GeForce NOW: How to Create a Mobile Touch Game Streamed From the Cloud

Design and Integration Reference

GFN_Mobile_Touch_v1.2 | 05 2022

Document History

Version	Date	Description of Change
1.0	*10/06/2021*	*Initial release revision*
1.1	*03/07/2022*	*Added touch support for common game engines*
1.2	*05/17/2022*	*Wording cleanup in various sections*

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.

As most PC games are designed to obtain input from keyboard and mouse input or XInput-based controllers, running these games on touch capable mobile devices creates a challenge that would traditionally be solved with a port of the game for mobile devices. GeForce NOW has the ability to translate touch-based inputs into mouse inputs, with little work by game developers. For games to work well with touch inputs, NVIDIA recommends that game developers support touch inputs natively. This allows PC games to be enjoyed by a very large install base of users that are signing up for GFN to instantly stream high quality PC games.

NVIDIA has discovered that many developers who port PC games to mobile do their development on PCs. Therefore, it is relatively easy to enable the touch controls on the PC version of games for onboarding onto GeForce NOW's cloud servers. This allows the full version of these games 10-50GB in size to be streamed to mobile devices with high quality textures and graphics. Users benefit by not having to download, install and update games, which saves valuable storage space on mobile games as well as extended battery life since the processing load and power is transferred to the cloud. Also older phones and tablets will now be able to play the latest games through the power of GFN cloud servers eliminating problems meeting minimum specs for games.

Audience

This document is directed towards game developers that wish to onboard their games into GFN, and would like to have their games readily playable by GFN users utilizing touch-based devices.

Overview

This document provides an overview of the GFN cloud environment and Mobile Touch support implemented in GFN, and describes how game developers can modify their games to leverage GFN's "Mobile Touch support" capability and stream PC games to mobile device users.

Key Concepts

1. **Overview of the GFN ecosystem**. This section provides a high-level architecture for how games are run on GFN servers.
2. **Overview of Mobile Touch**. This section provides a high-level architecture overview of Mobile Touch support.
3. **Technical Requirements to Support Mobile Touch.** This section discusses the technical aspects of leveraging Mobile Touch in your game.
4. **Touch Support for Unmodified Games.** This section describes touch support options for GFN-supported games that have not fulfilled all the requirements for full touch support.

1. Overview of the GFN ecosystem

GFN is an ecosystem that facilitates cloud-based PC gaming on a wide range of devices. The PC games run on high-performance Windows-based Virtual Machines located in various data centers around the world, while variants of the GFN client that run on Windows, macOS, Android phones, Android TV, Chromebook, iOS, and NVIDIA SHIELD products host the gaming experience for the user. This allows almost any system, even underpowered or older ones, to play the latest and most hardware-demanding games.

For more information on GFN, including getting access to GFN if you do not already have it, please visit https://www.nvidia.com/en-us/geforce-now/. For a listing of the latest global data centers and zones check out the GeForce NOW status page https://status.geforcenow.com/

2. Overview of Mobile Touch Support

GFN's mobile touch experience aims to provide a seamless native gaming experience on mobile devices, letting users interact with the game using touch instead of a gamepad. The experience is tuned to wider screens with higher pixel densities and enables using touch as the primary input method. To realize this experience, GFN detects specific information about the user's device and then provides information cues to a game to create an optimal visual and input experience for that device. In addition, the GFN Software Development Kit (SDK) provides the ability to integrate deeper with GFN by integrating PC game UI with on-screen keyboard input.

Mobile Touch support is currently available for GFN clients on Android and iOS devices.

GFN Mobile Touch experience comprises the following features:

- Game using touch input
    - GFN supports gaming using touch input on mobile devices for all games that support touch inputs on Windows operating systems. GFN relays touch input from the user to the GFN gaming servers, where they are injected to the operating system which then generates touch or gesture messages, as the game requires.
- Full-screen gaming
    - GFN streams the game video in wider aspect ratios typically used in mobile screens to provide full screen gaming experience.
- Intuitive text input
    - Text input is essential to enter usernames, passwords and chat messages in games. GFN provides APIs to the game to indicate edit boxes that it then uses to auto bring up / dismiss keyboards, auto pan to center the edit box on the screen and so on.
- Context sensitive touch
    - While streaming a game, GFN clients intelligently determine when to respond to user touch in-application or to send inputs to the game, enabling intuitive usage of the application
- Support for other input types
    - On Android devices, users can also utilize other input devices such as keyboard, mouse and gamepad, as defined by game support. On iOS devices, users can utilize gamepads in addition to touch for input in games.

3. Technical Requirements

This section describes how GFN realizes the touch-based mobile gaming experience and how game developers can leverage GFN features and interfaces in their games to deliver the best mobile gaming experience to users.

Mobile Touch support feature	How GFN supports the feature	How PC games can leverage the feature
Touch input	The GFN client detects touch capability of the game using a GFN game metadata store and requests the touch input mode from the GFN server. The GFN client determines, based on the context within a game session, if touch inputs must be relayed to the GFN servers or must be interpreted and responded to at the client.	Games must support standard Windows touch events or gestures. See here for Windows guidelines that games must follow. Games should customize the touch interface for small screens.
Full-screen gaming	The GFN client auto-detects the screen resolution for the user device, and requests the game to render at a resolution that allows full-screen gaming on the user's device.	Games should support rendering at mobile device aspect ratios. See here for a list of aspect ratios and resolutions. Games should customize UI elements to avoid edges and notches. See here for description of unusable screen areas
Text Input	GFN provides C-based APIs to let the game signal when the game UI contains edit boxes for user input. These APIs detect when the edit box comes into / loses focus, brings up the keyboard, auto-pans and centers the edit box over the keyboard.	Games should call into GFN to signal screen areas containing edit boxes. See here for description of relevant GFN SDK API.
Other input devices	GFN clients can detect a range of devices paired with mobile devices in addition to touch. GFN can seamlessly switch to the input device the user prefers.	If the game supports switching from one input device to another during a given session, users can move between input devices.

Touch Input

This section describes how games must support touch on Windows in order to support Mobile Touch through GFN.

Developing a Game for Mobile Touch

If your game already supports touch input on Windows, then it will also support Mobile Touch through GFN. Games can improve the gaming experience by adapting for mobile screens.

In some cases your game's engine or framework will support some or all of these modifications. If so, we recommend enabling that support through the engine's or framework's configuration, and handle any additional work directly in your game.

Touch input on Windows

Games that wish to make use of touch input must be developed in accordance with Microsoft's guidelines for Touch Input. Please see https://docs.microsoft.com/en-us/windows/win32/wintouch/windows-touch-portal for more information.

For Windows-based games to support touch, they must:

- Configure their window for receiving touch events (if using `WM_TOUCH` messages).
- Add handling of `WM_TOUCH` or `WM_GESTURE` messages.
- Specifically ignore other window messages that are synthesized from touch input.

