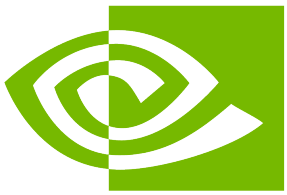

NVIDIA Audio Effects (AFX) SDK User Guide

Release 3.0.0

NVIDIA Corporation

Sep 10, 2026

Contents

Download this guide as a PDF

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

- *Audio Super-Resolution*: This effect improves the sound quality by adding higher-frequency content to the audio stream. For low-frequency audio, this feature predicts the higher frequency spectrum of input audio, which improves audio quality.
- *Noise Removal/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 such background noise from audio.
- *Noise Removal and Room Echo Removal/Denoise and Dereverb*: The effect combines both the above effects to remove/suppress both noise and reverberations from audio. This offers much better performance than applying these effects separately.
- *Room Echo Removal/Dereverb/Room Echo Cancellation*: Recordings of speech might contain reverberations from the recording environment, affecting speech clarity. The dereverb effect helps remove or suppress such reverberations from audio.
- *Speaker Focus*: This effect identifies and isolates the prominent speaker by removing all background speakers from the input audio, improving the intelligibility of the primary speaker.
- *Studio Voice*: Studio Voice enhances/recovers degraded speech recorded using low-end microphones in less-than-ideal acoustic environments. The resulting audio should perceptually sound better than the original, removing/reducing degradation and artifacts to make the audio sound more like a recording in a professional studio setup.

Note

The Acoustic Echo Cancellation (AEC) effect and Voice Font effect have been depreciated.

Note

The Windows SDK supports x64 systems and NVIDIA N1X systems running Windows on Arm (ARM64). It is optimized for client-side application integration. The Linux SDK is designed and optimized for server-side (datacenter or cloud) deployments. Using these SDKs outside these use cases is not officially supported.

Chapter 1. Get Started on Windows

The Audio Effects SDK requires specific GPUs, a specific version of the Windows OS, and other software dependencies.

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, or deployment of this SDK in a datacenter or cloud environment.

1.1. Hardware Requirements

The Audio Effects SDK is supported on NVIDIA GPUs with Tensor Cores on the following Windows architectures:

- x64 systems with a supported discrete NVIDIA GPU.
- NVIDIA N1X systems running Windows on Arm (ARM64).

1.2. Software Requirements

The NVIDIA CUDA® and inference runtime dependencies are bundled with the SDK: TensorRT™ for x64 and TensorRT-RTX for ARM64. (See *Install the AFX SDK*.)

Software	Required Version
Windows OS	64-bit Windows 10 or Windows 11
Microsoft Visual Studio	2017 (MSVC15.0) or later
CMake	3.9 or later
NVIDIA Graphics Driver for Windows, x64	572.61 or later
NVIDIA Graphics Driver for Windows, ARM64 (NVIDIA N1X)	615.15 or later

Note

All libraries that are required to use the SDK are in the package, under `external`, and do not need to be separately installed.

1.2.1. Install the AFX SDK

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

- A core package that contains the API implementation along with dependencies and scripts for downloading AI features and sample applications.
- AI features, which you can download from NGC by using the download script.
- A sample application, hosted on GitHub, that demonstrates how to use the SDK APIs.

To develop applications with the Audio Effects SDK, you must download the core SDK package, the AI features, and the sample application. Provide the path to the package by setting the environment variable `AFX_SDK_ROOT` before compilation and linking. Your app will use the SDK functions that are exposed by the SDK header and feature headers and dynamically link against the provided libraries.

1.2.1.1 Install the Core Windows SDK

To install the SDK, extract the contents of the Audio Effects SDK `.zip` file to the required location on your computer.

Before using the SDK, you must download and install AI features.

For an offline target, download the required features on a connected machine. Place the features under the extracted package's `features` directory, and transfer the complete package as one archive. Extract the archive without changing its directory structure so that the core headers, libraries, runtime files, public samples, and downloaded feature roots remain together.

1.2.1.2 Obtain Windows SDK Features

You can download SDK AI features from the NGC website or by using the provided download script. Remember to unzip the AI features before use.

> **Note**
>
> If the target machine cannot access NGC because of firewall or airgap rules, you can download features to another machine and transfer them to the target machine. The machine used for downloading features does not require GPUs.

To download features using the NGC website:

1. Refer to the NGC SDK collection for the required feature (under `Entities` > `Models`).
2. Download the files corresponding to the target machine's GPU architecture, SDK version, and sample rate along with library and header files. For example, if the SDK version is `3.0.0`, the 16-kHz effect is required. If the target GPU architecture is Ampere, download the features for version `3.0.0-16k-ampere`.

To download AI features using the download script:

1. Obtain an NGC API key from the NGC website. For details, refer to the NGC User Guide. The download script prefers the `NGC_CLI_API_KEY` environment variable and also accepts the legacy `NGC_API_KEY` name.
2. Obtain the NGC organization and team from the SDK collection URL. In a URL containing `/orgs/<organization>/teams/<team>/`, use those exact values for `-NGC_Org` and `-NGC_Team`. The following public-release examples use these values:
    - NGC organization is `nvidia`.

- NGC team is `maxine`.

3. Unzip the SDK and navigate to the `features` folder.
4. Download the features using the following commands:

```
# Set the NGC API key. Do not place the literal key in scripts or source
 ↪control.

    $env:NGC_CLI_API_KEY = "<your NGC key>"

# Use the organization and team from the SDK collection URL.
# Replace these public-release example values for a prerelease collection.

$ngcOrg = "nvidia"
$ngcTeam = "maxine"

# Download features

# Syntax: powershell -ExecutionPolicy Bypass -File ./download_features.
 ↪ps1 [-GPU_Architecture <turing|ampere|ada|blackwell>] [-NGC_Org <NGC
 ↪org>] [-NGC_Team <NGC team>] [-Effects effect1,effect2,effect3] [-
 ↪Version <X.Y.Z>] [-Output_Directory <path>]
# -Effects specifies the list of effects to download; if omitted,
 ↪downloads all non-Early-Access effects.

# For example, the following command downloads the features for Ampere
 ↪GPU Architecture to features (under the ampere folder) from NGC org
 ↪"nvidia" and team "maxine" (excluding Early Access features):

powershell -ExecutionPolicy Bypass -File ./download_features.ps1 -GPU_
 ↪Architecture ampere -NGC_Org $ngcOrg -NGC_Team $ngcTeam -Version 3.0.0 -
 ↪Output_Directory .

# If -GPU_Architecture is omitted, the x64 script detects the current GPU.
 ↪ To download Early Access features, explicitly name them:

powershell -ExecutionPolicy Bypass -File ./download_features.ps1 -NGC_Org
 ↪$ngcOrg -NGC_Team $ngcTeam -Effects "speaker_focus-16k,speaker_focus-48k
 ↪"

# To download only the Superresolution and Dereverb features from
 ↪organization 'nvidia' and team 'maxine' for Turing architecture, use
 ↪the following command:

powershell -ExecutionPolicy Bypass -File ./download_features.ps1 -GPU_
 ↪Architecture turing -NGC_Org $ngcOrg -NGC_Team $ngcTeam -Effects
 ↪superres-16k_to_48k,superres-8k_to_16k,dereverb-16k,dereverb-48k

# Discover supported features without downloading payloads.
powershell -ExecutionPolicy Bypass -File ./download_features.ps1 -list_
 ↪features -NGC_Org $ngcOrg -NGC_Team $ngcTeam

# List published versions for selected effects without downloading
```

(continues on next page)

(continued from previous page)

```
↪payloads.
powershell -ExecutionPolicy Bypass -File ./download_features.ps1 -list_
↪versions -Effects denoiser-16k,denoiser-48k -NGC_Org $ngcOrg -NGC_Team
↪$ngcTeam

# Show the complete CLI contract.
powershell -ExecutionPolicy Bypass -File ./download_features.ps1 -Help
```

`-Output_Directory` selects the installation root. `-Version` pins all requested effects; when omitted, the script resolves and records the latest published version for each effect. `-list_features` and `-list_versions` are discovery-only modes. The downloader stages the complete request under a contained transaction directory, validates NGC MD5 sidecars, records SHA-256 file hashes and resolved artifact identifiers in `features-download-manifest.json`, and atomically promotes the feature folders. Existing feature folders remain byte-for-byte unchanged if any requested artifact fails. Transient network failures use bounded exponential backoff; HTTP 401, 402, and 403 fail immediately. API keys are read only from `NGC_CLI_API_KEY` or the legacy `NGC_API_KEY` and are never written to the manifest or output.

On NVIDIA N1X (ARM64), Blackwell is the only supported architecture. The ARM64 package defaults to Blackwell when `-GPU_Architecture` is omitted and installs models under each feature's `models\blackwell` directory. The following native PowerShell parameter syntax is valid on both x64 and ARM64:

```
powershell -ExecutionPolicy Bypass -File ./download_features.ps1 -GPU_
↪Architecture blackwell -NGC_Org $ngcOrg -NGC_Team $ngcTeam
```

1.2.2. Build the Sample Application

The sample application demonstrates the SDK APIs and is hosted on GitHub.

This sample includes the source file `effects_demo.cpp`, which you can compile and run along with CMake scripts to automatically find the required dependencies.

The release package includes the sample tree under `samples`. If you receive a core-only package, acquire the same release-controlled sample revision by opening PowerShell at the extracted SDK root and running the following:

```
$sampleCommit = '73cbdb8d4e988d4873d4347923de48c179259694'
.\samples\download_samples.ps1 -Destination .\samples -Ref $sampleCommit
```

The downloader fetches that exact commit and writes `samples\samples-download-manifest.json` with the source URL, resolved commit, file sizes, and SHA-256 hashes. After validation, the downloader promotes the complete tree. A repeated identical acquisition verifies and skips identical files. Destination-only files are preserved and reported. A conflicting file causes no changes and returns nonzero; `-Force` replaces only the listed conflicting sample files and reports every replacement. Fetch failures use bounded backoff, and failed staging directories are removed.

Successful output names the resolved commit, installed-file count, destination, and manifest path. You can also manually fetch the same immutable commit with GitHub Desktop or Git, but do not build an unpinned default branch for an R22 release.

If the sample application and core package reside in different locations, you must set the `AFX_SDK_ROOT` environment variable to the path of the core package.

1.2.2.1 Command-Line Quick Start (x64 and N1X ARM64)

Open PowerShell in the extracted SDK package's `samples` directory. The following commands use only paths inside that package to configure an x64 Visual Studio build, build the Release binary, and verify its output:

```
$env:AFX_SDK_ROOT = (Resolve-Path ..).Path
cmake -S . -B build -A x64
cmake --build build --config Release
if (-not (Test-Path .\build\Release\effects_demo.exe)) { throw 'effects_demo.
↪exe was not built' }
```

For an NVIDIA N1X ARM64 package, use its `samples` directory and run the following commands:

```
$env:AFX_SDK_ROOT = (Resolve-Path ..).Path
cmake -S . -B build-arm64 -A ARM64
cmake --build build-arm64 --config Release
if (-not (Test-Path .\build-arm64\Release\effects_demo.exe)) { throw 'effects_
↪demo.exe was not built' }
```

Run helpers from the build-produced `Release\scripts` directory so that the helper and executable stay together. For example, after the N1X build:

```
Set-Location .\build-arm64\Release\scripts
.\run_effects_demo.bat -e denoiser -isr 16k -osr 16k -ir 1.0 -ev 2 -vad 1
```

The copies under `samples\apps\effects_demo\scripts` are build inputs and must not be run directly. Run only the generated helpers in the build-produced `Release\scripts` directory.

Do not use ARM64-only headers or selectors when building against the x64 package.

1.2.2.2 CMake GUI Workflow

To build the sample application, follow these steps:

1. Start the CMake GUI and specify the source folder and a build folder for the binary files.
    1. For the source folder, ensure that the path ends in `package`.
    2. For the build folder, ensure that the path ends in `package/build`.
2. Use CMake to configure and generate the Visual Studio solution file.
    1. Click **Configure**.
    2. When prompted to confirm whether CMake can create the build folder, click **OK**.
    3. Select **Visual Studio** for the generator. Select **x64** for an x64 package or **ARM64** for an NVIDIA N1X package.
    4. To finish configuring the Visual Studio solution file, click **Finish**.
    5. 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.
    1. In CMake, click **Open Project** to open Visual Studio.
    2. In Visual Studio, select **Build** > **Build Solution**.

1.2.3. Run the Sample Application

To run the application directly from a Command Prompt window, use `.\effects_demo.exe -c <config-file>` and replace `<config-file>` with the path of an existing effect configuration file.

`<config-file>` specifies the path of an existing effect configuration file. The helper scripts create temporary configuration files in the sample application directory, run `effects_demo.exe`, and remove those files when processing completes.

Configurations for effects can be generated on the fly using `run_effects_demo.bat`. For more information, see `samples\README.txt`. After building, open `samples\build\Release\scripts` for x64 or `samples\build-arm64\Release\scripts` for NVIDIA N1X and run the helper from that build-produced directory.

The sample application also includes `run_denoiser_48k.bat` for the 48-kHz denoiser model. The following example generates its configuration and runs the `effects_demo.exe` sample application:

```
.\run_denoiser_48k.bat
```

The helper detects the GPU architecture on x64 and selects Blackwell on NVIDIA N1X. Use `-g` only to override detection on x64.

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

`effect <effect>`

Specifies the effect that will be applied, such as `denoiser`. For a complete list of supported effects, refer to the *introduction*.

`effect_version <version>` (Windows only)

Specifies the effect version for the denoiser. 1 executes `BNR 1.0` and 2 executes `BNR 2.0`. Any other value is rejected with a nonzero exit code. Other effects use version 1 and reject an explicit non-1 value.

`input_sample_rate <sample-rate>`

Specifies the input sample rate for the model file that will be used in the sample application; for example, `16000` or `48000`. The value must match both the loaded model's input rate and the input WAV rate.

`output_sample_rate <sample-rate>`

Specifies the output sample rate for the model file that will be used in the sample application; for example, `16000` or `48000`. The value must match the loaded model's output rate.

On Windows x64 and NVIDIA N1X, `effects_demo` compares the configured rates with the loaded model before creating output. It also verifies the input WAV rate. A mismatch returns a nonzero exit code and does not produce an output WAV.

`model <model-file>`

Specifies the path of the model file that will be used in the sample application fetch from the core SDK package; for example, `denoiser_48k.trtpkg`.

> **Note**
>
> In the previous version of the SDK, the models were in the `bin/models` folder. In the core SDK package, the models are now in `features/<feature>/models`.

On NVIDIA N1X (ARM64), models are under `features/<feature>/models/blackwell`. The helper scripts select this directory automatically, so omit `-g`.

`mic_profile_model <model-file>`

(Studio Voice Low Latency only) Specifies the path to the microphone-profile companion model. This entry is required when `mic_profile_id` is 0 through 5.

`mic_profile_id <profile-id>`

(Studio Voice Low Latency only) Selects a microphone profile. The default, `-1`, disables microphone profiling. Values 0 through 5 select the profiles described in *About the Studio Voice Effect*.

Note

In the previous version of the SDK, the models were in the `bin/models` folder. In the core SDK package, the models are now in `features/<feature>/models`.

`input_wav <input-audio-file>`

Specifies the path of the noisy input audio `.wav` file to use, for example, `noisy_48k.wav`. The file should contain mono channel audio in signed 16-bit or 32-bit float format with a basic WAV header

Note

The sample inputs (present in folder `samples/effects_demo` in previous versions of the SDK) have been moved to `samples/effects_demo/input_files`.

Note

Sample input audio files are 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`.

Note

Only the .wav file format is supported.

`intensity_ratio <intensity-ratio>`

Specifies the effect 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 of input audio.

`real_time <enable>`

Simulates real-time audio input, set to 1 to enable or 0 to disable (disabled by default). When this option is enabled each audio frame is passed to the SDK with a delay of 10 ms, similar to how audio is received from a physical device or stream.

`enable_vad <enable>`

Specifies whether to enable the Voice Activity Detection (VAD) algorithm:

- 1 to enable
- 0 to disable

Any other value is rejected with a nonzero exit code. The default is 1 for denoiser version 2 at 16 kHz and 0 otherwise; an explicitly supplied 0 is honored. When this option is enabled, the sample application passes each audio frame to the VAD algorithm to check voice activity, and zeros out the frames that do not have any activity.

1.2.3.1 Use the Helper Script to Run the Sample Application

`run_effects_demo.bat` is a Windows batch file that can be used to run the sample application for various effects. This script generates a config file for the specified effect, the GPU, and sample inputs for that effect and runs `effects_demo.exe` on the sample files.

If the effect is to be applied to custom input files, the input files can be placed in the input sample folder that corresponds to the effect/sample rate. When you run the helper script, the effect is applied on the inputs, and the processed audio outputs will be placed in the output folder that corresponds to the effect/output sample rate.

For example, to apply the Background Noise Removal (Denoiser effect) on custom 48-kHz files, copy the files to `input_files/denoiser/48k` and run `.\run_effects_demo.bat`. Processed outputs will be generated in `output_files/denoiser/48k`. Refer to `samples\README.txt` for further details.

The full command syntax is `.\run_effects_demo.bat -g <architecture> -e <effect> -isr <input_sr> -osr <output_sr> -ir <intensity_ratio> -ev <effect_version> -vad <enable_vad>`. Replace each bracketed value with one of the following supported values:

- `architecture`: GPU architecture. Supported values are `turing`, `ampere`, `ada`, and `blackwell`. On NVIDIA N1X (ARM64), omit `-g`; the script selects the `blackwell` Windows on Arm model directory automatically.
- `effect`: Effect to be applied. Some of the supported values are `denoiser`, `dereverb`, `dereverb_denoiser`, and `superres`.
- `input_sr`: Input sample rate for the effect. Supported values are 8k, 16k, and 48k.
- `output_sr`: Output sample rate for the effect. Supported values are 16k and 48k.
- `effect_version`: Denoiser version 1 or 2. Other values are rejected instead of being silently normalized.
- `enable_vad`: 0 or 1. Other values are rejected. Omit this option to use the documented default.

The same argument and sample-rate validation applies to x64 and NVIDIA N1X.

For example, to run the 16-kHz Dereverb effect on the current GPU, use the following command:

```
.\run_effects_demo.bat -e dereverb -isr 16k -osr 16k
```

To run Studio Voice Low Latency with the Warm microphone profile on NVIDIA N1X, use the following command:

```
.\run_effects_demo.bat -e studio_voice_low_latency -isr 48k -osr 48k -mp 5
```

1.2.4. Chain Multiple Effects

The `effects_demo` sample application also provides config files in which multiple effects are run in a chain. For more information, see *Create a Chained Audio Effect*.

