GeForce NOW

Software Development Kit Quick Start Guide

Document History

Version	Date	Description of Change
1.0	*11/09/2023*	*Initial release revision*
1.1	*05/24/2024*	*Extended information on callback purposes and uses*

Introduction

NVIDIA GeForce NOW (GFN) allows users to enjoy their library of PC games on many internet-enabled devices via the power of cloud gaming. This includes PCs, Macs, Smart TVs, and mobile devices such as smartphones and tablets.

GFN has the ability to onboard your Windows and Linux games and applications as-is for your users to enjoy. For tighter integration of these applications with GFN, NVIDIA provides the GeForce NOW Runtime Software Development Kit (GFN SDK) to developers so that their applications can perform various tasks; from knowing when the application is running in GFN to getting information about the GFN session or the user's client system.

This document is provided as part of the SDK to aid in getting the developer up to speed with the SDK quickly, as well as providing source code pointers for the most common of integration scenarios.

Audience

This document is directed towards developers of applications and games that are part of the GFN ecosystem, and wish to provide their users a seamless experience between their application and GFN.

For the most complete understanding of the SDK, this document should be read to completion. If time does not permit that, then it is suggested to cover Section 3, then the pertinent integration use cases for your integration needs as found in Section 5, and review Section 8 for best practices in using SDK API functions.

SDK Overview

The GFN Runtime SDK consists of an ever-growing collection of C-based native APIs that support both Windows and Linux applications. Along with the C source files that define the APIs, the SDK also includes various documentation to aid in development, as well as a collection of samples in source code form that provide example execution of the APIs in various scenarios.

The SDK is designed with ease of use in mind, with API design meant to allow function calls and data consumption to be straightforward. In addition, the SDK is designed to support backwards compatibility as a core requirement, which allows developers to move to new

versions of the SDK without the worry of code breakage related to existing API calls due to API definition alterations or deprecation.

1. GFN SDK First Steps

The GFN Runtime SDK distribution is made available publicly via GitHub: https://github.com/NVIDIAGameWorks/GeForceNOW-SDK/, with new releases targeting a monthly cadence. Obtaining the SDK can be accomplished either by cloning the GitHub repository, or by downloading the latest release via the “Releases” list.

2. Building the GFN SDK and Samples

To ease SDK integration, the SDK is distributed as pre-built dynamic libraries and C-based API definition header files. As such, the SDK itself does not need to be built to obtain the binaries to integrate with your applications and games.

The SDK does include the sources of three sample applications, each with a specific layout that is designed to show what the APIs do and how to call them based on the application type. These samples can be built from source for Windows x86/x64, and Linux x64 platforms. The steps and tools needed to build these samples depend on the platform they will be built for. In general, you will need cmake 3.27 or greater (https://cmake.org), as well as a build system manager such as ninja-builder (https://ninja-build.org/). If building for Windows, the SDK build system is set up to take advantage of Visual Studio’s support for cmake. If not familiar with this support, please visit https://learn.microsoft.com/en-us/cpp/build/cmake-projects-in-visual-studio for more information.

