PROP_INPUT_IMAGE_PITCH	Type: NVCALIB_DATATYPE_UINT32 The pitch of the input image data, that is, the byte stride between pixels vertically. Optional. Default = NVCALIB_CAM_PROP_WIDTH * 4
Intrinsics	
NVCALIB_CAM_PROP_INTR_PRINCIPAL_POINT	Type: NVCALIB_DATATYPE_FLOAT32 array[2] The (X,Y) coordinates in pixels, relative to the upper-left corner, that specify the location of the principal point. Note that the Y-coordinates increase downward. Optional. Default = (Width-1)/2, (Height-1)/2
NVCALIB_CAM_PROP_INTR_FOCAL_LENGTH	Type: NVCALIB_DATATYPE_FLOAT32 The focal length of the camera in pixels. Optional.
Extrinsics	
NVCALIB_CAM_PROP_EXTR_ROT_YPR	Type: NVCALIB_DATATYPE_FLOAT32 array[3] Extrinsic rotation in radians, specified in aircraft principal axes: yaw, pitch, and roll. They are applied in this order: <ol style="list-style-type: none"> 1. Roll - increases clockwise 2. Pitch - increases tilting upward 3. Yaw - increases toward the right Optional.
NVCALIB_CAM_PROP_EXTR_AXES	Type: CalibExtrinsicsMatrixAxes, NVCALIB_DATATYPE_UINT32 The coordinate system axes in which the rotation matrix and translation are specified. The standard coordinate system has Y-up. All coordinate systems have X to the right. Other coordinate systems have Z-up or Y-down. In any event, all coordinate systems are right-handed.

Enumerator	Description
NVCALIB_CAM_PROP_EXTR_ROT_MATRIX	<p>Type: NVCALIB_DATATYPE_FLOAT32 array[9]</p> <p>The extrinsic rotation transformation, which places the camera into the world.</p> <p>In the standard coordinate system with Y-up, the vector {array[3], array[4], array[5]} aligns with the camera's Y axis (up) when placed into the world. Likewise, {array[6], array[7], array[8]} is the camera's Z-axis, which is opposite the camera's look-at direction.</p> <p>Specify NVCALIB_CAM_PROP_EXTR_AXES to choose a different coordinate system than Y-up.</p> <p>Optional</p>
NVCALIB_CAM_PROP_EXTR_TRANSLATION	<p>Type: NVCALIB_DATATYPE_FLOAT32 array[3]</p> <p>Translation X,Y,Z from the rig center in axes. Typically, a negative multiple of elements 6-8 of the rotation transform. Default = {0,0,0}.</p> <p>Optional.</p>
Distortion	
NVCALIB_CAM_PROP_DISTORTION_TYPE	<p>Type: CameraLensModelType, NVCALIB_DATATYPE_UINT32</p> <p>The lens type of the camera. Default=fisheye. Required.</p>
NVCALIB_CAM_PROP_DISTORTION_COEFFS	<p>Type: NVCALIB_DATATYPE_FLOAT32 array[1-5]</p> <p>Coefficients of the camera lens distortion model</p> <p>Optional. Default = (0,0,0,0,0). Optional.</p>
NVCALIB_CAM_PROP_FISHEYE_RADIUS	<p>The circular clipping radius in pixels for fisheye lens cameras. The clipping circle is centered on the principal point. Zero implies no circular clipping. Optional.</p>
NVCALIB_CAM_PROP_LAYOUT	<p>Type: nvstitchCameraLayout, NVCALIB_DATATYPE_UINT32</p> <p>The layout of the camera in the rig. Default = Equatorial. Optional.</p>

3.1.3 nvcalibDataType

This enumeration defines the calibration API data types. It is used in `nvcalibGetCameraProperty()`, `nvcalibSetCameraProperty()`, `nvcalibGetResultParameter()`.

Enumerator	Description
NVCALIB_DATATYPE_UINT32 = 0	32-bit unsigned integer.

Enumerator	Description
NVCALIB_DATATYPE_FLOAT32	32-bit single-precision IEEE floating-point.
NVCALIB_DATATYPE_CSTRING	C-string, null terminated.

3.1.4 nvcalibExtrinsicsAxes

This enumeration defines the axes of the extrinsics coordinate system. It is used in `nvcalibGetCameraProperty()` and `nvcalibSetCameraProperty()`. All coordinate systems are right handed with X increasing to the right. Therefore, only the orientation of the Y or Z axis need be specified.

