BRIDGE

SDK Description

for Bridge Overlay SDK 1.0.1

Last update Aug 16th

Contact: bridgesdk@logitech.com

Introduction:

The BRIDGE SDK is a Development kit aimed at helping app makers and SW developers solve the issues arising when a user needs a Keyboard in Virtual Reality.

Motivation:

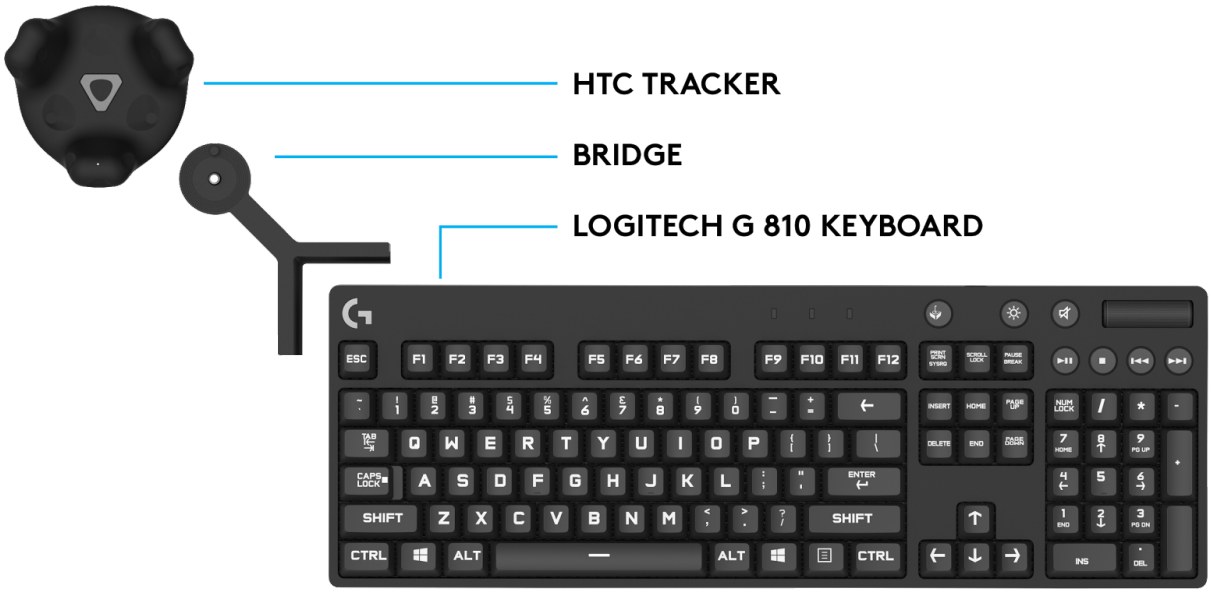
Our motivation comes from the research-backed understanding that in certain situations the user still needs a keyboard to interact with applications, particularly in productivity-driven or desktop scenarios, but also in games, social applications and content browsing.

We believe that a physical keyboard should be present, as it delivers essential tactile feedback and the universal experience that people value.

Components:

The BRIDGE SDK requires the following elements:

- Hardware
  - A Logitech G810 Keyboard (off-the-shelf)
  - A Logitech BRIDGE adapter
  - A HTC VIVE Tracker
- Software
  - BRIDGE OVERLAY Software SDK
    - The SW installer sets up the BRIDGE OVERLAY software on the user's system.
    - Includes a pairing utility to associate a specific VIVE Tracker
    - Overlays a 3D VR keyboard that appears on top of the VR environment.
    - (In progress) SDK to allow developer to control elements of the VR keyboard overlay.
    - (In progress) a representation of user's hands overlaid (capture from the VIVE HMD Passthrough camera) on the VR keyboard.



Setup instructions:

1. Attaching the VIVE Tracker to the Keyboard

1. Ensure that the BRIDGE *Locator Pin* is aligned with the VIVE Tracker locator hole when placing the Tracker on BRIDGE.
2. Secure the VIVE Tracker to BRIDGE by tightening the BRIDGE *Mounting Screw*.
3. Attach the assembled BRIDGE & VIVE Tracker to the top left corner of the Logitech G810 keyboard. Following the leaflet contained in the BRIDGE box, first align BRIDGE to the left side of the keyboard, and then position the other leg on the top of the keyboard and push to make sure is it well secured.

2. Install the BRIDGE OVERLAY SW package

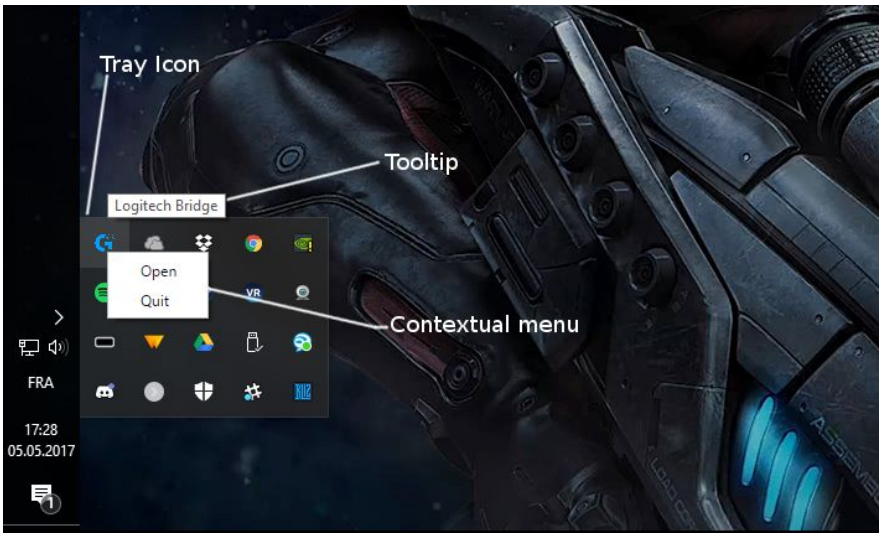
Head to our private GitHub repository: https://github.com/Logitech/logi_bridge_sdk, and clone or download the full content. Follow the README.md instructions.

Download the full zip package and extract it to a folder of your choice.

Once extracted, double click on **Logitech_Bridge.exe** to launch it.