For example, the following command runs the 16-kHz Denoiser effect followed by the 16-kHz to 48-kHz Superresolution effect on input audio. It auto-detects the GPU on x64 and selects Blackwell on NVIDIA N1X:

```
.\run_effects_demo.bat -e denoiser -isr 16k -osr 16k -e2 superres -isr2 16k -
↪osr2 48k
```

To run effects in a chain, the configuration file uses syntax similar to the syntax used when running single effects, with the following changes:

`effect <effect>`

Specifies the effects in sequence to be used for chaining. For more information about possible chaining combinations, see *Create a Chained Audio Effect*.

> **Note**
>
> Chaining effects only support combinations of the Superres and the Denoiser/Dereverb/Combined Denoiser+Dereverb effect and Denoiser + Speaker Focus. Other effect chains are *not* supported. If combining the Denoiser effect and Dereverb effect, use the combined Denoiser+Dereverb model. For more information, see *About the Noise Removal/Background Noise Suppression Effect*.

`model <model-file-1,model-file-2>`

Specifies the path of the model files in sequence that will be used in the sample application, for example, `superres_16kto48k.trtpkg,denoiser_48k.trtpkg`.

`intensity_ratio <intensity-ratio-1,intensity-ratio-2>`

Specifies the intensity ratio for the effects. 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 of input audio.

1.2.4.1 Use the Helper Script to Chain Multiple Effects

Users can modify the sample config/batch files provided with the SDK and use these files with `effects_demo.exe`. If the effect is to be applied on custom input files, the files can be placed in the input sample folder that corresponds to the effect/sample rate. Running the helper script will apply the effect on these inputs, and the processed audio outputs will be placed in the output folder corresponding to the effect/output sample rate.

For example, to apply the Background Noise Removal effect on custom 48kHz files, copy the files to `input_files/denoiser/48k` and run `.\run_effects_demo.bat`. The processed outputs will be generated in `output_files/denoiser/48k`.

To apply the Background Noise Removal (Denoiser effect) + Superres effect on 16k files, copy the files in `input_files/chaining/denoiser/16k` folder and use `.\run_effects_demo.bat`. The outputs will be generated in `output_files/chaining/denoiser16k_superres16kto48k`.

The full helper syntax is `.\run_effects_demo.bat [-g <architecture>] -e <effect_1> -isr <in_sr_1> -osr <out_sr_1> -e2 <effect_2> -isr2 <in_sr_2> -osr2 <out_sr_2>`. Replace each bracketed value with one of the following values. Run the script from the build-produced `Release\scripts` directory described in *Build the Sample Application*.

- `architecture`: GPU Supported Architecture. Supported values are `turing`, `ampere`, `ada`, and `blackwell`. On NVIDIA N1X, omit `-g`; the helper selects `blackwell` automatically.
- `effect_1`: 1st Effect to be applied. Supported values are `denoiser`, `dereverb`, `dereverb_denoiser`, `superres`, and `speaker_focus`.
- `in_sr_1`: Input Sample Rate for 1st effect. Supported values are 8k, 16k, and 48k.
- `out_sr_1`: Output Sample Rate for 1st effect. Supported values are 16k and 48k.
- `effect_2`: 2nd Effect to be applied. Supported values are `denoiser`, `dereverb`, `dereverb_denoiser`, and `superres`.
- `in_sr_2`: Input Sample Rate for 2nd effect. Supported values are 8k, 16k, and 48k.
- `out_sr_2`: Output Sample Rate for 2nd effect. Supported values are 16k and 48k.

The first effect's output rate must equal the second effect's input rate. The configured rates must also match both loaded models, and the input WAV rate must match the first model's input rate. These checks are identical on x64 and NVIDIA N1X; a mismatch returns nonzero before an output WAV is created.

For next steps, refer to *Use the Audio Effects SDK in Applications*.

1.2.5. Windows Performance and Deployment Guide

Use the measurements on this page to validate an integration and to decide whether the Windows Audio Effects SDK matches the intended deployment. These numbers are reference measurements from one system, not guarantees for every GPU, driver, audio corpus, or application architecture.

1.2.5.1 R22 x64 Reference Measurement

The final R22 x64 package was measured on Windows with SDK version `3.0.0.25`, an NVIDIA RTX 3500 Ada Generation Laptop GPU, and the packaged 16-kHz mono sample audio. The single-effect workload used Denoiser v2 with 10-ms frames (160 samples) and processed five complete recordings per process. The representative chain used 16-kHz Denoiser followed by 16-kHz-to-48-kHz Super Resolution and processed two recordings per process. Each workload was started in a new process three times. These results measure separate process invocations; they do not represent a cold operating-system boot. The first process includes effect creation and model loading; later processes might benefit from operating-system, driver, and model caches.

Table 1: Final x64 Package Controlled Benchmark

Measurement	Denoiser v2	Denoiser + Super Resolution
Process invocations	3	3
Per-file NvAFX_Run averages	15	6
p50 NvAFX_Run time per 10-ms frame	0.374 ms	0.442 ms
p95 NvAFX_Run time per 10-ms frame	0.410 ms	0.479 ms
p50 processing throughput	26.74x real time	22.48x real time
p05 processing throughput (lower tail)	24.40x real time	20.90x real time
p50 / p95 system CPU utilization	46% / 68%	2% / 47%
p50 / p95 GPU utilization	11% / 68%	1% / 58%
p95 GPU memory use	215 MiB	181 MiB
First-process wall time	8.844 s	4.485 s
Repeated-process wall times	10.361 s and 10.112 s	5.832 s and 5.814 s

The release acceptance thresholds for both reference workloads are a p95 NvAFX_Run time below the 10-ms frame budget and p05 processing throughput above 1.0x real time. Throughput uses p05 because higher values are better; p05 represents the lower-performing tail. The measured x64 package passes both thresholds for both workloads. CPU utilization is system-wide Windows processor load. GPU utilization and memory are sampled from `nvidia-smi`. Command duration includes process startup, effect creation, model loading, all selected audio files, file I/O, and teardown; it is not frame latency.

1.2.5.2 Reproduce the x64 Measurement

Build the packaged sample for x64 Release, acquire the Ada Denoiser and Super Resolution features, and run `run_effects_demo.bat -g ada -e denoiser -isr 16k -osr 16k -ir 1.0 -ev 2 -vad 1` three times. Then run `run_effects_demo.bat -g ada -e denoiser -isr 16k -osr 16k -e2 superres -isr2 16k -osr2 48k` three times. Capture the `NvAFX API Call Timing Summary` emitted for each file and calculate p50/p95 from the per-file NvAFX_Run averages. Calculate p50/p05 from the real-time factors (audio duration divided by processing time). Report Windows system CPU load and `nvidia-smi` GPU utilization and memory samples while each command runs.

Open PowerShell at the extracted x64 SDK root and reproduce the process invocations described in the preceding paragraph. This block is x64-only; it contains no ARM64 commands:

```
$env:AFX_SDK_ROOT = (Resolve-Path .).Path
cmake -S .\samples -B .\samples\build -A x64
cmake --build .\samples\build --config Release
$helper = '.\samples\build\Release\scripts\run_effects_demo.bat'
1..3 | ForEach-Object { & $helper -g ada -e denoiser -isr 16k -osr 16k -ir 1.
↪0 -ev 2 -vad 1; if ($LASTEXITCODE -ne 0) { throw "Denoiser run $_ failed" }
↪}
1..3 | ForEach-Object { & $helper -g ada -e denoiser -isr 16k -osr 16k -e2
```

(continues on next page)

(continued from previous page)

```
↪superres -isr2 16k -osr2 48k; if ($LASTEXITCODE -ne 0) { throw "Chain run $_
↪ failed" } }
```

1.2.5.3 Reproduce the Measurement on NVIDIA N1X (ARM64)

On NVIDIA N1X, open PowerShell at the extracted Windows ARM64 SDK root, build the samples for ARM64, and run the build-produced helpers without `-g`. Five invocations of each workload provide enough repeated samples for percentile reporting while keeping model-load and per-frame timing separate:

```
$env:AFX_SDK_ROOT = (Resolve-Path .).Path
cmake -S .\samples -B .\samples\build-arm64 -A ARM64
cmake --build .\samples\build-arm64 --config Release
$helper = '.\samples\build-arm64\Release\scripts\run_effects_demo.bat'
1..5 | ForEach-Object { & $helper -e denoiser -isr 16k -osr 16k -ir 1.0 -ev 2
↪-vad 1; if ($LASTEXITCODE -ne 0) { throw "Denoiser run $_ failed" } }
1..5 | ForEach-Object { & $helper -e denoiser -isr 16k -osr 16k -e2 superres -
↪isr2 16k -osr2 48k; if ($LASTEXITCODE -ne 0) { throw "Chain run $_ failed" }
↪ }
```

Report p50/p95 frame latency, p50/p05 real-time factor, model-load time, process wall time, CPU, GPU, and memory fields. Treat the x64 values in the preceding section as x64 reference evidence, not as N1X targets. Apply the documented real-time thresholds to the N1X results.

For an application-specific release decision, repeat the procedure on every supported GPU tier with representative audio, both cold and warm application state, and every production effect or chain. Record CPU and GPU utilization alongside frame latency so that resource contention is visible. A production integration should define its own thresholds before collecting results.

Measure warm in-process operation by creating and loading an effect once, discarding an explicitly documented warm-up interval, and then processing at least 30 representative recordings through the same handle. Report p50/p95 frame latency, real-time factor, and resource utilization. Measure cold startup separately in a new process after a documented reboot or cache-reset procedure, and report effect creation and model-load time separately from `NvAFX_Run`. For concurrency testing, repeat at each supported production stream count and fail the release if the predeclared frame or throughput threshold is missed.

1.2.5.4 Deployment and Integration Fit

The following comparison is a deployment-selection guide. It makes no unmeasured claim about perceptual audio quality and does not substitute for a listening evaluation on the application's own corpus.

Decision Area	Windows Audio Effects SDK R22	Evaluate Another Deployment When
Effect breadth	Denoiser v1/v2, dereverb, combined dereverb and denoiser, Super Resolution, Speaker Focus, and Studio Voice HQ/LL; documented compatible chains are also available.	The product requires transcription, codecs, acoustic echo cancellation, or another capability outside this set.
Hardware and OS	Local Windows 10/11 x64 with a supported NVIDIA Tensor Core GPU; NVIDIA N1X uses the Windows ARM64 package.	CPU-only execution, a non-NVIDIA GPU, mobile, or browser-only execution is a hard requirement.
Integration surface	Native C API, import library and DLLs, dynamic loading, CMake consumer sample, and batch helpers.	A managed-language-only or hosted HTTP API is required without a native bridge.
Feature acquisition	Core SDK plus independently downloaded NGC feature payloads selected by GPU architecture and sample rate.	Deployment must be a single offline artifact with no authenticated feature acquisition step.
Deployment target	Client-side local Windows integration; the separate Linux SDK targets server deployments.	The primary target is a server or cloud service, where the Linux SDK or a service architecture should be evaluated.
Performance evidence	Frame-level timing is available from the sample and can be measured against the application's real-time frame budget.	A decision requires results for a GPU, concurrency level, or audio corpus that has not been benchmarked.

1.2.5.5 Market Alternatives and Differentiation

Use the following matrix for architectural selection. It compares documented product surfaces, not perceptual quality. Product capabilities and licensing can change, so verify the linked primary documentation during procurement.

Table 2: Documented Integration Alternatives

Option	Documented Execution Surface	Documented Audio Scope	Best Fit to Evaluate
Windows Audio Effects SDK R22	Native C API on Windows x64 with a supported NVIDIA Tensor Core GPU; Windows ARM64 on NVIDIA N1X.	Denoiser, dereverb, combined dereverb and denoiser, Super Resolution, Speaker Focus, Studio Voice HQ/LL, and compatible chains.	A local Windows product that can use RTX acceleration and needs several enhancement effects through one API.
Krisp RTC SDK	CPU execution across documented server, desktop, mobile, and browser targets.	Its RTC SDK documents background-voice cancellation and noise cancellation models.	CPU-only, mobile, browser, or broad cross-platform RTC deployment.
WebRTC Audio Processing Module	Native WebRTC pipeline component that can also be used standalone.	Echo cancellation, noise suppression, and automatic gain control.	An RTC stack that needs WebRTC-native capture processing, AEC, or AGC.
RNNoise	Open-source C noise-suppression library with a command-line example.	Suppression of recurrent neural-network noise; the example documents 48-kHz mono raw PCM.	An open-source denoising baseline or a narrowly scoped C integration.

The R22-specific differentiation is effect breadth in a single Windows-native API plus measured RTX frame-latency headroom for the two reference workloads discussed earlier. This statement does not claim superior perceptual quality. Before a competitive decision, run every candidate on the same licensed audio corpus, hardware class, frame size, concurrency, and warm-up policy. Predeclare the latency, throughput, CPU/GPU, memory, objective-quality, and listening-test criteria; retain raw outputs and report both passing and failing samples.

1.2.5.6 Release Checklist

Before shipping an integration:

1. Verify the exact SDK and feature versions and retain the feature-download receipt.
2. Build a clean consumer against only the packaged public headers and libraries.
3. Exercise every selected effect, supported chain, and production intensity value, including malformed and out-of-range inputs.
4. Validate output sample rate, channels, duration, finite samples, and non-silent output; use a listening test for perceptual quality.
5. Measure cold and warm frame latency, p50/p95, real-time factor, and CPU/GPU utilization on each supported hardware tier.
6. Confirm that logs and published reports do not contain credentials.

Chapter 2. Get Started on Linux

The Audio Effects SDK requires specific GPUs, specific distros of Linux and other software dependencies.

The Audio Effects SDK is designed and optimized for server-side (datacenter and cloud) deployment. Using this SDK for testing, experimentation, or production deployment of these SDKs outside of this use case is not officially supported.

2.1. Hardware Requirements

The Linux SDK is supported on systems that have a minimum of 10 GB RAM and NVIDIA GPUs with Tensor Cores.

Hardware	Required Version
NVIDIA GPU	NVIDIA GPUs with Tensor Cores (NVIDIA Turing™/NVIDIA Ampere/Ada/Hopper/Blackwell) • Turing: NVIDIA Tesla® T4 • Ampere: A2, A10, A16, A30, A40, A100 • ADA: L4, L40 • Hopper: H100 • Blackwell: B100, B200, RTX PRO 6000

Note

The SDK supports Multi-Instance GPU (MIG) only on Tesla A30 and Ampere A100. When MIG is enabled, the GPU instance and corresponding compute instance must be defined, regardless of whether the SDK is executed on a specific GPU instance or on the entire GPU.

Note

For best performance with NVIDIA T4 and other server GPUs, be sure to use a server that meets the thermal and airflow requirements for these types of products. For the latest list of qualified servers, refer to the Qualified System Catalog.

2.2. Software Requirements

Software	Required Version
Linux distribution	64-bit Linux distribution. Supported distros: • Ubuntu (18.04) • RHEL8 • CentOS8 • Debian 10+
NVIDIA Graphics Driver for Linux	• 570.26 or later. • 470/535/550 can be used with CUDA Forward-Compatible Upgrade. For more information, refer to *CUDA Forward-Compatible Upgrade*.
CUDA/TensorRT/CuDNN **Note**: All libraries that are required to use the SDK are in the package, under `external/cuda`, and do not need to be separately installed.	• CUDA: 12.8.1 • TensorRT: 10.9.0.34 • CuDNN: 9.7.1
Sample program compilation	• CMake: 3.10 or later • g++

2.2.1. Install the AFX SDK

The Audio Effects SDK for Linux consists of the following components:

- The core SDK package, which contains the base SDK library, headers, dependent libraries, and scripts for downloading feature packages and sample applications. You can download the core SDK package from NGC (from the SDK resource page).
- Feature packages, which contain the model library and GPU specific model files for effects. You can download feature packages from NGC (from the model page) or use the feature download script included in the core SDK package.
- Optional sample applications, which demonstrate how the core SDK and feature packages can be used. You can download the sample applications from GitHub or use the sample package download script included in the core SDK package.

To install the Audio Effects SDK, you must install both the base SDK and one or more feature packages.

2.2.1.1 Install the Core SDK Package

To install the Core SDK package, extract the contents of the Audio Effects SDK archive to the required location on your computer. For example, use the following command:

```
tar xvf --one-top-level Audio_Effects_SDK.tar.gz
```

Before using the SDK, you must download and install one or more feature packages.

2.2.1.2 Obtaining SDK Feature Packages

An SDK feature package consists of the effect library and the effect model file. Feature packages are hosted on NGC; you must download and unzip them before use.

You can download SDK feature packages directly from the NGC website or by using the download script included in the core SDK package.

Note

If the machine on which the SDK is to be deployed is not granted access to NGC (because of firewall or airgap rules), you can download feature packages to another machine and then transfer them to the target machine. The machine used for downloading feature packages does not require GPUs.

To download feature packages from the NGC website:

1. Refer to the NGC SDK collection for the required feature (under **Entities** > **Models**). The feature page contains both the library for the effect and the models.
2. Download the effect library corresponding to the SDK version. For example, if the SDK version is `3.0.0`, download the files under version `3.0.0-lib`.
3. Extract the file under this version under the `features` folder in the SDK (for example, `features/denoiser`). After extraction, this folder should contain include files (under `includes`) and the feature libraries (under `libs`).
4. Download the model files corresponding to the target machine GPU, SDK version, and sample rate. For example, if the SDK version is `3.0.0`, the `16kHz` effect is required, and the target GPU is A100 (SM 80), download the models for version `3.0.0-16k-sm80`.
5. Extract the model files under the `models` folder of the feature in a folder corresponding to the target machine GPU. For example, if the feature is `denoiser` (has feature libraries already extracted under `features/denoiser`) and the model files correspond to L40 GPU (SM 89), place the model files under `features/denoiser/models/sm_89`).