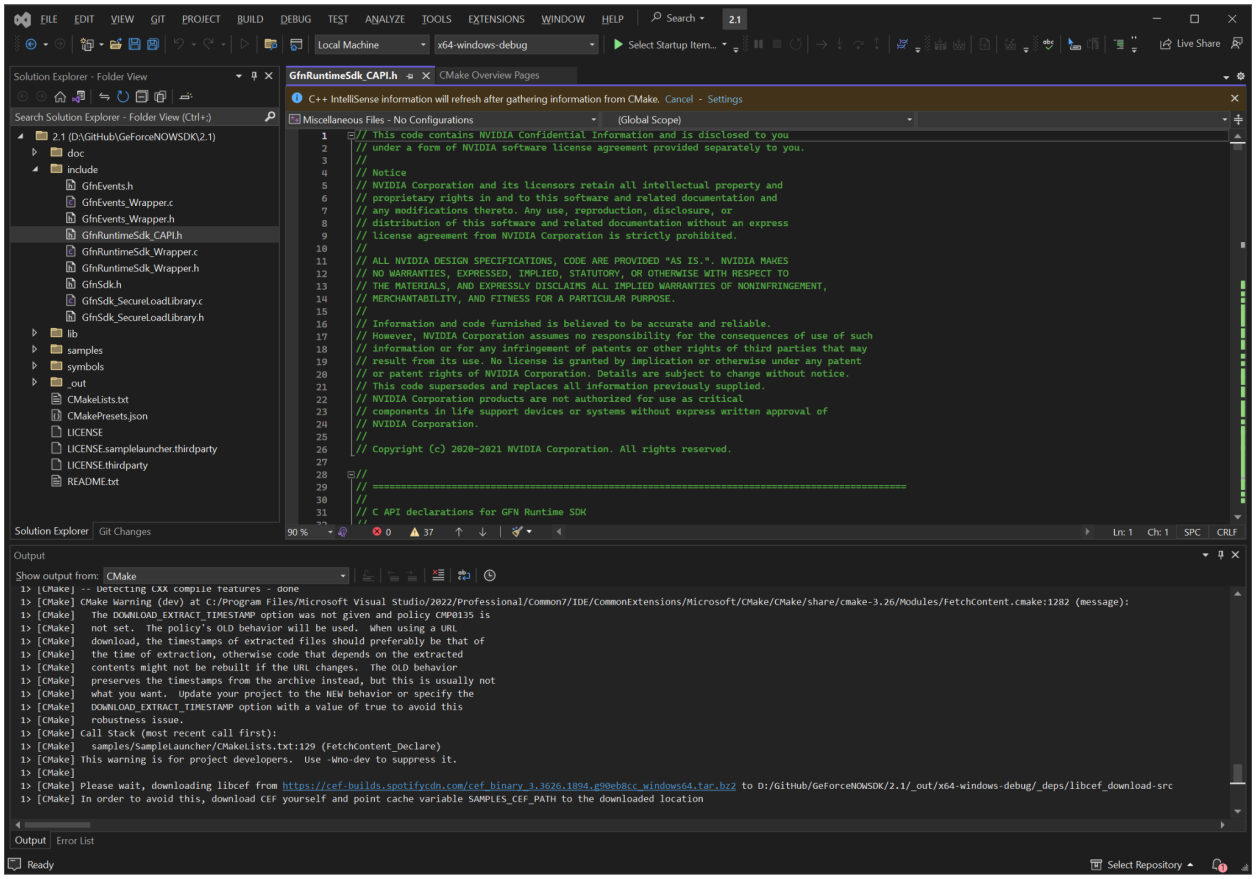
The use of cmake is to make use of cmake’s preset feature. If you are not familiar with this feature, please visit https://cmake.org/cmake/help/latest/manual/cmake-presets.7.html for more information.

In general, building the SDK Samples is a matter of calling cmake with one of the supported presets:

-x64-windows-debug
-x64-windows-release
-x86-windows-debug
-x86-windows-release
-x64-linux-debug
-x64-linux-release

Example:
cmake --preset x64-windows-release

Alternatively with Visual Studio's cmake support, you can open Visual Studio, and then use the "Open a local folder" option to open the root folder of the SDK distribution. This will result in Visual Studio reading the SDK's make files, and generating a Visual Studio solution to build and debug from within:



For more detailed information and build commands for building the SDK, please refer to the README file at the root of the SDK distribution.

3. Integrating SDK Into a Game or Application

Integration of the SDK into a game or application supported by GeForce NOW (also known as a "GFN title") is performed by pulling the appropriate header file(s) and source(s) from ./include into the applications build source tree, as well as the appropriate dynamic library from ./lib into the GFN title's binary structure. The tables below provide details on what to pull in the various scenarios.

For scenarios that wish to call the API functions as library exports:

Target OS	Headers (./include)	Library
Windows 64-bit	GfnRuntimeSdk_CAPI.h GfnSdk.h	.\lib\win\x64\GfnRuntimeSdk.dll
Windows 32-bit	GfnRuntimeSdk_CAPI.h GfnSdk.h	.\lib\win\x86\GfnRuntimeSdk.dll
Linux 64-bit	GfnRuntimeSdk_CAPI.h GfnSdk.h	./lib/linux/x64/GfnRuntimeSdk.so

For scenarios that wish to leverage the export helpers that allow C-based function calls for API functions instead of library exports, add these files to the table above:

