

GFN Link Software Development Kit

Original Author: Kevin Klemmick

Date: 2/14/2014

Updated: 9/10/2015

Updated: 8/25/2016

Table of Contents

[Introduction 4](#_Toc459902825)

[Overview 4](#_Toc459902826)

[C API 5](#_Toc459902827)

[1. Project Setup 5](#_Toc459902828)

[2. GFN Setup and Shutdown 5](#_Toc459902829)

[3. Implement GFN Application Methods 6](#_Toc459902830)

[4. Add GFN Link API Calls 6](#_Toc459902831)

[C++ API 7](#_Toc459902832)

[1. Project Setup 7](#_Toc459902833)

[2. GFN Setup and Shutdown 7](#_Toc459902834)

[3. Implement GFN Application Methods 8](#_Toc459902835)

[4. Add GFN Link API Calls 8](#_Toc459902836)

[GFN Link Framework, Standards and Types 9](#_Toc459902837)

[Memory 9](#_Toc459902838)

[Versioning 9](#_Toc459902839)

[GFN Initialization/Shutdown Methods 10](#_Toc459902840)

[InitializeGFNLinkSDK 10](#_Toc459902841)

[ShutdownGFNLinkSDK 10](#_Toc459902842)

[GFN Link API Methods (IGFNLink Interface) 11](#_Toc459902843)

[IsGFNEnabled 11](#_Toc459902844)

[RequestKeyboardOverlayOpen 11](#_Toc459902845)

[RequestKeyboardOverlayClose 12](#_Toc459902846)

[RequestGFNAccessToken 12](#_Toc459902847)

[Request3rdPartyToken 13](#_Toc459902848)

[Set3rdPartyToken 13](#_Toc459902849)

[GetStorageLocation 14](#_Toc459902850)

[NotifyStorageChange 14](#_Toc459902851)

[NotifyErrorEncountered 15](#_Toc459902852)

[GFN Application Methods (IGFNApplication Interface) 16](#_Toc459902853)

[RequestApplicationPause 16](#_Toc459902854)

[RequestApplicationSave 16](#_Toc459902855)

[RequestApplicationExit 17](#_Toc459902856)

[LockUserOptions 17](#_Toc459902857)

[SetLocale 18](#_Toc459902858)

[IsUpdateRequired 18](#_Toc459902859)

[Architecture 20](#_Toc459902860)

[Authentication and Federation 21](#_Toc459902861)

[Development and Debugging 22](#_Toc459902862)

[GFN Link Host Emulator 22](#_Toc459902863)

[GFN 22](#_Toc459902864)

[Environment 22](#_Toc459902865)

# Introduction

The NVIDIA GFN Link SDK allows application developers to easily prepare their software to run in NVIDIA’s cloud environment. Integration is simple to do and doesn’t require a separate build process or target executable. The light weight library adds no performance cost when operating outside of the GFN environment, so it is safe to include in builds that run on desktops or other environments.

This document provides step by step details of how to integrate the GFN Link SDK into your application. For more detailed technical information about the architecture, refer to the architecture section later in this document.

## Overview

The GFN Link SDK developer package consists of the following:

Samples\... Sample application implementations and tools.

GFNLinkSDK\dll\... Prebuilt development dlls for running in a test environment

GFNLinkSDK\include\... GFN Link public include files

GFNLinkSDK\lib\... Prebuilt library files for various configurations

GFNLinkSDK\props\... VisualStudio .props files for easy integration

GFNLinkSDK\stubs\... Stubbed off implementation files that can be used for integration

At present, NVIDIA provides two different methods you can use to integrate with GFN. Which choice is best for you depends on what language your application is written in and what works best for your application's architecture.

**C API**: A traditional C library linked into your application. This option is best for applications written in C or other unmanaged languages.

**C++ API**: An object oriented API in C++. This may be preferable to the C API for those using C++.

**C# API**: A C-Sharp assembly for integration with C# applications.

# C API

The C API is the most straightforward integration method, as it uses a traditional C library statically linked to your application. It provides a simple set of global methods to be called from your code, as well as a set of global method stubs that should be implemented by you.

For an example implementation, you can refer to the “SampleCApplication” project included in the GFN Link SDK package.

These steps will guide you with the integration.