The core Bridge Overlay functionality will run as a service and the main UI will be available in the **system tray** to be accessed whenever needed:

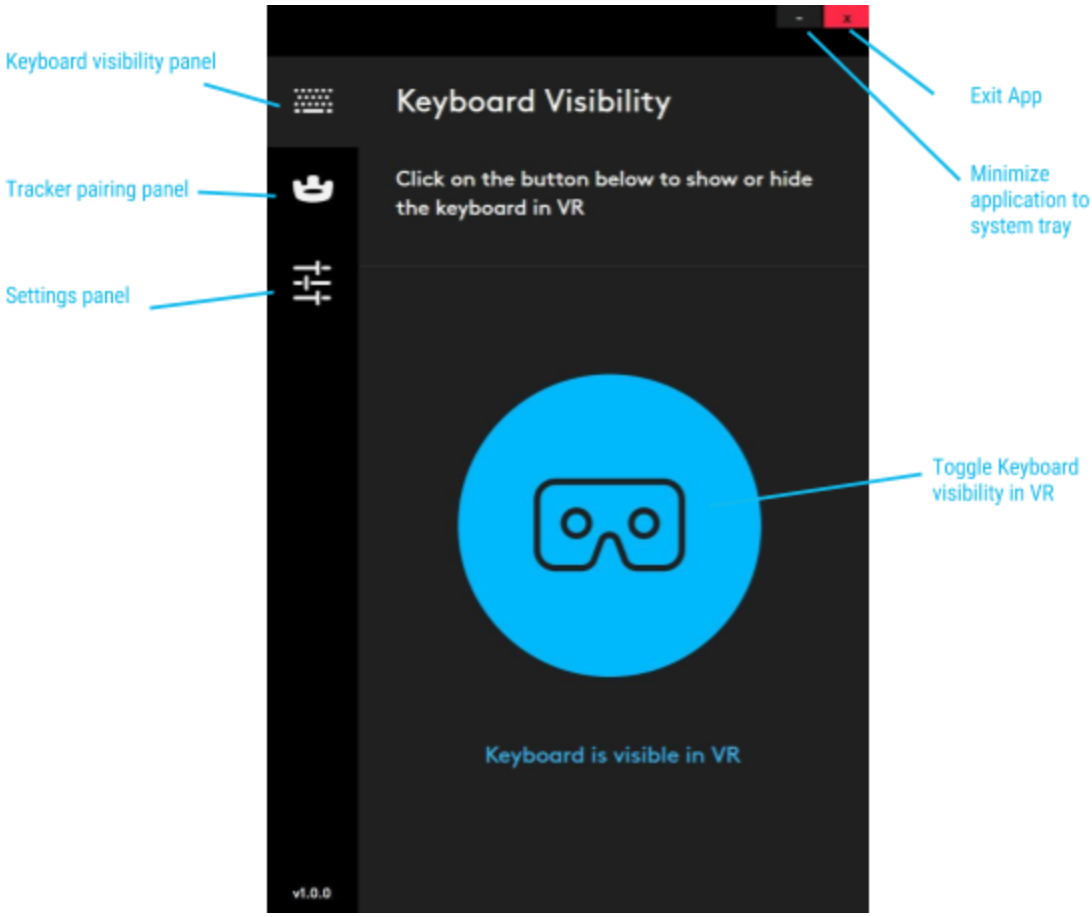
Logitech BRIDGE OVERLAY tray icon



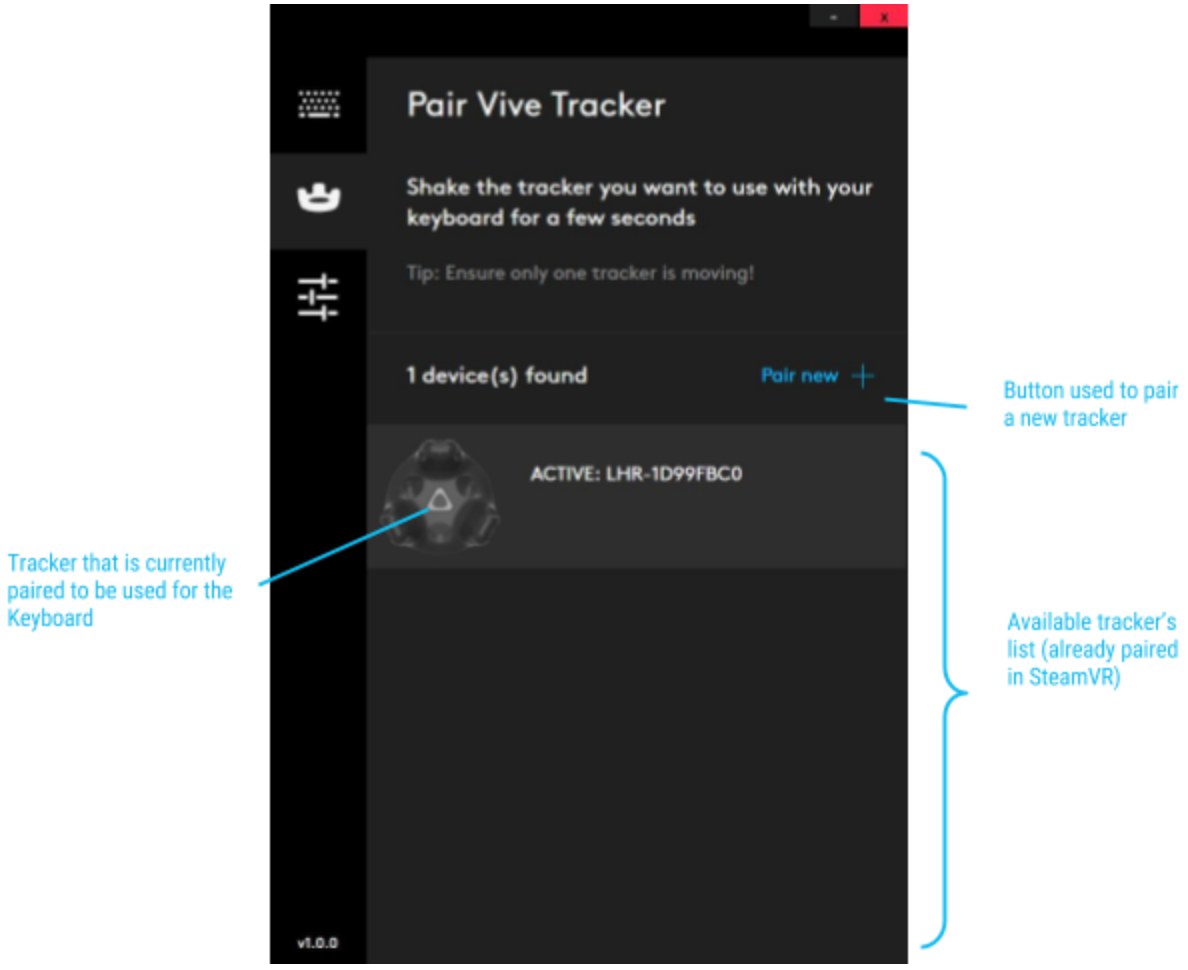
Logitech BRIDGE SDK SW that runs in system tray.

UI overview

The UI allows you to setup the first steps and offers access to various settings to customize your overlay:



The Keyboard visibility panel



Pairing panel with one tracker assigned to the keyboard

The settings panel

The settings panels will be described further down in this document, in the functionality chapter.

Minimizing the application and bringing it back

In addition to notifications, the **notification area** of the taskbar (sometimes referred to as the "**system tray**") is also used to display icons for system and program features that usually have no presence on the desktop; Bridge is one such application.

When the software is in foreground, pressing the minimize button will hide the Bridge application. From the notification area, a single click on the Bridge software icon will bring it back to the foreground.