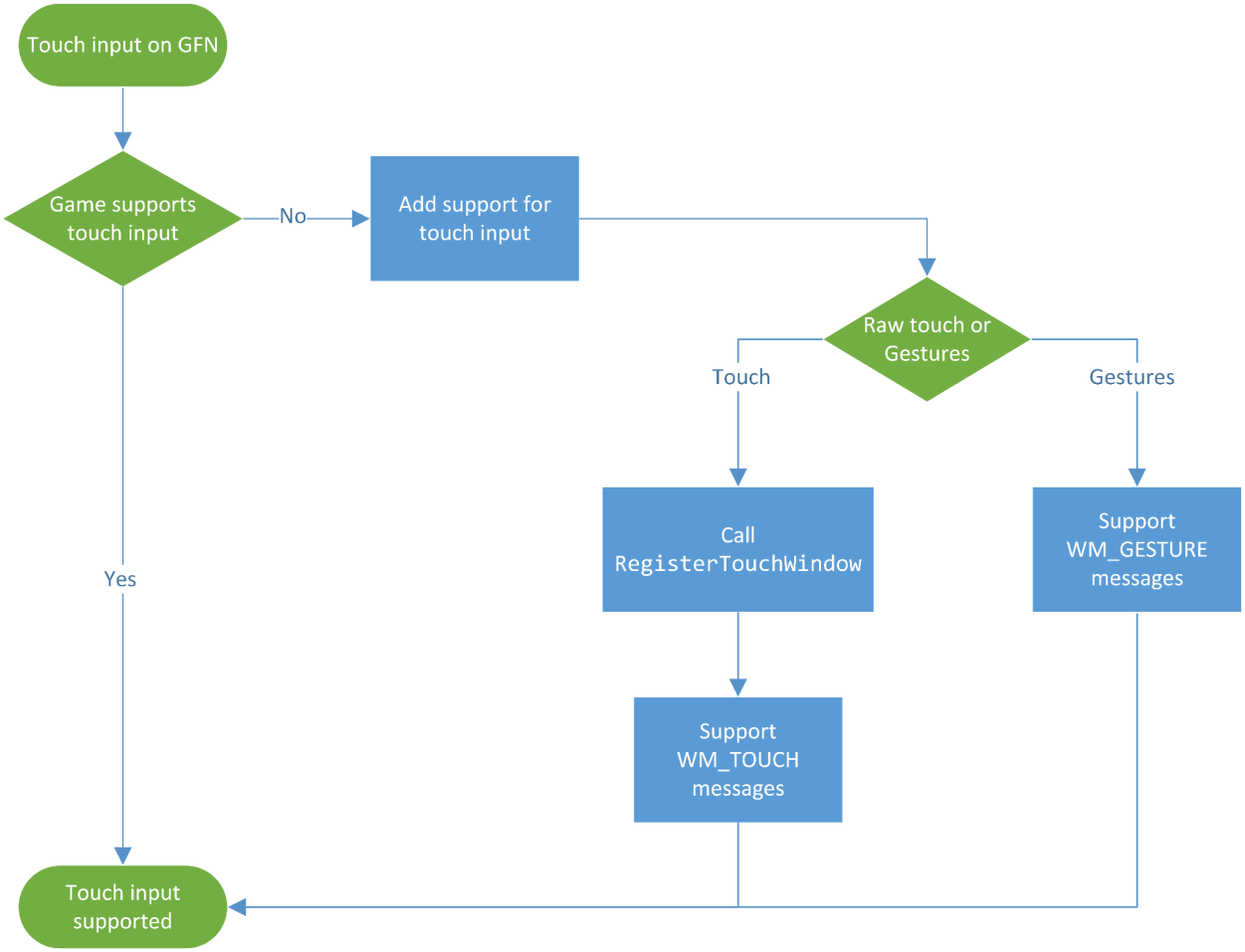
Additionally, it can provide a more integrated experience to modify the touch experience by doing some of the following:

- Disabling Windows touch visualizations, such as the translucent square that is shown for long-press animations.
- Choosing whether to have Windows disable palm detection of touches.
- Deciding whether to have non-coalesced input.

Configuring the Game Window

If a game uses `WM_TOUCH` it must call `RegisterTouchWindow()` to start receiving `WM_TOUCH` messages instead of `WM_GESTURE` messages.

If a game prefers `WM_GESTURE`, no special setup is required.



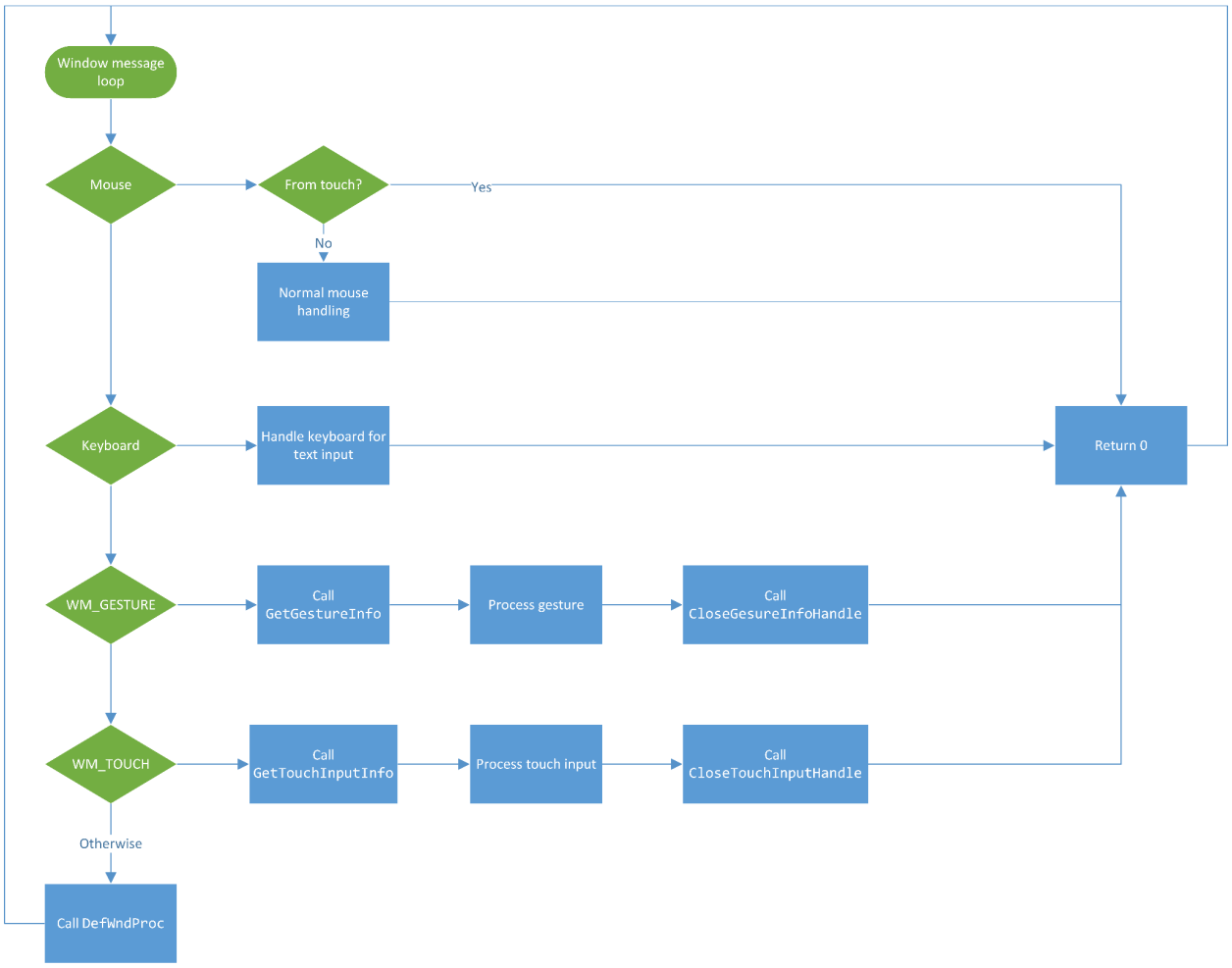
Receiving messages

`WM_TOUCH` and `WM_GESTURE` are delivered to a game in the same way that other input messages are - through the window procedure (`WindowProc`).