Enumerator	Description
NVCALIB_AXES_Y_UP = 0	Right hand system with Y up: <ul style="list-style-type: none"> • X = Right • Y = Up • Z = Inward, which is opposite to the look-at direction
NVCALIB_AXES_Y_DOWN	Right hand system with Y down: <ul style="list-style-type: none"> • X = Right • Y = Down • Z = Outward, which is the look-at direction
NVCALIB_AXES_Z_UP	Right hand system with Z up: <ul style="list-style-type: none"> • X = Right • Y = Outward, which is the look-at direction • Z = Up

3.1.5 nvcalibInputImageFormat

This enumeration defines the supported input image types. It is used in `nvcalibGetCameraProperty()` and `nvcalibSetCameraProperty()`. In this release, only the following image types are supported:

- ▶ RGBA8
- ▶ RGA8
- ▶ BGR8
- ▶ BGRA8

Enumerator	Description
NVCALIB_IN_FORMAT_RGBA8 = 0	4-byte pixels, arranged as {R,G,B,A} in memory.
NVCALIB_IN_FORMAT_RGB8	3-byte pixels, arranged as {R,G,B} in memory.
NVCALIB_IN_FORMAT_BGRA8	4-byte pixels, arranged as {B,G,R,A} in memory.
NVCALIB_IN_FORMAT_BGR8	3-byte pixels, arranged as {B,G,R} in memory.
NVCALIB_IN_FORMAT_GRAY8	1-byte grayscale pixels.

Enumerator	Description
NVCALIB_IN_FORMAT_PNG	PNG (unimplemented).
NVCALIB_IN_FORMAT_JPG	JPEG (unimplemented).

3.1.6 nvcalibOption

This enumeration defines calibration options. It is used in `nvcalibSetOption()`.

Enumerator	Description
NVCALIB_OPTION_USE_TRANS = 0	Type: NVCALIB_DATATYPE_UINT32 Whether to use rig translation parameters for triangulation-based calibration. <ul style="list-style-type: none"> 0=Don't Use (Default) 1=Use
NVCALIB_OPTION_OPTIMIZE_PARAMS	Type: nvcalibOptionOptimizeParams (uint32_t) Camera parameters to be optimized. NVCALIB_OPTION_OPTIMIZE_PARAMS_ALL (Default).
NVCALIB_OPTION_MODE	Type: NVCALIB_DATATYPE_UINT32 The calibration mode. <ul style="list-style-type: none"> 0= High Quality Mode (Default) 1=Fast Mode.
NVCALIB_OPTION_CORRESPONDENCE_DIRECTORY	Type NVCALIB_DATATYPE_CSTRING Write images with feature points and correspondence overlaid to the specified directory. Set to "" or NULL to clear.
NVCALIB_OPTION_PANORAMA_ORIENTATION	Type NVCALIB_DATATYPE_UINT32 or NVCALIB_DATATYPE_FLOAT32 The orientation of the panorama: <ul style="list-style-type: none"> NVCALIB_ORIENT_PANO_AUTO_BALANCE (default) NVCALIB_ORIENT_PANO_FREEZE_FIRST_CAMERA NVCALIB_ORIENT_PANO_APPLY_TRANSFORM NVCALIB_ORIENT_PANO_APPLY_YPR

3.1.7 nvcalibOptionOptimizeParams

This enumeration defines the sets of camera parameters to be optimized. It is used in `nvcalibSetOption()`.

Enumerator	Description
NVCALIB_OPTION_OPTIMIZE_PARAMS_ALL = 0	Optimize all parameters (intrinsics, extrinsics and distortion).

Enumerator	Description
NVCALIB_OPTION_OPTIMIZE_PARAMS_EXTRINSICS	Optimize only extrinsic parameters.

3.1.8 nvOrientationOption

This enumeration is used to indicate the desired orientation of the panorama. It is used in `nvcalibSetOption()`.

Some enumerators require extra arguments. The data type of enumerators that require extra arguments should be declared as `NVCALIB_DATATYPE_FLOAT32`. If no extra arguments are needed, `NVCALIB_DATATYPE_UINT32` can be used.

The APPLY enumerators can be used to level the horizon if the rig was not upright when the material was captured.