### 1. Project Setup

Copy the following file from the GFN Link SDK directory to your application’s source code tree:

GFNLinkSDK\stubs\C stubs\GFNApplication.c

Add the following include path to your compiler commands:

GFNLinkSDK\include

Link to the appropriate .lib file in the following location:

GFNLinkSDK\lib\...

Precompiled libraries are included for Win32 (x86) and x64 architectures, Debug and Release configurations and MS v100, v110 and v120 platform toolsets.

Note for VS 2010 – 2013 users:

You can easily integrate the correct library into your project by adding the appropriate .props file using the PropertyManager. There are two props files located here:

GFNLinkSDK\props

GFNLinkSDK.props should be used if you’re using Multithreaded Dll CRT linkage (/MD option).

GFNLinkSDKMT.props should be used if you’re using Multithreaded CRT linkage (/MT option).

No further action should be required to correctly compile and link in this case.

### 2. GFN Setup and Shutdown

Add a call to glInitializeGFNLinkSDK() to your application’s startup code.

Add a call to glShutdownGFNLinkSDK() to your application’s shutdown code.

You will need to include "GFNLinkSDK\_CAPI.h" for these definitions.

An Initialize/Shutdown pair should be added each time a process that needs to communicate with GFN is started. For example, if you have a launcher that implements the patching methods and a game executable that implements the log-in methods, both should call Initialize at startup and Shutdown during shutdown.

### 3. Implement GFN Application Methods

You should now begin implementing the methods stubbed out in the GFNApplication.c file that you copied into your project in step 1. See the GFN Application Methods section for a detailed explanation of what each method should do.

### 4. Add GFN Link API Calls

Lastly, you'll need to place GFN API calls in the appropriate locations in your code. See the GFN Link API Methods section for a detailed explanation of each method.

You will need to include "GFNLinkSDK\_CAPI.h" from any file that calls into the GFN API.

Note that in order to prevent any name collisions, all GFN Link C API methods are prefixed by “gl”.

# C++ API

The C++ API provides an object oriented API as well as the advantage of a unique namespace, but is otherwise similar to the C API. C++ developers can choose to implement the stub class provided by NVidia or interface with GFN only through the C API methods using the ‘GFNLinkSDK’ namespace.

For an example implementation, you can refer to the “SampleCPPApplication” project included in the GFN Link SDK package.

These steps will guide you with your integration.

### 1. Project Setup

Copy one the following files from the GFN Link SDK directory to your application’s source code tree. Use the .c file if you wish to use C API methods or use the .cpp file if you wish to use the C++ API:

GFNLinkSDK\stubs\C stubs\GFNApplication.c

GFNLinkSDK\stubs\C++ stubs\GFNApplication.cpp

Add the following include path to your compiler commands:

GFNLinkSDK\include

Link to the appropriate .lib file in the following location:

GFNLinkSDK\lib\...

Precompiled libraries are included for Win32 (x86) and x64 architectures, Debug and Release configurations and MS v100, v110 and v120 platform toolsets.

Note for VS 2010 – 2013 users:

You can easily integrate the correct library into your project by adding the appropriate .props file using the PropertyManager. There are two props files located here:

GFNLinkSDK\props

GFNLinkSDK.props should be used if you’re using Multithreaded Dll CRT linkage (/MD option).

GFNLinkSDKNT.props should be used if you’re using Multithreaded CRT linkage (/MT option).

No further action should be required to correctly compile and link in this case.

### 2. GFN Setup and Shutdown

Add a call to GFNLinkSDK::InitializeGFNLinkSDK() to your application’s startup code.

Add a call to GFNLinkSDK::ShutdownGFNLinkSDK() to your application’s shutdown code.

You will need to include "GFNLinkSDK\_CAPI.hpp" for these definitions.

An Initialize/Shutdown pair should be added each time a process that needs to communicate with GFN is started. For example, if you have a launcher that implements the patching methods and a game executable that implements the log-in methods, both should call Initialize at startup and Shutdown during shutdown.

### 3. Implement GFN Application Methods

You should now begin implementing the methods stubbed out in the GFNApplication.cpp file that you copied into your project in step 1. See the GFN Link Application Methods section for a detailed explanation of what each method should do.