Such messages should be received by the usual message loop and interpreted as applicable to the game's needs.

The default window procedure (`DefWindowProc`) will clean up and invalidate the message's data, so it must not be called before handling the message. If the message is not handled, `DefWindowProc` **must** be called on the message.

For any messages that are handled, clean up must occur - `CloseTouchInputHandle()` for `WM_TOUCH`, and `CloseGestureInfoHandle()` for `WM_GESTURE`.



Ignoring Synthesized Messages

Windows synthesizes mouse messages when touch input occurs. This allows applications that are not aware of touch input to continue to work with a touchscreen setup, as touches generate appropriate mouse input messages as well as the touch messages.

This can cause your game to double-respond, as it will receive both message types.

The canonical way to fix this issue is to check in your mouse handling code whether the message has come from a touch input or not. For full details, see https://docs.microsoft.com/en-gb/windows/win32/tablet/system-events-and-mouse-messages

Essentially, the game must check the extra message information (from `GetMessageExtraInfo()`) to see if it matches the bitmask `0xFF515700`:

```
#define MOUSEEVENTF_FROMTOUCH 0xFF515700

if ((GetMessageExtraInfo() & MOUSEEVENTF_FROMTOUCH) == MOUSEEVENTF_FROMTOUCH)
{
	// Message was generated by touch input; ignore
} else {
	// Message was generated by the mouse; process
}
```

Note that `GetMessageExtraInfo()` must be called while handling the message; the information is only valid for the current message and cannot be obtained after the message has been handled or passed to `DefWindowProc`.

User Interface and Experience

A game designed for touch input will often have a different user interface and user experience from one designed solely for gamepad or keyboard and mouse input.

There are many options for such a game. However in general when touch input is used:

- Mouse-style movement is better replaced by an alternative:
  - For example, if a game uses mouse movement to move a virtual-world camera, then that needs to be replaced by a different input mechanism - commonly, dragging a finger around on-screen - to provide that same functionality.
- Scroll-wheel gestures are no longer easy
  - Instead of having a scroll-wheel style selection of items, a replacement would be to have an explicit inventory mechanism that allows selection via touching the inventory item.
  - If your game provides its own zooming, you must all respect and support the pinch-to-zoom gesture that is now the common way of zooming on touch devices.
- Gamepad controls are not present
  - Use on-screen tappable areas, or provide an alternative input system, rather than relying on “always available” gamepad controls.

- Keyboard input isn't viable
    - Whilst text entry might be possible, for gameplay the keyboard isn't available. Try to provide different user interface elements to access any game functions that the keyboard would be used for. For example, a touchable fire button or an automatic firing system could be an option.
- Physical screen size is smaller
    - Mobile touch devices tend to have physically small screens, even if their screen resolutions are large. Ensure that your game interface works when the elements are physically small; make them large enough to see and touch cleanly.

Fullscreen Gaming

All games on GFN are played in a fullscreen mode, taking up most or all of the device's screen area. When playing on mobile devices, this can have some implications that need to be considered and handled to achieve the best gaming experience.

Screen resolutions

Touch devices often come in wider screen resolutions and aspect ratios than standard PCs, and games must support these mobile screen aspect ratios. Typical aspect ratios used in mobile screens are 16:9, 18:9, 18.5:9, 19:9, 19.5:9, 20:9. GFN clients will auto detect the mobile screen aspect ratio and request the game to render at a resolution corresponding to the detected aspect ratio as in the table below.

Supporting the resolutions in the table below will ensure fullscreen game video and avoid letterboxing / pillarboxing.

Aspect ratio	Resolutions	Example Devices
4:3	1024x768, 1080x810, 1194x834, 1366x1024	iPad
3:2	1133x744	iPad Mini 6 (2021)
10:7	1194x834	iPad Pro 11" (2018-2020)
16:9	1920x1080, 1280x720	Samsung S6/S7 and iPhone 8/9
18:9	2036x1018, 1356x678	
18.5:9	2046x990, 1364x660	Samsung S8/S9
19:9	2052x972, 1368x648	Samsung S10
19.5:9	2106X972, 1404X648	iPhone 11/12

20:9	2120x954, 1400x630	Samsung S20

Unavailable Screen Areas

Most mobile touch devices have areas of the screen that do not respond if touched, or that should not be used for displaying important information. Touch warping and Safe areas are techniques to mitigate this.

Safe Areas

Some devices have screens that are not entirely rectangular. For example, a number of phones have notches or cutouts for buttons or user-facing cameras. Regions close to the edges may also be used by operating system-level UI elements. Display elements in such regions may also be cropped unpleasantly.

It is recommended that games provide a safe area, where important elements are not placed. Any UI elements should be moved away from the edges, and out of the region where notches, *etc.* might be found.

In the diagram above, the white area is suitable for displaying game content. The blue area should be avoided.

Touch Warping

The GFN client will ‘warp’ touches that are at or near the edges of the touchable area to be closer to the edge, to allow for touching of on-screen elements or controls that are near the edges of the screen.

However, this is only intended for the non-touchable areas at the very edges of the screen, and does not accommodate safe areas in their entirety.

Further, touch warping does not prevent your game elements from being shown on a part of the screen that is unavailable for context.

In the diagram above, touches in the red area would not be detected, and so touches near to the red area in the white area adjacent are warped to appear to be within the red area.

Text Input

On-screen Keyboard Control

There are instances during game play where the user must provide keyboard-based input, for example, entering their username or password, or typing the name of a friend to message via an in-game friends list. Traditionally this would mean the game would need to implement some keyboard interface, along with support for the major keyboard layouts and locales.

GFN provides a way for that input to be obtained by the game via the native on-screen keyboard of touch-based devices, thus removing the need for the game to obtain input directly.

Since only the game knows when keyed input is necessary, control over the on-screen keyboard is managed by the game via APIs provided by the GFN Software Development Kit (GFN SDK).

The *`gfnSetActionZone`* API provides a way for a game to define areas of the screen that should trigger keyboard-based input. The API's definition is:

```
GfnRuntimeError gfnSetActionZone(GfnActionType type, unsigned int id, GfnRect* zone);
```

A correct call to this API includes:

- `type` as `gfnEditBox`
- `id` as a caller-defined unique identifier
- `zone` as the bounds of the on-screen region. See SDK documentation on `GfnRect` for more information on how to define this region.

An example code snippet for registering an edit box that is 20 pixels high and 100 pixels wide, with top-left XY coordinates of 5,10:

```
GfnRect editboxRect = {5, 10, 100, 20, false, gfnRectXYWH};
GfnRuntimeError error = gfnSetActionZone(gfnEditBox, 1, &editboxRect};
```

When called, the API signals the GFN client running on the user's host device to monitor the defined screen area for touch input. When the client receives a tap in the monitored area, it will bring up the virtual keyboard and shift up the video view to place the edit box above the keyboard.

This API is designed to be safe to call for all client types; the game does not need to know the client is a mobile device before calling the API. In cases where the client is not touch-enabled, the API call is a no-op.

When a display region should no longer be monitored for such input, for example, the user leaves the game menu that hosts the edit box, the same API is called with the same `id` value, and with the `zone` parameter set to null pointer. This will delete the rectangular screen region associated with the `id` from being monitored. For example:

```
GfnRuntimeError error = gfnSetActionZone(gfnEditBox, 1, nullptr};
```

For more detailed information on the API as well as different ways a `GfnRect` can be defined, refer to the GFN SDK API documentation.