After you install the feature, the folder structure should look similar to the following structure:

```
$ tree Audio_Effects_SDK/features/
Audio_Effects_SDK/features/
    README.md
    denoiser
        include
            denoiser.h
        lib
            libnv_audiofx_denoiser.so -> libnv_audiofx_denoiser.so.2
            libnv_audiofx_denoiser.so.2 -> libnv_audiofx_denoiser.so.2.0.0
            libnv_audiofx_denoiser.so.2.0.0
        models
            sm_89
                denoiser_16k.trtpkg
                denoiser_16k_6656.trtpkg
                denoiser_48k.trtpkg
                denoiser_48k_6656.trtpkg
                denoiser_v2_16k.trtpkg
```

(continues on next page)

(continued from previous page)

```
                denoiser_v2_16k_7680.trtpkg
                denoiser_v2_48k.trtpkg
                denoiser_v2_48k_3840.trtpkg
```

The download script included in the core SDK packages automates the preceding steps. To download the feature package by using the download script:

1. Obtain an NGC API key from the NGC website. For details, refer to Generating NGC API Keys in the NGC User Guide.
2. Obtain the NGC organization and team from the SDK download page. For example, if the SDK is present at `https://catalog.ngc.nvidia.com/orgs/nvidia/teams/maxine/resources/maxine_linux_audio_effects_sdk`, the NGC organization is `nvidia` and the NGC team is `maxine`.
3. Unzip the SDK tarball and navigate to the `features` folder.
4. Download the features by using the following commands:

```
# Set the NGC API key

export NGC_API_KEY="<replace with API key>"

# Download models

# Syntax: ./download_features.sh [--gpu <GPU> --ngc-org <NGC org> --ngc-
↪team <NGC team> --effects effect1,effect2,effect3]

# GPU can be any of the following values:
# t4, a100, a10, a2, a16, a30, a40, l4, l40, h100, b100, b200, or rtx_pro_
↪6000.
# If not specified, the GPU is automatically detected.

# '--effects' specifies the list of effects to be downloaded.
# If not specified, all effects are downloaded (excluding Early Access
↪effects).
# To see the full list of effects supported, run './download_features.sh -
↪-help'.

# For example, the following command downloads all feature packages for
↪the automatically detected GPU (excluding Early Access features).

./download_features.sh

# To download all feature packages for the automatically detected GPU
↪including Early Access features:

./download_features.sh --effects dereverb-16k,dereverb-48k,dereverb_
↪denoiser-16k,dereverb_denoiser-48k,denoiser-16k,denoiser-48k,superres-
↪8k_to_16k,superres-16k_to_48k,studio_voice-16k,studio_voice-48k,speaker_
↪focus-16k,speaker_focus-48k

# To download all feature packages for T4 (and override the automatically
↪detected GPU):
```

(continues on next page)

(continued from previous page)

```
./download_features.sh --gpu t4

# To download just the Superresolution and Dereverb features for T4:

./download_features.sh --gpu t4 --effects superres-16k_to_48k,superres-8k_
↪to_16k,dereverb-16k,dereverb-48k
```

2.2.1.3 For Older Drivers, Use CUDA Forward-Compatible Upgrade

Applications can use the SDK with older drivers (470/535/550) by using the CUDA Forward-Compatible upgrade path. (For more information, refer to Forward Compatibility.)

To use older supported drivers with the SDK, download the user-mode CUDA libraries (`libcuda.so.*`) and the JIT compiler libraries for PTX files (`libnvidia-ptxjitcompiler.so.*`) from one of the following locations:

- The CUDA 12.8 Toolkit/datacenter drivers.
- The CUDA network repositories (`cuda-compat-12-8`).

Before you run any applications using the SDK, ensure that `LD_LIBRARY_PATH` contains the location of these libraries.

For example, to use the CUDA network repository on an Ubuntu 18.04 system with older drivers:

1. Go to CUDA Toolkit 12.8.1 Downloads.
   a. Under **Operating System**, click **Linux**.
   b. Under **Distribution**, click **Ubuntu**.
   c. Under **Installer Type**, click **deb (network)**.
   d. To add the CUDA repository to the system, follow the steps under **Installation Instructions**.
2. Update the apt repository cache:

   ```
   $ sudo apt-get update
   ```

3. Install the compatibility package:

   ```
   $ sudo apt-get install -y cuda-compat-12-8
   ```

   The command in this step installs the compatibility package libraries under `/usr/local/cuda-12.8/compat`.
4. When you run the SDK applications, append this path to `LD_LIBRARY_PATH`:

   ```
   # Add path to LD_LIBRARY_PATH

   # Note: This command works only for the current terminal session. Add to ~
   ↪/.bashrc or similar to make this permanent.
   # For more details, refer to your distribution's documentation.

   $ export LD_LIBRARY_PATH=/usr/local/cuda-12.8/compat:$LD_LIBRARY_PATH
   # Run application
   ./run_effect.sh -e denoiser -s 48
   ```

For more information, refer to Forward Compatibility.

2.2.2. Sample Applications

The AFX SDK provides the following sample applications for Linux:

- *effects_demo*
- *effects_delayed_streams_demo*

Note

CMake (3.10 or later), g++, and Make are required to compile and run the sample applications.

2.2.2.1 effects_demo Application

This application demonstrates how to use the AFX SDK to apply effects to audio.

2.2.2.1.1 Build the Application

1. Navigate to the `samples/effects_demo` directory.
2. Run the application using one of the following scripts, which automatically build the sample application.
   - To automatically detect the current GPU and load the corresponding models:

```
./run_effect.sh -s 16 -b 1 -e denoiser
./run_effect.sh -s 48 -b 1 -e dereverb
./run_effect.sh -s 16 -b 400 -e denoiser
./run_effect.sh -s 48 -b 400 -e dereverb_denoiser
./run_effect.sh -s 8 -o 16 -b 400 -e superres
./run_effect.sh -s 16 -b 1 -e studio_voice_high_quality
./run_effect.sh -s 48 -b 1 -e studio_voice_high_quality
./run_effect.sh -s 48 -b 1 -e studio_voice_low_latency
./run_effect.sh -s 16 -b 1 -e speaker_focus
./run_effect.sh -s 48 -b 1 -e speaker_focus
```

   - To explicitly specify the GPU:

```
./run_effect.sh -g t4 -s 16 -b 1 -e denoiser
./run_effect.sh -g t4 -s 48 -b 1 -e dereverb
./run_effect.sh -g t4 -s 16 -b 400 -e denoiser
./run_effect.sh -g t4 -s 48 -b 400 -e dereverb_denoiser
./run_effect.sh -g t4 -s 8 -o 16 -b 96 -e superres
./run_effect.sh -g t4 -s 16 -b 1 -e studio_voice_high_quality
./run_effect.sh -g t4 -s 48 -b 1 -e studio_voice_high_quality
./run_effect.sh -g t4 -s 48 -b 1 -e studio_voice_low_latency
./run_effect.sh -g t4 -s 16 -b 1 -e speaker_focus
./run_effect.sh -g t4 -s 48 -b 1 -e speaker_focus
```

Note

Ensure that the application uses the TensorRT (requires the exact version) or CUDA libraries (requires the exact version or later) that is bundled with the SDK (under `external/cuda/lib`).

If the distro exports `LD_LIBRARY_PATH` from `~/.bashrc` or similar, the TensorRT and CUDA paths might be overridden. As a result, the SDK might load incompatible CUDA or TensorRT library versions. To avoid this issue, before you run the sample program, append the external directory to `LD_LIBRARY_PATH` by executing the following command:

```
$ export LD_LIBRARY_PATH=external/cuda/lib:$LD_LIBRARY_PATH
```

Note

The sample app might hit the limit for the maximum number of open files that is imposed by default by the Linux kernel, especially for large batch sizes. When this occurs, the sample application exits with the following error message:

```
[Error] Unable to read wav file:
../input_files/denoiser/48k/Fan_48k.wav.

Open file limit reached.
```

To increase this limit, before you run the sample application, use the `ulimit` command in the same shell to increase the number of open files. For example, `ulimit -n 20000` increases the open file limit to 20,000 for that shell. For more information, refer to the documentation of your distribution on how to increase open file descriptor limits.

2.2.2.1.2 Run the Application

You can run the sample application by using the `run_effect.sh` helper script or by directly using the effects_demo executable.

Note

This sample app processes files in an offline manner. Hence, for Studio Voice, the processing time for the low latency effect is expected to be equal to or higher than the processing time for the high quality effect. This is because the high quality effect processes audio in larger input audio chunks, and requires fewer processing calls to the effect when compared to the low latency effect (which processes audio in very small input chunks). For more details, refer to *About the Studio Voice Effect*.

2.2.2.1.2.1 Use the Helper Script to Run the Sample Application

The `run_effect.sh` helper script is a wrapper around the effects_demo application. Depending on the arguments passed to `run_effect.sh`, the script generates a temporary config file and runs the effects_demo application with this generated config file.

Run the helper script by using the following command:

```
./run_effect.sh -g <gpu> -e <effect> -s <input_sample_rate> -o <output_sample_
↪rate> -b <batch_size> -i <input_file_or_folder>
```

For example, to run the sample application on T4 with the 16k denoiser effect with batch size of 10, run the following command:

```
./run_effect.sh -g t4 -s 16 -b 10 -e denoiser
```

This command generates a config file at `/tmp/tmp_cfg.txt` with the 16kHz denoiser model for T4, the sample rate initialized to 16kHz, and input/output file list based on batch size, and then runs effects_demo using this configuration file.

The helper script supports the following parameters:

- `-i/--input-file` specifies the input files/folder on which to run the effect.

  If this parameter is not specified, the helper script will use the sample files that are distributed with the SDK (in the `samples/input_files` directory). If this parameter specifies a file/folder, the helper script will use this file/files in this folder.

  The supported value is a path to the input file in correct format (for more information, refer to *Directly Running the Sample Application*), or a folder that contains multiple input files in correct formats. If a folder is specified, only the files at the top level will be processed. For example, if the input folder is `folder1`, `folder1/a.wav`, `folder1/b.wav`, and so on are processed, but `folder1/subfolder/a.wav` is not processed.

- `-g/--gpu` specifies the GPU on which to run the effect. If not specified, the current GPU (device 0) is used. Supported values are `a2`, `a16`, `a100`, `a10`, `t4`, `a30`, `a40`, `l4`, `l40`, `h100`, `b100`, `b200`, and `rtx_pro_6000`. The helper script selects the appropriate model based on the value of this parameter. If a model is not specified, the default value is `t4`.

- `-e/--effect` specifies which effect to use:
  - `denoiser`
  - `dereverb`
  - `dereverb_denoiser`
  - `superres`
  - `studio_voice_high_quality`
  - `studio_voice_low_latency`
  - `speaker_focus`

  If an effect is not specified, the default value is `denoiser`.

- `-s/--sample_rate` specifies the sample rate of input audio in kHz (can be either of 48/16/8). If the rate is not specified, the default value is 16.

- `-o/--output_sample_rate` (**Superresolution only**) specifies the sample rate of output audio. If the rate is not specified, the default value is 16.

- `-b/--batch_size` specifies the batch size to use.

  Depending on this parameter, the script generates an input file list (in `samples/input_files`) of the specified batch size by using the sample audio files provided with the SDK and a corresponding list of the output files. If the batch size is not specified, the default value is 1. The maximum value supported by this parameter is 1024.

- `-c/--cfg-file` specifies the path to which the temporary configuration file will be written. If the path is not specified, the default location is `/tmp/tmp_cfg.txt`.

- `-f/--frame_size` specifies the frame size (10 or 20) that will be used in milliseconds. If the frame size is not specified, the default value is 10.

- `-m/--effect_version` (**Denoiser only**) specifies the version of the effect to use. Set to 1 for the legacy Denoiser model or 2 for the experimental BNR 2.0 model. For more details, refer to *About the Noise Removal/Background Noise Suppression Effect*.
- `-p/--mic_profile` (**Studio Voice Low Latency only**) specifies the microphone profile to use. Supported values are `passthrough`, `bright`, `vibrant`, `flat`, `full`, `thin`, and `warm`. If you do not specify a profile, microphone profiling is disabled. For more details, refer to *Microphone Profiles*.
- `-h/--help` prints the parameters that are supported by this script.

2.2.2.1.2.2 Run the Sample Application Directly

Before running the sample application, ensure that it is built (either by running `./build.sh` or by calling either `run_effect.sh` or `run_effect_chained.sh`, which builds the sample app automatically). The sample app is built in the `build` folder.

To run the sample application after it is built, run the following command:

```
build/effects_demo -c <config-file>
```

`<config-file>` specifies the path of the sample config file, such as `t4_denoise48k_1_cfg.txt`. Sample config files for 16-kHz and 48-kHz audio are provided with the sample application.

Note

Config files that are used by the sample app can be generated by using the `run_effect.sh` script, which accepts a path specified by the `-c` or the `--cfg-file` flag. If this path is specified, the script writes a config file with the specified configuration parameters to that path. This config file can be reused by the `effects_demo` sample app.

For example, the following command writes the configuration to the file `t4_dereverb.cfg`:

```
./run_effect.sh -e dereverb -s 48 -g t4 -c t4_dereverb.cfg
```

For example, to denoise a 48-kHz stream on a T4 GPU for a batch size of 1, run:

```
build/effects_demo -c t4_denoise48k_1_cfg.txt
```

The configuration files contain pairs of parameters and their values, with one pair per line. Currently, the following parameters are supported:

`reset <list-of-stream-ids>`

Specifies the stream identifiers to reset, starting with 1. Multiple identifiers are separated by spaces.

`effect <effect-name>`

Specifies the name of the effect to apply:

- `denoiser`
- `dereverb`
- `dereverb_denoiser`
- `superres`
- `studio_voice_high_quality`

- `studio_voice_low_latency`
- `speaker_focus`

`sample_rate <audio-sample-rate>`

Specifies the sample rate of the input audio in Hz. Supported values are `8000`, `16000`, and `48000`.

`model <model-file>`

Specifies the path of the model file to be used in the sample application, such as `models/sm_70/denoiser_48k_1152.trtpkg`. The model file should match the audio sample rate that was specified in the `sample_rate` parameter and the sample rate of input wav files specified in the `input_wav_list` parameter. (For more information, refer to *Set the Parameters of an Audio Effect*.)

`frame_size <frame-size-value-in-milliseconds>`

Specifies the input frame size (in milliseconds) to be used in the *NvAFX_Run()* call.

The supported values are as follows:

- `10` or `20` for all effects unless another value is explicitly stated.
- `6000` for Studio Voice High Quality and `10` for Studio Voice Low Latency.

`input_wav_list <input-audio-file-list>`

Specifies a list of paths to input audio .wav files to use. Each file should contain mono channel audio in signed 16-bit or 32-bit float format with basic WAV header. Multiple files are separated by spaces. The number of input files must match the number of streams/batch size. In a stream, the files that are separated by a semicolon (;) are processed one after another in the same stream. In addition, if the stream ID exists in the reset list, *NvAFX_Reset* is called on the stream identifiers when switching between files.

For example, the following configuration specifies that streams 1, 2, and 4 use `file1.wav`, `file2.wav`, and `file6.wav` as the input to the stream, and stream 3 uses multiple files (`file3.wav`, `file4.wav`, `file5.wav`) as the input to the stream:

```
input_wav_list file1.wav file2.wav file3.wav;file4.wav;file5.wav
↪file6.wav
```

Note

Sample input audio files are included with the sample application in the `samples/input_files/16k` directory and the `samples/input_files/48k` directory.

`output_wav_list <output-audio-file-list>`

Specifies the files to which the output audio will be written. Output files contain mono audio in 32-bit float format. Multiple files are separated by spaces. In a stream, if multiple input files are specified (separated using semicolon), multiple output files will be created with the same name followed by _1, _2, and so on.

For example, in the following configuration, the output will be written to `out1.wav` (output of `file1.wav`), `out2.wav` (output of `file2.wav`), `out3.wav` (output of `file3.wav`), `out3_1.wav` (output of `file4.wav`), `out3_2.wav` (output of `file5.wav`), and `out4.wav` (output of `file6.wav`):

```
input_wav_list file1.wav file2.wav file3.wav;file4.wav;file5.wav
↪file6.wav

output_wav_list out1.wav out2.wav out3.wav out4.wav
```

Note

In input/output .wav files, only the basic WAV header is supported.

`real_time <enable>`

Simulates real-time audio input, set to 1 to enable or 0 to disable (disabled by default). When this option is enabled, each audio frame is passed to the SDK with a delay, similar to how audio is received from a physical device or stream. For example, if the frame size is 10ms, each frame is passed in every 10ms, like how audio is received from a physical microphone (10 ms audio received from the microphone approximately every 10 ms).

`intensity_ratio <[ratio|list_of_ratios]>`

Specifies the denoising intensity ratio. This value can either be a single value (applied to all streams) or a value per stream (individual values applied to each corresponding stream).

The value of this parameter ranges from 0.0 to 1.0 (inclusive), where a higher value indicates a stronger suppression of noise/reverb. A value of 0.0 is equivalent to passing out input audio without applying noise removal/dereverb.

`effect_version <effect_version>`

(Denoiser only) Specifies the version for the effect. Set to 1 to use the original BNR effect and 2 to use the experimental BNR 2.0 effect. For more details, see *About the Noise Removal/Background Noise Suppression Effect*.

`mic_profile_model <model-file>`

(Studio Voice Low Latency only) Specifies the path to the microphone-profile companion model. This entry is required when `mic_profile_id` is set to a value from 0 through 5.

`mic_profile_id <profile-id>`

(Studio Voice Low Latency only) Selects a microphone profile. The default, -1, disables microphone profiling. Values 0 through 5 select the profiles described in *Microphone Profiles*.

2.2.2.1.2.3 Chain Multiple Effects

This sample application also supports chaining multiple effects. (For more information, refer to *Run Multiple Audio Effects in a Chain*.)

To run the application in chaining mode, use `run_effect_chained.sh`:

```
./run_effect_chained.sh -g <gpu> -e1 <effect1> -s1 <input_sample_rate_1> -o1
↪<output_sample_rate_1> -e2 <effect2> -s2 <input_sample_rate_2> -o2 <output_
↪sample_rate_2> [-c <path_to_save_config_file>] [-i <input_file_or_folder>]
```

The preceding script above generates a config file that can be used with the effects_demo sample to run multiple effects in a chain and runs the application with this file.

The config file that is used for chaining follows the same format and parameters as effects_demo, with the following modifications:

`effect <effect-name-1> <effect-name-2>`

Specifies the names of the effects to apply to input audio (*effect-name-1* will be applied to input audio first, and *effect-name-2* will be applied to this output). For more information about chaining combinations, see *Create a Chained Audio Effect*.

> **Note**
>
> Chaining effects only support combinations of Superres+Denoiser/Dereverb/Combined Denoiser+Dereverb effect and Denoiser + Speaker Focus. Other effect chains are *not* supported.
>
> If you combine the Denoiser effect and Dereverb effect, use the combined Denoiser+Dereverb model. For more information, see *About the Noise Removal/Background Noise Suppression Effect*.

`sample_rate <audio-sample-rate-1> <audio-sample-rate-2>`

Specifies the input sample rate of the audio in Hz for the effects. The supported values are `8000`, `16000`, and `48000`.