### 4. Add GFN Link API Calls

Lastly, you'll need to place GFN Link API calls in the appropriate locations in your code. See the GFN Link API Methods section for a detailed explanation of each method.

You will need to include "GFNLinkSDK \_CAPI.hpp" from any file that calls into the GFN Link API.

Note that in order to prevent any name collisions, all C/C++ API methods use the GFNLinkSDK namespace.

# GFN Link Framework, Standards and Types

GFN Link methods fall into three categories:

1. Initialization/Shutdown Methods: Static methods called at initialization and shutdown.
2. GFN Link API Methods: Methods called by you application to query or notify GFN.
3. GFN Application Methods: Method implemented by you to expose functionality to GFN.

In most cases, GFN Link API methods will return a GFNLinkError type, defined in GFNLinkSDK\_CAPI.h. The application can check the return value in order to verify a given method succeeded.

All GFN Application methods return an ApplicationResult type that is used by GFN to determine if a given method has been implemented or not, and if so if a call to it has succeeded. See the GFN Application Methods section for more details.

### Memory

Any memory allocated by GFN Link API methods will be managed by the library and cleaned up during shutdown. The application is not responsible for freeing memory. However, this also means that any pointers or strings returned by GFN Link API methods are only valid until the shutdown call is made.

### Versioning

The GFN Link SDK provides a version number defined in “GFNLinkSDK\_CAPI.h” that can be used by the application to check which version is currently linked, if necessary.

# GFN Initialization/Shutdown Methods

### InitializeGFNLinkSDK

|  |  |
| --- | --- |
| C | GFNLinkError **glInitializeGFNLinkSDK**() |
| C++ | GFNLinkError GFNLinkSDK::**InitializeGFNLinkSDK**() |

**Description**

Should be called at application startup and prior to any GFN Link API methods.

When running outside of a GFN environment (a game seat virtual machine or development environment) it is expected for this method to return a result other than success. In this case all GFN methods become no-ops and have no performance impact on your application.

**Usage**

Call as soon as possible during application startup.

**Return values**

GFNLinkError::gleSuccess on success when running in a GFN environment.

GFNLinkError::gleGFNDLLNotPresent if running outside a GFN environment (no GFN.dll present)

GFNLinkError::gleGFNComNotEstablished if running outside a GFN environment (no GFN host or test application running)

GFNLinkError::gleIncompatibleVersion Linked GFN Link SDK library is not compatible with the existing GFN.dll)

### ShutdownGFNLinkSDK

|  |  |
| --- | --- |
| C | void **glShutdownGFNLinkSDK**() |
| C++ | void GFNLinkSDK::**ShutdownGFNLinkSDK**() |

**Description**

Should be called at Application shutdown. Frees up memory allocated by GFN and disconnects from GFN backend.

**Usage**

Call during application shutdown or when GFN Link API methods are no longer needed.

# GFN Link API Methods (IGFNLink Interface)

GFN Link API methods are used to make request from or to notify the GFN backend.

When your application is operating outside of the GFN environment, these methods are simple stubs that incur almost no cost, so it's safe to add these to your main build.

The calling convention differs by which API you've chosen to use. Examples of usage and the required include/using statement are given below:

|  |  |
| --- | --- |
| C | #include "GFNLinkSDK\_CAPI.h"  **glRequestKeyboardOverlayClose**(); |
| C++ | #include "GFNLinkSDK\_CAPI.hpp"  GFNLinkSDK::Instance()->**RequestKeyboardOverlayClose**(); |

In most cases IGFNLink methods return a GFNLinkError result, which can be used by the application to check for errors – however, in practice it is unlikely to be useful for the application to check this value as an ideal implementation would not make logic changes based on the result. For example, should RequestKeyboardOverlayOpen fail due to running outside of a GFN environment, the application would still continue to accept keyboard input from the native input handler and/or text control. Additionally, since calling RequestKeyboardOverlayClose is safe to do even if RequestKeyboardOverlayOpen failed, the application can call it when input is no longer needed in all cases.

### IsGFNEnabled

|  |  |
| --- | --- |
| C | bool **glIsGFNEnabled**() |
| C++ | bool IGFNLink::**IsGFNEnabled**() |

**Description**

Determines if application is running in GFN environment or not.

