NVIDIA Audio Effects

Programming Guide (Windows)
Table of Contents

Chapter 1. Introduction to NVIDIA Audio Effects SDK in Windows................................. 1

Chapter 2. Getting Started with the Audio Effects SDK in Windows............................... 2
  2.1. Hardware Requirements............................................................................................. 2
  2.2. Software Requirements............................................................................................ 2
  2.3. Installing the Audio Effects SDK............................................................................. 3
  2.4. Audio Effects SDK Sample Application................................................................... 3
      2.4.1. Building the Sample Application........................................................................ 3
      2.4.2. Running the Sample Application........................................................................ 4

Chapter 3. Using the Audio Effects SDK in Applications.................................................. 6
  3.1. About the Background Noise Suppression Effect....................................................... 6
  3.2. About the Room Echo Cancellation Effect (BETA)................................................... 7
  3.3. About the Room Echo Cancellation + Background Noise Suppression Effect (BETA)....... 8
  3.4. Creating an Audio Effect............................................................................................ 8
  3.5. Setting the Parameters of an Effect.............................................................................. 8
  3.6. Getting the Parameters of an Effect............................................................................ 8
  3.7. Getting Supported Devices......................................................................................... 9
  3.8. Loading an Audio Effect............................................................................................. 10
  3.9. Running an Audio Effect........................................................................................... 10
  3.10. Destroying an Audio Effect....................................................................................... 11
  3.11. Use Multiple GPUs.................................................................................................. 11
      3.11.1. Selecting a GPU for Audio Effects Processing in a Multi-GPU Environment.......... 11
      3.11.2. Offloading GPU Selection to the SDK for Audio Effects Processing in a Multi-GPU Environment........................................................................................................................... 12
  3.12. Selecting Different GPUs for Different Tasks............................................................ 12

Chapter 4. Audio Effects SDK for Windows API Reference.............................................. 14
  4.1. Type Definitions......................................................................................................... 14
      4.1.1. NvAFX_EffectSelector......................................................................................... 14
      4.1.2. NvAFX_ParameterSelector................................................................................ 14
      4.1.3. Nvafx_Handle..................................................................................................... 15
  4.2. Functions................................................................................................................... 15
      4.2.1. Nvafx_GetEffectList......................................................................................... 15
      4.2.2. Nvafx_CreateEffect......................................................................................... 16
      4.2.3. Nvafx_DestroyEffect....................................................................................... 17
      4.2.4. Nvafx_SetString............................................................................................... 17
      4.2.5. Nvafx_SetU32.................................................................................................. 18
4.2.6. NvAFX_SetFloat................................................................................................................ 19
4.2.7. NvAFX_GetString...............................................................................................................19
4.2.8. NvAFX_GetU32.................................................................................................................. 20
4.2.9. NvAFX_GetFloat................................................................................................................ 21
4.2.10. NvAFX_GetSupportedDevices.........................................................................................22
4.2.11. NvAFX_Load.................................................................................................................... 23
4.2.12. NvAFX_Run......................................................................................................................23
4.2.13. NvAFX_Reset...................................................................................................................23
4.3. Run Codes............................................................................................................................... 24
Chapter 1. Introduction to NVIDIA Audio Effects SDK in Windows

The NVIDIA® Audio Effects SDK for Windows is powered by NVIDIA RTX™ graphic processor units (GPUs) with Tensor Cores, so the algorithm throughput is greatly accelerated, and latency is reduced. Refer to Tensor Cores for more information. By leveraging the capabilities of NVIDIA RTX GPUs, developers can use the SDK to build audio plugins and add sound effects for broadcasting.

The Audio Effects SDK provides the following audio effects for broadcast use cases with real-time audio processing:

- **Denoising**: Recordings of speech made outside of a recording studio can contain a lot of background noise, which causes the speech to be garbled and difficult to understand. The audio denoising effect removes the background noise.

- **Dereverb/Room echo cancellation**: Recordings of speech might contain the reverberation from the recording environment. Excessive reverb reduces speech clarity, and the dereverb effect helps remove or suppress the reverb.

- **Denoise plus Dereverb**: The effects listed above are combined to remove/suppress the noise and reverb.

Note: In this guide, the term *Room Echo Cancellation* is used interchangeably with *Dereverb*. 
Chapter 2. Getting Started with the Audio Effects SDK in Windows

This section provides information about the hardware and software requirements to install the SDK.

2.1. Hardware Requirements

Here is the hardware requirement to install and use the Audio Effects SDK in Windows. The SDK is supported on NVIDIA GPUs with Tensor Cores.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Required Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA GPU</td>
<td>NVIDIA GPUs with Tensor Cores</td>
</tr>
</tbody>
</table>

2.2. Software Requirements

The Audio Effects SDK requires a specific version of the Windows OS and other associated software on which the SDK depends.