`model <model-file-1> <model-file-2>`

Specifies the path of the model file to be used by the effects; for example, `models/sm_70/denoiser_48k_1152.trtpkg`. The model file should match the audio sample rate that was specified in the `sample_rate` parameter and the sample rate of input wav files specified in the `input_wav_list` parameter. (For more information, refer to *Set the Parameters of an Audio Effect*.)

`intensity_ratio <ratio-1> <ratio-2>`

Specifies the denoising intensity ratio of the first and the second effect. The value of this parameter ranges from `0.0` to `1.0` (inclusive), where a higher value indicates a stronger suppression of noise/reverb. A value of `0.0` is equivalent to passing out input audio without applying noise removal/dereverb.

`chained_effect_gpu_list <gpu-1> <gpu-2>`

In a multi-GPU system, specifies the GPU device ID that will be used for the first and the second effect in the chain.

The supported value for this parameter is a path to the input file in the correct format (refer to *Directly Running the Sample Application*) or a folder that contains multiple input files in the correct format. If a folder is specified, only the files at the top level are processed. For example, if the input folder is `folder1`, the files `folder1/a.wav`, `folder1/b.wav`, and so on are processed, but `folder1/subfolder/a.wav` is not processed.

2.2.2.2 effects_delayed_streams_demo Application

This application demonstrates the use case for handling delayed streams. (For more information, refer to *Run an Audio Effect on Delayed Audio Streams on Linux*.) In this sample, each of the input streams falls under one of the following categories:

`one_step_delay_streams`

These streams have a delay of one frame. For example, if the frame size is 10ms, these streams will have a delay of 10ms. This means that these streams will be active every alternate iteration. When active, it will receive data for both frames (20ms). As a result, when

data from these streams arrive, *NvAFX_Run* should be called two times, once with the delayed data and once with the current data.

`two_step_delay_streams`

These streams have a delay of two frames. For example, if the frame size is 10ms, these streams will have a delay of 20ms. This means that these streams will be active after every two iterations, and when active, will receive data for three iterations (30ms). As a result, when data from these streams arrive, *NvAFX_Run* should be called three times, twice with the delayed data and once with the current data.

`always_active_streams`

These streams have no delay and are always active, with one *NvAFX_Run* call per iteration.

NvAFX_Run() calls are made based on the preceding description to generate processed audio output. The configuration files provide a parameter to specify `one_step_delay_streams` and `two_step_delay_streams`. (For more information, refer to *Run the Application*.) These values and the batch size are used to infer the list of `always_active_streams`.

2.2.2.2.1 Build the Application

1. Navigate to the `samples/effects_delayed_streams_demo` directory.
2. To compile the application, run the following command:

```
./build.sh
```

2.2.2.2.2 Run the Application

You can run the sample application by using the `run_effect.sh` helper script or directly by using the `effects_delayed_streams_demo` executable located in `builds`.

2.2.2.2.3 Use the Helper Script to Run the Sample Application

The `run_effect.sh` helper script is a wrapper around the `effects_delayed_streams_demo` application and runs like *the helper script for effects_demo*.

This script supports ten streams that are always pre-configured into active streams, streams with a one-step delay, and streams with a two-step delay. In addition to the parameters available in `run_effect.sh` that were specified, this script also supports the `-t/--all_streams_active` parameter, which specifies that all ten streams are always active. If this parameter is not specified, several streams are configured with a one-step or two-step delay.

For example, to run the sample application on T4 with the 16k denoiser effect, a batch size of 10, and with all streams active, run the following command:

```
./run_effect.sh -g t4 -s 16 -b 10 -e denoiser -a
```

2.2.2.2.4 Run the Sample Application Directly

To directly run the sample application after it is built, run the following command:

```
build/effects_delayed_streams_demo -c <config-file>
```

`<config-file>` specifies the path of the config file, such as `t4_denoise48k_10_cfg.txt`.

For example:

```
build/effects_delayed_streams_demo -c t4_denoise48k_10_cfg.txt
```

Sample config files are provided with the application.

Like `effects_demo`, the configuration files contain pairs of parameters and their values, with one pair per line. In addition to the configuration parameters used by `effects_demo`, `effects_delayed_streams_demo` requires the following parameters:

- `one_step_delay_streams <list-of-stream-id>`

  Specifies the stream identifiers that belong to the `one_step_delay_streams` category as mentioned in the previous section. If none of the streams are in this category, this value should be set to `none`.

- `two_step_delay_streams <list-of-stream-id>`

  Specifies the stream identifiers that belong to the `two_step_delay_streams` category as mentioned in the previous section. If none of the streams are in this category, this value should be set to `none`.

2.2.2.2.5 Chain Multiple Effects

This sample application also supports chaining multiple effects. (For more information, refer to *Create a Chained Audio Effect*.)

To run the application in chaining mode, use `run_effect_chained.sh`:

```
./run_effect_chained.sh -g <gpu> -e1 <effect1> -s1 <input_sample_rate_1> -o1
↪<output_sample_rate_1> -e2 <effect2> -s2 <input_sample_rate_2> -o2 <output_
↪sample_rate_2> [-c <path_to_save_config_file>]
```

This script generates a config file at `/tmp/tmp_cfg.txt` and runs the `effects_delayed_streams_demo` sample with this config.

The configuration used for this script is the same as the configuration used for `effects_demo`. (For more information, refer to *Chain Multiple Effects*.) The script also uses the same parameters that are used by `effects_delayed_streams_demo`. (For more information, refer to *Run the Application*.)

2.2.3. Use the AFX SDK in Containers

The AFX SDK can be used with any containerization technology that supports the NVIDIA Container Toolkit (such as Docker) without requiring any changes.

Note

Using the AFX SDK in containers might introduce negligible differences in performance compared to bare-metal deployments due to virtualization overheads, depending on the containerization technology used and how it is configured. For further details, refer to the containerization technology's performance optimization guide.

For reference on how to package the SDK for containerization, a sample container configuration based on Docker is provided under `samples/container`.

To use the sample Docker container:

- Ensure that you have Docker, NVIDIA Container Toolkit, and NVIDIA drivers installed on the host system.
- Download features for the targeted GPUs.

> **Note**
>
> This sample downloads the model files from NGC and embeds them in the container. Although doing so is not required for containerization, we highly recommend having the models pre-downloaded inside the container. Downloading the models after the container is launched might not be optimal because the time required for the container to be ready (including downloading the required models) can vary.

- Change the current working directory to `samples/container`: `cd samples/container`.
- Build container (takes a few minutes): `./build_container_image.sh`
  - The built container will have the SDK with downloaded models copied to `/opt/nvidia/maxine/Audio_Effects_SDK`.
- Create a folder for container output: `mkdir out`
- Run the container mapping output folder using `-m`. (The path must not end with `/`.) This starts a new prompt inside the container:

```
./run_container_image.sh -m out
```

- Change the working directory to `samples/effects_demo` and run the effects_demo sample for any effect.
- Copy the output to `/host` (which is mapped to the output folder using `-m`):

```
root@d3a064d21267:/opt/nvidia/maxine/Audio_Effects_SDK#
cd samples/effects_demo/

root@d3a064d21267:/opt/nvidia/maxine/Audio_Effects_SDK/samples/effects_
↪demo#
./run_effect.sh -g a16 -e denoiser -s 48

...

root@d3a064d21267:/opt/nvidia/maxine/Audio_Effects_SDK/samples/effects_
↪demo#
cp -r denoiser/ /host/

root@d3a064d21267:/opt/nvidia/maxine/Audio_Effects_SDK/samples/effects_
↪demo#
exit
```

- The output will be present on the host system under the mapped folder (`out` in the preceding example):

```
ls out
```

Chapter 3. About the Audio Super-Resolution Effect

The Audio Super-Resolution effect upsamples the audio. For low-frequency audio, this feature predicts the higher frequency spectrum of input audio, which improves audio quality.

Note

In this guide, the term *Super-Resolution* is used interchangeably with *Superres* and *Superresolution* (referred to as `superres` in the API).

This effect has the following characteristics:

- Supported input/output audio format is 32-bit float audio.

Note

The main purpose of this effect is to enhance the sampling rate of input audio. The level of enhancement seen in the output audio depends on the type of audio.

Audio that is captured on Windows with the audio enhancement settings disabled produces better superres outputs than when this setting is enabled.

Acquire the Windows feature files once from the SDK `features` directory:

```
powershell -ExecutionPolicy Bypass -File .\download_features.ps1 -Effects
↪superres-8k_to_16k,superres-16k_to_48k -Version 3.0.0 -Output_Directory .
```

Run from an existing package without network access. The x64 helper directory is shown; on NVIDIA N1X, use `samples\build-arm64\Release\scripts`:

```
Set-Location $env:AFX_SDK_ROOT\samples\build\Release\scripts
.\run_effects_demo.bat -e superres -isr 8k -osr 16k
.\run_effects_demo.bat -e superres -isr 16k -osr 48k
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

Acquire the Linux models once from the SDK `models` directory:

```
./download_models.sh --gpu <gpu> --effects superres-8k_to_16k,superres-16k_to_
↪48k
```

Run locally from the built Linux `effects_demo` sample directory:

```
./run_effect.sh -g t4 -s 16 -e superres
./run_effect.sh -g t4 -s 48 -e superres
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

- Supported upsampling of 8-kHz input audio to 16-kHz output (2x) and 16-kHz input audio to 48-kHz output (3x).
- In the Linux SDK, this effect has the following maximum throughput (the number of batches supported in real time):

Architecture	Maximum Throughput for the 8K-to-16K Effect	Maximum Throughput for the 16K-to-48K Effect
T4	410	180
A100	2110	920
A10	910	380
L40	3450	1350
H100	2430	1030
B100	5720	2330
RTX PRO 6000	3940	1650

Chapter 4. About the Noise Removal/Background Noise Suppression Effect

Recordings of speech made outside of a recording studio contain a lot of background noise. The Audio Denoiser Effect removes a variety of background noises from audio recordings. (For the types of noise that this effect removes, see *Types of Background Noise Removed*.)

This effect retains emotive tones in speech, such as happy, sad, excited, and angry tones, which were removed as noise in previous versions of the SDK. Extreme emotive cases, such as loud laughing, shrieking, screaming, and crying might not be retained. Extremely loud noises in input (SNR < 5 dB) might result in distorted output.

Note

In this guide, the term *Background Noise Suppression* is used interchangeably with *Denoising* and *Noise Removal* (referred to as `denoiser` in the API).

Acquire the Windows feature files once from the SDK `features` directory:

```
powershell -ExecutionPolicy Bypass -File .\download_features.ps1 -Effects
↪denoiser-16k,denoiser-48k -Version 3.0.0 -Output_Directory .
```

Run from an existing package without network access. The x64 helper directory is shown; on NVIDIA N1X, use `samples\build-arm64\Release\scripts`:

```
Set-Location $env:AFX_SDK_ROOT\samples\build\Release\scripts
.\run_effects_demo.bat -e denoiser -isr 16k -osr 16k
.\run_effects_demo.bat -e denoiser -isr 48k -osr 48k
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

Acquire the Linux models once from the SDK `models` directory:

```
./download_models.sh --gpu <gpu> --effects denoiser-16k,denoiser-48k
```

Run locally from the built Linux `effects_demo` sample directory:

```
./run_effect.sh -g t4 -s 16 -e denoiser
./run_effect.sh -g t4 -s 48 -e denoiser
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

This effect has the following characteristics:

- Supported input/output audio format is 32-bit float audio with a sampling rate of 16 kHz or 48 kHz.
- In the Linux SDK, this effect has the following maximum throughput (the number of batches supported in real time):

Architecture	Maximum Throughput for the 16K Effect	Maximum Throughput for the 48K Effect
T4	3240	1520
A100	15760	6750
A10	6410	2960
L40	14440	5400
H100	18500	8030
B100	25030	12160
RTX PRO 6000	23640	10640

Note

This effect might miss some noises in the first 1–2 seconds of input audio. Low-volume noises during speech might also be missed.

4.1. BNR 2.0 Experimental Model

BNR 2.0 is a newer version of the Noise Removal effect that improves the speech recognition accuracies of ASR systems and integrates seamlessly in pipelines where audio cleaning is required before being used in other subsystems. It supports all the noise profiles supported by the stable denoiser effect.

The BNR 2.0 preview is currently supported using the `effect_version` parameter. For details on how this mode is enabled, see *Set the Parameters of an Audio Effect*.

The same acquired feature set contains the Denoiser v2 model. Run it locally with `-ev 2`; do not edit the helper script:

```
Set-Location $env:AFX_SDK_ROOT\samples\build\Release\scripts
.\run_effects_demo.bat -e denoiser -isr 16k -osr 16k -ev 2
.\run_effects_demo.bat -e denoiser -isr 48k -osr 48k -ev 2
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*. This effect cannot be chained with other effects using the Chaining API.

Note

With this release, Voice Activity Detection is enabled by default for this effect.

Run the already-acquired Linux v2 model locally:

```
./run_effect.sh -g t4 -s 16 -e denoiser -m 2
./run_effect.sh -g t4 -s 48 -e denoiser -m 2
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

This effect has the following characteristics:

- Supported input/output audio format is 32-bit float audio with a sampling rate of 16 kHz or 48 kHz.
- In the Linux SDK, this effect has the following maximum throughput (the number of batches supported in real time):

Architecture	Maximum Throughput for the 16K Effect	Maximum Throughput for the 48K Effect
T4	1350	430
A100	7880	3960
A10	3160	1840
L40	7220	4860
H100	9670	6940
B100	12160	4160
RTX Pro 6000	12160	5610

Chapter 5. About the Noise Removal and Room Echo Removal/Room Echo Cancellation + Background Noise Suppression Effect

This effect applies a denoising and a dereverb effect on the input audio. (For the types of noise that this effect removes, see *Types of Background Noise Removed*.)

Note

In this guide, the term *Room Echo Cancellation + Background Noise Suppression* is used interchangeably with *Dereverb+Denoiser* and *Noise Removal and Room Echo Removal* (referred to as `dereverb_denoiser` in the API).

Acquire the Windows feature files once from the SDK `features` directory:

```
powershell -ExecutionPolicy Bypass -File .\download_features.ps1 -Effects
↪dereverb_denoiser-16k,dereverb_denoiser-48k -Version 3.0.0 -Output_
↪Directory .
```

Run from an existing package without network access. The x64 helper directory is shown; on NVIDIA N1X, use `samples\build-arm64\Release\scripts`:

```
Set-Location $env:AFX_SDK_ROOT\samples\build\Release\scripts
.\run_effects_demo.bat -e dereverb_denoiser -isr 16k -osr 16k
.\run_effects_demo.bat -e dereverb_denoiser -isr 48k -osr 48k
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

Acquire the Linux models once from the SDK `models` directory:

```
./download_models.sh --gpu <gpu> --effects dereverb_denoiser-16k,dereverb_
↪denoiser-48k
```

Run locally from the built Linux `effects_demo` sample directory:

```
./run_effect.sh -g t4 -s 16 -e dereverb_denoiser
./run_effect.sh -g t4 -s 48 -e dereverb_denoiser
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

This effect has the following characteristics:

- Supported input/output audio is 32-bit float audio with a sampling rate of 16 kHz or 48 kHz.
- In the Linux SDK, this effect has the following maximum throughput (the number of batches supported in real time):

Architecture	Maximum Throughput for the 16K Effect	Maximum Throughput for the 48K Effect
T4	1230	510
A100	7600	2860
A10	6600	3000
L40	7480	2760
H100	8860	3350
B100	12160	4860
RTX Pro 6000	11500	4260

Chapter 6. About the Room Echo Removal/Room Echo Cancellation Effect

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

Note

In this guide, the term *Room Echo Cancellation* is used interchangeably with *Dereverb* and *Room Echo Removal* (referred to as `dereverb` in the API).

Acquire the Windows feature files once from the SDK `features` directory:

```
powershell -ExecutionPolicy Bypass -File .\download_features.ps1 -Effects
↪dereverb-16k,dereverb-48k -Version 3.0.0 -Output_Directory .
```

Run from an existing package without network access. The x64 helper directory is shown; on NVIDIA N1X, use `samples\build-arm64\Release\scripts`:

```
Set-Location $env:AFX_SDK_ROOT\samples\build\Release\scripts
.\run_effects_demo.bat -e dereverb -isr 16k -osr 16k
.\run_effects_demo.bat -e dereverb -isr 48k -osr 48k
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

Acquire the Linux models once from the SDK `models` directory:

```
./download_models.sh --gpu <gpu> --effects dereverb-16k,dereverb-48k
```

Run locally from the built Linux `effects_demo` sample directory:

```
./run_effect.sh -g t4 -s 16 -e dereverb
./run_effect.sh -g t4 -s 48 -e dereverb
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

This effect has the following characteristics:

- Supported input/output format is 32-bit float audio with a sampling rate of 16 kHz or 48 kHz.
- In the Linux SDK, this effect has the following maximum throughput (the number of batches supported in real time):

Architecture	Maximum Throughput for the 16K Effect	Maximum Throughput for the 48K Effect
T4	3240	1450
A100	17020	6970
A10	6600	3000
L40	14440	5720
H100	19340	8180
B100	26600	12510
RTX PRO 6000	25110	10640

Chapter 7. About the Speaker Focus Effect

Important

The Speaker Focus effect is currently available under the Early Access program. For more details, refer to NVIDIA Maxine Early Access Program.

Note

In this guide, the *Speaker Separation Effect* is used interchangeably with *Speaker Focus* (referred to as `speaker_focus` in the API).

Audio with people speaking in the background is generally unintelligible. The Speaker Focus Effect identifies and isolates the primary speaker from all other speakers and removes the speech of all other speakers from the input audio. This significantly improves the intelligibility of the speech of the primary speaker in the input audio.