**Usage**

Use to enable any GFN specific application logic.

**Return value**

true: Application is running on a game seat virtual machine or GFN test environment

false: Application is not running in a GFN Environment

### RequestKeyboardOverlayOpen

|  |  |
| --- | --- |
| C | GFNLinkError **glRequestKeyboardOverlayOpen**(GFNScreenPosition gspPosition) |
| C++ | GFNLinkError IGFNLink**::RequestKeyboardOverlayOpen**(GFNScreenPosition gspPosition) |

**Description**

Called from application when it is expecting text input from user. Calling this API would trigger a native keyboard overlay to be shown to the GFN user such that he/she can most easily enter text, based on the particular GFN client platform being used.

There's no special input handling needed from application; input will be injected into application by GFN (as is done in all other times running in GFN). Note GFN is not displaying any text input box or prompt to user, only a keyboard overlay.

**Usage**

This API should be called as a pair with RequestKeyboardOverlayClose. Multiple calls to RequestKeyboardOverlayOpen will have no effect after the first call.

**Parameters**

gspPosition: the desired screen positioning of text input element (i.e. Android keyboard). Should be one of the following values:

gspBottom

gspTop

gspLeft

gspRight

gspCenter

gspTopLeft

gspTopRight

gspBottomLeft

gspBottomRight

**Return value**

gleSuccess: On success

Otherwise, appropriate error code

### RequestKeyboardOverlayClose

|  |  |
| --- | --- |
| C | GFNLinkError **glRequestKeyboardOverlayClose**() |
| C++ | GFNLinkError IGFNLink**::RequestKeyboardOverlayClose**() |

**Description**

Called from application when necessary text input has been processed and user can continue. This would cause a previously requested keyboard overlay on the GFN user's client display to be dismissed

**Usage**

RequestKeyboardOverlayOpen should be called before this method. If not, RequestKeyboardOverlayClose will have no effect.

**Return value**

gleSuccess: On success

Otherwise, appropriate error code

### RequestGFNAccessToken

|  |  |
| --- | --- |
| C | GFNLinkError **glRequestGFNAccessToken**(const char\*\* ppchToken) |
| C++ | GFNLinkError IGFNLink**::RequestGFNAccessToken**(const char\*\* ppchToken) |

**Description**

Request to obtain a user specific access token to allow access to the GFN backend service (IDM endpoint).

**Usage**

The access token provided can be used by the application’s backend servers to validate the user and obtain user data from the GFN backend service. The GFN backend service provides an OAuth2 interface for validating users and retrieving data. See Account Federation section for more information.

**Parameters**

ppchToken: Populated with a user specific GFN access token.

**Return value**

gleSuccess: On success

Otherwise, appropriate error code

### Request3rdPartyToken

|  |  |
| --- | --- |
| C | GFNLinkError **glRequest3rdPartyToken**(const char\* pchProviderId, const char\*\* ppchToken) |
| C++ | GFNLinkError IGFNLink**::Request3rdPartyToken**(const char\* pchProviderId, const char\*\* ppchToken) |

**Description**

Request to obtain a 3rd party token from the GFN Link database.

**Usage**

This and the Set3rdPartyToken methods can be used by developers to store and retrieve their own tokens in the GFN Link database. GFN Link does nothing with this token, it is simply a storage mechanism for developers to use. One usage might be to store an access or credential token that allows the user access to the developer’s backend services.

**Parameters**

pchProviderId: ProviderId string provided by GFN Link during account signup.

ppchToken: Populated by the token saved in the GFN database.

**Return value**

gleSuccess: On success

Otherwise, appropriate error code

### Set3rdPartyToken

|  |  |
| --- | --- |
| C | GFNLinkError **glSet3rdPartyToken**(const char\* pchProviderId, const char\* pchToken) |
| C++ | GFNLinkError IGFNLink**::Set3rdPartyToken**(const char\* pchProviderId, const char\* pchToken) |

**Description**

Store a 3rd party token in the GFN Link database.

**Usage**

This and the Request3rdPartyToken methods can be used by developers to store and retrieve their own tokens in the GFN Link database. GFN Link does nothing with this token, it is simply a storage mechanism for developers to use. One usage might be to store an access or credential token that allows the user access to the developer’s backend services.