The NVIDIA CUDA® and TensorRT™ dependencies are bundled with the SDK Installer. See Installing the NVIDIA Audio Effects SDK for Windows.

The Audio Effects SDK is designed and optimized for client-side application integration and for local deployment. We do not officially support the testing, experimentation, deployment of this SDK in a datacenter/cloud environment.

<table>
<thead>
<tr>
<th>Software</th>
<th>Required Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows OS</td>
<td>64-bit Windows 10</td>
</tr>
<tr>
<td>Microsoft Visual Studio</td>
<td>2017 (MSVC15.0) or later</td>
</tr>
<tr>
<td>CMake</td>
<td>3.9 or later</td>
</tr>
<tr>
<td>NVIDIA Graphics Driver for Windows</td>
<td>462.46 or later</td>
</tr>
</tbody>
</table>
2.3. Installing the Audio Effects SDK

The Audio Effects SDK for Windows is distributed in the following parts:

- A developer package that contains the AI models, binaries, header file, and a sample app.
- A redistributable package that contains only the AI models and binaries.

This package streamlines the installation and usage of the SDK on the end-user’s computer.

To develop applications with the Audio Effects SDK, you must install the developer package and provide the path to this package during compilation and linking. The app will use the SDK functions that are exposed by the SDK header and dynamically link against the provided libraries.

During deployment, the redistributable package installer helps install the necessary runtime components. To help your app access the runtime components on the end-user’s computer, after the redistributable package is installed, the installer completes the following tasks:

- Copies the AI models and binaries to the install location.
- Sets the NVAFX_SDK_DIR environment variable that points to the directory where the redistributable package is installed and that contains the AI models and binaries.

Your app needs to use this environment variable to locate and load the binaries and the AI model.

2.4. Audio Effects SDK Sample Application

This section provides information about the sample application in the SDK.

To demonstrate the various audio effects, the SDK provides the following options:

- The sample includes the source file `effects_demo.cpp` that can be compiled and run.
- The sample application is also available as a binary file `effects_demo.exe` that can be executed directly.

2.4.1. Building the Sample Application

The SDK includes the source code for building the sample applications.

1. Start the CMake GUI and specify the source folder and a build folder for the binary files.
   a). For the source folder, ensure that the path ends in `package`.
   b). For the build folder, ensure that the path ends in `package/build`.
2. Use CMake to configure and generate the Visual Studio solution file.
   a). Click Configure.
Getting Started with the Audio Effects SDK in Windows

b). When prompted to confirm whether CMake can create the build folder, click OK.
c). To enable CMake to locate the CUDA compiler, select Visual Studio for the generator and x64 for the platform.
d). To finish configuring the Visual Studio solution file, click Finish.
e). To generate the Visual Studio solution file, click Generate.

3. Use Visual Studio to generate the application binary (.exe) file from the solution file that was generated in the previous step.
   a). In CMake, click Open Project to open Visual Studio.
   b). In Visual Studio, click Build > Build Solution.

2.4.2. Running the Sample Application

Here is some information about how to run the sample application.

In a Command Prompt window, enter the following command:

```
effects_demo.exe -c config-file
```

where `-c config-file` specifies the path of the effect sample config file, for example, denoise48k_cfg_turing.txt. A few sample config files are supplied with the sample application.

The following example runs the effects_demo.exe sample application:

```
effects_demo.exe -c denoise48k_cfg_turing.txt
```

The config files contain the following parameters with one pair per line:

- **effect “effect to be applied”**
  Specifies the effect that will be applied, for example, denoiser. See Introduction to NVIDIA Audio Effects SDK for a complete list of supported effects.
- **model model-file**
  Specifies the path of the model file that will be used in the sample application, for example, denoiser_48k.trtpkg.

**Note:** A 48kHz model file and a 16kHz model file is included with the SDK for all effects.

- **input_wav input-audio-file**
  Specifies the path of the noisy input audio .wav file to use, for example, noisy_48k.wav.

**Note:** A sample input audio file is included with the sample application.

- **output_wav output-audio-file**
  Specifies the path of the file to which the applied effect audio output is to be written, for example, denoised_48k.wav. The audio format of the output file will match the audio format of the input file.

**Note:** Only the .wav file format is supported.

- **intensity_ratio intensity-ratio**
 Specifies the denoising intensity ratio. The value of this parameter ranges from 0.0f to 1.0f, where a higher value indicates a stronger suppression of noise/reverb. A value of 0.0f is equivalent to a passthrough.
By using the Audio Effects SDK in Windows, you can enable an application to apply effects to audio. The Audio Effects API is a C API but can also be used with applications that are built using C++.