This effect is robust against the following types of background noises and suppresses them to a certain extent:

- AC noise
- Clapping
- Fan noise
- Keyboard
- Mouse clicks
- PC noise
- Sounds of a vacuum cleaner
- Tapping

Acquire the Windows feature files once from the SDK `features` directory:

```
powershell -ExecutionPolicy Bypass -File .\download_features.ps1 -Effects
↪speaker_focus-16k,speaker_focus-48k -Version 3.0.0 -Output_Directory .
```

Run from an existing package without network access. The x64 helper directory is shown; on NVIDIA N1X, use `samples\build-arm64\Release\scripts`:

```
Set-Location $env:AFX_SDK_ROOT\samples\build\Release\scripts
.\run_effects_demo.bat -e speaker_focus -isr 16k -osr 16k
.\run_effects_demo.bat -e speaker_focus -isr 48k -osr 48k
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

Acquire the Linux models once from the SDK `models` directory:

```
./download_models.sh --gpu <gpu> --effects speaker_focus-16k,speaker_focus-48k
```

Run locally from the built Linux `effects_demo` sample directory:

```
./run_effect.sh -g t4 -s 16 -e speaker_focus
./run_effect.sh -g t4 -s 48 -e speaker_focus
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

This effect has the following characteristics:

- Supported input/output format is 32-bit float audio with a sampling rate of 16 kHz or 48 kHz.
- Supports mono-channel input and output.
- Supports audio with up to four speakers and the noises listed earlier.
- In the Linux SDK, this effect has the following maximum throughput (the number of batches supported in real time):

Architecture	Maximum Throughput for the 16K Effect	Maximum Throughput for the 48K Effect
T4	3040	1350
A100	15760	6700
A10	6480	2900
L40	13890	5120
H100	18500	8030
B100	25030	11500
RTX PRO 6000	25030	10380

Chapter 8. About the Studio Voice Effect

Speech recorded with low-end microphones in less-than-ideal acoustic environments can contain distortions such as reverberations or static noise. The Studio Voice effect can enhance and recover degraded speech recorded using low-end microphones in non-ideal acoustic environments (for example, with reverb and static). This effect also recovers speech degraded by noise-reduction filters or beamforming algorithms. The resulting audio sounds better than the original because of the removal or reduction of degradation and artifacts, making the audio sound more like a recording in a professional studio setup.

This effect has two variations:

- Studio Voice High Quality
- Studio Voice Low Latency

8.1. Studio Voice High Quality

This effect is intended for offline use cases, such as post-processing video and audio files.

For a batch of recordings, create and load one effect instance, process all recordings with that instance, call *NvAFX_Reset* between unrelated audio sources, and destroy the effect after the batch. Avoid starting a new process or reloading the model for every file; effect creation and model loading are initialization costs, not per-frame latency.

This effect supports the following configurations:

- 16-kHz degraded speech to 16-kHz enhanced speech.
- 48-kHz degraded speech to 48-kHz enhanced speech.

Acquire the Windows feature files once from the SDK `features` directory:

```
powershell -ExecutionPolicy Bypass -File .\download_features.ps1 -Effects
↪studio_voice_high_quality-16k,studio_voice_high_quality-48k -Version 3.0.0 -
↪Output_Directory .
```

Run from an existing package without network access. The x64 helper directory is shown; on NVIDIA N1X, use `samples\build-arm64\Release\scripts`:

```
Set-Location $env:AFX_SDK_ROOT\samples\build\Release\scripts
.\run_effects_demo.bat -e studio_voice_high_quality -isr 16k -osr 16k
.\run_effects_demo.bat -e studio_voice_high_quality -isr 48k -osr 48k
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

Acquire the Linux models once from the SDK `models` directory:

```
./download_models.sh --gpu <gpu> --effects studio_voice-16k,studio_voice-48k
```

Run locally from the built Linux `effects_demo` sample directory:

```
./run_effect.sh -g t4 -s 16 -e studio_voice_high_quality
./run_effect.sh -g t4 -s 48 -e studio_voice_high_quality
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

Note

This effect is not optimized for real-time use cases or for optimal GPU usage. Outputs might have extra latency (up to 6 seconds) and, depending on your system specifications, are not guaranteed to be generated in real time.

This effect uses large input/output chunk sizes (6 seconds per inference, which can be queried using *NvAFX_GetU32*) and currently does not support multiple batches or delayed streams.

8.2. Studio Voice Low Latency

This effect is intended for online or real-time use cases (for example, online voice conferences or broadcast apps).

This effect supports the following configuration:

- 48-kHz degraded speech to 48-kHz enhanced speech.

8.2.1. Microphone Profiles

Studio Voice Low Latency can optionally apply a microphone profile to its enhanced output. The default is passthrough, which leaves the Studio Voice output unchanged.

The following profiles are available:

Profile	Character
Passthrough	Microphone profile disabled.
Bright	Clear and sharp, with audible detail.
Vibrant	Rich sound with added character.
Flat	Natural, balanced sound.
Full	Rich, deep, clear sound.
Thin	Light sound with less depth.
Warm	Soft, smooth, bass-heavy sound.

Note

This effect is optimized specifically for real-time use cases, where input audio is provided in small chunks and output audio is expected back with minimal delay.

This effect should not be used for offline use cases (such as post-processing) because it processes audio in very small input chunks. Use the high-quality effect for offline cases. Do not compare the two variants by API-call time alone: they use different frame sizes and target different latency requirements. Compare end-to-end throughput for offline work and frame latency against the real-time budget for interactive work.

Acquire the Windows low-latency feature once from the SDK `features` directory:

```
powershell -ExecutionPolicy Bypass -File .\download_features.ps1 -Effects
↪studio_voice_low_latency-48k -Version 3.0.0 -Output_Directory .
```

Run from the existing package without network access:

```
Set-Location $env:AFX_SDK_ROOT\samples\build\Release\scripts
.\run_effects_demo.bat -e studio_voice_low_latency -isr 48k -osr 48k
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

Acquire the Linux Studio Voice models once from the SDK `models` directory:

```
./download_models.sh --gpu <gpu> --effects studio_voice-16k,studio_voice-48k
```

Then run the low-latency effect locally from the built sample directory:

```
./run_effect.sh -g t4 -s 48 -e studio_voice_low_latency
```

Note

For more information, see *Use the Helper Script to Run the Sample Application*.

Note

This effect is not optimized for optimal GPU usage. Outputs might have extra latency (up to 110 ms) and might not be generated in real time on lower-end GPUs. Also, this effect does not currently support multiple batches or delayed streams.

Chapter 9. Use the Audio Effects SDK in Applications

The Audio Effects API is a C API but can also be used with applications that are built using C++.

This page describes the typical workflow for using an effect in applications.

This flow is a simplified version of the sample programs (`effects_demo` and, for Linux, `effects_delayed_streams_demo`). The same flow is also used for chained effects, with a few differences in API calls.

1. Create an effect handle for the effect:

```
#include <nvAudioEffects.h>
#include <nvAFXDenoiser.h>  // From features/nvafxdenoiser/include

NvAFX_Handle handle;

// Create single effect
NvAFX_Status status = NvAFX_CreateEffect(NVAFX_EFFECT_DENOISER, &handle);

// Create chained effect
NvAFX_CreateChainedEffect(NVAFX_CHAINED_EFFECT_SUPERRES_8k_TO_16k_
↪DENOISER_16k, &handle);
```

2. Set the required parameters:

   Windows

```
// Set model name

// Single effect (can also use SetStringList with size 1)
NvAFX_SetString(handle, NVAFX_PARAM_MODEL_PATH, "denoiser_48k.trtpkg");

// Chained effect
NvAFX_SetStringList(handle, NVAFX_PARAM_MODEL_PATH, model_files, num_
↪model_files);

// Set input and output sample rates
NvAFX_SetU32(handle, NVAFX_PARAM_INPUT_SAMPLE_RATE, 48000);
NvAFX_SetU32(handle, NVAFX_PARAM_OUTPUT_SAMPLE_RATE, 48000);
```

Linux

```
// Set model name

// Single effect (can also use SetStringList with size 1)
NvAFX_SetString(handle, NVAFX_PARAM_MODEL_PATH, "denoiser_48k.trtpkg");

// Chained effect
NvAFX_SetStringList(handle, NVAFX_PARAM_MODEL_PATH, model_files, num_
↪model_files);

// Set input sample rate and number of streams
NvAFX_SetU32(handle, NVAFX_PARAM_INPUT_SAMPLE_RATE, 48000);
NvAFX_SetU32(handle, NVAFX_PARAM_NUM_STREAMS, 20);
```

3. Set the optional parameters by using the `NvAFX_SetU32`/`NvAFX_SetFloat` parameters:
   - Intensity ratio.
   - Use default GPU.
   - VAD enable/disable.
   - CUDA graph enable/disable (Windows only).
   - Delayed streams enable/disable (Linux only).
   - Samples per input frame. A list of supported input sample rates can be queried by using *NvAFX_GetU32List*. (Refer to *Get the Parameters of an Audio Effect*.)
   - Effect version (by using `NvAFX_SetU32`), if using the experimental Denoiser effect.
   - GPU on which the model will be loaded. For more information, refer to *Use Multiple GPUs*.
4. Load the model:

```
NvAFX_Load(handle);
```

5. After a successful load, query the input/output sample rate, channels, and samples per frame for the effect:

```
// Sample rate
NvAFX_GetU32(handle, NVAFX_PARAM_INPUT_SAMPLE_RATE, &input_sample_rate);
NvAFX_GetU32(handle, NVAFX_PARAM_OUTPUT_SAMPLE_RATE, &output_sample_rate);

// Channels
NvAFX_GetU32(handle, NVAFX_PARAM_NUM_INPUT_CHANNELS, &num_input_channels);
NvAFX_GetU32(handle, NVAFX_PARAM_NUM_OUTPUT_CHANNELS, &num_output_
↪channels);

// Samples per frame
NvAFX_GetU32(handle, NVAFX_PARAM_NUM_SAMPLES_PER_INPUT_FRAME, &num_
↪samples_per_input_frame);
NvAFX_GetU32(handle, NVAFX_PARAM_NUM_SAMPLES_PER_OUTPUT_FRAME, &num_
↪samples_per_output_frame);
```

6. For each input sample, process the audio by using *NvAFX_Run*:

```
const float* input_buffers[] = { input_samples };
float* output_buffers[] = { output_samples };
NvAFX_Run(handle, input_buffers, output_buffers,
          num_samples_per_input_frame, num_input_channels);
```

7. If a disconnection occurs in audio processing (for example, a batch that was reused for a different audio source), reset the internal effect states by calling *NvAFX_Reset*:

   Windows

   ```
   NvAFX_Reset(handle);
   ```

   Linux

   ```
   NvAFX_Reset(handle, states_array, input_wav_list.size());
   ```

8. On Linux only: During batching, to temporarily pause streams (for example, if data is not ready for that stream but is available for processing for other streams) use `NVAFX_PARAM_ACTIVE_STREAMS` as required.

After audio processing is complete, to free resources, use *NvAFX_DestroyEffect*. For more information, see *Run an Audio Effect on Delayed Audio Streams on Linux*.

Chapter 10. Build Applications with the AFX SDK

The SDK includes dependent libraries in `external/cuda/lib`, which are required to compile and run applications and do not require libraries to be separately installed. Refer to *Software Requirements* (Windows) or *Software Requirements* (Linux) for runtime requirements for applications that use the SDK.

10.1. Windows

Add both the core `include` directory and the selected feature's include directory to the compiler search path. For Denoiser, these are `<AFX_SDK_ROOT>\include` and `<AFX_SDK_ROOT>\features\nvafxdenoiser\include`. The feature header defines selectors such as `NVAFX_EFFECT_DENOISER`.

To build applications with SDK on Windows, use one of the following methods:

- Statically link the library in Visual Studio by using the `.lib` file (`NVAudioEffects.lib`).

  For more information, refer to *Build the Sample Application*.

- Load the SDK DLL at runtime by using `LoadLibrary/GetProcAddress`.

  For example, *NvAFX_CreateEffect* can be called in the following way:

```
#include <Windows.h>
#include <nvAudioEffects.h>
#include <nvAFXDenoiser.h>  // From features/nvafxdenoiser/include

typedef NvAFX_Status (*NVAFX_CREATEEFFECT)(NvAFX_EffectSelector, NvAFX_
↪Handle*);
HMODULE module = LoadLibraryW(L"NVAudioEffects.dll");
if (module == nullptr) {
   // Handle DLL load failure.
}
NVAFX_CREATEEFFECT create_effect = reinterpret_cast<NVAFX_CREATEEFFECT>(
   GetProcAddress(module, "NvAFX_CreateEffect"));
if (create_effect == nullptr) {
   FreeLibrary(module);
   // Handle missing API entry point.
}
NvAFX_Handle nv_handle = nullptr;
NvAFX_Status status = create_effect(NVAFX_EFFECT_DENOISER, &nv_handle);
```

(continues on next page)

(continued from previous page)

```
// Similarly for other APIs
FreeLibrary(module);
```

Previous releases of TensorRT might have a bug where cuBLAS is not unloaded after you unload the SDK DLL, and this issue might cause a memory leak.

To work around this issue, run the following workaround:

```
int maxLoopCount = 5;

while (maxLoopCount--) {
   HMODULE cublas_handle = GetModuleHandleW(L"cublasLt64_11");
   if (!cublas_handle) break;
   if (FreeLibrary(cublas_handle) == false) break;
}
```

10.2. Linux

To build applications with SDK on Linux, use one of the following methods:

- At compile time, link to the core SDK library, `libnv_audiofx.so`. For example, with `gcc`:

  ```
  gcc -L "../../nvafx/lib" -l "nv_audiofx" -L "../../external/cuda/lib/" -I
  ↪"../../nvafx/include" source.c
  ```

- Dynamically load the core SDK library, `libnv_audiofx.so`, by using `dlopen`/`dlsym` with the correct library paths set (via `LD_LIBRARY_PATH` or similar). For more information, refer to the `dlopen(3)`/`dlsym(3)` man pages.

  For example:

  ```
  #include <cassert>
  #include <dlfcn.h>
  #include <nvAudioEffects.h>
  #include <nvAFXDenoiser.h>  // From the downloaded Denoiser feature

  // Typedefs for functions; similarly define for other functions required

  typedef NvAFX_Status (*fnNvAFX_CreateEffect)(NvAFX_EffectSelector code,
  ↪NvAFX_Handle* effect);

  // Load library and bind

  // Note: ensure that external/cuda/lib is in library path, or fix via
  ↪RPATH/similar

  void* handle = dlopen("libnv_audiofx.so", RTLD_LAZY);

  assert(handle);

  fnNvAFX_CreateEffect f = (fnNvAFX_CreateEffect) dlsym(handle, "NvAFX_
  ```

(continues on next page)

(continued from previous page)

```
↪CreateEffect");

assert(f);

// Call functions

NvAFX_Handle effect;

auto ret = f(NVAFX_EFFECT_DENOISER, &effect);

assert(ret == NVAFX_STATUS_SUCCESS);
```

Chapter 11. Create an Audio Effect

To create an audio effect, call the *NvAFX_CreateEffect()* function with the following parameters:

- The *NvAFX_EffectSelector* type includes the following values:
  - `NVAFX_EFFECT_DENOISER`
  - `NVAFX_EFFECT_DEREVERB`
  - `NVAFX_EFFECT_DEREVERB_DENOISER`
  - `NVAFX_EFFECT_SUPERRES`
  - `NVAFX_EFFECT_STUDIO_VOICE_HIGH_QUALITY`
  - `NVAFX_EFFECT_STUDIO_VOICE_LOW_LATENCY`
  - `NVAFX_EFFECT_SPEAKER_FOCUS`
- The pointer to the location that stores the handle to the newly created audio effect.

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

The following example creates a denoiser audio effect:

```
#include <nvAudioEffects.h>
#include <nvAFXDenoiser.h>  // From features/nvafxdenoiser/include

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

11.1. Linux

When creating an effect, the SDK attempta to load the effect library (`libnv_audiofx_<effect>.so`, where `<effect>` is the name of the feature), located in the `features/<effect>/lib` folder in the SDK. For example, creating an instance of the Studio Voice Low Latency effect attempts to load `libnv_audiofx_studio_voice_low_latency.so`, following the dynamic loader search path.

This file is normally located at `features/studio_voice/libs/libnv_audiofx_studio_voice_low_latency.so` (using the feature download script). If you download this file to another location, however, you can set the path by using `LD_LIBRARY_PATH` or similar. For example, the `effects_demo` and `effects_delayed_streams_demo` applications find all detected effects by using the `RPATH` of the sample application binary. Similarly, you can use `LD_LIBRARY_PATH` and other methods if the effect library is located in a different location.

For example, if an application has the feature library located under `/usr/lib/afx`, the application can use any of the following approaches:

- Use `LD_LIBRARY_PATH` when loading the application; for example, `LD_LIBRARY_PATH=/usr/lib/afx:$LD_LIBRARY_PATH ./my_app`.
- Use `RPATH/RUNPATH` when linking the application; for example, `g++ my_app.cpp .. -Wl, -rpath=/usr/lib/afx`.
- Register the file in the linker cache file, or use other methods that the dynamic loader/linker provides.

For further details, refer to the man pages of the Linux dynamic loader/linker (`man 8 ld.so`)

11.2. Windows

When creating an effect, the SDK attempts to load the effect library (`features\nvafx<effect>\bin\nvafx<effect>.dll`), where `<effect>` is the name of the feature), located in the `features/<effect>/bin` folder in the SDK. For example, creating an instance of the Studio Voice Low Latency effect attempts to load `nvafxstudiovoicelowlatency.dll` from either the `features\nvafxstudiovoicelowlatency\bin\` folder (located in the SDK root) or the same folder as the core SDK DLL.

Chapter 12. Create a Chained Audio Effect

The SDK supports running multiple effects in a chain where the output from one effect is passed as the input to the second effect without performing unnecessary pre- and post-processing computations. For example, the SDK can chain the denoiser and Superresolution effects, which take in 16-kHz input data, remove the noise from this audio, and upsample it to 48 kHz.

This technique is more efficient than creating two standalone audio effect objects and passing the output of the first object to the next, and also avoids unnecessary device-to-host and host-to-device copies.

The following effect chains are supported by the SDK models:

- Superresolution effect (8 kHz to 16 kHz) + Background Noise Removal effect (16 kHz).
- Superresolution effect (8 kHz to 16 kHz) + Room Echo Removal effect (16 kHz).
- Superresolution effect (8 kHz to 16 kHz) + Combined Background Noise Removal/Room Echo Removal effect (16 kHz).
- Background Noise Removal effect (16 kHz) + Superresolution effect (16 kHz to 48 kHz).
- Room Echo Removal effect (16 kHz) + Superresolution effect (16 kHz to 48 kHz).
- Combined Background Noise Removal/Room Echo Removal effect (16 kHz) + Superresolution effect (16 kHz to 48 kHz).
- Speaker Focus effect (16 kHz) + Background Noise Removal effect (16 kHz).
- Speaker Focus effect (48 kHz) + Background Noise Removal effect (48 kHz).

Note

No other effect chains are supported by the SDK models. Using an unsupported chain by manually chaining individual effects together might result in degraded audio quality.

If you combine the Denoiser effect and the Dereverb effect, use the combined Denoiser+Dereverb model. (For more information, see *About the Noise Removal/Background Noise Suppression Effect*.)