Double click on the system tray icon should have no effect. Using the right-click on the tray icon of the Logitech Bridge Software, the user can choose from either opening the app, or exiting it.

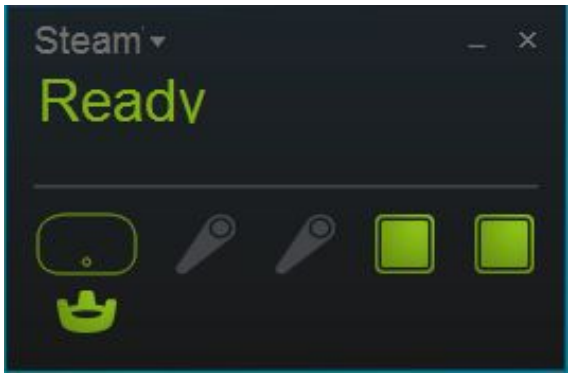
3. Pairing a tracker

A) in Steam VR

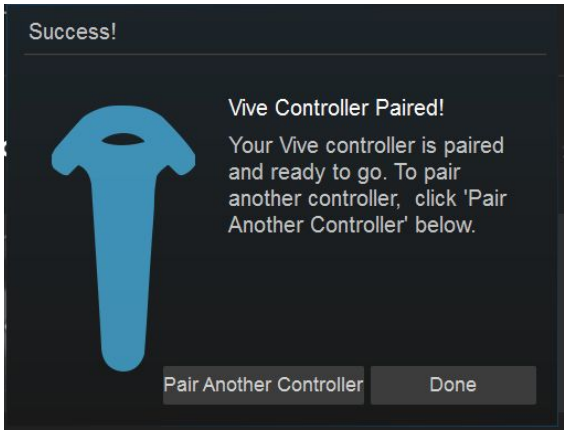
First, pair the VIVE Tracker as per HTC instructions (http://link.vive.com/tracker/guideline).

Make sure you switch the Tracker on in PAIRING MODE, by long pressing the center button, indicated by the Tracker LED blinking.

Use the SteamVR drop down menu ">DEVICES > PAIR CONTROLLER" to pair a new device. Follow the steps there (even if the UI references controllers rather than Trackers). When successful, the new Tracker should appear as below in SteamVR.



Use the SteamVR menu, and head to DEVICES -> PAIR CONTROLLER



After successful pairing, the tracker LED should turn green and you should have this confirmation screen.

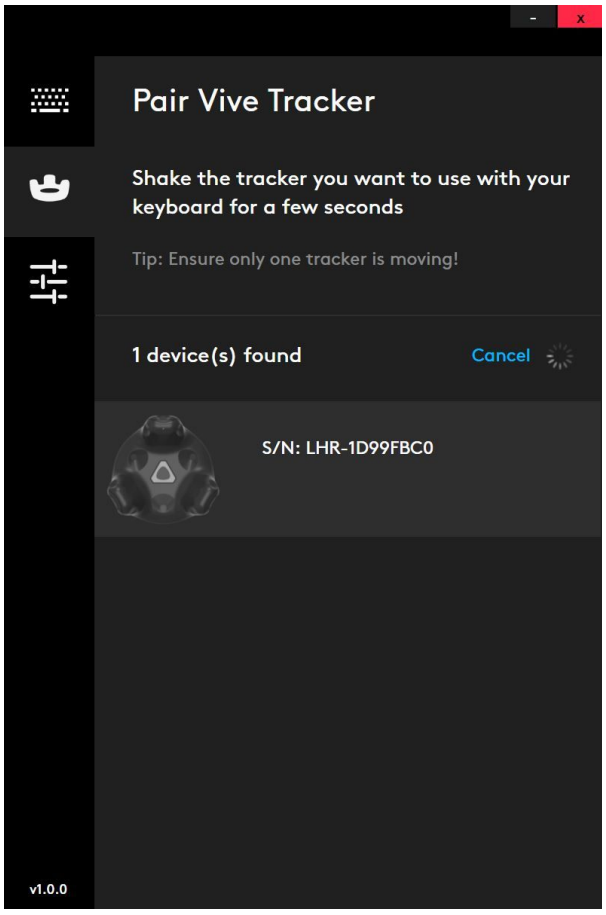
B) in Logitech BRIDGE OVERLAY UI

Launch the Logitech BRIDGE Software by right clicking in the system tray and selecting OPEN.

If a Vive system is found, the application will launch as expected and bring the pairing panel forward . This panel displays any HTC tracker paired with SteamVR should appear in the form of a list. The next steps are described in the section “Pairing an HTC tracker”.

Pairing wizard

Upon clicking on the Pairing menu icon (HTC tracker icon), when launching the application for the first time, or when trying to display the keyboard while no tracker has been paired, the user is presented with a screen to help them assign an HTC tracker to the keyboard.

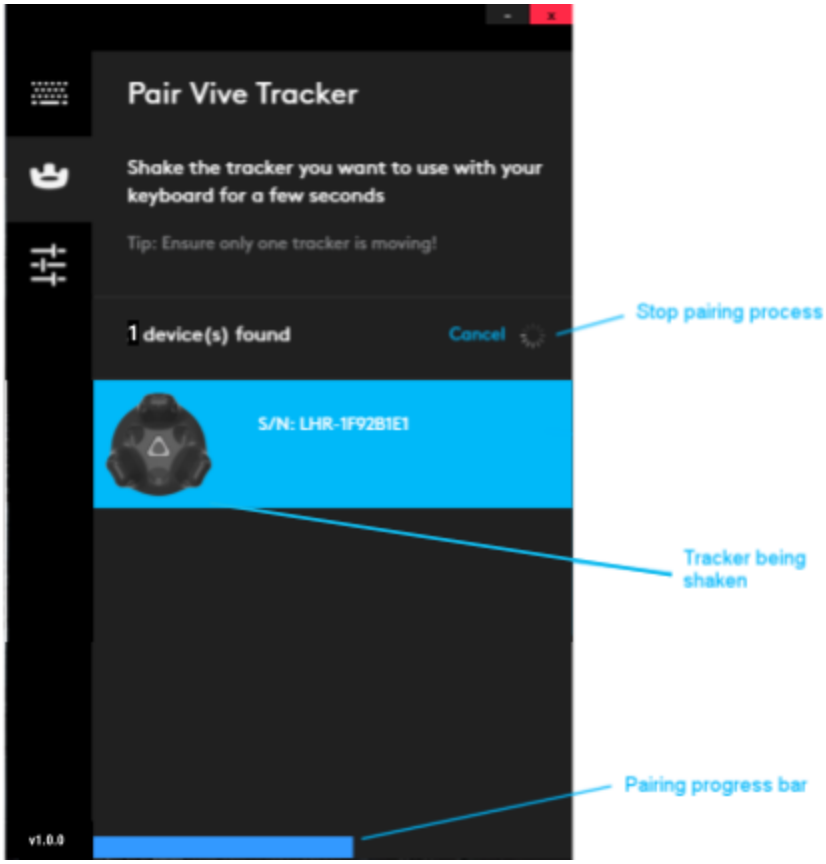


How the first time launch screen should look if a tracker is found

Any currently turned on and paired with SteamVR HTC tracker will appear in the list on this screen. To pair a new tracker with the keyboard, press the “Pair new” button. The tracker list will update and include all found trackers (not only the currently paired one).