### 3.1. About the Background Noise Suppression Effect

Recordings of speech made outside of a recording studio contain a lot of background noise. The Audio Denoiser Effect removes the following types of background noise from audio recordings:

- AC noise
- PC noise
- Babble / crowd noise
- Chatter from other people
- Keyboard
- Fan noise
- Sirens
- Clapping
- Tapping
- Sounds of furniture moving
- Sounds of glass breaking
- Traffic noise
- Mouse clicks

**Note:** In this guide, the term *Background Noise Suppression* is used interchangeably with *Denoising.*
Sounds of a train passing by
Sounds of a vacuum cleaner
Washing machine
Metal sounds
Baby crying
Wrappers (plastic / non-plastic rustling)
Water taps / running water
Cooking sounds (cutting, cooker, etc)
Construction site sounds
Rains
Pet sounds
Drums
Door slamming
Bird chirping
Phone ringing

The Background Noise Suppression effect retains female voices and male voices the same way and also retains emotive speech such as happy, sad, excited, and angry tones. The effect does not include extreme emotive cases such as just loud laughing, shrieking, screaming, and excessive crying.

Here is a list of the Background Noise Suppression Effect characteristics:

- The audio format is 48kHz and 16KHz sample rate and 32-bit float type.

### 3.2. About the Room Echo Cancellation Effect (BETA)

Recordings of speech made in a large room/hall contain echoes and reverbs. The Audio Room Echo Cancellation Effect removes/suppresses such echoes and reverbs from audio recordings.

**Note:** In this guide, the term *Room Echo Cancellation* is used interchangeably with *Dereverb*.

This effect has the following characteristics:

- The audio format is 48kHz and 16KHz sample rate and 32-bit float type.
3.3. About the Room Echo Cancellation + Background Noise Suppression Effect (BETA)

This effect combines the denoiser effect (see About the Background Noise Suppression Effect) and Dereverb effect (see About the Room Echo Cancellation Effect) into one effect. It applies to the denoising and dereverb effects on the input audio.

**Note:** In this guide, the term Room Echo Cancellation + Background Noise Suppression is used interchangeably with Dereverb+Denoiser.

Here is a list of the Audio Room Echo Cancellation Effect + Background Noise Removal characteristics:

- The audio format is 48kHz and 16KHz sample rate and 32-bit float type.

3.4. Creating an Audio Effect

Call the `NvAFX_CreateEffect()` function and specify the following information as parameters:

- The `NvAFX_EffectSelector` type `NVAFX_EFFECT_DENOISER` or any other parameters that are supported.
- A pointer to the location where the handle to the newly created audio effect will be stored.

The `NvAFX_CreateEffect()` function creates a handle to the audio effect instance for use in additional API calls.

This example creates a denoiser audio effect:

```c
NvAFX_Status err = NvAFX_CreateEffect(NVAFX_EFFECT_DENOISER, &handle);
```

3.5. Setting the Parameters of an Effect

An audio effect requires a model to transform the input audio, and each model supports a specific audio sample rate. You must set the path to the model file.

3.6. Getting the Parameters of an Effect

The number of I/O samples per frame and number of I/O audio channels for an Audio Effect and cannot be changed. Before running an audio effect, you must get the number of I/O
samples per frame and the number of I/O channels to pass as parameters to the function. See Running an Audio Effect for more information.

Note: To ensure that the sample rate of the audio that you are transforming is compatible with the audio effect, you can also get the sample rate.

To get one of these parameters, call the NvAFX_GetU32() function and specify the following information as parameters:

- The effect handle that was created.

  See Creating an Audio Effect for more information.

- The selector string for the parameter that you want to get:
  - To get the number of input samples per frame, specify NVAFX_PARAM_NUM_INPUT_SAMPLES_PER_FRAME.
  - To get the number of output samples per frame, specify NVAFX_PARAM_NUM_OUTPUT_SAMPLES_PER_FRAME.
  - To get the number of input audio channels, specify NVAFX_PARAM_NUM_INPUT_CHANNELS.
  - To get the number of output audio channels, specify NVAFX_PARAM_NUM_OUTPUT_CHANNELS.
  - To get the input sample rate, specify NVAFX_PARAM_INPUT_SAMPLE_RATE.
  - To get the output sample rate, specify NVAFX_PARAM_OUTPUT_SAMPLE_RATE.
- A pointer to a location where to store the value that you want to get.