To create a chained effect, call *NvAFX_CreateChainedEffect* with the following parameters:

- **One** of the following effect selectors:

```
NVAFX_CHAINED_EFFECT_DENOISER_16k_SUPERRES_16k_TO_48k
NVAFX_CHAINED_EFFECT_DEREVERB_16k_SUPERRES_16k_TO_48k
```

(continues on next page)

(continued from previous page)

```
NVAFX_CHAINED_EFFECT_DEREVERB_DENOISER_16k_SUPERRES_16k_TO_48k
NVAFX_CHAINED_EFFECT_SUPERRES_8k_TO_16k_DENOISER_16k
NVAFX_CHAINED_EFFECT_SUPERRES_8k_TO_16k_DEREVERB_16k
NVAFX_CHAINED_EFFECT_SUPERRES_8k_TO_16k_DEREVERB_DENOISER_16k
NVAFX_CHAINED_EFFECT_SPEAKER_FOCUS_16k_DENOISER_16k
NVAFX_CHAINED_EFFECT_SPEAKER_FOCUS_48k_DENOISER_48k
```

- The pointer to the location that stores the handle to the newly created audio effect.

When a chained effect is created, the effect loads the effect DLLs for both effects in the chain. For further details, refer to the section *Create an Audio Effect*.

The following example creates a chained audio Background Noise Removal effect (16 kHz) + Superresolution effect (16 kHz to 48 kHz) effect:

```
NvAFX_Status err = NvAFX_CreateChainedEffect(NVAFX_CHAINED_EFFECT_DENOISER_
↪16k_SUPERRES_16k_TO_48k, &handle);
```

Note

Running effects in a chain might impact the performance and latency of the audio pipeline.

Chapter 13. Set the Parameters of an Audio Effect

An audio effect requires a model to transform the input audio. Each model supports a specific audio sample rate. The path to the model file and input audio sample rate (Linux SDK only) must be set in the SDK. After required parameters for the effect are set, the effect can be loaded using *NvAFX_Load*.

The SDK also supports several frame sizes (the number of samples per frame), which can be queried and set in the SDK. (For more information, see *Get the Parameters of an Audio Effect*.)

To set U32 values, call the *NvAFX_SetU32()* function with the following parameters:

- Previously created effect handle.
- The selector string for the parameter to be set:
  - To set the number of samples per input frame, specify `NVAFX_PARAM_NUM_SAMPLES_PER_INPUT_FRAME`.
  - In a Multi-GPU setup, to have SDK automatically select the GPU compatible with the model set in SDK, set the `NVAFX_PARAM_USE_DEFAULT_GPU` parameter to 1 (The default value is `0`). This parameter is not supported by chained effects.
  - The Noise Removal and Room Echo Removal/Room Echo Cancellation effect support VAD, which can indicate whether the audio data frame supplied to the SDK through *NvAFX_Run* contains a voice.

    When `NVAFX_PARAM_ENABLE_VAD` is enabled, this feature also removes low-volume noise and all non-speech data from the *NvAFX_Run* output without degrading performance.

    To enable this feature, set the following parameter to 1. (The default value is 1 for BNR 2.0 and `0` for all other effects.) This parameter is not supported by chained effects.

    `NVAFX_PARAM_ENABLE_VAD` is a U32 selector on Windows x64 and Windows on Arm. Set it with *NvAFX_SetU32()*, not `NvAFX_SetBool()`. Only `0` and 1 are valid:

    ```
    unsigned int enable_vad = 1;
    NvAFX_Status err = NvAFX_SetU32(handle, NVAFX_PARAM_ENABLE_VAD,
                                    enable_vad);
    ```

  - To set the sample rate (not supported by chained effects), specify `NVAFX_PARAM_INPUT_SAMPLE_RATE`. This value must be set before querying the frame sizes supported the effect using *NvAFX_GetU32* or *NvAFX_GetU32List*.
  - (Denoiser v2 only) To set the version of the denoiser effect, specify `NVAFX_PARAM_EFFECT_VERSION`.
  - **Linux only**

        - To set the number of audio streams, specify `NVAFX_PARAM_NUM_STREAMS`.
        - In a multi-GPU environment, to run the constituent effects of a chained effect on separate GPUs, specify `NVAFX_PARAM_CHAINED_EFFECT_GPU_LIST`.
    - **Windows only**
        - To set the output sample rate (not supported by chained effects), specify `NVAFX_PARAM_OUTPUT_SAMPLE_RATE`.
        - To create and manage the CUDA context used by the SDK, set the `NVAFX_PARAM_USER_CUDA_CONTEXT` parameter to 1 (the default value is `0`).
        - To disable CUDA graphs, set the `NVAFX_PARAM_DISABLE_CUDA_GRAPH` parameter to 1 (the default value is 1).
        - To control the TRT-RTX engine cache on Windows on Arm, set `NVAFX_MODEL_CACHE_MODE` before loading the effect: `0` selects automatic mode (the default), 1 disables caching, and 2 forces regeneration.
- An unsigned integer value that specifies the value for the selector.

To set the model, call the *NvAFX_SetString()* function with the following parameters:

- Previously created effect handle.
- A null-terminated string specifying the path to the model file.
    - For Linux:
        - Each model file supports a specific sample rate and a maximum number of audio streams.
        - Model files for specific GPU compute versions are located in the `models/<compute_version>` directory in the SDK.

          The following GPU compute versions can be used for the following GPUs:
            - Turing (T4): `models/sm_75`
            - Ampere
                - A100 (ga100 based GPUs): `models/sm_80`
                - A10 (ga102 or later GPUs): `models/sm_86`
            - ADA (L4/L40): `models/sm_89`
            - Hopper (H100): `models/sm_90`
            - Blackwell
                - B100/B200: `models/sm_100`
                - RTX PRO 6000: `models/sm_120`
        - The specified model should match the sample rate and a specified number of audio streams.
        - The model file name uses the following format: `<effect> <samplerate> <max-streams>.trtpkg`
            - For convenience, each folder includes a symlink, for example `denoise_16k.trtpkg` and `denoiser_48k.trtpkg`, which points to the actual model.
            - `samplerate` can be `8k`, `16k` or `48k`.

- Number of audio streams should be within the range 1 and `max-streams` (both inclusive).
- The model gives the best throughput performance when the number of audio streams is set to 64 or a multiple of 256 (256, 512, 768, and so on).

For example, the `denoiser_48k_1152.trtpkg` model can be used for 48 kHz and between 1 to 1152 audio streams but will be optimal for 64, 256, 512, 768, and 1024 streams. Code that uses this model can also directly use the symlink `denoiser_48k.trtpkg` in the same folder, which allows the underlying model to be changed without code changes.

- For Windows:
  - Each model file supports a specific sample rate.
  - Model files for specific GPU compute versions are located in the models directory in the SDK.
  - On x64, models are organized by GPU architecture. On NVIDIA N1X (ARM64), Windows on Arm models are stored in `models/blackwell` and are selected automatically by the packaged helper scripts.
  - After loading a model, query `NVAFX_PARAM_INPUT_SAMPLE_RATE` and `NVAFX_PARAM_OUTPUT_SAMPLE_RATE` with *NvAFX_GetU32()*. Applications must reject input audio whose rate does not match the loaded model rather than silently processing it at a different configured rate.

- For chained effects, call the *NvAFX_SetStringList* function with the following parameters:
  - The previously created effect handle.
  - An array of null-terminated strings, each specifying the path to the model file of the effect to be chained.

    For example, for a Denoiser 16k + Superres 16k to 48k chain, an array that contains two paths should be passed in the following paths:
    - To the 16k Denoiser model.
    - To the 16k to 48k Superres model. The model paths should follow the same conventions as the conventions of the standalone effect.
  - The length of the array.

For example, the following platform-neutral code sets the model path:

```
NvAFX_Status err;

err = NvAFX_SetString(handle, NVAFX_PARAM_MODEL_PATH, model_file.c_str());
```

On Linux, set the input sample rate and stream count separately before loading the model:

```
err = NvAFX_SetU32(handle, NVAFX_PARAM_INPUT_SAMPLE_RATE, sample_rate);
err = NvAFX_SetU32(handle, NVAFX_PARAM_NUM_STREAMS, num_streams);
```

The following example is for the Windows ARM64 package only. The x64 headers do not expose these TRT-RTX cache selectors. On Windows on Arm, you can optionally configure the cache before calling *NvAFX_Load*. An empty directory selects the default `cache` directory in the same directory as the model:

```
// Windows ARM64 only; do not compile this block against x64 SDK headers.
#if defined(_M_ARM64)
unsigned int model_cache_mode = 0;  // Auto
err = NvAFX_SetU32(handle, NVAFX_MODEL_CACHE_MODE, model_cache_mode);

std::string model_cache_directory;
err = NvAFX_SetString(handle, NVAFX_MODEL_CACHE_DIRECTORY,
                      model_cache_directory.c_str());
#endif
```

To use microphone profiles in Studio Voice Low Latency, set the companion model path before loading the effect. After the effect is loaded, set the profile ID:

```
NvAFX_SetString(handle, NVAFX_PARAM_MIC_PROFILE_MODEL, mic_profile_model_file.
↪c_str());
NvAFX_Load(handle);
NvAFX_SetU32(handle, NVAFX_PARAM_MIC_PROFILE_ID,
             static_cast<unsigned int>(NVAFX_MIC_PROFILE_WARM));
```

To disable the profile, set `NVAFX_PARAM_MIC_PROFILE_ID` to `static_cast<unsigned int>(NVAFX_MIC_PROFILE_PASSTHROUGH)`. Values outside the range -1 through 5 are invalid.

Chapter 14. Get the Parameters of an Audio Effect

The number of channels in input/output audio are fixed for the Audio Effect and cannot be changed. Before running an audio effect, the input and output sample rates and number of channels supported by the effect must be queried.

The SDK also supports several frame sizes (number of samples per frame), which can be queried and set by using the set API. (For more information, see *Set the Parameters of an Audio Effect*.) The application can also query and use the default frame size supported by the SDK, as demonstrated in the following example.

Chained effects currently support *only* a 10ms frame size.

Note

Effect parameters (except for supported frame size list) can be queried only after the effect is loaded.

Querying parameters before model load might return invalid values or the function might fail with an error code.

To ensure that the sample rate of the input audio is compatible with the effect, query the sample rate first.

To query these parameters, call the *NvAFX_GetU32()* function with the following parameters:

- Previously created effect handle.
- The selector string for the parameter to be queried:
  - To get the default number of samples per input frame, specify `NVAFX_PARAM_NUM_SAMPLES_PER_INPUT_FRAME`.
  - To get the default number of samples per output frame per channel, specify `NVAFX_PARAM_NUM_SAMPLES_PER_OUTPUT_FRAME`.

    The number of samples for each input frame and output frame is the same for all effects except Superresolution. In the Superresolution effect, output frames will contain more data than input frames.
  - To get the number of channels in input/output audio, specify `NVAFX_PARAM_NUM_INPUT_CHANNELS/NVAFX_PARAM_NUM_OUTPUT_CHANNELS`.
  - To get the sample rate for input/output audio, specify `NVAFX_PARAM_INPUT_SAMPLE_RATE/NVAFX_PARAM_OUTPUT_SAMPLE_RATE`.

  - To get the TRT-RTX cache mode on Windows on Arm, specify `NVAFX_MODEL_CACHE_MODE`.
- A pointer to the location where the value will be stored.

To get the resolved TRT-RTX cache directory on Windows on Arm, call *NvAFX_GetString()* with `NVAFX_MODEL_CACHE_DIRECTORY` after the effect is loaded.

To query lists, the user must first query the list size and allocate memory for the output and then pass in the newly allocated memory and size to *NvAFX_GetU32List()*.

To query the list size, call the *NvAFX_GetU32List()* function with the following parameters:

- Previously created effect handle.
- The selector string for the parameter to be queried.

  To get the list of the number of supported samples per frame, specify `NVAFX_PARAM_SUPPORTED_NUM_SAMPLES_PER_FRAME`.

  Input sample rate must be set before querying this list.
- An output pointer, set to `nullptr` (or `NULL`) to query size.
- A pointer to the location where the size of the list is to be stored. The size should be initialized to zero and will be updated with the actual size when this function is called.

  The preceding *NvAFX_GetU32List()* call retrieves the size of the list for the corresponding parameter selector with an `NVAFX_STATUS_OUTPUT_BUFFER_TOO_SMALL` error status. To obtain the actual list values, allocate memory for the list with the returned size and call the *NvAFX_GetU32List()* function again with the following parameters:

  - The selector string for the parameter to be queried. To get the list of supported number of samples per frame, specify `NVAFX_PARAM_SUPPORTED_NUM_SAMPLES_PER_FRAME`.
  - A pointer to a U32 array of size at least of the list size retrieved from the above call. The list values are written to this array.
  - A pointer to a location where the value of the size of the list is stored.

The VAD status for the last *NvAFX_Run* call can also be queried by using the *NvAFX_GetBoolList()* function. This query can be helpful when the audio output pipeline has an alternative packet loss concealment algorithm.

The following example queries an effect for the supported number of samples per frame, the number of channels in input/output audio, the sample rate, and the supported frame sizes:

```
NvAFX_Status err;

std::unique_ptr<unsigned int[]> supported_list = nullptr;
unsigned int list_size = 0;

// Query with list_size set to zero to get the actual list size
err = NvAFX_GetU32List(handle, NVAFX_PARAM_SUPPORTED_NUM_SAMPLES_PER_FRAME,
↪supported_list.get(), &list_size);
if (err != NVAFX_STATUS_OUTPUT_BUFFER_TOO_SMALL) {
            // This indicates API failure
            return;
}

// Allocate memory based on the returned size, and query with this pointer
supported_list.reset(new unsigned int[list_size]);
```

(continues on next page)

(continued from previous page)

```
err = NvAFX_GetU32List(handle,
NVAFX_PARAM_SUPPORTED_NUM_SAMPLES_PER_FRAME, supported_list.get(), &list_
↪size);

// Load model
unsigned num_samples_per_frame, num_channels, sample_rate;
err = NvAFX_GetU32(handle, NVAFX_PARAM_NUM_SAMPLES_PER_INPUT_FRAME, &num_
↪samples_per_frame);
err = NvAFX_GetU32(handle, NVAFX_PARAM_NUM_INPUT_CHANNELS, &num_channels);
err = NvAFX_GetU32(handle, NVAFX_PARAM_OUTPUT_SAMPLE_RATE, &sample_rate);

// Querying VAD results

// VAD must be supported and enabled on the handle
NvAFX_Run(...); // Process input first

// Query results for last input

// Note: If no voice is detected, output audio is zeroed by the effect
// However, result is also returned (in case custom packet loss concealment
↪is desired).
     // Returns NVAFX_TRUE if input audio had voice, NVAFX_FALSE otherwise

std::vector<NvAFX_Bool> out(num_streams, 0);
uint32_t s = out.size();
auto status = NvAFX_GetBoolList(handle, NVAFX_PARAM_VAD_RESULT, out.data(), &
↪s);
assert(status == NVAFX_STATUS_SUCCESS);

// Use VAD result
```

Chapter 15. Get Supported Devices on Windows

On Windows, the *NvAFX_GetSupportedDevices()* function can be used to determine the GPUs supported by the currently selected model. This function is not supported for chained effects.

Note

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

This function should be called with the following parameters:

- The effect handle.
- The size of the input array. If the call succeeds, this value is set by the function.
- An array with `numSupportedDevices` elements.

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

The first sizing call intentionally returns `NVAFX_STATUS_OUTPUT_BUFFER_TOO_SMALL` while setting `numSupportedDevices`. Treat that status as the successful sizing result, and then require `NVAFX_STATUS_SUCCESS` from the second call. This contract is identical on Windows x64 and Windows on Arm.

The following example fetches the list of supported GPUs for the model:

```
int numSupportedDevices = 0;
NvAFX_Status status =
    NvAFX_GetSupportedDevices(handle, &numSupportedDevices, nullptr);
if (status != NVAFX_STATUS_OUTPUT_BUFFER_TOO_SMALL ||
    numSupportedDevices <= 0) {
    // Handle a model-path, device-discovery, or API error.
    return;
}

std::vector<int> devices(numSupportedDevices);
status = NvAFX_GetSupportedDevices(handle, &numSupportedDevices,
                                   devices.data());
if (status != NVAFX_STATUS_SUCCESS) {
    // Handle the device-list query failure.
```

(continues on next page)

(continued from previous page)

```
    return;
}
```

Chapter 16. Load, Run, and Destroy an Audio Effect

16.1. Load an Audio Effect

Loading an effect involves validating the parameters that were set for the effect and loading the specified model into GPU memory.

To load an audio effect, set the parameters for the effect (as described in *Set the Parameters of an Audio Effect*) and call *NvAFX_Load()* with the effect handle:

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

If the function succeeds, `NVAFX_STATUS_SUCCESS` is returned. Otherwise, an error code is returned.

16.2. Run an Audio Effect

After the effect is loaded, it can be applied to input audio by calling the *NvAFX_Run()* function. When the 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.

Note

Input/output memory buffers are in CPU memory. Copying to or from GPU memory is handled internally by the SDK.

To run an audio effect, call the *NvAFX_Run()* function with the following parameters:

- Previously created effect handle.
- The input memory buffer.
- The output memory buffer.

  For the Super Resolution effect, the size of input and output memory buffer will differ and should be queried by the user using `NVAFX_PARAM_NUM_SAMPLES_PER_OUTPUT_FRAME` and `NVAFX_PARAM_NUM_SAMPLES_PER_INPUT_FRAME`.
- The number of samples per frame per stream of input data.
- The number of channels in input audio.

For more information, refer to *Get the Parameters of an Audio Effect*.

The following example runs an audio effect:

```
// input_samples and output_samples point to allocated frame buffers.
const float* input_buffers[] = { input_samples };
float* output_buffers[] = { output_samples };
NvAFX_Status err = NvAFX_Run(handle, input_buffers, output_buffers,
                             num_samples, num_channels);
```

16.3. Run Multiple Audio Effects in a Chain

The following example demonstrates running two effects in a chain:

```
NvAFX_Status err;
NvAFX_Handle chained_handle = nullptr;
err = NvAFX_CreateChainedEffect(
   NVAFX_CHAINED_EFFECT_DENOISER_16k_SUPERRES_16k_TO_48k,
   &chained_handle);

// Set effect parameters & load effect
const float* input_buffers[] = { input_audio };
float* output_buffers[] = { output_audio };
err = NvAFX_Run(chained_handle, input_buffers, output_buffers,
                num_samples, num_channels);
```

Note

Running effects in a chain might impact the performance and latency of the audio pipeline.

16.4. Run an Audio Effect on Delayed Audio Streams on Linux

The Linux SDK supports cases where some streams do not arrive at the expected time, such as when the audio is being processed in real-time. These streams are referred to as delayed streams. To support handling these streams, the SDK allows applications to specify a list that indicates whether the corresponding stream is currently active or delayed/inactive.