Target OS	Headers and Sources (./include)
Windows 64-bit	GfnRuntimeSdk_Wrapper.h GfnRuntimeSdk_Wrapper.c GfnSdk_SecureLoadLibrary.h GfnSdk_SecureLoadLibrary.c
Windows 32-bit	GfnRuntimeSdk_Wrapper.h GfnRuntimeSdk_Wrapper.c GfnSdk_SecureLoadLibrary.h GfnSdk_SecureLoadLibrary.c
Linux 64-bit	GfnRuntimeSdk_Wrapper.h GfnRuntimeSdk_Wrapper.c

The optional “wrapper” files provide a bridge from native C/C++ to the library export functions. In addition for Windows builds, they also take care of making sure the SDK DLL is authentic by checking for the appropriate NVIDIA Digital Signature on the DLL.

Default placement of the SDK dynamic library is expected to be in the same folder as the process executable. However this can be customized to another folder, for example, if all 3rd party libraries are placed in a specific subfolder. See “Initializing the SDK” for more information for each of these cases.

4. Information on SDK Callbacks

In addition to the API functions provided to obtain information on-demand, the SDK provides several C-style callback functions to receive asynchronous notifications on state or data changes. These callback functions are designed to keep applications running on the client or game seat up-to-date with the state of the streaming session. The table below provides an overview of these callbacks and their purpose. Refer to Section 6 of this document for more

information on registering for a callback and the proper definition of the callback handler function.

Callback Name	Purpose	Typical Use Case(s)
RegisterExitCallback	Notifies session exit is requested	Saving game and user data before application shutdown due to session ending
RegisterPauseCallback	Notifies the application should enter paused state	Pausing title when user changes focus away from streaming window
RegisterSaveCallback	Notifies the application to save user data	Saving data on session exit to allow GeForce NOW to save data for next session
RegisterInstallCallback	Notifies when base install of application is done	Performing post-install activities before title launch
RegisterStreamStatusCallback	Notifies when streaming status changes	Allows client that launched streaming session to know various session states to act on current state
RegisterSessionInitCallback	Notifies when session is starting	Allows pre-launched application to know when user session has started to load user data and start title
RegisterMessageCallback	Notifies when a custom message is received	Used for two-way communication between a custom client and streaming application for real-time data exchange
RegisterClientInfoCallback	Notifies when client data changes during session	Allows to know when client information changes, such as a session resuming on a different device
RegisterNetworkStatusCallback	Notifies when network data changes during session	Provides network latency data for matchmaking or other latency-based decisions

5. Understanding the SDK Samples

The SDK includes three samples, each that are meant to be examples of common game or game launcher application types, with API calls applicable to those types:

CGameAPISample:
This C-based simple command-line sample demonstrates usage of the game-focused APIs to detect the GeForce NOW cloud environment and control behavior of a game in that environment. This sample focuses on use of callbacks to notify a title of GeForce NOW cloud environment state changes.

CloudCheckAPI:
This C++-based simple command-line sample demonstrates usage of the APIs that determine if the application is running inside the GeForce NOW streaming environment. The Secure Cloud Check Guide provides information that corresponds to the API calls in this sample.

CubeSample:
This C++-based sample specializes in demonstrating two-way communication between app and the client, leveraging the SendMessage API and its corresponding callback. The sample code is based on the vkCube Sample that ships with the Vulkan SDK to provide a visual reference to custom message handling using keyboard and mouse input.

PartnerDataAPI:
This C++-based simple command-line sample demonstrates usage of the APIs for obtaining both insecure and secure data provided by the partner during initialization of a streaming session.

SampleLauncher:
This C++-based sample demonstrates usage of the Launcher/Publisher application-focused APIs, including getting a list of titles supported by GeForce NOW, as well as invoking the GeForce NOW Windows client to start a streaming session of a title. This sample is meant to be run on both the local client and GeForce NOW cloud environment to understand how all the APIs behave in each environment.

In addition to supporting both execution environments, this sample is built on Chromium Embedded Framework (CEF) to provide a view of how a CEF-based application can make use of the API. The sample source is a fork of the CEF project's example source found at https://bitbucket.org/chromiumembedded/cef-project/src.
For applications that are not CEF-based, users can focus on API calls as found in ./SampleLauncher/src/gfn_sdk_demo/gfn_sdk_helper.cc file.