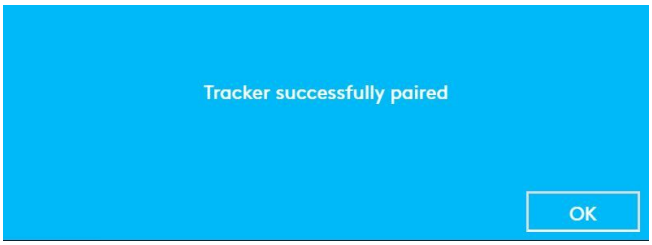
Shake it !

The user then has to shake for roughly 5 seconds the tracker they want to assign to the keyboard. As they shake this tracker, a progress bar will appear at the bottom of the application.



While the corresponding tracker is being "shaken" it will highlight and show a progress bar.

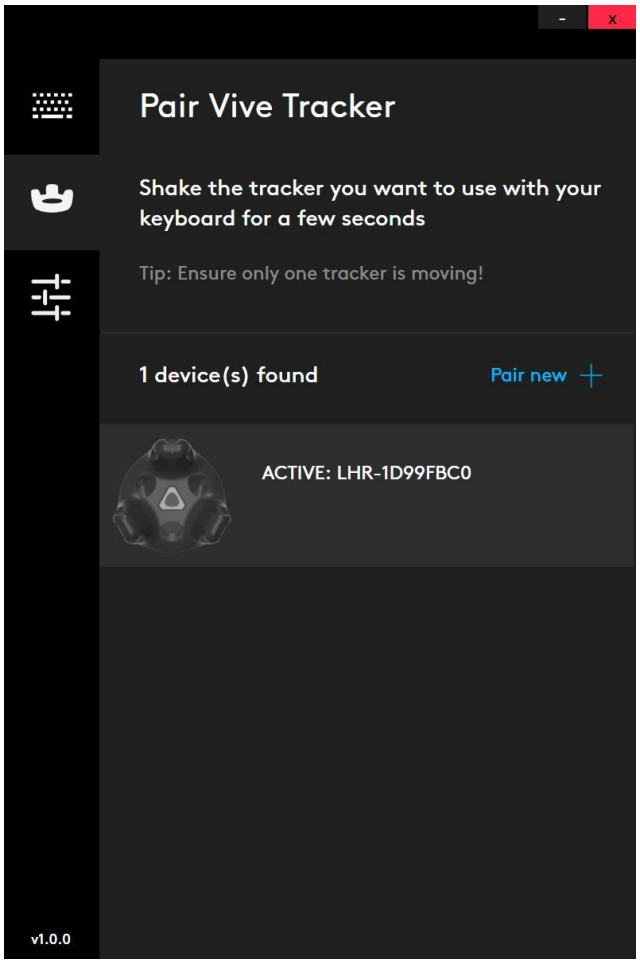
Note: If a second tracker is moved, all progress is reset. If the user stops moving a tracker, its progress will pause and decrement overtime. If the user then resumes shaking the tracker, the progress will resume and move on. When the progress bar reaches the right side of the panel, a popup will appear to confirm the tracker has been successfully paired.



Confirmation message after the "shaking".

Note: if a tracker is not correctly seen by the lighthouses, it may be look like it's drifting, hence causing an automatic pairing as it is moving. Ensure your trackers are well tracked and steady before starting the pairing process.

Going back to the PAIRING PANEL will show the tracker that is currently paired as **ACTIVE**. If a paired tracker is not detected or turned off, it will appear as INACTIVE in the following screen:



A successful pairing: one tracker should be set as ACTIVE

Reassigning a new tracker

If a tracker has already been assigned to the keyboard, it is possible to reassign another one instead. To do that open the pairing panel, and go through the process of section "Pairing an HTC tracker".

The new tracker will be automatically used to position and orient the keyboard in VR once successfully paired.

Startup Steps (IMPORTANT)

You might want to see how that looks like right away … but hold on to that for a second ! In order to avoid any side issues (and some limitations with the current version), you should first:

- **CLOSE the BridgeUI app**
- And before restarting it ...

For any subsequent startup, as a good practice, you might want to follow these steps:

1. Make sure **SteamVR is open** and running
2. Verify that the **HTC tracker is turned ON** and its led is **GREEN**
3. Your **HMD is tracking** correctly (led is GREEN or WHITE)
4. Check that the **VIVE CAMERA** (SteamVR -> Settings -> Camera -> Test Camera Rate) is ok.

5. … now you can **start the Logitech_Bridge.exe** app

Also note that:

- When CLOSING a session, first QUIT the Logitech Bridge app
- Only after that close SteamVR (not before)

Functionality:

Requirements:

The BRIDGE OVERLAY SW package follows these requirements:

- Needs Steam and SteamVR installed
- Needs an HTC Vive kit and at least one HTC Tracker
- Runs on Windows x64 only
- Uses Open VR API's
- Compatible with all apps that are developed based on Steam VR (©Valve)

Keyboard Model Overlay

It is the SW piece that supports the BRIDGE SDK and presents the user with an overlaid virtual representation of their keyboard in any VR application: it acts as an additional virtual screen that is placed in front of the user's HMD view.

The system will get the paired VIVE Tracker pose and render a 3D representation of a Logitech G810 keyboard, complete with animations when the keys are pressed.

Fig: Skin example where fonts are bigger (more readable)