**Parameters**

pchProviderId: ProviderId string provided by GFN Link during account signup.

pchToken: The token to save in the GFN database.

**Return value**

gleSuccess: On success

Otherwise, appropriate error code

### GetStorageLocation

|  |  |
| --- | --- |
| C | GFNLinkError **glGetStorageLocation**(const char\*\* ppchStoragePath) |
| C++ | GFNLinkError IGFNLink**::GetStorageLocation**(const char\*\* ppchStoragePath) |

**Description**

Provides a path to a GFN managed storage location for the current application and user. All files and folders under this location will be persisted in GFN cloud storage.

**Usage**

Application developers should use the provided location for storing user-specific application state files and options files. NotifyStorageChange() should be called once all files have been in order to trigger an immediate backup of these files.

**Parameters**

ppchStoragePath: Populated with a local directory path for the application to store files to.

**Return value**

gleSuccess: On success

Otherwise, appropriate error code

### NotifyStorageChange

|  |  |
| --- | --- |
| C | GFNLinkError **glNotifyStorageChange**() |
| C++ | GFNLinkError IGFNLink**::NotifyStorageChange**() |

**Description**

Notifies GFN that file saves have completed and that it should immediately backup the local files to cloud storage. Note that all files are automatically saved at the end of a GFN gaming session, so this is only necessary in order to increase robustness.

**Usage**

Called from application when a set of file operations, as defined by the application, are completed at the GFN Storage location. GFN systems may use this notification to provide additional Cloud backup functionality. Application does not need to call this on every single file change.

**Return value**

gleSuccess: On success

Otherwise, appropriate error code

### NotifyErrorEncountered

|  |  |
| --- | --- |
| C | GFNLinkError **glNotifyErrorEncountered**(GFNLinkAppError appError, const char\* pchErrorMessage) |
| C++ | GFNLinkError IGFNLink**::NotifyErrorEncountered**(GFNLinkAppError appError, const char\* pchErrorMessage) |

**Description**

Notifies GFN that an error occurred in the application.

**Usage**

Called from application when a GFNLink related error occurred where notification to GFN Link would be useful. In most cases critical errors will end the user session. This gives developers a way to end the user session from within the application as well as notify the GFN system an error occurred which can be useful for debugging and metrics purposes.

**Parameters**

appError: Set to an appropriate error in the Enum or a private error code.

pchErrorMessage: A human readable error message.

**Return value**

gleSuccess: On success

Otherwise, appropriate error code

# GFN Application Methods (IGFNApplication Interface)

In order for GFN to make requests of your application, you will need to implement a set of methods stubbed off in the GFNApplication files provided by NVIDIA. You only need to implement those method that are applicable for your application and your business model, and should leave the default implementation for the remainder. The default implementation for these methods return ‘arNotImplemented’, which indicates to GFN that you’ve not implement that method.

Implementation in most cases will involve calling into your code in order to perform the requested operation and returning ‘arSuccess’ or ‘aFailure’ instead of the default ‘arNotImplemented’. In cases where the requested operation is asynchronous but no response is required, your application should not block until the operation is complete, but rather return success if the operation was successfully initiated and failure otherwise.

### RequestApplicationPause

|  |  |
| --- | --- |
| C | ApplicationResult **glRequestApplicationPause**() |
| C++ | ApplicationResult IGFNApplication**::RequestApplicationPause**() |

**Description**

Should pause the application when called, if possible.

For Multiplayer games, it is recommended that this is implemented similar to a client disconnect.

**Usage**

GFN will call this method in cases where the user loses network connection with the game seat virtual machine or otherwise disconnects from his/her GRID session.

**Return value**

arNotImplemented: Method not implemented for this application

arSuccess: Command accepted successfully

arFailure: Command not possible at this time

### RequestApplicationSave

|  |  |
| --- | --- |
| C | ApplicationResult **glRequestApplicationSave**() |
| C++ | ApplicationResult IGFNApplication**::RequestApplicationSave**() |

**Description**

Should save all user game and option data.

It is recommended this be implemented as an autosave if such a feature is supported by your application.

**Usage**

GFN will call this method in order to save user progress in cases where the user has gone idle or otherwise disconnected from the game seat virtual machine.