SDKDllDirectRefSample:
This C-based sample demonstrates basic SDK usage without relying on the wrapper helper functions. This can be useful for partners who are unable to utilize the wrapper in their build

environment or want finer control on SDK library loading and how the library exports are called from an application.

6. Sample Code for Common Integration Scenarios

The sections below provide code snippets and guidelines for API calls in the most common SDK integration and use cases. For ease in integration, the sample code provided in this section will be in the form of the C-based API wrapper functions. If directly using the library exports, replace the function calls with the corresponding export function names. All parameters are the same between both API call types.

Note: To keep the length of the document manageable for the reader, not every data element of an API is shown in the code examples, nor is every error result handled. See the API documentation for complete information on all data types and error values.

Initializing and Shutting Down the SDK

Before any SDK API function can be called, the SDK must be initialized. To do this, call the appropriate initialization function:

Location of SDK Dynamic Library	Initialization API to Use
In the same folder as process executable	GfnInitializeSdk
In a different folder from process executable	GfnInitializeSdkFromPath

```
C/C++
GfnError err = GfnInitializeSdk(gfnDefaultLanguage);
if (GFNSDK_SUCCEEDED(err)) {
	// Other calls to SDK APIs expected to succeed
} else {
	// SDK is in failed state, do not call other APIs
}
```

If using GfnInitializeSdkFromPath, make sure to use the full path to the library, including the library name, and not a relative path.

The language parameter is used for the limited cases where the SDK APIs will directly show UI to the user, for example, if a call to gfnStartStream must download and install the GFN client. The use of gfnDefaultLanguage will display in the system language, is appropriate in most cases. Define another language if this needs to be overridden.

The return value should be checked for success. This can be one of three success values:

Return Value	Meaning
gfnSuccess	SDK initialized, and cloud and client APIs can be called
gfnInitSuccessClientOnly	SDK initialized, and looks like a local client environment
gfnInitSuccessCloudOnly	SDK initialized, and looks like the GFN cloud environment

Once you have confirmed the SDK is successfully initialized, you can call other APIs for scenarios in the next sections.

When finished calling any SDK API function, for example, when the GFN title is shutting down, the SDK must also be shut down. This is accomplished with a call to gfnShutdownSdk:

```
GfnShutdownSdk();
```

That's it, no return values to handle.

Checking if Running in GFN

Since games and applications do not need to be specially built to run on GeForce NOW, the most common use of the SDK is to determine if running inside the GeForce NOW cloud environment.

There are currently two flavors of APIs to accomplish this, depending on the criticality that the API call result be authentic.

For non-critical checks, for example, to bypass a graphics driver check that would only be applicable to a user's PC:

```
C/C++
 bool bIsCloudEnvironment = false;
 GfnIsRunningInCloud(&bIsCloudEnvironment);
 if (bIsCloudEnvironment) {
       // Running inside GFN
} else {
       // Not running in GFN
}
```

If the check is more critical to running the GFN title, for example, to disable anti-cheat mechanisms when running inside the secure GeForce NOW environment:

```
C/C++
GfnIsRunningInCloudAssurance assurance =
GfnIsRunningInCloudAssurance::gfnNotCloud;
GfnError err = GfnIsRunningInCloudSecure(&assurance);
if (GFNSDK_SUCCEEDED(err) {
    // success, look at the assurance value the confidence level of running in
GFN
       switch(assurance){
       case gfnIsCloudHighAssurance:
              // Is GFN based on HW crypto heuristics, near impossible to spoof
       break;
       case gfnIsCloudMidAssurance:
              // Is GFN based on network heuristics not easily spoofed
       break;
case gfnIsCloudLowAssurance:
              // Is GFN based on software heuristics, but can be spoofed
       break;
case gfnNotCloud:
default:
       // Consider to be local PC
break;
       }
}
```

Because of the sensitivity of calling `GfnIsRunningInCloudeSecure`, the application calling it must be granted permissions to call the API by NVIDIA. Otherwise the call will fail. Refer to the “Secure Cloud Check API” document in ./doc for more information.