Enumerator	Description
NVCALIB_ORIENT_PANO_AUTO_BALANCE	Type <code>NVCALIB_DATATYPE_INT32</code> or <code>NVCALIB_DATATYPE_FLOAT32</code> . Count 1. Balance the panorama on the dominant axis of the rig, as determined from the camera poses.
NVCALIB_ORIENT_PANO_FREEZE_FIRST_CAMERA	Type <code>NVCALIB_DATATYPE_INT32</code> or <code>NVCALIB_DATATYPE_FLOAT32</code> . Count 1. Maintain the orientation of the first camera.
NVCALIB_ORIENT_PANO_APPLY_TRANSFORM	Type <code>NVCALIB_DATATYPE_FLOAT32</code> . Determine the pose of the panorama from a linear transformation. This enumerator must be followed by 9 additional single-precision floating-point numbers that comprise the desired pose of the panorama expressed as a linear transformation. For example, if the pose of the fourth camera is supplied, the fourth camera will be in the center of the stitched panorama. The total count is 10, and the data is <code>float[10]</code> .
NVCALIB_ORIENT_PANO_APPLY_YPR	Type <code>NVCALIB_DATATYPE_FLOAT32</code> . Determine the pose of the panorama from yaw, pitch, and roll angles. This enumerator must be followed by 3 additional single-precision floating-point numbers that comprise the desired pose of the panorama expressed using the yaw, pitch and roll angles in radians. The total count is 4, and the data is <code>float[4]</code> .

3.1.9 nvcalibResultParameter

This enumeration is used to indicate the computation of the accuracy of a calibration. It is used in `nvcalibGetResultParameter()`.

Enumeration	Description
NVCALIB_RESULTS_ACCURACY	Type NVCALIB_DATATYPE_FLOAT32 Accuracy indication of results from [0-1].

3.2 COMMON TYPES

3.2.1 nvcalibInstance

```
typedef uint32_t nvcalibInstance;
```

3.3 FUNCTIONS

3.3.1 nvcalibCalibrate

```
nvcalibResult NVCALIBAPI nvcalibCalibrate(
    nvcalibInstance hInstance
);
```

3.3.1.1 Parameters

`hInstance`

Type: `nvcalibInstance`

The calibration instance on which to run the calibration.

3.3.1.2 Return Value

Returns one of the following values:

- ▶ `NVCALIB_SUCCESS` on success
- ▶ `NVCALIB_ERROR_HANDLE_INVALID` if `nvcalibInstance` is invalid
- ▶ `NVCALIB_ERROR_ARGUMENT_INVALID` if an argument is invalid
- ▶ `NVCALIB_ERROR_ARGUMENT_MISSING` if required properties are not set
- ▶ `NVCALIB_ERROR_ADD_IMAGES_COUNT_MISMATCH` if the frames added do not match the frames

- ▶ `NVCALIB_ERROR_INSUFFICIENT_FEATURES` if the camera does not have enough features
- ▶ `NVCALIB_ERROR_FAILED_CONVERGENCE` if the calibration failed to converge to optimum values

3.3.1.3 Remarks

This function runs the main calibration process.

3.3.2 `nvcalibCreateInstance`

```
nvcalibResult NVCALIBAPI nvcalibCreateInstance(
    [in]      const uint32_t    framesCount,
    [in]      const uint32_t    camerasCount,
    [out]     nvcalibInstance*  hInstance
);
```

3.3.2.1 Parameters

`framesCount` [in]

Type: `uint32_t`

The number of frames to be used for calibration.

`camerasCount` [in]

Type: `uint32_t`

The number of cameras in the rig to calibrate.

`hInstance` [out]

Type: `nvcalibInstance*`

Pointer to the `nvcalibInstance` instance created.

3.3.2.2 Return Value

Returns one of the following values:

- ▶ `NVCALIB_SUCCESS` on success
- ▶ `NVCALIB_ERROR_HANDLE_INVALID` if the handle `nvcalibInstance` is invalid
- ▶ `NVCALIB_ERROR_ARGUMENT_INVALID` if input arguments are invalid

3.3.2.3 Remarks

This function creates the calibration instance.

3.3.3 nvcalibDestroyInstance

```
nvcalibResult NVCALIBAPI nvcalibDestroyInstance(
    nvcalibInstance hInstance);
```

3.3.3.1 Parameters

hInstance

Type: nvcalibInstance

The calibration instance to destroy.