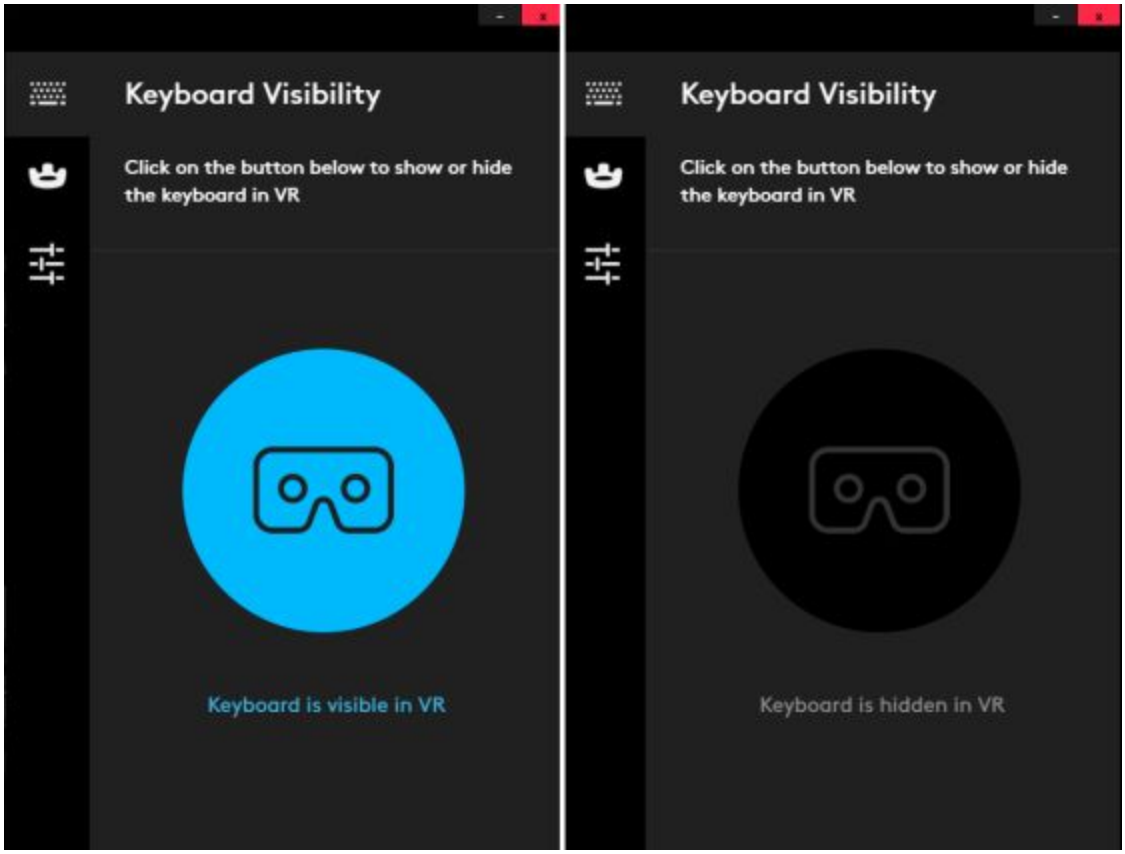
How does that work ?

The developer's application does not need to manage anything, the overlay appears automatically as soon as the associated Tracker (see pairing chapter) is turned on.

Starting from version 1.0.0 the developer's application is able to interface (see API chapter below) with the BRIDGE OVERLAY SW in order to control the keyboard's appearance, skins, layout and other elements.

Toggle the keyboard in VR

Toggling the keyboard in VR allows the user to show or hide the keyboard in the VR world. Once an HTC tracker is paired, the keyboard visibility panel allows the user to toggle the keyboard in VR. The large, round button in the center of the panel acts as a toggle button. When the button is in dark grey, it means that the keyboard is hidden. If the button is light blue, the keyboard is visible in VR.

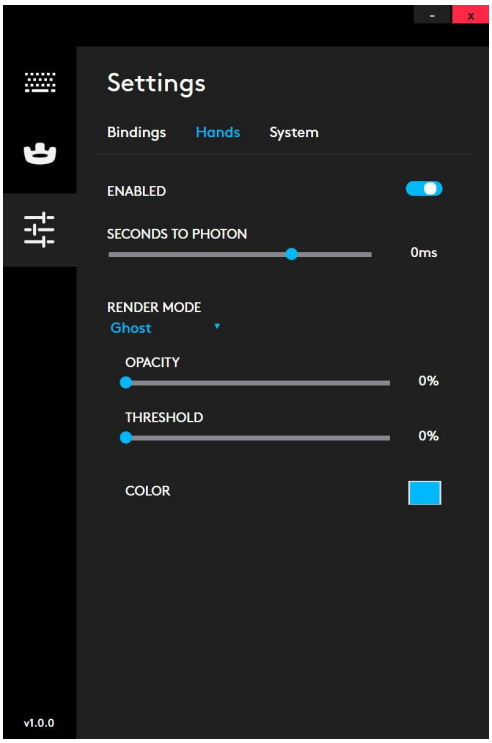


Hand Segmentation via Vive Front-facing Camera

To bring your hands into the virtual space, the feed from the Vive front-facing camera is processed, the image of your hands is extracted, and added on top of the VR keyboard model in the overlay.

HANDS MODES OVERVIEW

You can change the RENDER MODE of your hands in this settings panel :



The settings panel for hands

See-through

The see-through hands mode is a frontal projection of the camera feed on the 3D model of the keyboard. Therefore the user sees their real keyboard and hands with no post-processing. It opens a tracked window to reality. This is intended as a demonstration/reference mode only.

Note: due to the low resolution of the HTC Vive camera, the keyboard will be difficult to read as the keycaps prints will look blurred.

Ghost

This is the **default hands mode.** The ghost mode is a luminosity-based segmentation of the hands as captured by the HTC Vive camera. The segmented hands are overlaid on top of the 3D model of the keyboard. The result is a very readable keyboard with “ghost-like” hands visible.

This mode allows to customize the opacity level of the hands, their color and a threshold. The threshold will impact the segmentation process: a lower threshold will be more selective, a higher one will have more outliers.

Blue Keyboard

The blue keyboard hands mode is a solution that should help users in case the ghost mode provides poor segmentation results. This can happen mostly because the lighting conditions as well as difference of contrast between the user’s hands and the keyboard are sometimes not enough to yield a good segmentation.

If you experience bad results with the segmentation even after trying various settings in the app, please contact us and we will ship you a blue keyboard to go with this hands mode. (see FAQ/Known issues, point H)

***NB:** This mode is intended to be used only with a blue G810 keyboard not the regular black one you received as part of the development kit.*

Seconds to Photon

This allows you to change the delay (in ms) of the hands overlay representation.

Opacity

This settings allows you to decide the amount of opacity of your hands layer (you can actually see through your hands). Some people use 75% opacity as a good tradeoff and this allows them to better see the animation of the keys when pressed.

Threshold

This setting (also see Setup Hands chapter below) allows you to change the threshold (luminance in GHOST mode or red component in BLUE mode) that is used to cut out your hands from the background (ie the keyboard) in the camera image feed.

Setup (your) Hands

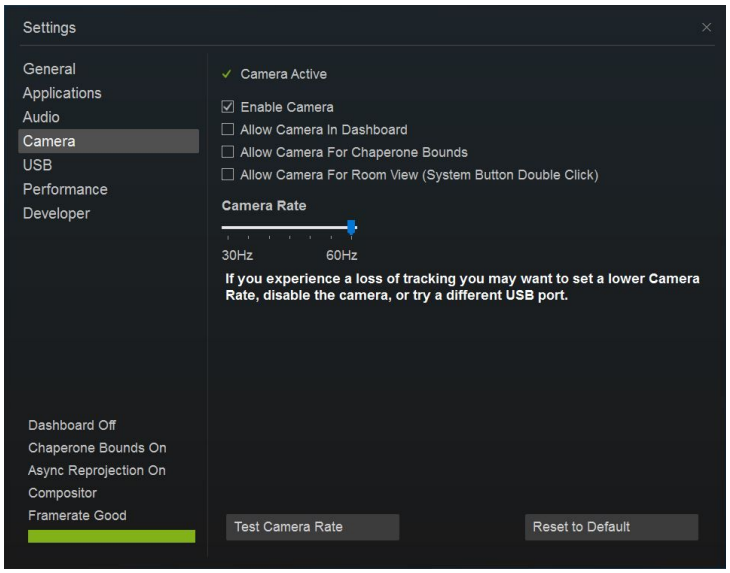
There are a few steps outlined below which will help you to get this working for your particular VR setup. This component of the software is evolving as we try different approaches. It's a work-in-progress, and known issues that we're trying to improve are listed in the section *Known Issues* below.