Once the edit box is in focus, the keyboard is automatically brought on screen, and the viewport of the client is panned to center the edit box over the keyboard. As the user enters text via the keyboard, the text is sent to the game server and into the game. Once the keyboard is dismissed, the viewport is panned back to allow the full game screen to be displayed.

4. Game Engine Support

This section provides information on how to enable touch support in games that utilize common game engines to allow games to support Mobile Touch when running in GFN.

Unreal Engine 4 and 5

Epic Games' Unreal Engine 4 and 5 (UE4/UE5) provides support for touch input as well as on-screen control overlays. As of version 4.27.2 and 5.0 Preview, Mobile Touch support is enabled by several changes in a project:

- Setting up Mobile Platforms
- Enabling and handling Touch Events
- Setting up the Touch Control Overlay
- Invoking the game in touch mode

Note that the steps below focus on UE4. The same exact steps apply in UE5 as well; the difference is slight differences in option access/placement in the new Editor UI.

Setup Mobile Platforms

While GFN hosts user games in a Windows environment, GFN supports user playing games on multiple mobile platforms. For the best experience on these platforms, the game should enable the platform support at a minimum and preferably have the user interface optimized for these platforms. Support for these platforms should be enabled as follows:

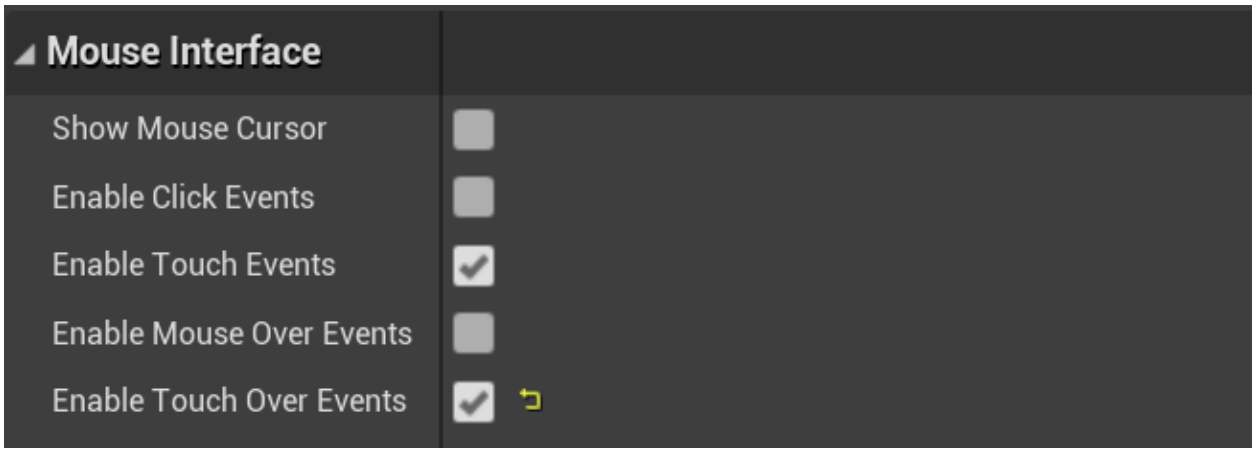
- Open the Project settings, and navigate to the "Platforms" section.
- Under "iOS", configure project settings and make sure "Supports IPad" and "Supports IPhone" are checked:

- Under "Android", configure the package information. The rest of the settings can be left to their default values.

These options will enable touch overlay options for these platforms in the UE4 Editor.

Enable Touch Events

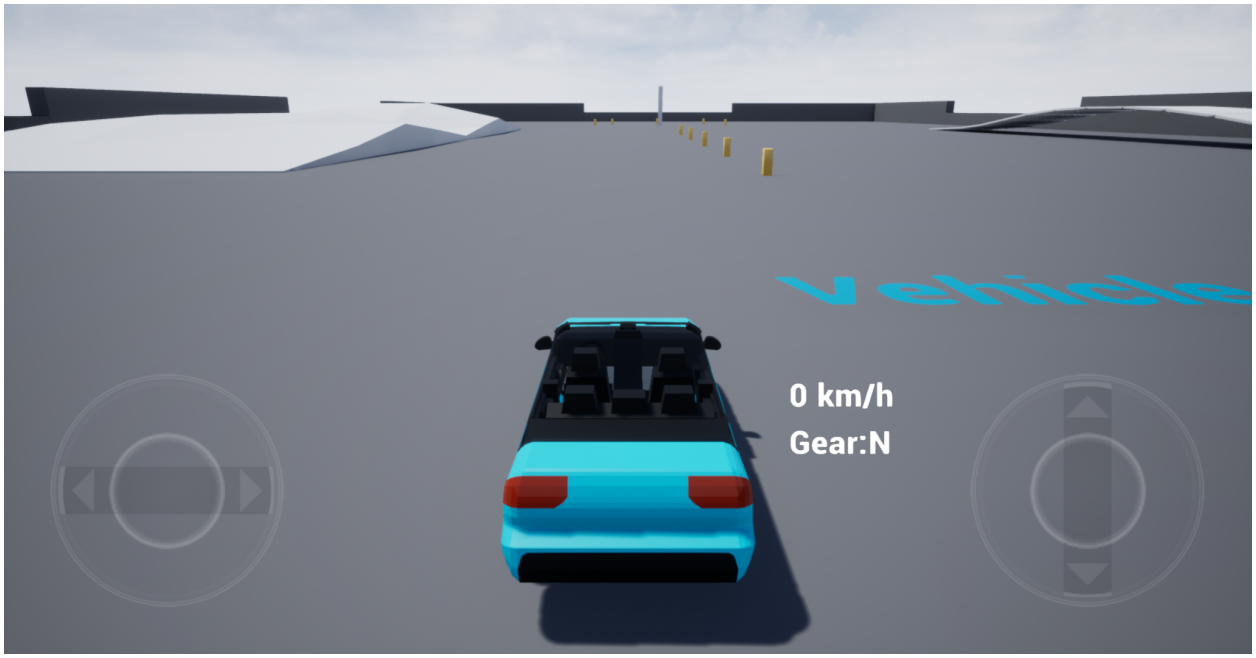
- Open the Player Controller Class and expand the "Mouse Interface" group.
- Check "Enable Touch Events" to receive Touch Events of
    - On Input Touch Begin
    - On Input Touch End
- Check "Enable Touch Over Events" to receive the Touch Over Events of
    - On Input Touch Enter
    - On Input Touch Leave



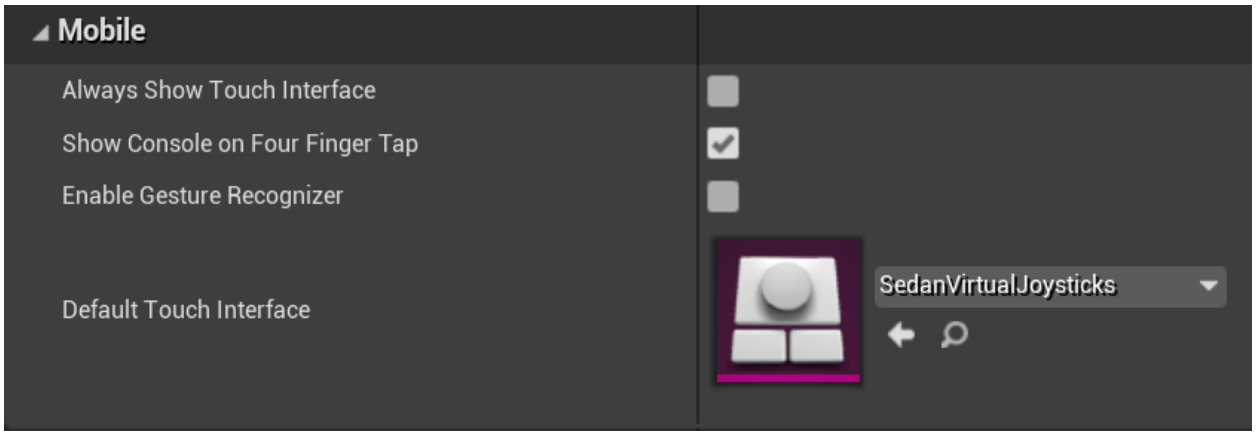
For more information about the events and how to connect them to specific actions, refer to the UE4 documentation found at https://docs.unrealengine.com/4.27/en-US/BlueprintAPI/Input/TouchInput/, or whichever version of UE4 is used by the game.