**Return value**

arNotImplemented: Method not implemented for this application

arSuccess: Command accepted successfully

arFailure: Command not possible at this time

### RequestApplicationExit

|  |  |
| --- | --- |
| C | ApplicationResult **glRequestApplicationExit**() |
| C++ | ApplicationResult IGFNApplication**::RequestApplicationExit**() |

**Description**

Should perform a graceful exit of the application when called.

**Usage**

GFN will call this method in cases where a GFN gaming session has completed prior to the user closing the application themselves; For example, after a prolonged network connection loss.

**Return value**

arNotImplemented: Method not implemented for this application

arSuccess: Command accepted successfully

arFailure: Command not possible at this time

### LockUserOptions

|  |  |
| --- | --- |
| C | ApplicationResult **glLockUserOptions**(UserOptions uoOptions) |
| C++ | ApplicationResult IGFNApplication**::LockUserOptions**(UserOptions uoOptions) |

**Description**

Should disable certain user option menus in the application, such as graphics options, screen resolution changes and windowed/fullscreen mode.

**Usage**

GFN will call this method shortly after being initialized in order to inform the application which user options should be disabled. Alternatively, application developers can use calls to IsGFNEnabled at the appropriate locations in order to disable these options themselves.

**Parameters**

uoOptions: Set of flags representing the types of options to disable

At present only a single option flag has been specified:

uoGraphicsSettings: Disable all optional graphics settings, including screen resolution, windowed mode and graphics quality options.

**Return value**

arNotImplemented: Method not implemented for this application

arSuccess: Command accepted successfully

arFailure: Command not possible at this time

### SetLocale

|  |  |
| --- | --- |
| C | ApplicationResult **glSetLocale**(const char\* pchlanguageCode) |
| C++ | ApplicationResult IGFNApplication**::SetLocale**(const char\* pchlanguageCode) |

**Description**

Should set application’s localization settings to those provided by GFN.

Since GFN does not perform language specific installs for every language an application supports or have separate game seat virtual machine for each language, it is necessary to set the application locale to the user’s locale dynamically.

In order to properly implement this feature applications must be able to switch locales at runtime without restarting the application.

**Usage**

GFN will call this method shortly after being initialized in order to set the application’s locale to that requested by the user. If the requested locale is not supported, the application should return arFailure and use “en-US” as a default.

**Parameters**

pchLanguageCode: Code following [ISO 639-1](http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes) and [ISO 3166-1](http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2) standards (i.e. 'en-US')

**Return value**

arNotImplemented: Method not implemented for this application

arSuccess: Command accepted successfully

arFailure: Command not possible at this time

### IsUpdateRequired

|  |  |
| --- | --- |
| C | ApplicationResult **glIsUpdateRequired**(bool\* pbUpdate) |
| C++ | ApplicationResult IGFNApplication**::IsUpdateRequired**(bool\* pbUpdate) |

**Description**

Determines if the application requires an update or patch in order to continue execution.

In most cases Grid will keep applications updated to the most recent version, but in cases where an application developer releases an update without providing this to NVIDIA, it’s possible that the application is temporarily out of date. Applications should not self patch in the Grid environment.

Note that this is querying if it is possible to run with the current executable, not whether the application is fully up to date. For example, if a game’s backend is backward compatible with slightly out of date clients, then this should return true rather than false.

**Usage**

GFN will call this method shortly after being initialized in order to determine if the application is able to run with the current version.

**Parameters**

pbUpdate: Should be set to true by the application if the current version of the application is not able to run, false otherwise.