3.3.3.2 Return Value

Returns one of the following values:

- ▶ NVCALIB_SUCCESS on success
- ▶ NVCALIB_ERROR_HANDLE_INVALID if the handle nvcalibInstance is invalid

3.3.3.3 Remarks

This function destroys the calibration instance, freeing the memory that was allocated to it.

3.3.4 nvcalibGetCameraProperty

```
nvcalibResult NVCALIBAPI nvcalibGetCameraProperty(
    nvcalibInstance      hInstance,
    uint32_t             cameraIndex,
    nvcalibCameraProperties cameraProperty,
    nvcalibDataType      propertyDataType,
    uint32_t             dataCount,
    void*                propertyData
);
```

3.3.4.1 Parameters

hInstance [in]

Type: nvcalibInstance

The calibration instance from which to get a camera property.

cameraIndex [in]

Type: uint32_t

The index of the camera from which to get a property.

cameraProperty

Type: nvcalibCameraProperties

The name of the camera property to get, which must be one of the properties defined in “nvcalibCameraProperties” on page 9: width, height, pitch, format, rotation, principal point, focal length, translation, axes, lens type, distortion coefficients, fisheye radius, layout.

propertyDataType

Type: nvcalibDataType

The data type of the option to get, which must be one of the data types defined in “nvcalibDataType” on page 11. If this parameter is set to any other data type, an error occurs.

dataCount

Type: uint32_t

The number of data elements.

propertyData

Type: void*

A pointer to the destination data buffer. The destination must have the appropriate data size.

3.3.4.2 Return Value

Returns one of the following values:

- ▶ NVCALIB_SUCCESS on success
- ▶ NVCALIB_ERROR_HANDLE_INVALID if the handle nvcalibInstance is invalid
- ▶ NVCALIB_ERROR_ARGUMENT_UNSUPPORTED if cameraProperty specifies an unsupported property
- ▶ NVCALIB_ERROR_ARGUMENT_COUNT_INVALID if the number of property data elements is invalid
- ▶ NVCALIB_ERROR_ARGUMENT_DATATYPE_INVALID if the data type of the property is invalid

3.3.4.3 Remarks

This function gets an individual camera property.

3.3.5 nvcalibGetLastErrorString

```
nvcalibResult NVCALIBAPI nvcalibGetLastErrorString(
    nvcalibInstance hInstance,
    const char**    errorStringPtr
);
```

3.3.5.1 Parameters

`hInstance` [in]

Type: `nvcalibInstance`

The calibration instance from which to get the last error string.

`errorStringPtr` [in]

Type: `const char**`

A pointer to a NULL-terminated C-string. This string provides more detail about the most recently received error code. Both the pointer and the string should be considered to be ephemeral, and will change the next time a call to a function in this API returns an error. Typically, the string is immediately printed and discarded. The string would need to be copied to be retained for a longer period.

3.3.5.2 Return Value

Returns one of the following values:

- ▶ `NVCALIB_SUCCESS` on success
- ▶ `NVCALIB_ERROR_HANDLE_INVALID` if the handle `nvcalibInstance` is invalid

3.3.5.3 Remarks

After a result other than `NVCALIB_SUCCESS` is received, call this function to get additional information about the error.

3.3.6 nvcalibGetResultParameter

```
nvcalibResult NVCALIBAPI nvcalibGetResultParameter(
    nvcalibInstance      hInstance,
    nvcalibResultParameter resultParameter,
    nvcalibDataType      resParamDataType,
    uint32_t             dataCount,
    void*                resultParameterData
);
```

3.3.6.1 Parameters

`hInstance`

Type: `nvcalibInstance`

The calibration instance for which to get the result parameter.

`resultParameter`

Type: `nvcalibResultParameter`

The name of the result parameter to get, which must be one of the parameters defined in “`nvcalibResultParameter`” on page 14.

`resParamDataType`

Type: `nvcalibDataType`

The data type of the parameter to get, which must be one of the data types defined in “`nvcalibDataType`” on page 11. If this parameter is set to any other data type, an error occurs.

`dataCount`

Type: `uint32_t`

The number of data elements.

`resultParameterData`

Type: `void*`

A pointer to the destination data buffer.