Getting Session Information

If the call to the cloud check API determines the GFN title is running in GFN, then certain information about the current GFN session can be obtained. This is useful for determining if the session supports RTX, or how much time is remaining in the session to see if the user would be able to finish another round or chapter of the game. This information is obtained from `GetSessionInfo`:

```
C/C++
GfnSessionInfo sessionInfo = { 0 };
GfnError err = GfnGetSessionInfo(&sessionInfo);
if (GFNSDK_SUCCEEDED(err){
       gfnGameSessionId = sessionInfo.sessionId;
hasRayTracing = sessionInfo.sessionRTXEnabled;
gfnSessionLength = sessionInfo.sessionMaxDurationSec;
       gfnSessionSecondsLeft = sessionInfo.sessionTimeRemainingSec;
}
```

Getting Client Information

If the call to the cloud check API determines the GFN title is running in GFN, then certain information about the client system connected to the current GFN session can be obtained. This is useful for analytics, knowing what country the user is connected from to display the correct currency, or knowing client system resolution for scaling purposes. This information is obtained from `GetClientInfo`:

```
C/C++
GfnClientInfo clientInfo = { 0 };
GfnError err = GfnGetClientInfo(&clientInfo);
if (GFNSDK_SUCCEEDED(err) {
       userCountryCode = clientInfo.country;
       userIP = clientInfo.ipV4;
       userPingtoGFN = clientInfo.RTDAverageLatencyMs;
}
```

Registering a Callback

Some GFN information and state data is asynchronous or changes. In those cases, a C-style callback is made available to be notified of these changes. Following the C language for callbacks, to receive a callback, your GFN title must register a function handler for the callback. The SDK uses unique callback methods for each of the information or state change types in order to allow developers to pick the information they wish to receive instead of having to filter through all callback notifications.

Some of the common callback methods are to be notified if a user reconnects to the session on a different client, or if the network latency changes:

```
C/C++
GfnError err =
GfnRegisterClientInfoCallback(reinterpret_cast<ClientInfoCallbackSig>(&handleCl
ientInfoCallback), nullptr);
if (GFNSDK_SUCCEEDED(err) {
       // callback is good
}

// Client Info callback handler
void HELPER_CALLBACK handleClientInfoCallback(GfnClientInfoUpdateData*
pClientUpdate, void* context)
{
    if (!!s_registerClientInfoCallback){
         // callback still registered, process what data changed
        switch (pClientUpdate->updateType){
        case gfnOs:
            // User's Operating System changed when they reconnected to session
            break;
        case gfnIP:
            // User's IP changed when they reconnected to session
            // This usually happens when user goes wifi-LTE or vice versa
            break;
        case gfnClientResolution:
            // User's device resolution changed when they reconnected to session
            break;
        default:
            // data that changed is not important
            return;
        }
    }
}


GfnError err =
GfnRegisterNetworkStatusCallback(reinterpret_cast<NetworkStatusCallbackSig>(&ha
ndleNetworkStatusCallback), 5 * 1000, nullptr);
if (GFNSDK_SUCCEEDED(err) {
       // callback is good, will receive latency changes every 5 seconds
}

// Session Latency callback handler
```

```
void HELPER_CALLBACK handleNetworkStatusCallback(GfnNetworkStatusUpdateData*
pNetworkStatus, void* context)
{
    if (pNetworkStatus->updateType == gfnRTDAverageLatency){
       userLatency = pNetworkStatus->data.RTDAverageLatencyMs;
    }
}
```

Obtaining Partner Data Passed into the Seat from a Stream Start

When a streaming session is started via the GeForce NOW SDK, the API functions provide mechanisms to pass in data. Depending on how the data was passed in, either via the standard or secure data paths, the data can be obtained by GetPartnerData or GetParnerSecureData respectively.

```
C/C++
char const* partnerData = nullptr;
GfnError err = GfnGetPartnerData(&partnerData);
if (GFNSDK_SUCCEEDED(err) {
       // pointer now points to allocated memory that holds the data
       // do not forget to call GfnFree when done with it!
} else if (err  == gfnNoData {
       // no data was found, pointer is untouched
}
else {
       // generic error case
}

char const* partnerSecureData = nullptr;
GfnError err = GfnGetPartnerSecureData(&partnerSecureData);
if (GFNSDK_SUCCEEDED(err) {
       // pointer now points to allocated memory that holds the data
       // do not forget to call GfnFree when done with it!
} else if (err  == gfnNoData {
       // no data was found, pointer is untouched
}
else {
       // generic error case
}
```

A couple of points to keep in mind when calling these functions:

- “Secure” means that the data is transferred between backends via cryptographic means, and is not subject to interception and tampering by a user or user service.
- The data is stringified; it is up to the application to convert it to whatever data type needed to process it.
- If data is found, the SDK will allocate memory for it. To avoid a leak, call `GfnFree` when done with the data
- If data is not found, SDK will not allocate memory for it, and there is no need to call `GfnFree`. If the memory is uninitialized, calling `GfnFree` can result in an application crash.