Setup Touch Overlay

Now that the game supports touch events, the next step is to inform the user where to touch on screen for game actions. UE4 provides default settings of two virtual joysticks to designate screen areas for touch input, and the sample projects provide a few customizations of this overlay:



This overlay is enabled and customized via Project Settings under “Input”->”Mobile”:

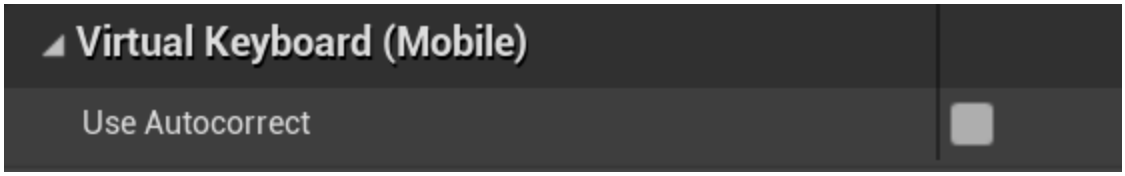


It is recommended to customize this overlay to provide users a great experience when playing on a touch device, making sure that the controls do not clutter the UI or block game viewing as well as adhere to other user experience recommendations found in this document. For more information on how to customize the overlay, please refer to the UE4 documentation.

It is not recommended to enable “Always Show Touch Interface” as this would show the touch control overlay in scenarios where touch is not enabled on the user’s client device.

Virtual Keyboard Support

UE4 provides support for an on-screen virtual keyboard when run on mobile devices and is enabled in the same “Input” section of the Project Settings:



This keyboard can be used if the game requires text input by the user. However, as mentioned in the previous section of this document, GFN provides support for displaying the user’s preferred keyboard on their mobile device and sending the game input from that keyboard. Using this approach is preferred over the UE4 virtual keyboard.

When to Enable Touch Input

UE4 will detect the system hardware and determine if touch should be enabled. However, since GFN utilizes Windows-based systems to host games, UE4 will not consider these systems as touch-enabled. While GFN will send touch events for user input and UE4 will process them, the touch overlay will not be enabled by default and can confuse a user expecting to provide touch input. Therefore, there is additional work necessary to invoke the game with overlay support, which is accomplished by integrating the GFN SDK.

On game startup, the game should:

1. Initialize the SDK via *`gfnInitializeRuntimeSdk`*.
2. Call *`gfnIsRunningInCloud`* to determine if the game is running in GFN.
3. If *`gfnIsRunningInCloud`* returns true, call *`gfnGetClientInfo`* to get information about the client Operating System.
4. If the API returns one of the mobile OS types, then it can be considered to support touch, and the game *may* manually enable the on-screen overlay.
    a. A future SDK release will expand *`gfnGetClientInfo`* with direct knowledge of input type.

Note that there is a measurable number of GFN users on touch-enabled devices that do not use touch input, opting for controllers or other connected devices instead. For this set of users, displaying a touch-oriented interface can lead to a poor experience. To prevent this, a game must provide a way for a user to set a persisted game option that defaults input to their preferred type. This option should be used on all GFN launches going forward when *`gfnGetClientInfo`* `returns the touch-oriented client device type.`

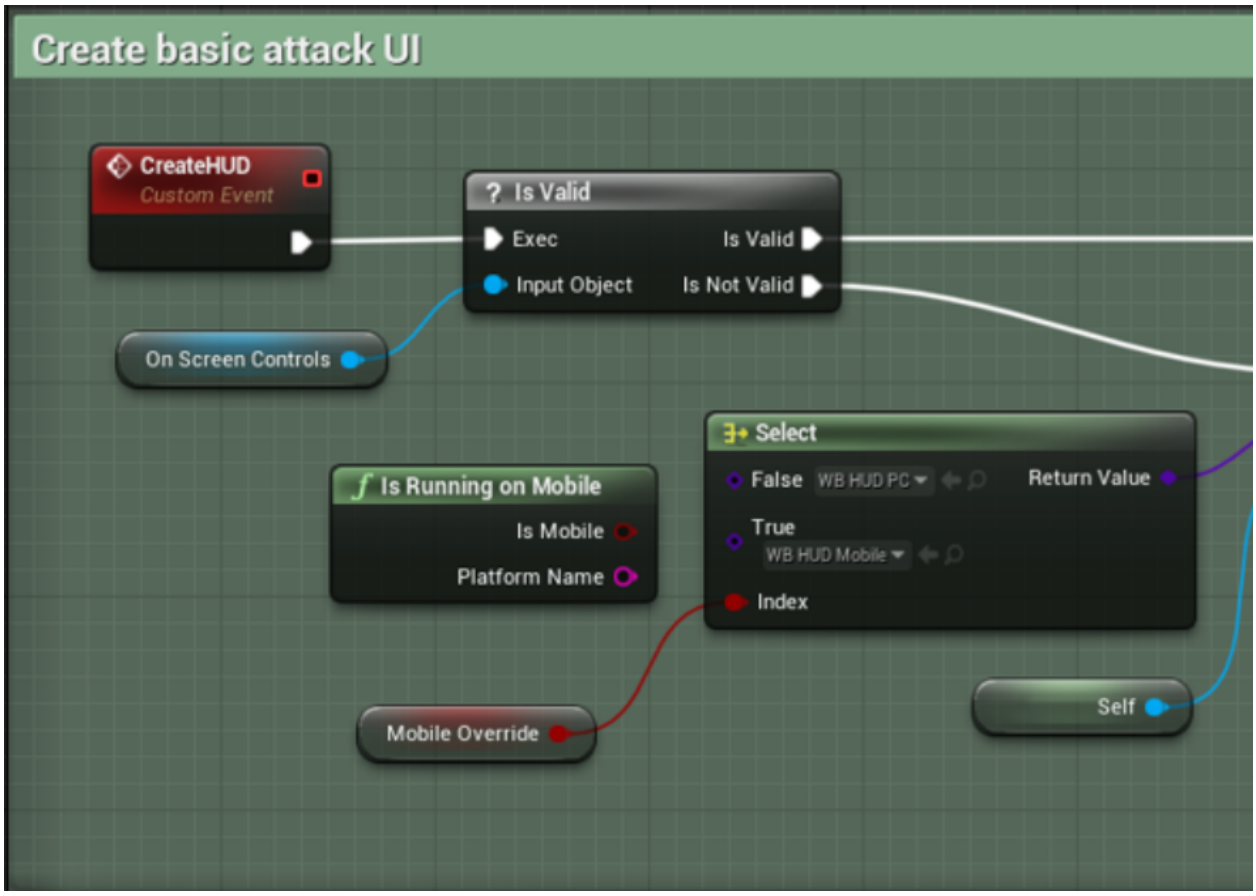
Alternatively, it is possible to pass a startup command line option to the game to denote when the client device is touch-enabled. As this method requires custom configuration with GFN,

please contact your NVIDIA Representative or NVIDIA Developer Relations for more information on this method.