This example gets the number of samples per frame, number of I/O channels, and sample rate for an audio effect.

```c
unsigned num_input_samples_per_frame, num_output_samples_per_frame,
    num_input_channels, num_output_channels, input_sample_rate, output_sample_rate;
NvAFX_Status err;
err = NvAFX_GetU32(handle, NVAFX_PARAM_NUM_INPUT_SAMPLES_PER_FRAME, &num_input_samples_per_frame);
err = NvAFX_GetU32(handle, NVAFX_PARAM_NUM_OUTPUT_SAMPLES_PER_FRAME, &num_output_samples_per_frame);
err = NvAFX_GetU32(handle, NVAFX_PARAM_NUM_INPUT_CHANNELS, &num_input_channels);
err = NvAFX_GetU32(handle, NVAFX_PARAM_NUM_OUTPUT_CHANNELS, &num_output_channels);
err = NvAFX_GetU32(handle, NVAFX_PARAM_INPUT_SAMPLE_RATE, &input_sample_rate);
err = NvAFX_GetU32(handle, NVAFX_PARAM_OUTPUT_SAMPLE_RATE, &output_sample_rate);
```

3.7. Getting Supported Devices

You can call the NvAFX_GetSupportedDevices() function to fetch the number of supported GPUs, by model.

Note: This method must be called after you set the model path.

Here is a list of the parameters:

- The effect handle that was created.
See Creating an Audio Effect for more information.

- The size of the input array.
  - If the call succeeds, this value will be set by the function.
- Array of size num.

The function will fill the array with the CUDA device indices of devices that are supported by the model, in descending order of preference, where the first device is the most preferred device.

This example fetches the list of supported GPUs by the model:

```c
int numSupportedDevices = 0;
NvAFX_GetSupportedDevices(handle, &numSupportedDevices, nullptr);
std::vector<int> ret(num);
NvAFX_GetSupportedDevices(handle, &numSupportedDevices, ret.data());
```

### 3.8. Loading an Audio Effect

Loading an effect selects and loads a model and validates the parameters that were set for the effect.

To load an audio effect, call the `NvAFX_Load()` function and specify the effect handle that was created. See Creating an Audio Effect for more information.

```c
NvAFX_Status err = NvAFX_Load(handle);
```

### 3.9. Running an Audio Effect

After loading an audio effect, run the effect to apply the desired effect. After an effect is run, the contents of the input memory buffer are read, the audio effect is applied, and the output is written to the output memory buffer.

To run an audio effect, call the `NvAFX_Run()` function and pass the following information as parameters:

- The effect handle that was created.
  - See Creating an Audio Effect for more information.
- The input memory buffer to be read.
- The output memory buffer to which the information will be written.
- The number of samples per frame that were obtained.
  - See Getting the Parameters of an Effect for more information.
- The number of I/O audio channels that were obtained.
  - See Getting the Parameters of an Effect for more information.

This example runs an audio effect:

```c
NvAFX_Status err = NvAFX_Run(effect, input, output, num_samples, num_channels);
```
3.10. Destroying an Audio Effect

When an audio effect is no longer required, destroy it to free the resources and the memory that were allocated for the effect.

To destroy an audio effect, call the \texttt{NvAFX\_DestroyEffect()} function and specify the effect handle that was created. See \hyperref[Creating an Audio Effect]{Creating an Audio Effect} for more information.

\begin{verbatim}
NvAFXObject err = NvAFX\_DestroyEffect(handle);
\end{verbatim}

3.11. Use Multiple GPUs

Applications that are developed with the Audio Effects SDK can be used with multiple GPUs. By default, the SDK assumes that the application will set the GPU. The SDK can optionally select the best GPU to run the effect(s).

3.11.1. Selecting a GPU for Audio Effects Processing in a Multi-GPU Environment

The GPU that will be used to run audio effect(s) in a multi-GPU environment can be controlled by using the \texttt{cudaSetDevice()} and \texttt{cudaGetDevice()} CUDA functions.

\begin{itemize}
  \item \textbf{Note:} The device should be set before calling \texttt{NvAFXObject}.\texttt{Load()}.
\end{itemize}

The SDK determines the GPU selection based on the compute capability of the currently selected GPU:

\begin{itemize}
  \item If the currently selected GPU supports the SDK, the effect is loaded.
  \item If the currently selected GPU does not support the SDK, \texttt{NvAFXObject}\texttt{Load()} fails.
\end{itemize}

\begin{verbatim}
int chosenGPU = 0; // or whatever GPU you want to use
cudaSetDevice(chosenGPU);
NvAFXObject effect;
err = NvAFXObject\_CreateEffect(code, &effect);
err = NvAFXObject\_Set...; // set parameters
... err = NvAFXObject\_Load(effect);
... err = NvAFXObject\_Run(effect, ...);
\end{verbatim}
3.11.2. Offloading GPU Selection to the SDK for Audio Effects Processing in a Multi-GPU Environment

You can use the Audio Effects `NvAFX_SetU32(effect, NVAFX_PARAM_USE_DEFAULT_GPU, 1)` set function to allow the SDK to select the GPU on which to run the audio effect(s).