Freeing memory dynamically allocated by the SDK

Any API function that allocates memory for the returned data must have its memory freed in order to prevent leaks. The SDK provides and API for this purpose

```
C/C++
GfnError err = GfnGetPartnerData(&partnerData);
if (GFNSDK_SUCCEEDED(err) {
       // pointer now points to allocated memory that holds the data
       // do something with the data here
       // now free the memory
       GfnFree(&partnerData);
}
else {
       // error case, no memory allocated
       // DO NOT call GfnFree
}
```

A couple of points to keep in mind when calling GfnFree:

- API functions that allocate memory are documented as such in the API documentation
- Call GfnFree only with parameters that had memory allocated by the SDK
- Only call in API call success cases. Do not blindly call, otherwise the application may crash with a memory exception for GfnFree accessing invalid memory.

Launching a Streaming Session via the GFN Windows Client

The GFN SDK includes two API functions that allow an application to trigger the Windows GFN Client application to launch a streaming session for a selected application. This allows launcher type applications to offer an option to either install and launch a game on the local PC, or skip the game download and install, and allow the user to play the game through GeForce NOW. There are two functions to allow the launch to be called synchronous (blocking calling application continued execution):

```
C/C++
StartStreamResponse response = { 0 };
StartStreamInput startStreamInput = { 0 };
startStreamInput.uiTitleId = 12345;    // The GFN-defined identified for a game
startStreamInput.pchPartnerSecureData = userLoginTokenAsString;
startStreamInput.pchPartnerData = "This is example partner data";
GfnError err = GfnStartStream(&startStreamInput, &response);
if (GFNSDK_SUCCEEDED(err)
       // game launched successfully
}

Or asynchronous to to the calling application's execution:

StartStreamInput startStreamInput = { 0 };
startStreamInput.uiTitleId = 12345;    // The GFN-defined identified for a game
startStreamInput.pchPartnerSecureData = userLoginTokenAsString;
startStreamInput.pchPartnerData = "This is example partner data";
GfnError err = GfnStartStreamAsync(&startStreamInput, &callback, context,
5000);
if (GFNSDK_SUCCEEDED(err)
       // init success, use callback for streaming state change notifications
}
```

A couple of points to keep in mind when calling these functions:

- If the Windows-native GFN client application is not installed, the SDK will automatically download and install the client. Because these steps can take considerable time, the SDK will show progress UI for the steps in the language set in the call to GfnInitializeSdk.
- For the asynchronous function, a callback function is not necessary, but it does provide detail on the progress of starting the stream, as well as any state-changes in the stream.
- If you want to know if the stream loses input focus, use the asynchronous function and register a callback function so that you can receive the `GfnStreamStatusLostInputFocus` and `GfnStreamStatusGotInputFocus` enum values via `GfnStreamStatusResponse`.

There is also an API to stop an in-progress stream, for example, if the user has pressed a “stop streaming” button in the calling application’s UI:

```
C/C++
GfnError err = GfnStopStream();
if (GFNSDK_SUCCEEDED(err)
	// stream stopped
}
```

Launching a Streaming Session via the GFN Web Browser Client

If the client PC is not Windows, it is still possible to invoke a streaming session via GFN’s client support on various web browsers. For details on this method, refer to the “SDK-GFN-DEEP-LINKING” PDF in the ./doc folder.