Manually Enabling Touch Control Overlay

Enabling the overlay when either the API checks return touch enabled or by existence of a touch command line can be accomplished by setting the “Is Mobile” option in the PlayerController Blueprint as well as defining the Platform Name that coincides with the OS type returned from *`gfnGetClientInfo`* API.

To set the Is Mobile option, set the “Is Mobile” option to true or create a Mobile Override that is set to true.



Unity 2018

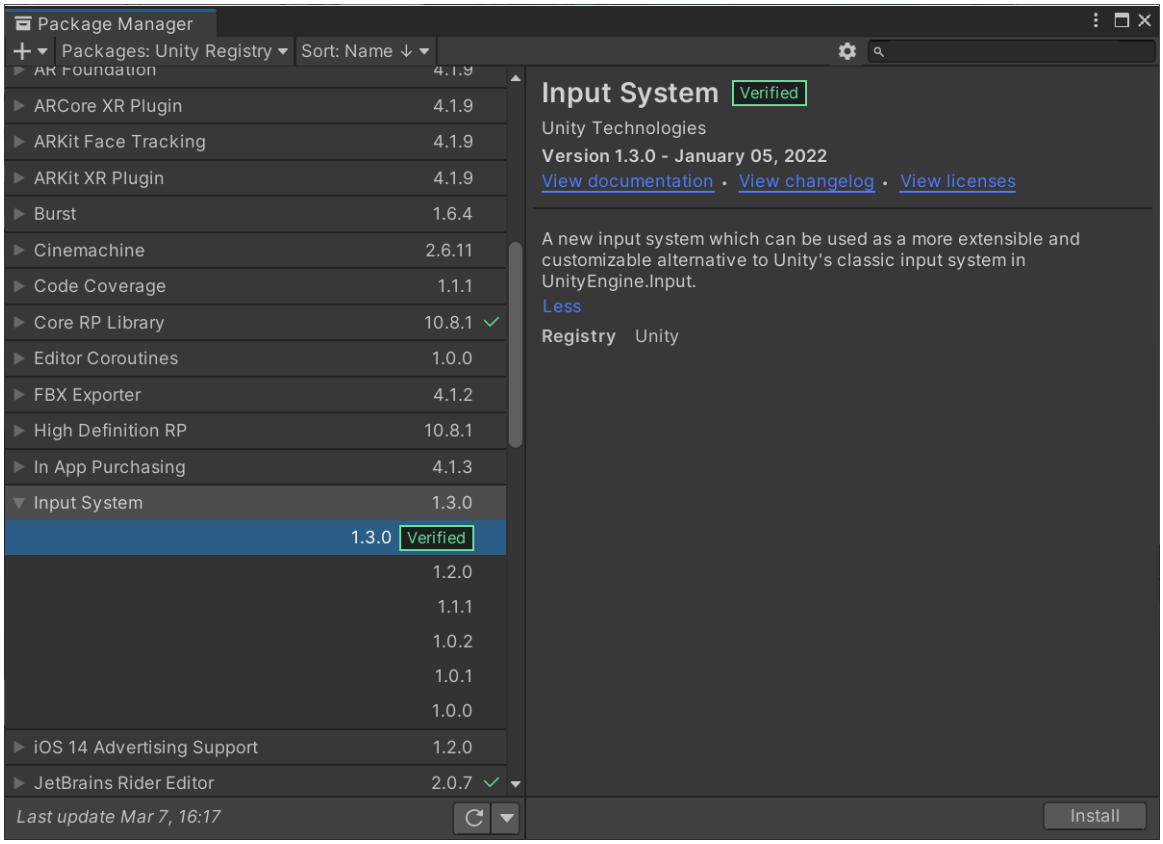
Unity 2018 and later versions support touch input, allowing Unity-based games to natively support touch-enabled devices alongside other input types. Enabling a game to support touch input requires a handful of steps. These steps include:

- Installing and enabling the new Input System (if not already installed)
- Configuring Touch Input Actions
- Configuring Input Overlay
- Invoking the game in touch mode

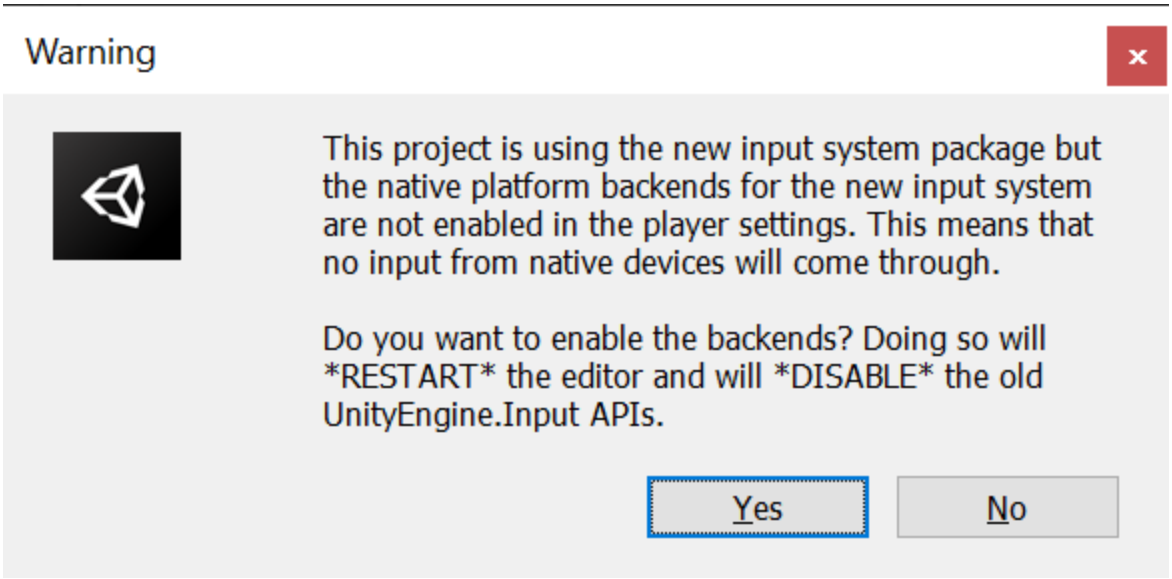
The steps defined below are specific to Unity 2018, and subject to change in newer versions.