The list can be set by calling *NvAFX_SetBoolList()* with the following function parameters:

- Previously created effect handle.
- The selector string `NVAFX_PARAM_ACTIVE_STREAMS` for the parameter that will be set.
- An array of type `NVAFX_BOOL` in which each element represents the status of the corresponding audio stream, with `NVAFX_TRUE` indicating an active stream and `NVAFX_FALSE` indicating an inactive stream.
- The length of the preceding array (equal to the number of audio streams).

Note

This functionality is not currently supported by Speaker Focus or Studio Voice.

For delayed audio streams, the effect can be initially applied on all delayed audio streams by setting them as active and setting the on-time audio streams as inactive. This should be followed by one or more *NvAFX_Run()* calls to apply the effect on the delayed audio streams. After the delayed audio streams are processed, the on-time audio streams are set to active, and *NvAFX_Run()* is executed once to apply the effect.

The following example demonstrates the procedure for processing four streams:

1. Consider an effect that accepts 10ms audio inputs.
2. Audio streams 1 and 3 are delayed by 10ms each and arrive with 20ms worth of data.
3. Audio streams 2 and 4 are on time and arrive with 10ms of data.
4. Streams can be processed in either of the following ways:
   - **Option 1:** Process the extra 10ms *only* in the delayed streams and then process on-time 10ms data for *all* streams. Initially, by using *NvAFX_SetBoolList*, streams 1 and 3 are set as active, and 2 and 4 are set as inactive:
     a. An *NvAFX_Run* call is executed where 10ms of data from streams 1 and 3 is populated in the input while the rest of the input is set to 0. This step processes the extra 10ms of data in streams 1 and 3.
     b. A second *NvAFX_SetBoolList* call is executed to set all streams (1, 2, 3, and 4) as active.
     c. An *NvAFX_Run* call is executed with the real-time 10ms data from all four streams.
   - **Option 2:** Process 10ms in *all* streams and then process the extra 10ms data *only* in delayed streams:
     a. Process 10ms of data from all streams (stale data from stream 1 and 3 and new data from stream 2 and 4) by calling *NvAFX_Run*.
     b. Set streams 1 and 3 to active and 2 and 4 to inactive by calling *NvAFX_SetBoolList*.
     c. Process the extra 10ms from stream 1 and 3 by calling *NvAFX_Run*.

The following example runs an audio effect after setting some of the audio streams as inactive:

```
NvAFX_Status err = NvAFX_SetBoolList(handle, NVAFX_PARAM_ACTIVE_STREAMS,
↪stream_active_list, num_streams);

const float* input_buffers[] = { input };
float* output_buffers[] = { output };
err = NvAFX_Run(handle, input_buffers, output_buffers, num_samples, num_
↪channels);
```

The internal state of each stream is updated during each *NvAFX_Run* call only for active streams. Setting a stream to inactive disables updating this state. If required, this state can also be reset by using *NvAFX_Reset*.

16.5. Destroy an Audio Effect

When an audio effect is no longer required, it should be destroyed to free the resources and memory used by the effect.

To destroy an audio effect, call *NvAFX_DestroyEffect()* and specify the handle to the effect to be destroyed:

```
NvAFX_Status err = NvAFX_DestroyEffect(handle);
```

Chapter 17. Use Multiple GPUs

Applications that are developed with the NVIDIA Audio Effects SDK can be used with multiple GPUs. By default, the SDK assumes that the application will set the GPU. Optionally, the SDK can select the best GPU to run the effects.

17.1. Select the GPU for Audio Effects Processing in a Multi-GPU Environment

The GPU to be used to run audio effects in a multi-GPU environment can be controlled by using the `cudaSetDevice()` and `cudaGetDevice()` CUDA functions.

The device should be set *before* *NvAFX_Load()* is called because *NvAFX_Load()* will succeed only when the currently selected GPU supports the SDK:

```
int chosenGPU = 0; // or whichever GPU you want to use
cudaSetDevice(chosenGPU);
NvAFX_Handle effect = nullptr;
NvAFX_Status err = NvAFX_CreateEffect(code, &effect);
err = NvAFX_Set...; // set parameters
...
err = NvAFX_Load(effect);
...
err = NvAFX_Run(effect, ...);
```

17.2. Select GPUs for Chained Audio Effects on Linux

When using chained effects in a multi-GPU environment on Linux, the SDK can optionally run the effects in the chain on separate GPUs. For example, in a Denoiser 16k + Superres 16k to 48k chain, the denoiser effect can be run entirely on one GPU and the Superres effect on another GPU.

To use this feature, create the chained effect and set `NVAFX_PARAM_CHAINED_EFFECT_GPU_LIST` to an array that specifies the GPU IDs by using *NvAFX_SetU32List*. This parameter must be set *before* you call *NvAFX_Load* on the effect.

The following sample demonstrates use of this parameter:

```
NvAFX_Handle effect = nullptr;
NvAFX_Status err = NvAFX_CreateChainedEffect(code, &effect);
...

// Run first effect on GPU id 3, second on GPU id 4
uint32_t gpus[] = {3, 4};
err = NvAFX_SetU32List(effect, NVAFX_PARAM_CHAINED_EFFECT_GPU_LIST, gpus, 2);
...
err = NvAFX_Load(effect);
...
```

17.3. Offload GPU Selection to the SDK for Audio Effects Processing in a Multi-GPU Environment

In a multi-GPU environment, the SDK can optionally determine the optimal GPU on which to run the audio effects. To use this feature, call *NvAFX_SetU32* with the parameters `NvAFX_SetU32(effect, NVAFX_PARAM_USE_DEFAULT_GPU, 1)` *before* loading effects. If called after an audio effect is loaded, this function has no effect.

Note

`NVAFX_PARAM_USE_DEFAULT_GPU` and `NVAFX_PARAM_USER_CUDA_CONTEXT` (Windows SDK only) cannot be used at the same time.

If the application sets `NVAFX_PARAM_USE_DEFAULT_GPU` to `0` (or does not set this parameter), the SDK does not explicitly select the GPU to run the effect. The application can set the device on which SDK calls are executed by using `cudaSetDevice()` to set the device. If this parameter is not set or is set to `0`, the SDK uses the default device (device `0`).

Note

This parameter is not supported for chained effects.

If the application sets `NVAFX_PARAM_USE_DEFAULT_GPU` to `1`, the application should not call `cudaSetDevice()`, and all other effects or multiple instances of an effect will use the GPU determined by the SDK. If the application does explicitly call `cudaSetDevice()` before calling:ref:*NvAFX_Load() <nvafx-load>*, the SDK might override the application's device preference. If the client calls `cudaSetDevice()` to set the GPU to a different GPU just before calling *NvAFX_Run()*, the *NvAFX_Run()* call will fail:

```
NvAFX_Handle effect = nullptr;
NvAFX_Status err = NvAFX_CreateEffect(code, &effect);
err = NvAFX_SetU32(effect, NVAFX_PARAM_USE_DEFAULT_GPU, 1);
...
err = NvAFX_Load(effect);
...
```

17.4. Select GPUs for Various Tasks

The applications that use the SDK might be designed to perform multiple tasks in a multi-GPU environment in addition to applying the audio effect filter. In this situation, the best GPU for each task should be selected before calling *NvAFX_Load()* and be set before each *NvAFX_Run()* call.

Switching to the appropriate GPU is the responsibility of the application. If the application does not switch to the appropriate GPU before calling *NvAFX_Run()*, the call will fail with an error.

The following steps demonstrate how to complete CUDA tasks and SDK calls on different GPUs.

1. Call `cudaGetDeviceCount()` to determine the number of available GPUs:

   ```
   cuErr = cudaGetDeviceCount(&deviceCount);
   ```

2. Determine the best GPU for the task.

   For example, this can be determined by iterating over the available GPUs and selecting the GPU with the highest number of SMs by using `cudaGetDeviceProperties()`.

3. In the loop that completes the application's tasks, call `cudaSetDevice(gpuOtherTask)` before the other CUDA workload, and then call `cudaSetDevice(gpuAFX)` immediately before `NvAFX_Run()`. This sequence is illustrative application logic; the application supplies its own loop, audio buffers, error handling, and task implementation.

17.5. CUDA Graph Support on Windows

The Windows SDK supports using CUDA graphs, which improve performance by reducing the CPU overheads seen with short-lived CUDA kernels.

By default, graphs are enabled in the Windows SDK, but this can cause issues if the SDK runs in parallel with other applications that are using CUDA graphs. The following example shows how to disable CUDA graphs:

```
// Call before loading model (NvAFX_Load) on Windows
err = NvAFX_SetU32(effect, NVAFX_PARAM_DISABLE_CUDA_GRAPH, 1);
...
```

Chapter 18. Audio Effects SDK API Reference

This section provides detailed information about the APIs in the NVIDIA® AFX SDK.

18.1. Type Definitions

This section describes the type definitions used in the AFX SDK.

18.1.1. NvAFX_EffectSelector

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

```
typedef const char* NvAFX_EffectSelector;
```

The currently supported selectors are:

`NVAFX_EFFECT_DENOISER` : `"denoiser"`

Denoiser audio effect (see *About the Noise Removal/Background Noise Suppression Effect*).

`NVAFX_EFFECT_DEREVERB` : `"dereverb"`

De-reverb effect (see *About the Room Echo Removal/Room Echo Cancellation Effect*).

`NVAFX_EFFECT_DEREVERB_DENOISER` : `"dereverb_denoiser"`

Combined De-reverb and Denoiser effects (see *About the Noise Removal/Background Noise Suppression Effect*).

`NVAFX_EFFECT_SUPERRES` : `"superres"`

Audio Super-Resolution effect (see *About the Audio Super-Resolution Effect*).

`NVAFX_EFFECT_STUDIO_VOICE_HIGH_QUALITY` : `"studio_voice_high_quality"`

Studio Voice High Quality effect (see *About the Studio Voice Effect*).

`NVAFX_EFFECT_STUDIO_VOICE_LOW_LATENCY` : `"studio_voice_low_latency"`

Studio Voice Low Latency effect (see *About the Studio Voice Effect*).

`NVAFX_EFFECT_SPEAKER_FOCUS` : `"speaker_focus"`

Speaker Separation effect (see *About the Speaker Focus Effect*).

`NVAFX_CHAINED_EFFECT_DENOISER_16k_SUPERRES_16k_TO_48k`

Chained effect (Denoiser 16k + Superres 16k to 48k)

`NVAFX_CHAINED_EFFECT_DEREVERB_16k_SUPERRES_16k_TO_48k`

Chained effect (Dereverb 16k + Superres 16k to 48k)

`NVAFX_CHAINED_EFFECT_DEREVERB_DENOISER_16k_SUPERRES_16k_TO_48k`

Chained effect (Dereverb+Denoiser 16k + Superres 16k to 48k)

`NVAFX_CHAINED_EFFECT_SUPERRES_8k_TO_16k_DENOISER_16k`

Chained effect (Superres 8k to 16k + Denoiser 16k)

`NVAFX_CHAINED_EFFECT_SUPERRES_8k_TO_16k_DEREVERB_16k`

Chained effect (Superres 8k to 16k + Dereverb 16k)

`NVAFX_CHAINED_EFFECT_SUPERRES_8k_TO_16k_DEREVERB_DENOISER_16k`

Chained effect (Superres 8k to 16k + Dereverb+Denoiser 16k)

`NVAFX_CHAINED_EFFECT_SPEAKER_FOCUS_16k_DENOISER_16k`

Chained effect (Speaker Focus 16k + Denoiser 16k)

`NVAFX_CHAINED_EFFECT_SPEAKER_FOCUS_48k_DENOISER_48k`

Chained effect (Speaker Focus 48k + Denoiser 48k)

18.1.2. NvAFX_ParameterSelector

This type definition provides selector strings for audio effect parameters:

```
typedef const char* NvAFX_ParameterSelector;
```

The currently supported selectors are:

`NVAFX_PARAM_MODEL_PATH` : `"model_path"`

A character string that specifies the path to the model file for the audio effect.

`NVAFX_PARAM_MIC_PROFILE_MODEL` : `"mic_profile_model"` **(Studio Voice Low Latency only)**

A character string that specifies the path to the microphone-profile companion model. Set this optional parameter before calling *NvAFX_Load*.

`NVAFX_PARAM_MIC_PROFILE_ID` : `"mic_profile_id"` **(Studio Voice Low Latency only)**

An unsigned integer that selects a microphone profile. Pass `static_cast<unsigned int>(NVAFX_MIC_PROFILE_PASSTHROUGH)` for passthrough (the default), or select a profile from `NVAFX_MIC_PROFILE_BRIGHT` through `NVAFX_MIC_PROFILE_WARM`. The supported profiles are:

Constant	ID	Profile	Character
NVAFX_MIC_PROFILE_PASSTHROUG	-1	Passthrough	Microphone profile disabled.
NVAFX_MIC_PROFILE_BRIGHT	0	Bright	Clear and sharp, with audible detail.
NVAFX_MIC_PROFILE_VIBRANT	1	Vibrant	Rich sound with added character.
NVAFX_MIC_PROFILE_FLAT	2	Flat	Natural, balanced sound.
NVAFX_MIC_PROFILE_FULL	3	Full	Rich, deep, clear sound.
NVAFX_MIC_PROFILE_THIN	4	Thin	Light sound with less depth.
NVAFX_MIC_PROFILE_WARM	5	Warm	Soft, smooth, bass-heavy sound.

`NVAFX_PARAM_INPUT_SAMPLE_RATE` : "`input_sample_rate`"

An unsigned integer that specifies the input audio sample rate for the audio effect.

`NVAFX_PARAM_OUTPUT_SAMPLE_RATE` : "`output_sample_rate`"

An unsigned integer that specifies the output audio sample rate for the audio effect.

`NVAFX_PARAM_NUM_INPUT_CHANNELS` : "`num_input_channels`"

An unsigned integer that specifies the number of audio channels for the Audio effect.

`NVAFX_PARAM_NUM_OUTPUT_CHANNELS` : "`num_output_channels`"

An unsigned integer that specifies the number of output audio channels for the Audio effect.

`NVAFX_PARAM_NUM_STREAMS` : "`num_streams`"

An unsigned integer that specifies the number of audio streams to be processed by the audio effect.

`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_EFFECT_VERSION` : "`effect_version`"

An integer value that specifies which version of the effect will be used. Supported only on the Denoiser model (BNR 2.0).

`NVAFX_PARAM_NUM_SAMPLES_PER_INPUT_FRAME` : "`num_samples_per_input_frame`"

An unsigned integer that specifies the number of samples per input frame per audio stream for the audio effect.

`NVAFX_PARAM_NUM_SAMPLES_PER_OUTPUT_FRAME` : "`num_samples_per_output_frame`"

An unsigned integer that specifies the number of samples per output frame per audio stream for the audio effect.

`NVAFX_PARAM_SUPPORTED_NUM_SAMPLES_PER_FRAME` : "`supported_num_samples_per_frame`"

A list of U32 values specifying the supported values for number of samples per frame. This value can be queried before model load after the input sample rate is set.

Linux SDK only:

`NVAFX_PARAM_ACTIVE_STREAMS` : `"active_streams"`

A list of NvAFX_Bool values that specify whether the corresponding stream is active.

`NVAFX_PARAM_CHAINED_EFFECT_GPU_LIST` : `"chained_effect_gpu_list"`

A list of U32 values specifying the GPUs to be used in chained effects. Each effect in the chain will use the corresponding GPU from the list.

Windows SDK Only:

`NVAFX_MODEL_CACHE_DIRECTORY` : `"ModelCacheDir"` **(Windows on Arm only)**

A character string that specifies the TRT-RTX engine-cache directory. Set it before calling *NvAFX_Load*. If it is empty or is not set, the SDK uses a `cache` directory in the same directory as the model file.

`NVAFX_MODEL_CACHE_MODE` : `"ModelCacheMode"` **(Windows on Arm only)**

An unsigned integer that controls the TRT-RTX engine cache. Set `0` for automatic cache reuse and generation (the default), 1 to disable the cache, or 2 to force cache regeneration. Set it before calling *NvAFX_Load*.

`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`. This value cannot be changed after the model is loaded for that particular session.

`NVAFX_PARAM_DISABLE_CUDA_GRAPH` : `"disable_cuda_graph"`

An unsigned integer value that specifies whether to enable CUDA graphs (enabled by default).

- To **disable** CUDA graphs, set this value to 1.
- To **enable** CUDA graphs, set this value to `0`.

For more information about CUDA Graphs, refer to Getting Started with CUDA Graphs.

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

The following selectors 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"`

18.1.3. NvAFX_Handle

An opaque handle that is associated with an instance of an audio effect:

```
typedef void* NvAFX_Handle;
```

18.1.4. NvAFX_Bool

This type definition is set to `NVAFX_TRUE` to represent `true`, else `NVAFX_FALSE` to represent `false`:

```
typedef char NvAFX_Bool;
```

18.1.5. Logging_cb_t

A callback function type that is used in the *NvAFX_InitializeLogger* API.

Linux SDK:

```
typedef void (*logging_cb_t)(LoggingSeverity level, const char* log,
                             void* userdata);
```

Windows SDK:

```
typedef void (*logging_cb_t)(void* user_data, const char* msg);
```

18.1.6. LoggingSeverity

Levels of LoggingSeverity used in the NvAFX_InitializeLogger API. Refer to *NvAFX_InitializeLogger* for more information:

```
typedef enum LoggingSeverity_t {
   LOG_LEVEL_NONE = -1,
   LOG_LEVEL_FATAL = 0,
   LOG_LEVEL_ERROR = 1,
   LOG_LEVEL_WARNING = 2,
   LOG_LEVEL_INFO = 3,
} LoggingSeverity;
```

18.1.7. LoggingTarget

Logging target used in NvAFX_InitializeLogger API. Refer to *NvAFX_InitializeLogger* for more information:

```
typedef enum LoggingTarget_t {
   LOG_TARGET_NONE = -1,
   LOG_TARGET_STDERR = 0,
   LOG_TARGET_FILE = 1,
   LOG_TARGET_CALLBACK = 2,
} LoggingTarget;
```

18.2. Functions

This section describes the functions provided in the AFX SDK.

18.2.1. NvAFX_CreateEffect

This function creates an effect instance:

```
NvAFX_Status NvAFX_CreateEffect(
   NvAFX_EffectSelector code,
   NvAFX_Handle* effect
);
```

18.2.1.1 Parameters

`code` *[in]*

Type: `NvAFX_EffectSelector`

The selection string for the type of audio effect to be created. Refer to *NvAFX_EffectSelector* for more information about the allowed selection strings.