Step 0: Deactivate the keyboard backlighting

The hand segmentation works much better if you deactivate the lighting of the G810 keyboard. The simplest way to do that is to use the lighting switch (button with the sun icon) on the top of the keyboard.

Step 1: Activate Vive Front-facing Camera

- Before starting, ensure that the Vive camera is enabled as outlined in the Vive manual.
- SteamVR app->Settings->Camera->Enable Camera
- (You can click on Test Camera Rate to check if it works correctly)

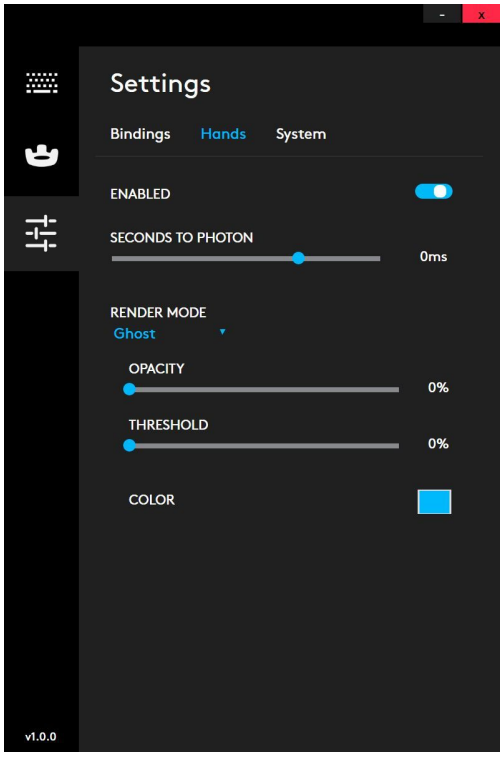


SteamVR Settings panel, Camera settings.

- For better performance, setup the camera rate to 60Hz.
- If you encounter any issues with that, you can try to unplug-replug your HMD, as well as close and restart SteamVR, this usually gets the camera working.

Step 2: Experiment to Find a Suitable *Threshold* Setting

- Make sure you have TURNED OFF the keyboard backlighting: simply use the “sun” round button just above PAUSE/BREAK key.
- In the Bridge app, if you open the SETTINGS -> HAND tab
- Make sure the hands layer is ENABLED
- Select the “GHOST” RENDER MODE
- you can adjust a slider to change the THRESHOLD value. Adjusting this value should help give you a clearer image of your hands without too much additional residues (on other regions of the keyboard).



- There is no one-size-fits-all universal setting for the THRESHOLD value. Factors such as the light level in your room, the source, direction and uniformity of lighting, the color of the keyboard and contrast with your hands may all have an impact, so you may have to spend a few minutes tweaking this for your specific setup.
- You can also personalize other settings, as the OPACITY, that will allow you to see through your hands (superpowers anyone?) or the COLOR (want to feel like hulk or a smurf?).

Step 3: Manually Align Hands and Keyboard Model

*Note: we are working on an **automated method** that will render the manual alignment operation irrelevant, but we are not there ready to share that yet. Stay tuned and use the manual mode described here below.*

- The image of your hands from the camera feed must be aligned with the keyboard model, so that what you see (your hands and the keyboard model) matches with what you touch (the real keyboard).

- The position offset of the image can be adjusted using the **RIGHT CTRL + ARROW keys.**

- If the keyboard appears slightly rotated, it can be aligned using the **RIGHT CTRL + PAGE UP/DOWN** keys (in the Bridge App Settings, see the Bindings tab for more details).

  ***NOTE**: be sure to keep the keys (ARROW or PAGE UP-DOWN) pressed long enough (while RIGHT CTRL is pressed) to allow the offset to be noticeable.*

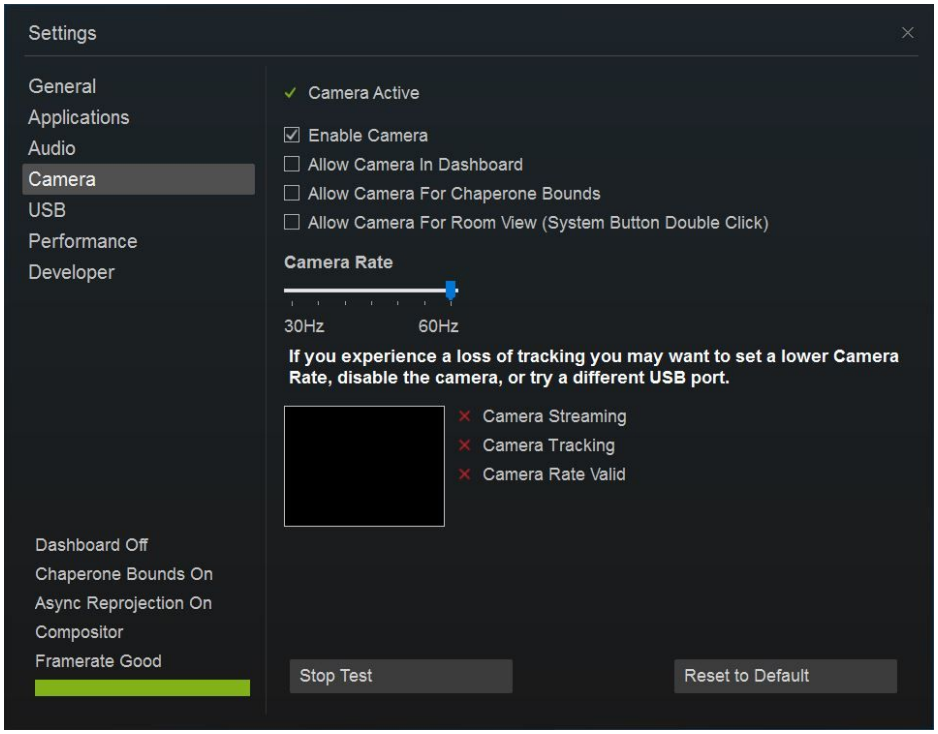
FAQ / Known Issues

There are several known issues we've encountered during development that we're actively working to improve:

A. **Keyboard position not stable when I move my head (HMD).**
When you move your head, the model of the keyboard and hands appears to move (aka "swims") in VR space, and we are working to improve the stability of this.

B. **Keyboard Model Rendering.**
The way the keyboard is currently rendered (projection on the overlay plane) will be improved in future versions. It does not support stereo-rendering as of now.

C. **No Camera Feed = No Hand Overlay.**
We've encountered situations where the Vive Camera stops working on certain systems. When this happens, restarting Steam VR or disconnecting/reconnecting the Vive HMD camera usually works for us, but check on SteamVR Troubleshooting for other solutions.