3.3.6.2 Return Value

Returns one of the following values:

- ▶ `NVCALIB_SUCCESS` on success
- ▶ `NVCALIB_ERROR_HANDLE_INVALID` if the handle `nvcalibInstance` is invalid
- ▶ `NVCALIB_ERROR_ARGUMENT_UNSUPPORTED` if `resultParameter` specifies an unsupported property
- ▶ `NVCALIB_ERROR_ARGUMENT_COUNT_INVALID` if the number of property data elements is invalid
- ▶ `NVCALIB_ERROR_ARGUMENT_DATATYPE_INVALID` if the data type of the property is invalid

3.3.6.3 Remarks

This function gets the result parameters after calibration. Call this function after call to `nvcalibCalibrate()`.

Result parameters that can be retrieved are specified in in “`nvcalibResultParameter`” on page 14.

3.3.7 nvcalibGetRigProperties

```
nvcalibResult NVCALIBAPI nvcalibGetRigProperties(
    nvcalibInstance          hInstance,
    nvstitchVideoRigProperties_t* cameraProperties
);
```

3.3.7.1 Parameters

`hInstance`

Type: `nvcalibInstance`

The calibration instance for which to get the camera rig properties.

`cameraRigProperties`

Type: `nvstitchVideoRigProperties_t*`

A pointer to the destination data buffer.

3.3.7.2 Return Value

Returns one of the following values:

- ▶ `NVCALIB_SUCCESS` on success
- ▶ `NVCALIB_ERROR_HANDLE_INVALID` if the handle `nvcalibInstance` is invalid

3.3.7.3 Remarks

This function gets all the camera rig properties in the structure `nvstitchVideoRigProperties_t` defined in `nvstitch_common_video.h`.

3.3.8 nvcalibSetCameraProperty

```

nvcalibResult NVCALIBAPI nvcalibSetCameraProperty(
    nvcalibInstance      hInstance,
    uint32_t             cameraIndex,
    nvcalibCameraProperties cameraProperty,
    nvcalibDataType       propertyDataType,
    uint32_t             dataCount,
    const void*          propertyData
);

```

3.3.8.1 Parameters

hInstance

Type: `nvcalibInstance`

The calibration instance for which to set the camera property.

cameraIndex

Type: `uint32_t`

The index of the camera for which set a property.

cameraProperty

Type: `nvcalibCameraProperties`

The name of the camera property to set, which must be one of the properties defined in “nvcalibCameraProperties” on page 9: width, height, pitch, format, rotation, principal point, focal length, translation, axes, lens type, distortion coefficients, fisheye radius, or layout.

propertyDataType

Type: `nvcalibDataType`

The data type of the option to set, which must be one of the data types defined in “nvcalibDataType” on page 11. If this parameter is set to any other data type, a typecasting error occurs.

dataCount

Type: `uint32_t`

The number of data elements.

propertyData

Type: `void*`

A pointer to the source data buffer.

3.3.8.2 Return Value

Returns one of the following values:

- ▶ `NVCALIB_SUCCESS` on success
- ▶ `NVCALIB_ERROR_HANDLE_INVALID` if the handle `nvcalibInstance` is invalid
- ▶ `NVCALIB_ERROR_ARGUMENT_UNSUPPORTED` if `cameraProperty` specifies an unsupported property
- ▶ `NVCALIB_ERROR_ARGUMENT_COUNT_INVALID` if the number of property data elements is invalid
- ▶ `NVCALIB_ERROR_ARGUMENT_DATATYPE_INVALID` if the data type of the property is invalid

3.3.8.3 Remarks

This function sets an individual camera property.

3.3.9 nvcalibSetImages

```
nvcalibResult NVCALIBAPI nvcalibSetImages(
    [in]      nvcalibInstance hInstance,
    [in]      const void*      imageData[]
);
```

3.3.9.1 Parameters

`hInstance`

Type: `nvcalibInstance`

The calibration instance to which to add the images.

`imageData`

Type: `void*`

An array of pointers to image data. The size of the array should be the number of cameras passed. Each pointer points to a buffer, the size of which is the product of the following values:

- ▶ `NVCALIB_CAM_PROP_HEIGHT`
- ▶ `NVCALIB_CAM_PROP_INPUT_IMAGE_PITCH` (or computed from `NVCALIB_CAM_PROP_WIDTH` if not supplied)
- ▶ The number of bytes corresponding to `nvcalibInputImageFormat`

3.3.9.2 Return Value

Returns `NVCALIB_SUCCESS` on success.