The `NvAFX_SetU32(NVAFX_PARAM_USE_DEFAULT_GPU)` call is optional and should be called only once before an effect is loaded. If this call is invoked after an audio effect is loaded, it will not have any effect.

If the application sets `NVAFX_PARAM_USE_DEFAULT_GPU` to 1, we assume that the application will not call `cudaSetDevice()`, and all other effects or multiple instances of an effect will use the default GPU selection. If the application calls `cudaSetDevice()` explicitly before `NvAFX_Load()`, the SDK will override the application’s device preference. If the client calls `cudaSetDevice()` to select a different GPU before calling `NvAFX_Run()`, the call will fail.

```c
NvAFX_Handle effect;
err = NvAFX_API NvAFX_CreateEffect(code, &effect);
err = Nvafx_API SetU32(effect, NVAFX_PARAM_USE_DEFAULT_GPU, 1);
...
err = Nvafx_API Nvafx_Load(effect);
...
```

3.12. Selecting Different GPUs for Different Tasks

In addition to applying the audio effects, if your application is designed to complete multiple tasks in a multi-GPU environment, you need to select the best GPU for each task before calling `NvAFX_Load()`.

**Note:** For performance concerns, switching to the appropriate GPU is the responsibility of the application.

If the application does not switch to the appropriate GPU before calling `NvFX_Run()`, the call will fail.

1. To determine the number of GPUs in your environment, call `cudaGetDeviceCount()`.
   ```c
   // Get the number of GPUs
   cuErr = cudaGetDeviceCount(&deviceCount);
   ```
2. To get the properties of each GPU and determine which GPU is the best GPU for each task, complete the following operations for each GPU in a loop:
   a. To set the current GPU, call `cudaSetDevice()`.
   b. To get the properties of the current GPU, call `cudaGetDeviceProperties()`.
c). To determine whether the GPU is the best GPU for each specific task, use the custom code in your application to analyze the properties that are retrieved by `cudaGetDeviceProperties()`. This example uses the compute capability, which determines whether a GPU’s properties should be analyzed and determines whether the GPU is the best GPU to apply an audio effect filter.

```c
// Loop through the GPUs to get the properties of each GPU and
// determine if it is the best GPU for each task based on the
// properties obtained.
for (int dev = 0; dev < deviceCount; ++dev) {
    cudaSetDevice(dev);
    cudaGetDeviceProperties(&deviceProp, dev);
    if (DeviceIsBestForAFX(&deviceProp))  gpuAFX = dev; // say 7.5 compute
    if (DeviceIsBestForOtherTask(&deviceProp)) gpuOtherTask = dev;
    ...
}
cudaSetDevice(gpuAFX);
err = NvAFX_Set...; // set parameters
err = NvAFX_Load(effect, ...);
```

3. To select the GPU for the task before performing the task, in the loop to complete the application’s tasks, call `cudaSetDevice()`.

```c
// Select the best GPU for each task and perform the task.
while (!done) {
    ...
    cudaSetDevice(gpuOtherTask);
    PerformOtherTask();
    cudaSetDevice(gpuAFX);
    err = NvAFX_Run(eff, ...);
```
Chapter 4. Audio Effects SDK for Windows API Reference

This section provides the details about the APIs in the Audio Effects SDK for Windows.

4.1. Type Definitions

The Audio Effects SDK type definitions provide selector strings for the audio effect and the parameters of an audio effect.

4.1.1. NvAFX_EffectSelector

This type definition provides selector strings for the various audio effect types.

typedef const char* NvAFX_EffectSelector;

NVAFX_EFFECT_DENOISER: "denoiser"
    Denoiser audio effect.

NVAFX_EFFECT_DEREVERB: "dereverb"
    Dereverb audio effect.

NVAFX_EFFECT_DEREVERB_DENOISER: "dereverb+denoiser"
    Dereverb+Denoiser audio effect.

4.1.2. NvAFX_ParameterSelector

This type definition provides selector strings for the audio effects parameters.

typedef const char* NvAFX_ParameterSelector;

NVAFX_PARAM_MODEL_PATH: "model_path"
    A character string that specifies the path to the model file for the effect.

NVAFX_PARAM_INPUT_SAMPLE_RATE: "input_sample_rate"
    An unsigned integer that specifies the input audio sample rate for the effect. This parameter is immutable for an audio effect and can only be read.