**Return value**

arNotImplemented: Method not implemented for this application

arSuccess: Command accepted successfully

arFailure: Command not possible at this time

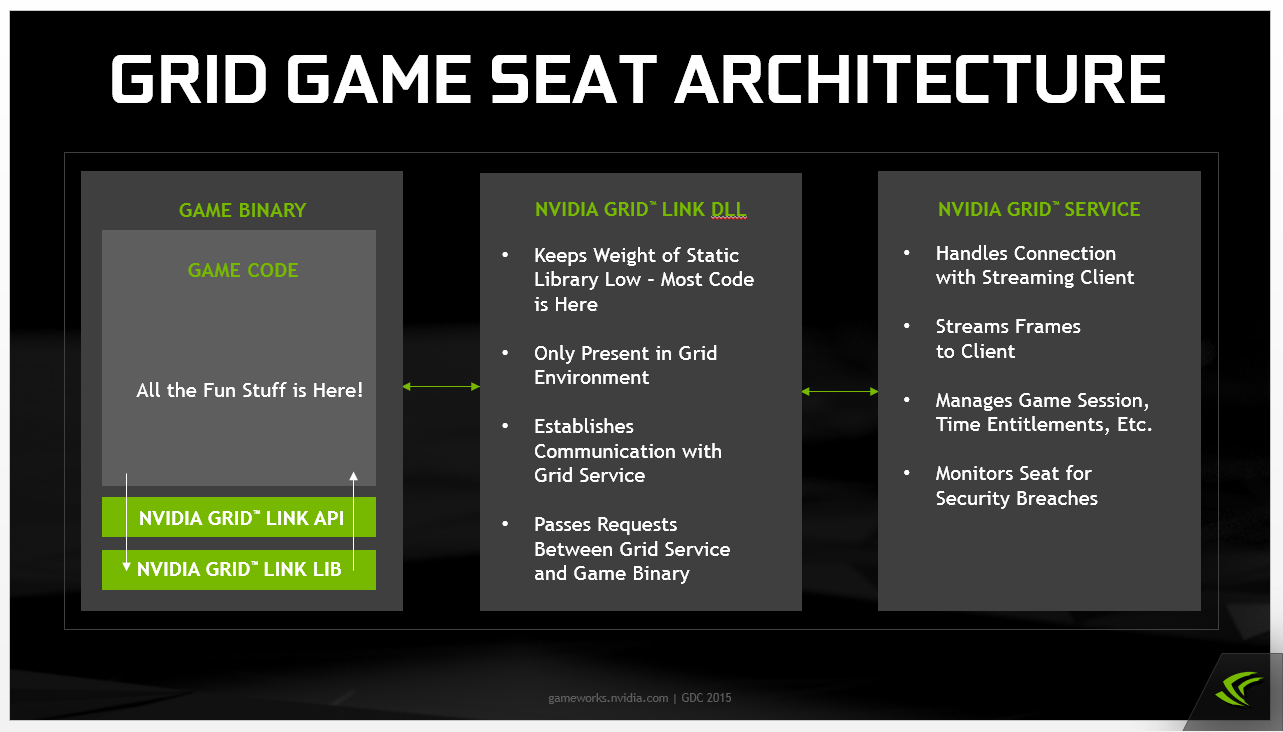
# Architecture

The GFN Link SDK provides a very thin static C/C++ library that is linked to the game/application.

This library checks for the presence of a GFN.dll at initialization time and if no dll is present essentially does nothing. All API methods are no-ops when operated this way so that an application could safely make these calls in all builds without worrying about performance problems or other errors. This way it is not necessary to make a GFN specific build or #ifdef out GFN API methods for other builds.

If the dll is there, the dll will attempt to establish communication with the GRID Service and passes any necessary information through that channel.