This is when the HMD camera is not available.

D. **Drifting of hand-keyboard alignment.**
Even after you align your hands with the keyboard in VR, we've noticed that hand layer position can move over time,.e.g, in VR it looks as if my finger is resting on one key, but in reality it is on another

key. We're currently trying to understand this causes of this offset. You can make changes to alignment with hotkeys. Currently, these alignment changes made via hotkeys are not automatically saved. However, if you made changes to any setting in the Bridge UI then that alignment values will also be saved.

E. **I see a strange overlay text (logi) on top of the keyboard**
This happens when the BridgeSDK was not able to acquire the camera feed. This can happen when you start the app and steamVR starts right after. To get around that, leave SteamVR open and close and start the Logitech_Bridge app again.

F. **I can see some colored dots where the keys are (same color as my hands)**
This can happen as we use luminance and color space to filter your hands (layer). Turning the keyboard backlighting OFF can help. Use the "sun" round button just above PAUSE/BREAK key.

G. **Changing Environmental / Lighting Conditions**
If the lighting in your environment changes a lot, then you'll probably have to spend time adjusting the *Threshold* setting to get acceptable representation of the hand and keyboard. Many factors impact the lighting condition, hand color, keyboard color, etc. We have a number of prototype blue G810 keyboards for experiments. These allow for more color differentiation between skin tones and the keyboard, and the matt finish is less susceptible to reflections. Let us know if you want one!

> **NB:** Our Logitech Gaming Software can be used to customize the keyboard lighting and it is highly recommended that set the lighting to pure blue in order to avoid the keyboard prints to appear as parts of your hands. You can download LGS for your Windows version on http://support.logitech.com/en_us/software/lgs.

H. **The system does not work with my hands**
If your skin tone is darker, the task for our algorithm is a bit tougher, especially on a black-colored keyboard. We can offer you an alternative version of it, with a blue painting. Send us an email (bridgesdk@logitech.com) if you are interested in getting one.

Standard black G810

Special version blue G810 (limited edition)

I. **Inconsistencies Caused by Nonuniform Lighting.**
Non-uniform lighting on the keyboard may result in a range of artifacts, e.g., seeing too little of your hands, seeing too much of the real-world keyboard.

J. **Task Manager Apps and Processes**
When Bridge application is running you can expect to see the following in Task Manager

- Bridge UI
  - 1 Application (labelled “Electron”)
  - 2 Background Processes (labelled “Electron”)
- Keyboard Model Overlay for SteamVR
  - SeraphOverlay_Runtime

When the Bridge application is not behaving as expected or crashed, the most common fix is to make sure that none of the processes listed above are still running. Kill everything and run the Bridge application again.

~~K.~~ ~~**Sometimes my settings are not saved**~~
~~This is a known issue an a fix is coming soon. For the Bridge application to prompt you to save your settings, make sure that you changed at least one setting through the application UI, like opacity for example (i.e. not only using shortcuts to change skins and adjust the alignment).~~
Note: Fixed in release 1.0.1 (Aug 16th)

Skins

As of now, the SDK integrates 3 keyboard skins. We plan on releasing new ones, depending on the need, as well as allowing developers play with that as well (see skin management chapter).

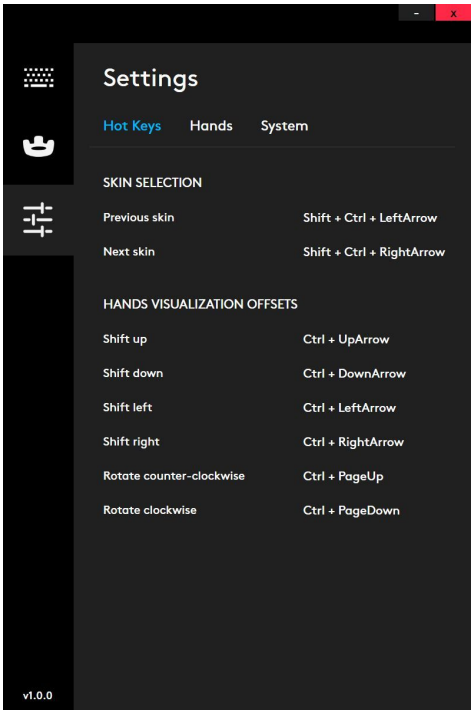
Generic skin (Logitech)

“Battered” skinned skin

Gaming Tech Skin

Keyboard shortcuts

Here are liste the keyboard shortcuts accessible in the app:



Note that in the UI we don't mention that it has to be the RIGHT control and shift keys !!!

shortcut	Keys	notes
Toggle Keyboard visibility	TBD	Not yet implemented.
Go to next skin	RIGHT SHIFT + RIGHT CTRL + RIGHT ARROW	(it cycles back at the end)
Go to previous skin	RIGHT SHIFT + RIGHT CTRL + LEFT ARROW	(it cycles back at the end)
Manual align hands layer over model	RIGHT CTRL + ARROWS	Move the hands layer alignment vs the keyboard model.
Manual align hands layer over model	RIGHT CTRL + PAGE DOWN/UP	Rotate the hands layer alignment clockwise / counterclockwise.

API (new in 1.0.X)

The Bridge SDK (*starting from version 1.0.0) allows to be accessed via API calls to a developer DLL (for C++ projects). By using the developer DLL your application will act as a client and the Bridge runtime will act as a server. The API allows a various set of configurations and settings. Here is an overview of the different functionalities which are exposed:

Generic functions:

- **Init**
- **Shutdown**

Keyboard layer functions:

- **SetKeyboardVisibility**
- **GetKeyboardStatus**

Hands layer functions:

- **SetHandsVisibility**
- **SetHandsColor**
- **SetHandsRepresentationMode**

Return Values

The interface is based on C++ enums which represent server responses (i.e. error codes) and predefined variables like the different hands representation modes that are available. Each API call will use its own enum, so for example:

if the SetHandsColor call returns

```
ESetHandsColorErrorCode::INVALID_INPUT
```

it will have a different meaning than if SetHandsRepresentationMode returns

```
ESetHandsRepresentationModeErrorCode::INVALID_INPUT .
```

You can find all the enums in the “BridgeEnums.h” header file.

Generic description of error codes:

There are some error codes which always mean the same thing and for the sake of readability, we will list them here and only detail the exceptions in the remainder of this document.

- **SUCCESS**: The operation ended as expected
- **INIT_REQUIRED**: You need to call the “Init” function first.
- **NO_CONNECTION**: We could not connect to the Bridge runtime. Try restarting the Bridge application.
- **FAILURE**: The message sent by the developer DLL was not containing the right kind of information and was rejected by the server. This should never happen so please let us know.
- **INVALID_SERVER_RESPONSE:** There was an internal error on the server side. This should never happen so please let us know. You should not assume the operation to have terminated successfully.