NVAFX_PARAM_OUTPUT_SAMPLE_RATE: "output_sample_rate"
    An unsigned integer that specifies the output audio sample rate for the effect. This parameter is immutable for an audio effect and can only be read.

NVAFX_PARAM_NUM_INPUT_SAMPLES_PER_FRAME: "num_input_samples_per_frame"
    An unsigned integer that specifies the number of input samples per frame for the effect. This parameter is immutable for an audio effect and can only be read.
**NVAFX_PARAM_NUM_OUTPUT_SAMPLES_PER_FRAME:** "num_output_samples_per_frame"

An unsigned integer that specifies the number of output samples per frame for the effect. This parameter is immutable for an audio effect and can only be read.

**NVAFX_PARAM_NUM_INPUT_CHANNELS:** "num_input_channels"

An unsigned integer that specifies the number of input audio channels for the effect. This parameter is immutable for an audio effect and can only be read.

**NVAFX_PARAM_NUM_OUTPUT_CHANNELS:** "num_output_channels"

An unsigned integer that specifies the number of output audio channels for the effect. This parameter is immutable for an audio effect and can only be read.

**NVAFX_PARAM_INTENSITY_RATIO:** "intensity_ratio"

A float value that specifies the factor that ranges from 0.0 to 1.0. Setting the factor to 0.0 is identical to a pass through, and a value of 1.0 provides the maximum possible impact of the effect.

**NVAFX_PARAM_USER_CUDA_CONTEXT:** "user_cuda_context"

An unsigned integer value that allows SDK users to disable SDK internal context management. To disable internal context management, set this value to 1. Other users can ignore this parameter. Once the value has been set to 1, this value cannot be unset for the current session (unsigned int).

---

**Note:** NVAFX_PARAM_USE_DEFAULT_GPU and NVAFX_PARAM_USER_CUDA_CONTEXT cannot be used at the same time.

The following macros have been deprecated:

- NVAFX_PARAM_NUM_CHANNELS: "num_channels"
- NVAFX_PARAM_SAMPLE_RATE: "sample_rate"
- NVAFX_PARAM_NUM_SAMPLES_PER_FRAME: "num_samples_per_frame"

### 4.1.3. NvAFX_Handle

This structure represents the opaque handle that is associated with each instance of an audio effect. Most audio effect function calls include this handle as the first parameter.

```c
typedef void* NvAFX_Handle;
```

### 4.2. Functions

This section provides information about the functions in the Audio Effects SDK for Windows.

#### 4.2.1. NvAFX_GetEffectList

This function retrieves a list of supported audio effects.

```c
NvAFX_Status NvAFX_GetEffectList(
    int* num_effects,
    NvAFX_EffectSelector* effects[]
);
```
Parameters

num_effects [out]
Type: int*
Address of the buffer that contains the number of effects that are returned in the effects array.

effects [out]
Type: NvAFX_EffectSelector* []
Address to a list of effect selection strings that are supported by the SDK. The list is statically allocated by the API implementation, so the caller does not need to allocate. See NvAFX_EffectSelector for more information about the selection strings.

Return Value

NVAFX_STATUS_SUCCESS on success.

Remarks

This function retrieves the list of audio effects that are supported by the SDK. The selection strings for the Audio Effects SDK are populated in the effects out parameter. The number of available effects are written to the num_effects out parameter.

4.2.2. NvAFX_CreateEffect

This function creates an audio effect.

NvAFX_Status NvAFX_CreateEffect(
    NvAFX_EffectSelector code,
    NvAFX_Handle* effect
);

Parameters

code [in]
Type: NvAFX_EffectSelector
The selection string for the type of audio effect that will be created. See NvAFX_EffectSelector for more information about the allowed selection strings.

effect [out]
Type: NvAFX_Handle*
The location where to store the handle to the newly created audio effect instance.

Return Value

NVAFX_STATUS_SUCCESS on success.
Remarks
This function creates an instance of the specified type of audio effect and also writes a handle to the audio effect instance to the effect out parameter.

4.2.3. NvAFX_DestroyEffect
This function retrieves a list of supported audio effects.

```
NvAFX_Status NvAFX_DestroyEffect(
    NvAFX_Handle effect
);
```

Parameters

effect [in]
  Type: NvAFX_Handle
  The handle to the audio effect instance that will be destroyed.

Return Value
NVAFX_STATUS_SUCCESS on success.

Remarks
This function destroys the audio effect instance with the specified handle and frees resources and memory that were allocated to the instance.