Sending Data To/From Applications Running in GFN

GeForce NOW SDK provides a mechanism for an application that launches and owns a streaming session to send data to and receive data from the GFN title running inside the stream session. The SDK defines a function to send stringified data as well as a callback to be notified of incoming data:

```
C/C++
std::string message = “This is a test message”;
GfnError err = GfnSendMessage(message.c_str(), (unsigned int)message.length());
if (GFNSDK_SUCCEEDED(err) {
	// message sent!
}
```

In order to receive messages, you must define and register a callback handler function, which will provide the data in stringified form.

```
C/C++
GfnError err =
GfnRegisterMessageCallback(reinterpret_cast<MessageCallbackSig>(&handleMsg),nul
lptr);
if (GFNSDK_SUCCEEDED(err) {
       // Message handler callback successfully registered
}

void HELPER_CALLBACK handleMsg(GfnString* pStrData, void* context) {
       std::string newMessage = std::string(pStrData->pchString);
       // do something with the new message
}
```

The SampleLauncher as well as the CGameAPISample provide extensive working code for the two-way messaging feature.

Defining Editbox Regions for On-Screen Keyboard Input

GeForce NOW supports streaming games and applications to mobile devices, along with support touch-based input from these devices. The GeForce NOW SDK includes the document “SDK-GFN-MOBILE-TOUCH-INTEGRATION-GUIDE” in ./doc folder that provides details on how games and applications can create a seamless integration for touch devices.

As part of that integration, there are times where the user will need to enter keyboard information into the GFN title. Rather than requiring the GFN title to create their own keyboard, GeForce NOW provides a way for the user to enter that input via their device’s on-screen virtual keyboard. To enable this feature, the GFN title would need to define the area of the screen that would trigger the keyboard to be shown, and the GeForce NOW SDK provides an API function to control that behavior:

```
C/C++
// Trigger region is 20 by 100 pixels, located 5 by 10 pixels from top left
corner
GfnRect editboxRect = {5, 10, 100, 20, false, gfnRectXYWH};

// Create the region as ID number 1
GfnError err = GfnSetActionZone(gfnEditBox, 1, &editboxRect);
if (GFNSDK_SUCCEEDED(err) {
       // Success, if user touches this region, GFN client will show the virtual
keyboard
...
```

```
}
```

For more details on using this API, including updating and destroying a trigger region, refer to Section 7 of “SDK-GFN-MOBILE-TOUCH-INTEGRATION-GUIDE” in ./doc folder.

A couple of points to keep in mind when calling this function:

- If the UI can scroll and move the touch point, it is up to the calling process to reset the edit box trigger region. Otherwise, a touch in the originally defined region will bring up the keyboard, and likely confuse the user on why the keyboard suddenly appeared
- Once the edit box trigger region leaves the UI, the API must be called to turn off that region for the same reasons in the previous bullet point.

Supporting Application Pre-Warm For Faster Session Startup

When a user starts a streaming session on GFN, GFN backend services allocate resources to host the GFN title the user selected to stream, then starts that process, and then starts streaming frames to the user. In most cases, the user sees “Loading” UI from the process, and depending on the process, can wait several minutes before loading is complete.

For GFN titles that have long loading times, GFN has a feature known as “Pre-Warm”, which minimizes the user’s wait for the title to load. The communication between the GFN title and GFN resources for this feature is made possible by a SDK API function and callback.

For a GFN title that support this feature, once the process loads all the generic data possible, the process then registers for a callback notification, which signals to GFN that the game is ready for a user connection when a user requests to stream that title:

```
C/C++
// Asset loading complete
GfnError err =
GfnRegisterMessageCallback(reinterpret_cast<SessionInitCallbackSig>(&handleSess
ionInit),nullptr);
if (GFNSDK_SUCCEEDED(err) {
       // callback registered, GFN notified, go idle until callback is triggered
}
```

Once a user is assigned to the process, GFN will copy all user data for the GFN title process running to the session, and then notify the process that it can finish loading via the callback.

Once the process has loaded everything and is ready for the user to see the stream, `GfnAppReady` is called:

```
C/C++
void HELPER_CALLBACK handleSessionInit(const char* partnerInfoMutable, void*
pUserContext) {
	std::string userTokenData = std::string(partnerInfoMutable);
	// Time to finish loading all user data, show main menu
	// Then notify GFN we are ready
	GfnError err = GfnAppReady(true, null);
	if (GFNSDK_SUCCEEDED(err) {
		// Success, GFN will now start showing frames to the user
		// Turn over to user input loop
	}
}
```

A couple of points to keep in mind when calling this function:

- In order to successfully use these APIs, NVIDIA GFN support services must specially configure the GFN title to work in this mode.
- It is expected that the GFN title still supports traditional “cold” launches in GeForce NOW; a special build of the game should not be required.
- For the best user experience, if the game requires the user to log into an account in-game, then the login token is to be passed in as part of the launch, and retrieved via GetPartnerSecureData API.
- Once the SessionInit callback is triggered, GFN will wait for the call to GfnAppReady API function or wait 30 seconds before frames are streamed to the user, whichever occurs first.