3.3.9.3 Remarks

This function adds images to a calibration instance and adds a single set of frames from all cameras. The number of calls to this function to add each frameset this must match `framesCount`.

3.3.10 nvcalibSetOption

```
nvcalibResult NVCALIBAPI nvcalibSetOption(
    nvcalibInstance hInstance,
    nvcalibOption   optionName,
    nvcalibDataType optionDataType,
    uint32_t        dataCount,
    void*           optionData
);
```

3.3.10.1 Parameters

`hInstance`

Type: `nvcalibInstance`

The calibration instance for which to set the options.

`optionName`

Type: `nvcalibOption`

The name of the option to set, which must be one of the options defined in “`nvcalibOption`” on page 13: whether to use translation, the parameters to optimize, quality mode, correspondence image directory, or panorama orientation.

`optionDataType`

Type: `nvcalibDataType`

The data type of the option to set, which must be one of the data types defined in “`nvcalibDataType`” on page 11.

`dataCount`

Type: `uint32_t`

The number of data elements.

`optionData`

Type: `void*`

A pointer to the source data buffer.

3.3.10.2 Return Value

Returns one of the following values:

- ▶ `NVCALIB_SUCCESS` on success
- ▶ `NVCALIB_ERROR_HANDLE_INVALID` if the handle `nvcalibInstance` is invalid
- ▶ `NVCALIB_ERROR_ARGUMENT_UNSUPPORTED` if `optionName` specifies an unsupported option
- ▶ `NVCALIB_ERROR_ARGUMENT_COUNT_INVALID` if the number of option data elements is invalid
- ▶ `NVCALIB_ERROR_ARGUMENT_DATATYPE_INVALID` if the data type of the option is invalid

3.3.10.3 Remarks

This function sets a particular calibration option from `nvcalibOption`. Call this function before a call to `nvcalibCalibrate()`.

3.3.11 `nvcalibSetRigProperties`

```
nvcalibResult NVCALIBAPI nvcalibSetRigProperties(
    nvcalibInstance          hInstance,
    const nvstitchVideoRigProperties_t* cameraRigProperties
);
```

3.3.11.1 Parameters

`hInstance`

Type: `nvcalibInstance`

The calibration instance for which to set the camera rig properties.

`cameraRigProperties`

Type: `nvstitchVideoRigProperties_t*`

A pointer to the source data buffer.

3.3.11.2 Return Value

Returns one of the following values:

- ▶ `NVCALIB_SUCCESS` on success
- ▶ `NVCALIB_ERROR_HANDLE_INVALID` if the handle `nvcalibInstance` is invalid

3.3.11.3 Remarks

This function sets all the camera rig properties from the structure `nvstitchVideoRigProperties_t` defined in `nvstitch_common_video.h`. Call this function after a call to `nvcalibCalibrate()` to get calibrated properties.

3.4 RETURN CODES

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

Return Code	Value	Description
NVCALIB_SUCCESS	0	The function call succeeded.
NVCALIB_ERROR_UNSUPPORTED		The function call is not supported.
NVCALIB_ERROR_ARGUMENT_INVALID		An argument passed to the function is invalid.
NVCALIB_ERROR_ARGUMENT_MISSING		One or more required properties is not provided.
NVCALIB_ERROR_ARGUMENT_UNSUPPORTED		The argument passed is unsupported.
NVCALIB_ERROR_ARGUMENT_COUNT_INVALID		The argument count is invalid.
NVCALIB_ERROR_ARGUMENT_DATATYPE_INVALID		The data type of an argument is incorrect.
NVCALIB_ERROR_ARGUMENT_DATA_INVALID		The data provided as argument is invalid.
NVCALIB_ERROR_HANDLE_INVALID		The handle <code>nvcalibInstance</code> is invalid.
NVCALIB_ERROR_ADD_IMAGES_COUNT_MISMATCH		The number of image frame sets added does not match <code>framesCount</code> .
NVCALIB_ERROR_CAMERA_COUNT_INVALID		The <code>camerasCount</code> parameter is invalid.
NVCALIB_ERROR_INSUFFICIENT_FEATURES		One or more frames has too few features to perform a calibration.
NVCALIB_ERROR_FAILED_CONVERGENCE		Calibration failed to converge to optimum values

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.