4.2.4. NvAFX_SetString
This function retrieves a list of supported audio effects.

```
NvAFX_Status NvAFX_SetString(
    NvAFX_Handle effect, 
    NvAFX_ParameterSelector param_name, 
    const char* val
);
```

Parameters

effect [in]
  Type: NvAFX_Handle
  The handle to the audio effect instance for which you want to set the specified character string parameter.

param_name [in]
  Type: NvAFX_ParameterSelector
  The selector string NVAFX_PARAM_MODEL_PATH.
Any other selector string returns an error.

val [in]

Type: char*

Pointer to the character string to which you want to set the parameter.

Return Value

NVAFX_STATUS_SUCCESS on success.

Remarks

This function sets the value of the specified character string parameter for the specified audio effect to the val parameter.

4.2.5. NvAFX_SetU32

This function retrieves a list of supported audio effects.

```c
NVAFX_Status Nvafx_SetU32(
    NVAFX_Handle effect,  
    NVAFX_ParameterSelector param_name,  
    unsigned int val
);
```

Parameters

effect [in]

Type: NVAFX_Handle

The handle to the audio effect instance for which you want to set the specified character string parameter.

Param_name [in]

Type: NVAFX_ParameterSelector

The selector string NVAFX_PARAM_STATE_RATE.

Any other selector string returns an error.

val [in]

Type: unsigned int

Value to be set for the parameter.

Return Value

NVAFX_STATUS_SUCCESS on success.
Remarks
This function sets the value of the specified 32-bit unsigned integer parameter for the specified audio effect to the val parameter.

4.2.6. NvAFX_SetInt
This function helps set integer type parameters of the SDK.

NvAFX_Status NvFX_SetInt(
    NvAFX_Handle effect,
    NvFX_ParameterSelector param_name,
    int val
);

Parameters

 effect [in]
  Type: NvAFX_Handle
  The handle to the audio effect instance for which you want to set the specified integer parameter.

 param_name [in]
  Type: NvFX_ParameterSelector
  The selector string NVAFX_PARAM_INTENSITY_RATIO.
  Any other selector string returns an error.

 val [in]
  Type: int
  Value to be set for the parameter.

Return Value
NvFX_STATUS_SUCCESS on success.

Remarks
This function sets the value of the specified integer parameter for the specified audio effect to the val parameter.

4.2.7. NvAFX_GetString
This function gets the current value of the set string parameter of the specified effect.

NvFX_Status NvFX_GetString(
    NvFX_Handle effect,
    NvFX_ParameterSelector param_name,
    char* val,
    int max_length
);
Parameters

effect [in]

Type: NvAFX_Handle
The handle to the audio effect instance from which you want to get the specified character string parameter.

Param_name [in]

Type: NvAFX_ParameterSelector
The selector string NVAFX_PARAM_MODEL_PATH.
Any other selector string returns an error.

val [out]

Type: char*
The address of the buffer where the requested character string will be stored.

max_length [in]

Type: int
The length in bytes of the buffer that is specified by the val parameter.

Return Value

NVAFX_STATUS_SUCCESS on success.

Remarks

This function gets the value of the character string parameter for the specified audio effect and writes the retrieved string to the buffer at the location specified by the val parameter.

4.2.8. NvAFX_GetU32

This function gets the value of a uint parameter of the specified effect.

NvAFX_Status NvAFX_GetU32(
    NvAFX_Handle effect,
    NvAFX_ParameterSelector param_name,
    unsigned int* val
);
**param_name [in]**

**Type:** NvAFX_ParameterSelector

One of the following selector strings for the specified 32-bit unsigned integer parameter that you want to get:

- NVAFX_PARAM_NUM_SAMPLES_PER_FRAME
- NVAFX_PARAM_NUM_CHANNELS
- NVAFX_PARAM_SAMPLE_RATE

Any other selector string returns an error.

**val [out]**

**Type:** unsigned int*

The address of the buffer in which the retrieved 32-bit unsigned integer parameter value will be written.

**Return Value**

NVAFX_STATUS_SUCCESS on success.

**Remarks**

This function gets the value of the specified 32-bit unsigned integer parameter for the specified audio effect and writes the retrieved value to the buffer that is specified by the val parameter.

### 4.2.9. NvAFX_GetFloat

This function helps get float type parameters of the SDK.

```c
NvAFX_Status NvAFX_GetFloat(
    NvFX_Handle effect,
    NvAFX_ParameterSelector param_name,
    float* val
);
```

**Parameters**

**effect [in]**

**Type:** NvFX_Handle

The handle to the audio effect instance from which you want to get the specified float parameter.
**param_name [in]**

_Type: NvAFX_ParameterSelector_

One of the following selector strings for the specified float parameter that you want to get:

_NVAFX_PARAM_INTENSITY_RATIO_

Any other selector string returns an error.

**val [out]**

_Type: float*_

The address of the buffer in which the retrieved float parameter value will be written.