7. Integrating SDK in Unreal Engine

For steps and best practices integrating the GFN SDK in Unreal Engine, please see the “SDK-GFN-UNREALENGINE-INTEGRATION-GUIDE.pdf” in the ./doc folder of the SDK release package.

8. Reviewing SDK's Log Files

If there are issues calling SDK functions, then the first step is to review the calls via a debugger. Outside of using a debugger, the SDK creates log files that can provide more information about the calls and the results. The logs can be found in the following OS-specific locations:

- Windows: %COMMONAPPDATA%\NVIDIA Corporation\GfnRuntimeSdk
- Linux: ~/.nvidia/GfnRuntimeSdk

9. SDK Usage Best Practices

This section includes helpful development tips for using the SDK in various cases:

Memory Management

As mentioned with example code scenarios in Section 5, certain API functions will allocate memory for variable-length data returned by the function. This approach allows more dynamic data use without setting static limits of data, however, this also means the memory must be managed. When the documentation mentions memory is allocated for a specific function, keep in mind for proper memory management.

- **DO** check the return value of the API function for success or failure. The API function will only allocate memory in a success case.
- **DO** call `GfnFree` when done with the data in the memory. This includes callback cases.
- **DO NOT** call the same API function again with the same variable before freeing the memory from the previous call. This will leak memory.
- **DO NOT** blindly call GfnFree after an API function call. Per the first bullet, if the API function returned any error, including `gfnNoData`, no memory is allocated, and the pointer is uninitialized. Attempting to free uninitialized memory can result in Access Violation or other unhandled exceptions.

Multiple Concurrent Instance Handling

The SDK does support multiple instances of the SDK across multiple processes, for example, a game launcher and a game process both initializing the SDK and calling API functions. However, in most cases, concurrency is not needed; a single process could initialize and call API functions and pass the data between processes to make sure the process's stay in sync in regards to SDK data.

Multiple Calls to APIs That Return Static Data

The SDK supports calling an API repeatedly, however, most APIs return static data/data that is not going to change during process lifetime. For example, if GfnIsRunningInCloud returns true during a session, that value is not going to change during the calling process's lifetime. In such cases, it is more efficient to call the API once, and cache the value for future use..

Thread Safety

Several SDK API functions either work asynchronously, or have an asynchronous variant. When calling such functions or registering for callbacks in multi-threaded processes, it is important to keep the calls and returned data straight. For example, care must be taken when one thread calls `GfnGetPartnerData` that allocates memory, and another thread is tasked with the call to `GfnFree`. In such cases, it is more robust to call `GfnFree` from the thread that called `GfnGetPartnerData`. Similar care must be taken when calling synchronous functions that block a thread while other threads continue to execute.

Complex Data into Generic Data Types

The SDK APIs are C-based for the widest range of compatibility to common game engines and other frameworks. This means that the APIs define common C-based generic data types for many parameters, for example, char*. This requires certain application-defined complex data objects, such as a JSON object, be stringified before passing into certain functions, or de-stringified when read from functions that return such data. Performing a type cast on these parameters versus turning the data into string form can have negative results on the data, or even cause an application crash.

Callback Handling

The SDK's callbacks described earlier in the document provide powerful methods to respond to changing states of the streaming session. It is encouraged to use them, however, keep in mind the execution of the callback function handlers should not be long-running or blocking functions. Many of the callbacks have timeouts. If the work required to process a callback is expected to take significant time, it is recommended to spin that work off as a separate thread and return from the callback function as quickly as possible.

10. Conclusion and Next Steps

This document is meant to provide an overview of the SDK and the most-used API methods. The code samples provided here are meant to provide enough code to get started, and is not complete drop-in code. For the sake of brevity, not all error cases are handled by the sample code provided, whereas production-level code must do so.

For complete information about the APIs, refer to the API documentation in the ./doc folder, as well as the working code supplied with the samples in ./samples. If there are still questions about the SDK, then please email the SDK development team at geforcenow-sdk-support@nvidia.com.