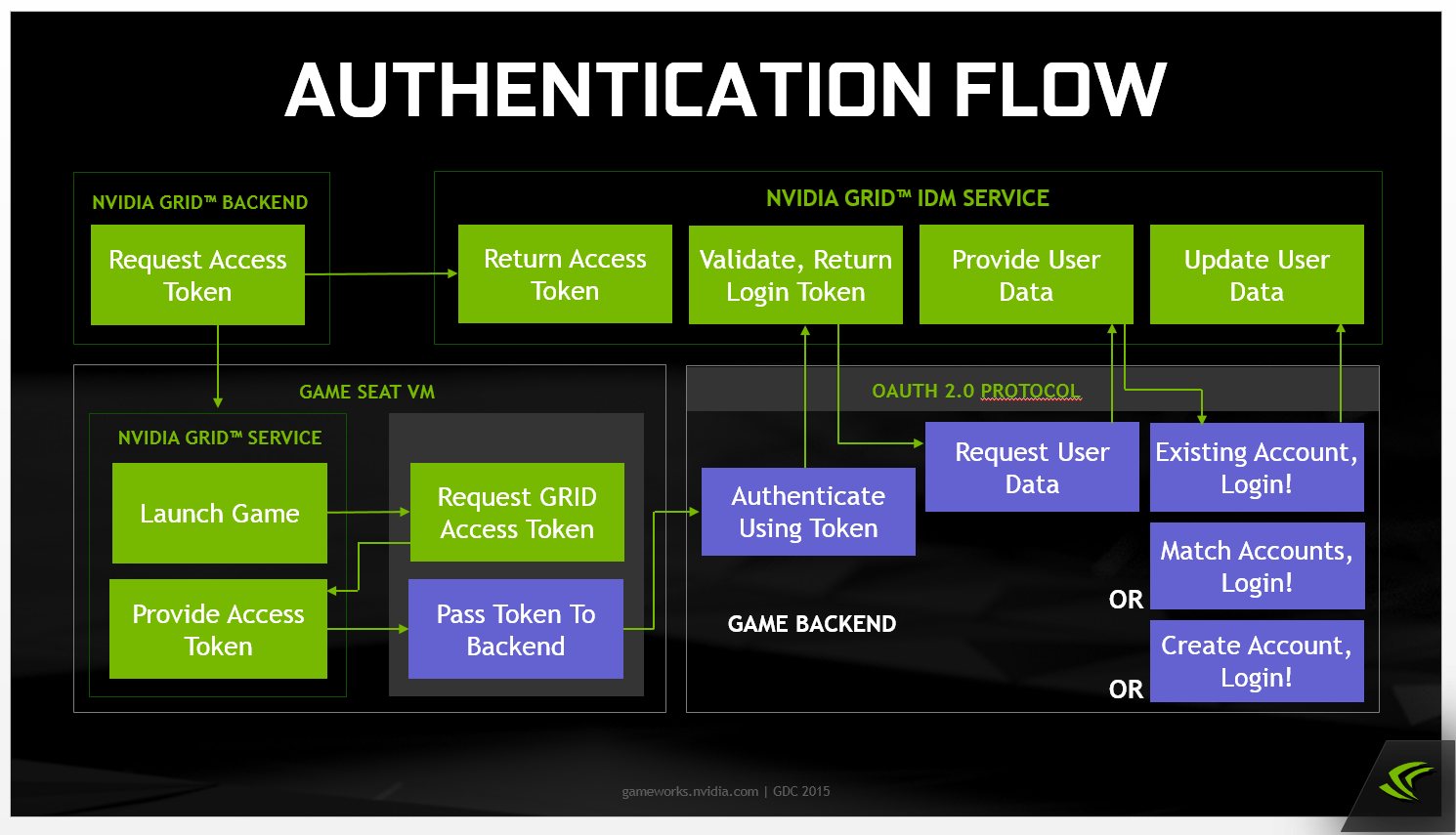
A similar situation is used when testing the application during development, where the GRID Service replaced with a test application provided in the GFN Link SDK package.



# Authentication and Federation

NVIDIA provides a backend IDM Service that application developers can use to validate users and obtain user information from.

A simplified flow diagram of how this functions is shown below:



Details of this system to be provided on an as needed basis. Please contact your NVIDIA representative for more information.

# Development and Debugging

### GFN Link Host Emulator

NVIDIA provides a simple test tool in order to help with testing and debugging your application. This is located in the Samples\HostEmulator directory.

In order to use this tool, you should run the tool prior to starting your application. You may then launch your application through the tool or separately.

For reference, you can use the SampleCPPGUIApplication to test out the tool. You'll need to either compile the sample application or run the precompiled executable. You can use the “Launch Game” button to select which sample to run.

The tool can be used to test sending the various IGFNApplication commands to your application and also displays and emulates responses to IGFNLink calls from your application. The test tool might display a failure response even if the resulting call succeed due to a time out condition. This will be most noticeable if you have a debugger attached to your project.

### GFN Environment

The “GFN Environment” is used to refer to the case where your application is running on a game seat virtual machine (see Architecture section). This can be emulated using the GFN Link Host Emulator described above.

Application developers should be sure to test their application both in and out of the GFN Environment.

From a technical standpoint, two things must happen in order for the application to be in a GFN Environment:

1. The GFN.dll must be present next to the application executable or in a registered dll path.
2. The GFN Service or host emulator must be running.