**Return Value**

_NVAFX_STATUS_SUCCESS_ on success.

**Remarks**

This function gets the value of the specified float parameter for the specified audio effect and writes the retrieved value to the buffer that is specified by the val parameter.

### 4.2.10. NvAFX_GetSupportedDevices

The function gets a list of compatible devices that are supported by the supplied model file.

_NvAFX_Status NvAFX_GetSupportedDevices(
    NvAFX_Handle effect,
    int *num,
    int *devices
);

**Parameters**

**effect [in]**

_Type: NvAFX_Handle_

The handle to the audio effect instance to load.

**num [in, out]**

_Type: int*_

The size of the input array. If the call succeeds, this value will be set by the function.

**devices [in, out]**

_Type: int*_

Array of size num. The function will fill the array with CUDA device indices of devices supported by the model, in descending order of preference, where the first device is the most preferred device.
Return Value

NVAFX_STATUS_SUCCESS on success.

Remarks

This function gets the devices supported by the model.

4.2.11. NvAFX_Load

This function helps load the programmed model file by using NvAFX_SetString() to the GPU to prepare for processing via the NvAFX_Run() call.

```
NVAFX_Status NvAFX_Load(
    NVAFX_Handle effect
);
```

Parameters

effect [in]

Type: NVAFX_Handle

The handle to the audio effect instance to load.

Return Value

NVAFX_STATUS_SUCCESS on success.

Remarks

This function loads the specified audio effect and validates the parameters that are set for the effect.

4.2.12. NvAFX_Run

This function applies the effect on the audio buffer and returns the processed buffer. This is a synchronous call.

```
NVAFX_Status NvAFX_Run(
    NVafx_Handle effect,
    const float** input,
    float** output,
    unsigned num_samples,
    unsigned num_channels
);
```

4.2.13. NvAFX_Reset

This function resets the algorithm state and helps flush the internal history. Every new denoising session can use this API instead of creating a new effect via NvAFX_CreateEffect() for each session.

```
NVAFX_Status NvAFX_Reset(
    NVAFX_Handle effect
)
```
Parameters

effect [in]

Type: NvAFX_Handle

The handle to the audio effect instance to run.

Return Value

NVAFX_STATUS_SUCCESS on success.

Remarks

Allows the state of an effect to be reset. This operation will reset the state selected in the next NvAFX_Run call.

4.3. Run Codes

The NVAFX_Status enumeration defines the following values that the Audio Effects functions might return to indicate error or success:

NVAFX_STATUS_SUCCESS
   Successful execution.

NVAFX_STATUS_FAILED
   Generic error code, which indicates that the function failed to execute for an unspecified reason.

NVAFX_STATUS_INVALID_HANDLE
   An invalid effect handle has been supplied.

NVAFX_STATUS_INVALID_PARAM
   An invalid parameter value has been supplied for this combination of effect and selector string.

NVAFX_STATUS_IMMUTABLE_PARAM
   User tried to modify an immutable parameter.

NVAFX_STATUS_INSUFFICIENT_DATA
   There is insufficient data to process.

NVAFX_STATUS_EFFECT_NOT_AVAILABLE
   The specified effect is not supported.

NVAFX_STATUS_OUTPUT_BUFFER_TOO_SMALL
   The output buffer length is too small to hold the requested data.

NVAFX_STATUS_MODEL_LOAD_FAILED
   The specified model file cannot be loaded.

NVAFX_STATUS_GPU_UNSUPPORTED
   The GPU is unsupported. Audio effects SDK requires Turing or later GPU with Tensor cores.

NVAFX_STATUS_GPU_UNSUPPORTED
   The selected GPU is not supported. The SDK requires a Turing and above GPU with Tensor cores.
**NVAFX_STATUS_NO_SUPPORTED_GPU_FOUND**  
No supported GPU found on the system.

**NVAFX_STATUS_WRONG_GPU**  
Current GPU is not the one selected.
Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. 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 document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information 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 ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

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 this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform 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 document. NVIDIA accepts no 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 document 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 document. 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 document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND 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. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 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 Terms 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.

ARM

ARM, AMBA and ARM Powered are registered trademarks of ARM Limited. Cortex, MPCore and Mali are trademarks of ARM Limited. All other brands or product names are the property of their respective holders. "ARM" is used to represent ARM Holdings plc; its operating company ARM Limited; and the regional subsidiaries ARM Inc.; ARM KK; ARM Korea Limited; ARM Taiwan Limited; ARM France SAS; ARM Consulting (Shanghai) Co. Ltd.; ARM Germany GmbH; ARM Embedded Technologies Pvt. Ltd.; ARM Norway, ÅS and ARM Sweden AB.

OpenCL

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