Using the New Input System

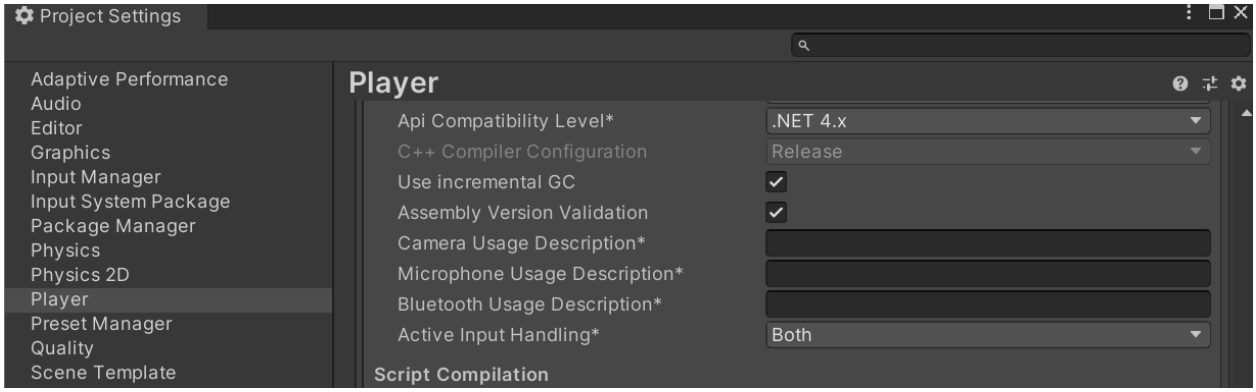
The first step is to make sure the new Input System that supports touch is available to the game. This is accomplished by opening the Package Manager under the “Window” menu item, and then setting the package source to “Unity Registry”. From there, scroll down to “Input System” and select “Install”.



Once installed, Unity will ask if the input backends should be enabled:



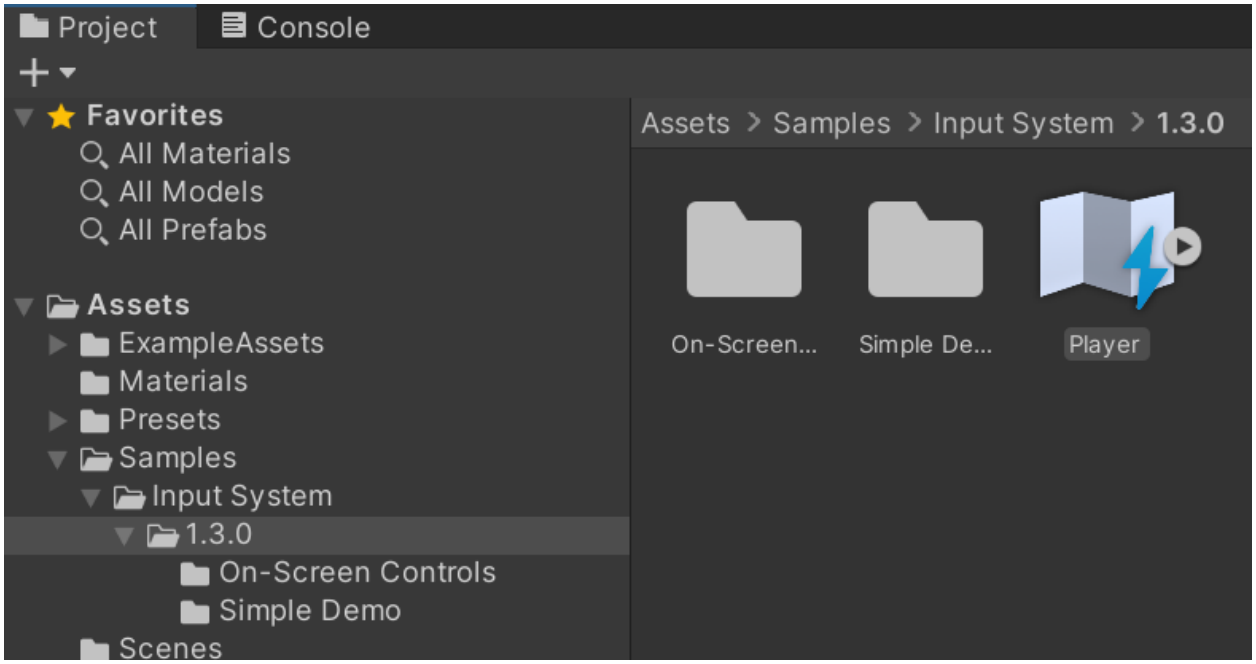
A selection can be made in this dialog, or it is possible to make changes in the Project Settings under the “Player” section, and then adjusting the value for “Active Input Handling”. For touch input, select either “New” or “Both” if legacy input options wish to be preserved.



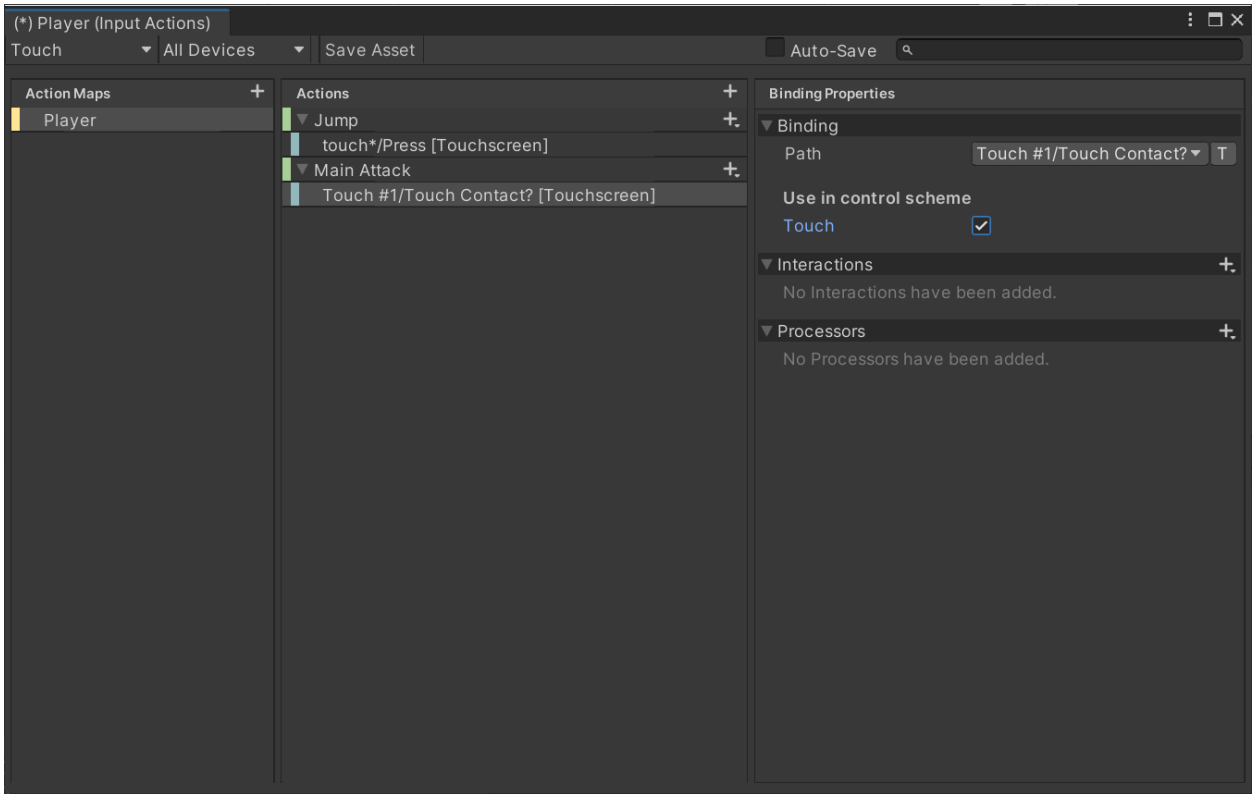
Any change to the Active Input Handling requires a Unity restart for the change to take effect.