`effect` *[out]*

Type: `NvAFX_Handle*`

The pointer to the location where the handle to the newly created audio effect instance will be stored.

18.2.1.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.1.3 Remarks

This function creates an instance of the specified type of audio effect and returns the handle to this effect via the effect output parameter.

18.2.2. NvAFX_CreateChainedEffect

This function creates a chained effect instance:

```
NvAFX_Status NvAFX_CreateChainedEffect(
   NvAFX_EffectSelector code,
   NvAFX_Handle* effect
);
```

18.2.2.1 Parameters

`code` *[in]*

Type: `NvAFX_EffectSelector`

The selection string for the type of chained audio effect to be created. For more information about the allowed selection strings, refer to *NvAFX_EffectSelector*.

`effect` *[out]*

Type: NvAFX_Handle*

The pointer to the location where the handle to the newly created audio effect instance will be stored.

18.2.2.2 Return Value

NVAFX_STATUS_SUCCESS on success.

18.2.2.3 Remarks

This function creates an instance of the specified type of chained audio effect and returns the handle to this effect via the effect output parameter.

18.2.3. NvAFX_DestroyEffect

This function destroys an effect instance:

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

18.2.3.1 Parameters

effect *[in]*

Type: NvAFX_Handle

The handle to the audio effect instance to be destroyed.

18.2.3.2 Return Value

NVAFX_STATUS_SUCCESS on success.

18.2.3.3 Remarks

This function destroys the audio effect instance with the specified handle and frees all resources and memory used by that instance.

18.2.4. NvAFX_SetString

This function sets a string parameter of the specified effect:

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

18.2.4.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect instance.

`param_name` *[in]*

Type: `NvAFX_ParameterSelector`

For the list of allowed options, see *Set the Parameters of an Audio Effect*.

`val` *[in]*

Type: `char*`

Pointer to the character string to be set.

18.2.4.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.4.3 Remarks

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

18.2.5. NvAFX_SetStringList

This function sets a string parameter of the specified effect:

```
NvAFX_Status NvAFX_SetStringList(
   NvAFX_Handle effect,
   NvAFX_ParameterSelector param_name,
   const char** list, unsigned int list_size
);
```

18.2.5.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect instance.

`param_name` *[in]*

Type: `NvAFX_ParameterSelector`

For the list of allowed options, see *Set the Parameters of an Audio Effect*.

`list` *[in]*

Type: `char**`

Pointer to the character array to be set.

`list_size` *[in]*

Type: `unsigned int`

Size of the character array to be set.

18.2.5.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.5.3 Remarks

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

18.2.6. NvAFX_SetU32

This function sets a unit parameter of the specified effect:

```
NvAFX_Status NvAFX_SetU32(
   NvAFX_Handle effect,
   NvAFX_ParameterSelector param_name,
   unsigned int val
);
```

18.2.6.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect.

`param_name` *[in]*

Type: `NvAFX_ParameterSelector`

For the list of allowed options, see *Set the Parameters of an Audio Effect*.

Any other selector string returns an error.

Note

Valid values for setting `NVAFX_PARAM_NUM_SAMPLES_PER_INPUT_FRAME` can be queried using `NvAFX_GetU32List()` function with `NVAFX_PARAM_SUPPORTED_NUM_SAMPLES_PER_FRAME` as the selector. Setting any other value would result in an error.

`val` *[in]*

Type: `unsigned int`

Value to be set for the parameter.

18.2.6.2 Return Value

NVAFX_STATUS_SUCCESS on success.

18.2.6.3 Remarks

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

18.2.7. NvAFX_GetString

This function gets the current value of the set string parameter of the specified effect:

```
NvAFX_Status NvAFX_GetString(
   NvAFX_Handle effect,
   NvAFX_ParameterSelector param_name,
   char* val,
   int max_length
);
```

18.2.7.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect instance.

`param_name` *[in]*

Type: `NvAFX_ParameterSelector`

For the list of allowed options, see *Get the Parameters of an Audio Effect*.

`val` *[out]*

Type: `char*`

The address of the buffer where the requested character string would be stored. This buffer must be allocated by and freed by the caller.

`max_length` *[i*

Type: `int`

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

18.2.7.2 Return Value

NVAFX_STATUS_SUCCESS on success.

18.2.7.3 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.

18.2.8. 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
   );
```

18.2.8.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect instance.

`param_name` *[in]*

Type: `NvAFX_ParameterSelector`

For the list of allowed options, see *Get the Parameters of an Audio Effect*. Any other selector string returns an error.

Note

Effect parameters (except for supported frame size list) can be queried only after the effect is loaded. Querying parameters before model load may return invalid values/function may fail with error code.

For SDK, while `NVAFX_PARAM_NUM_SAMPLES_PER_INPUT_FRAME` can be queried after model load using this API (returns the default number of samples per frame, if not set), you should use `NvAFX_GetU32List()` with the `NVAFX_PARAM_SUPPORTED_NUM_SAMPLES_PER_FRAME` parameter to get the list of supported values. You can then use `NvAFX_SetU32()` with the `NVAFX_PARAM_NUM_SAMPLES_PER_FRAME` parameter to set the value.

`val` *[out]*

Type: `unsigned int*`

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

18.2.9. NvAFX_GetU32List

This function gets the uint list parameter values for the specified effect:

```
NvAFX_Status NvAFX_GetU32List(
   NvAFX_Handle effect,
   NvAFX_ParameterSelector param_name,
   unsigned int* list,
   unsigned int* list_size
   );
```

18.2.9.1 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 specified by val.

18.2.9.2 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect instance.

`param_name` *[in]*

Type: `NvAFX_ParameterSelector`

The only selector supported for this function is `NVAFX_PARAM_SUPPORTED_NUM_SAMPLES_PER_FRAME`.

Any other selector string returns an error.

Note

Values returned for `NVAFX_PARAM_SUPPORTED_NUM_SAMPLES_PER_FRAME` as the selector depends on the sample rate. Sample rate must be set using `NvAFX_SetU32()` with the `NVAFX_PARAM_SAMPLE_RATE` selector before making this call.

`list` *[out]*

Type: `unsigned int*`

The address to a list containing the 32-bit unsigned values for the given selector.

Note

The application needs to call this API with `list_size` initialized to zero, and list set to `nullptr` to get the size of list to be allocated. The size will be returned in `list_size` parameter.The application can then allocate an U32 array of at least `list_size` and call the API again with list pointing to the array. For an example, see *Create an Audio Effect*.

`list_size` *[out]*

Type: `unsigned int*`

Pointer to an integer that contains the number of values returned in the list.

18.2.9.3 Return Values

- `NVAFX_STATUS_SUCCESS` on success.
- `NVAFX_STATUS_OUTPUT_BUFFER_TOO_SMALL` when the `list_size` is less than the minimum required size of the list array.

18.2.9.4 Remarks

This function gets the list of 32-bit unsigned integer values for the specified audio effect and writes the retrieved values to a buffer specified by list and writes the size of the returned list in the buffer specified by `list_size`.

18.2.10. NvAFX_GetBoolList

This function gets the boolean list parameter values for the specified effect:

```
NvAFX_Status NvAFX_GetBoolList(
   NvAFX_Handle effect,
   NvAFX_ParameterSelector param_name,
   NvAFX_Bool* list,
   unsigned int* list_size
);
```

18.2.10.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect instance.

`param_name` *[in]*

Type: `NvAFX_ParameterSelector`

The only selector supported for this function is `NVAFX_PARAM_VAD_RESULT`.

Any other selector string returns an error.

`list [out]`

Type: `NvAFX_Bool*`

The address to a list that contains the boolean values for the given selector.

Note

The application needs to call this API with `list_size` initialized to zero, and list set to nullptr to get the size of list to be allocated. The size will be returned in `list_size` parameter.The application can then allocate an array of at least `list_size` and call the API again with list pointing to the array. For an example, see *Create an Audio Effect*.

`list_size` *[out]*

Type: `unsigned int*`

Pointer to an integer that contains the number of values that were returned in the list.

18.2.10.2 Return Values

- `NVAFX_STATUS_SUCCESS` on success.
- `NVAFX_STATUS_OUTPUT_BUFFER_TOO_SMALL` when the `list_size` is less than the minimum required size of the list array.

18.2.10.3 Remarks

This function gets the list of boolean values for the specified audio effect, writes the retrieved values to a buffer specified by list, and writes the size of the returned list in the buffer specified by `list_size`.

18.2.11. NvAFX_GetSupportedDevices (Windows SDK only)

The function gets a list of compatible devices that are supported by the currently set model file but is not supported by chained effects:

```
NvAFX_Status NvAFX_GetSupportedDevices(
   NvAFX_Handle effect,
   int *num,
   int *devices
);
```

18.2.11.1 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.

18.2.11.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.11.3 Remarks

This function gets the devices supported by the model.

18.2.12. NvAFX_Load

This function validates effect parameters and loads the specified effect:

```
NvAFX_Status NvAFX_Load(
        NvAFX_Handle effect
);
```

18.2.12.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect instance to load.

18.2.12.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.12.3 Remarks

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

18.2.13. NvAFX_Run

This function runs the specified effect:

```
NvAFX_Status NvAFX_Run(
   NvAFX_Handle effect,
   const float** input,
   float** output,
   unsigned num_input_samples,
   unsigned num_input_channels
);
```

18.2.13.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect instance to run.

`input` *[in]*

Type: `const float**`

Pointer to a user-allocated array of buffers where each buffer holds the audio data for one channel.The size of the array must be equal to the number of input samples in the frame (set via `NVAFX_PARAM_NUM_SAMPLES_PER_INPUT_FRAME`) multiplied by the number of streams for which the effect is configured (set via `NVAFX_PARAM_NUM_STREAMS`, always 1 for Windows SDK). The number of channels must be equal to the value of the `NVAFX_PARAM_NUM_INPUT_CHANNELS` parameter that was obtained by the `NvAFX_GetU32()` function.

The sample rate of the audio data must be equal to the input sample rate of the effect (can be queried using *NvAFX_GetU32* using the `NVAFX_PARAM_INPUT_SAMPLE_RATE` selector).

`output` *[out]*

Type: `float**`

Pointer to a user-allocated array of buffers to which the output of the effect will be written. After this function returns, each buffer will contain audio data for one channel.

> **Note**
>
> The buffers must be allocated by and later freed by the calling program. NvAFX_Run internally copies input/output to/from the GPU, so pinning input/output buffers does not have any effect.

The size of the array must be equal to the number of output samples in the frame (set via `NVAFX_PARAM_NUM_SAMPLES_PER_OUTPUT_FRAME`) multiplied by the number of streams for which the effect is configured (set via `NVAFX_PARAM_NUM_STREAMS`, always 1 for Windows SDK).

`num_input_samples` *[in]*

Type: `unsigned`

The number of samples in the input buffer.

`num_input_channels` *[in]*

Type: `unsigned`

The number of input channels.

18.2.13.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.13.3 Remarks

This function runs the specified audio effect by reading the contents of the input buffer, applying the audio effect, and writing the output to the output buffer.

18.2.14. NvAFX_Reset

This function resets the internal state and flushes the internal history for the specified batches:

Windows SDK:

```
NvAFX_Status NvAFX_Reset(
   NvAFX_Handle effect
);
```

Linux SDK:

```
NvAFX_Status NvAFX_Reset(
   NvAFX_Handle effect,
   NvAFX_Bool* list,
   int length
);
```

The three-argument form is available only in the Linux SDK. The Windows header declares only `NvAFX_Reset(NvAFX_Handle effect)`.

Note

This functionality is not supported by Speaker Separation effects.

18.2.14.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect instance to run.

`list` *[in]* (Linux SDK only)

Type: `NvAFX_Bool *`

Pointer to a memory location which indicates the streams to be reset. The `i`-th element in this array should be set to `NVAFX_TRUE` to reset the `i`-th stream, and to `NVAFX_FALSE` otherwise.

`length` *[in]* (Linux SDK only)

Type: `int`

Number of elements in the array specified. Should be equal to the number of streams (batches).

18.2.14.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.15. NvAFX_SetBoolList

This function sets a list parameter of the specified effect:

```
NvAFX_Status NvAFX_SetBoolList(
   NvAFX_Handle effect,
   NvAFX_ParameterSelector param_name,
   const NvAFX_Bool* list,
   unsigned int list_size
);
```

18.2.15.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect.

`param_name` *[in]*

Type: `NvAFX_ParameterSelector`

The only selector supported for this function is `NVAFX_PARAM_ACTIVE_STREAMS`.

Any other selector string returns an error.

`list` *[in]*

Type: NvAFX_Bool*

Array of Boolean values to be set for the parameter.

`list_size` *[in]*

Type: `unsigned int`

Size of the Boolean array passed as input.

18.2.15.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.15.3 Remarks

This function sets the boolean values of the list parameter for the specified audio effect to the values from list.

18.2.16. NvAFX_SetU32List

This function sets a list parameter of the specified effect:

```
NvAFX_Status NvAFX_SetU32List(
   NvAFX_Handle effect,
   NvAFX_ParameterSelector param_name,
   unsigned int* list,
   unsigned int list_size
);
```

18.2.16.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect.

`param_name` *[in]*

Type: `NvAFX_ParameterSelector`

The only selector supported for this function is `NVAFX_PARAM_CHAINED_EFFECT_GPU_LIST`.

Any other selector string returns an error.

`list` *[in]*

Type: `unsigned int*`

Array of U32 values to be set for the parameter.

`list_size` *[in]*

Type: `unsigned int`

Size of the array that was passed as the input.

18.2.16.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.16.3 Remarks

This function sets the U32 values of the list parameter for the specified audio effect to the values from list.

18.2.17. NvAFX_SetFloatList

This function sets a list parameter of the specified effect:

```
NvAFX_Status NvAFX_SetFloatList(
   NvAFX_Handle effect,
   NvAFX_ParameterSelector param_name,
   float* list,
   unsigned int list_size
);
```

18.2.17.1 Parameters

`effect` *[in]*

Type: `NvAFX_Handle`

The handle to the audio effect.

`param_name` *[in]*

Type: `NvAFX_ParameterSelector`

The supported selector is `NVAFX_PARAM_INTENSITY_RATIO` for chained effects.

Any other selector string returns an error.

`list` *[in]*

Type: `float*`

Array of float values to be set for the parameter.

`list_size` *[in]*

Type: `unsigned int`

Size of the array that was passed as the input.

18.2.17.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.17.3 Remarks

This function sets the float values of the list parameter for the specified audio effect to the values from list. This function currently is supported only for setting per-stream intensity ratio for the Background Noise Removal effect

18.2.18. NvAFX_InitializeLogger

This function initializes the SDK logger:

```
NvAFX_Status NvAFX_InitializeLogger(
   LoggingSeverity level,
   LoggingTarget target,
   const char *filename,
   logging_cb_t cb,
   void* userdata
);
```

18.2.18.1 Parameters

`level` *[in]*

Type: `LoggingSeverity`

The logging level to enable. Enabling a level is inclusive of the levels preceding it.

For example, `LOG_LEVEL_INFO` also includes `LOG_LEVEL_WARNING` and `LOG_LEVEL_ERROR`.

This field can have either of the following values :

- `LOG_LEVEL_FATAL` (Windows Only)
- `LOG_LEVEL_ERROR`
- `LOG_LEVEL_WARNING`
- `LOG_LEVEL_INFO`

`target` *[in]*

Type: `LoggingTarget`

Select exactly one logging target. `LoggingTarget` is an enumeration, not a bitmask, so its values must not be combined with binary OR.

The following targets can be used:

- `LOG_TARGET_NONE` = `-1`
- `LOG_TARGET_STDERR` = `0`
- `LOG_TARGET_FILE` = `1`
- `LOG_TARGET_CALLBACK` = `2`

`filename` *[in]*

Type: `const char*`

The path of the file where to write logs. Used only when `LOG_TARGET_FILE` is enabled.

> **Note**
>
> The directory in which the log file resides must exist. For example, if the filename is `/foo/bar/log.txt`, the `/foo/bar` directory must already exist. If the `log.txt` file exists, it is overwritten.

cb *[in]*

Type: `logging_cb_t`

Callback to use if `LOG_TARGET_CALLBACK` is enabled. `NULL` can be passed if not using a callback target.

`userdata` *[in]*

Type: `void *`

Data passed back with log callback. Used only when `LOG_TARGET_CALLBACK` is enabled.

A null value can also be passed.

18.2.18.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.18.3 Remarks

This API enables logging in the SDK. Depending on the flags passed, logs are either redirected to `stderr`, file, or callback. Logging can be disabled using the `NvAFX_UninitializeLogger` API. For more information, refer to *NvAFX_UninitializeLogger*.

18.2.19. NvAFX_UninitializeLogger

```
NvAFX_Status NvAFX_UninitializeLogger(void);
```

18.2.19.1 Parameters

NvAFX_UninitializeLogger requires no parameters.

18.2.19.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

18.2.19.3 Remarks

This API disables all logging targets. Logging can be started again using the `NvAFX_InitializeLogger()` API. For more information, refer to *NvAFX_InitializeLogger*.

18.3. Return Codes

The `NvAFX_Status` enumeration defines the following values that the NVIDIA Audio Effects SDK 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_MODEL_NOT_LOADED`

Model is not loaded, and it has to be loaded for this operation.

`NVAFX_STATUS_INCOMPATIBLE_MODEL`

Selected model is incompatible.

`NVAFX_STATUS_GPU_UNSUPPORTED`

The selected GPU is not supported. The SDK requires an NVIDIA 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.

`NVAFX_STATUS_CUDA_ERROR`

CUDA operation failure.

`NVAFX_STATUS_INVALID_OPERATION`

Invalid operation performed.

Chapter 19. Types of Background Noise Removed

The Background Noise Removal (BNR) effect is available in both the *Noise Removal/Background Noise Suppression* (`denoiser`) and *Noise Removal and Room Echo Removal/Room Echo Cancellation + Background Noise Suppression* (`dereverb_denoiser`) effects.

BNR supports removing the following types of background noise:

- AC noise
- Babble and crowd noise
- Baby crying
- Bird chirping
- Body noises
- Chatter from other people
- Clapping
- Construction site sounds
- Cooking sounds (such as cutting or a cooker)
- Door slamming
- Drums
- Fan noise
- Furniture moving
- Gaussian/white noise
- Glass breaking
- Keyboard
- Metal sounds
- Mouse clicks
- PC noise
- Pet sounds
- Phone ringing
- Rains
- Sirens

- Tapping
- Traffic noise
- Train passing
- Vacuum cleaner
- Washing machine
- Water taps and running water
- Wrappers rustling (plastic or non-plastic)

Copyright

©2021-2026, NVIDIA Corporation