API detailed description

Init

Function prototype: *EInitErrorCode* Init(*void*)

Description: To be called (MANDATORY) to start a session and will initialize the communication channel with the Bridge runtime. This call is mandatory if the app wants to interact with the runtime for subsequents calls.

Returns one of the following EInitErrorCode values:

- SUCCESS
- FAILURE
- NO_CONNECTION
- INVALID_SERVER_RESPONSE

Shutdown

Function prototype: *EShutdownErrorCode* Shutdown(*void*)

Description: to be called to end an API session. This will not stop the Bridge runtime, but only unregister your application. If you want to connect to the runtime again after having called `Shutdown`, you will have to call `Init` again.

Returns one of the following EShutdownErrorCode values:

- SUCCESS
- FAILURE
- NO_CONNECTION
- INVALID_SERVER_RESPONSE

Set Keyboard Visibility

Function prototype: *ESetKeyboardVisibilityErrorCode* SetKeyboardVisibility(*bool* visible)

Description: Enables or disables the keyboard in the VR environment.

Returns one of the following ESetKeyboardVisibilityErrorCode values:

- SUCCESS
- INIT_REQUIRED
- NO_CONNECTION
- INVALID_SERVER_RESPONSE
- FAILURE

Set Hands Visibility

Function prototype: *ESetHandsVisibilityErrorCode* SetHandsVisibility(*bool* visible)

Description: Enables or disables the hands overlay in the VR environment.

Returns one of the following ESetHandsVisibilityErrorCode values:

- SUCCESS
- INIT_REQUIRED
- NO_CONNECTION
- INVALID_SERVER_RESPONSE
- FAILURE

Set Hands Representation Mode

Function prototype: *ESetHandsRepresentationModeErrorCode* SetHandsRepresentationMode(*EHandsRepresentationMode* mode)

Description: sets a predefined visibility mode for the hands from the HandsRepresentationMode enum:

- SEETHRU: full see through, no hand segmentation
- GHOST: shaded (i.e. tinted) view of segmented hands
- BLUE: use this mode in case you are using a (special) blue G810 keyboard.

Returns one of the following ESetHandsRepresentationModeErrorCode values:

- SUCCESS
- INIT_REQUIRED
- NO_CONNECTION
- INVALID_INPUT: Would happen if the hands representation mode you passed as a parameter is not supported.
- INVALID_SERVER_RESPONSE
- FAILURE

Set Hands Color

Function prototype: *ESetHandsColorErrorCode* SetHandsColor(E*HandsRepresentationMode* mode, *int* R, *int* G, *int* B)

Description: Set the color of the shading we apply to the hands for the specified visualization mode. Note that not all modes will make use of this parameter (e.g. passthru).

Returns one of the following ESetHandsColorErrorCode values:

- SUCCESS
- INIT_REQUIRED
- NO_CONNECTION
- INVALID_INPUT: Would happen if the hands representation mode you passed as a parameter is not supported or if one of the color component is not within the [0-255] range.
- INVALID_SERVER_RESPONSE
- FAILURE

Get Keyboard Status

Function prototype: *EGetKeyboardStatusErrorCode* GetKeyboardStatus (*KeyboardStatus** ks)

Description: In case of success, populate fields in the *KeyboardStatus* struct which you provided as an argument with the following fields:

- isVisible (bool)
- pairedTrackerID (string)

Returns one of the following EGetKeyboardStatusErrorCode values:

- SUCCESS
- INIT_REQUIRED
- NO_CONNECTION

- INVALID_SERVER_RESPONSE
- INVALID_INPUT: You did not provide a valid pointer to a *KeyboardStatus* struct.
- FAILURE

Get Hands Status

Function prototype: *EGetHandsStatusErrorCode* GetHandsStatus (*HandsStatus** hs)

Description: In case of success, populate fields in the *HandsStatus* struct which you provided as an argument with the following fields:

- isVisible (bool)
- handsMode (EHandsRepresentationMode)
- colorR, colorG, colorB (int)

Returns one of the following EGetHandsStatusErrorCode values:

- SUCCESS
- INIT_REQUIRED
- NO_CONNECTION
- INVALID_SERVER_RESPONSE
- INVALID_INPUT: You did not provide a valid pointer to a *HandsStatus* struct.
- FAILURE

Sample projects

The “samples” folder on Github contains a Visual Studio 2015 C++ project which loads the developer DLL and performs API calls as you hit some hotkeys. We also provide a C# wrapper made for Unity (Unity project and package file).

Our developer DLL is called “SeraphOverlay_Dev_DLL.dll” and it has the following dependencies:

- LIBEAY32.dll
- SSLEAY32.dll
- libuv.dll
- uWS.dll
- zlib1.dll

Notes:

- *You will get an error if you try to use the developer DLL without calling the Init function first, but please call the Shutdown function as well when your application exits.*

- *The developer DLL performs (local) I/O operations. It is therefore recommended that you call the DLL’s functions in a separate thread.*

C++

The BridgeSDK folder at the project’s root contains the .dll and .lib file that you need as well as the necessary header files.

After you build the project and run the application, you will be able to hit the ‘H’ key to display a help message.

Warning: At this stage of development, in order for an applications to use the developer DLL, you must make sure that:

- You compile in “release” mode. Debug is not supported yet.
- You deactivated the compiler’s SDL checks

To deactivate the SDL checks in Visual Studio, open the project’s ‘Configuration Properties’ and under ‘C/C++’ >> ‘General’, set the ‘SDL checks’ to ‘No’. Alternatively, you can manually add the compiler’s flag “/sdl-”.

Unity

We provide a sample Unity project with some scripts which are required to integrate the DLL (they are located in “Assets/BridgeSDK”). The libraries in the “Plugins” directory are also required. There is a sample script (in the Scripts directory) which will show you how to use the wrapper.

You can load and run the scene we provided to see a help message displayed in front of the main camera.

Warning: At this stage of development, the C# to C++ wrapper requires the C# compiler to run in ‘unsafe’ mode.

NB: There is no VR features (e.g. the SteamVR plugin) included in the sample project.

For more information, read the README.txt file at the Unity project’s root.

Feedback & Bug report procedure:

DISCLAIMER: Please be aware this is an ALPHA version of this SDK and it mainly meant as a POC and to spark discussion and feedback from your side. You can expect seeing bugs and robustness issues, but we are working to fix them continuously, so please make sure you have the latest release available on our GitHub repository (https://github.com/Logitech/logi_bridge_sdk).

We really hope this will be an ongoing discussion between Logitech and you, and for that to happen we will organise some sessions and meetings to get face to face (or CC based) discussions, whenever possible.

We value a lot your input on:

- possible bugs
- Shortcomings
- Issues
- imcompatilities

as well as:

- enhancements ideas
- possible new features

More importantly :

- What you think of the idea
- Is it useful
- Does it fit your app scenarii

We also strongly suggest to use our private GitHub repository for bug reports and features requests. Follow this link https://github.com/Logitech/logi_bridge_sdk/issues and post it there. This will allow easier tracking and followup.

If you have any other generic questions or comments, please feel free to contact us on bridgesdk@logitech.com.