Configuring Touch Events

User-based touch events are enabled similar to keyboard, mouse or joystick input. This is done by creating new player-based Action Maps under the “Touch” Control Scheme. For example, under “Assets” a new Input Action can be created under the Input System.

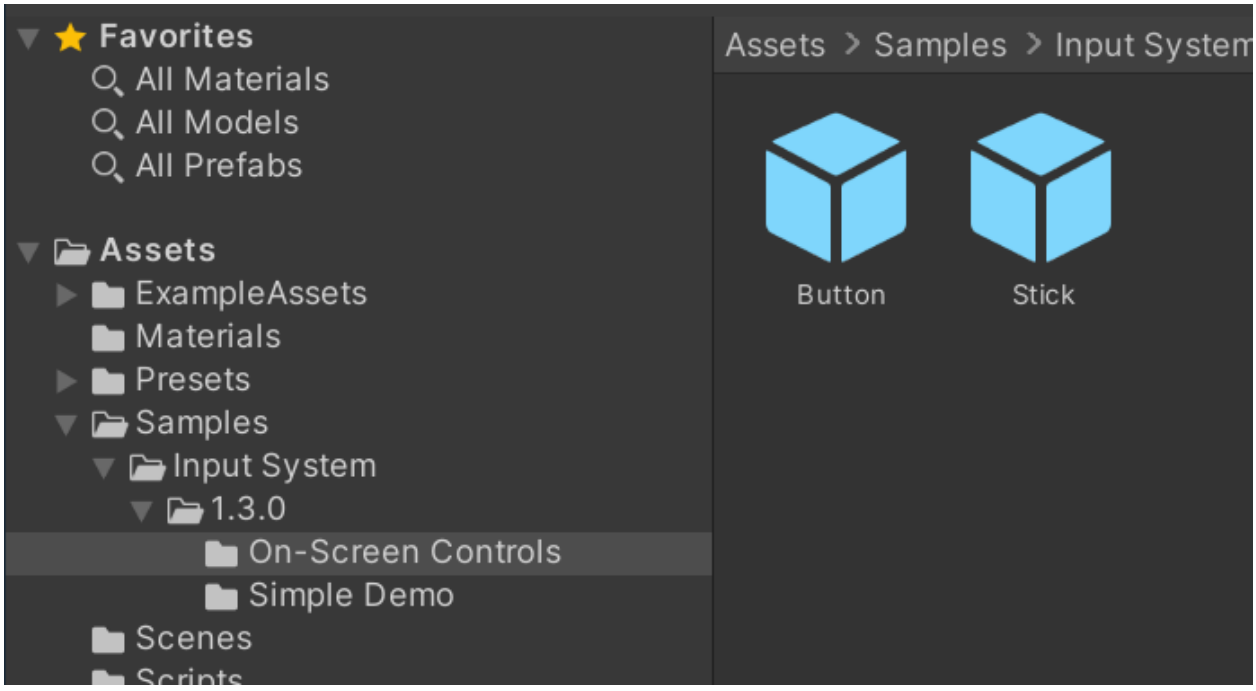


From there, a new “Touch” control scheme can be created, with various actions defined to to either default input paths, or custom ones:



Configuring Touch Overlay

Once touch support is enabled and various inputs are defined to actions, it is necessary to create an overlay of controls to educate the user where to touch on the screen for various input types. There are several ways to create an overlay, however, the easiest method is via the On-Screen Controls sample that comes with the Input System package. This includes a basic stock and button assets, which allow customization on placement on screen, or to be used as templates for various other actions.



For details on how to configure the various settings, please refer to the Unity editor documentation.

As stated earlier in this document, it is important that the overlay be configured to be intuitive to the user, but not block the user's view of the gameplay itself.

Manually Enabling Touch Control Overlay

As covered under the Unreal Engine section above, when a Unity-based game is run on GFN, it will receive Touch input, but it will not detect touch capabilities. As such, in order to know if the overlay should be enabled on game launch, at startup, the game should:

5. Initialize the SDK via *gfnInitializeRuntimeSdk*.
6. Call *gfnIsRunningInCloud* to determine if the game is running in GFN.
7. If *gfnIsRunningInCloud* returns true, call *gfnGetClientInfo* to get information about the client Operating System.

8. If the API returns one of the mobile OS types, then it can be considered to support touch, and the game should manually enable the on-screen overlay.
    a. A future SDK release will expand *`gfnGetClientInfo`* with direct knowledge of input type.

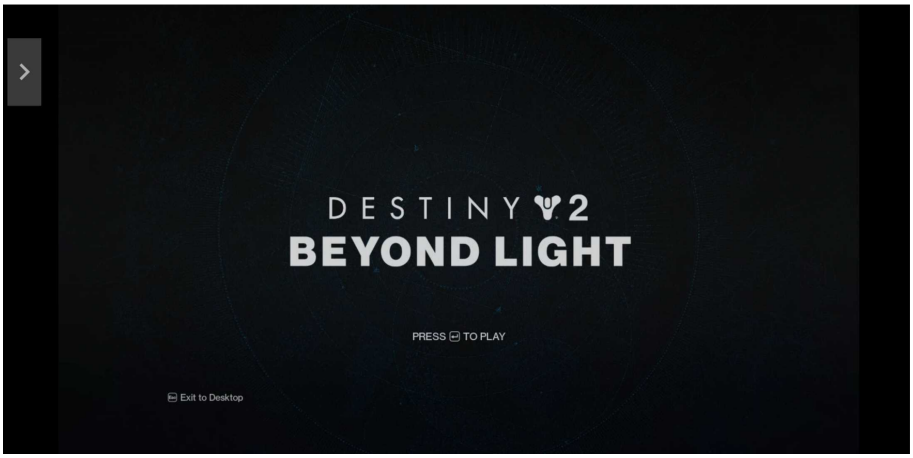
In addition, the same user input preference handling applies when the client device is reported as touch-enabled but the user prefers a different input type.

5. Support for Unmodified Games

In the event that a game cannot immediately support mobile touch as described above, the mobile GFN clients provide support for touch input via a gamepad-like overlay.

In this solution, touches in the specific control areas denoted in the overlay are mapped to XInput messages for the game process just as if a physical gamepad was providing the user input.

GFN clients will also support mouse through touch gestures such as one-tap click, two-tap right click and so on. The GFN client also supports input through on-screen keyboard, although the keyboard does have to be brought up manually using the chevron on top left.



In this case, auto-centering the edit box above the virtual keyboard is not supported.

Note that this input method is a less optimal solution than the mobile touch support described in this document, as it has several drawbacks:

- The overlay's UI elements are static and cannot be moved to accommodate the user's input preferences.
- The overlay's UI elements can interfere with game UI elements, requiring the game to rework the UI for a better experience.
- Depending on the relative position between the edit box and the keyboard, the keyboard may obscure the visual input.

As such, this method of input can be considered a first step to have a game support GFN mobile devices in parallel to full development of mobile touch, but is not desirable to be the final integration.

6. Conclusion

GeForce NOW provides a way for gamers to enjoy their games almost anywhere, and publishers to reach more gamers. With the ability of PC games to be played on mobile devices, the gamer audience is even more expansive and diverse.

Along with integrating other GFN-enabled features such as account linking with your user accounting system and enabling single sign-on to bypass the need for users to log into your game during the stream, enabling mobile touch in your game creates an efficient and seamless experience for your users to enjoy your games on GFN.

To have your games take advantage of the Mobile Touch feature, or any other feature GFN provides, please contact your NVIDIA Developer Relations representative for more information on onboarding and testing your game on GFN..