BRIDGE SDK
User Manual

for Bridge SDK 1.1.X
Last update Nov 8th

Contact: supportsdk@logitech.com

1.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.

2.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.

3.Components:

The BRIDGE SDK requires the following elements:

- Hardware
    - A Logitech G810 Keyboard (off-the-shelf, US English layout)
    - A Logitech G BRIDGE
    - A HTC VIVE Tracker
- Software
    - BRIDGE SDK
        - The SW installer sets up the BRIDGE 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 any app.
        - A representation of user's hands overlaid on that VR keyboard (capture from the VIVE HMD Passthrough camera)
        - (In progress) SDK to allow developer to control elements of the VR keyboard overlay.

HTC TRACKER

BRIDGE

LOGITECH G 810 KEYBOARD
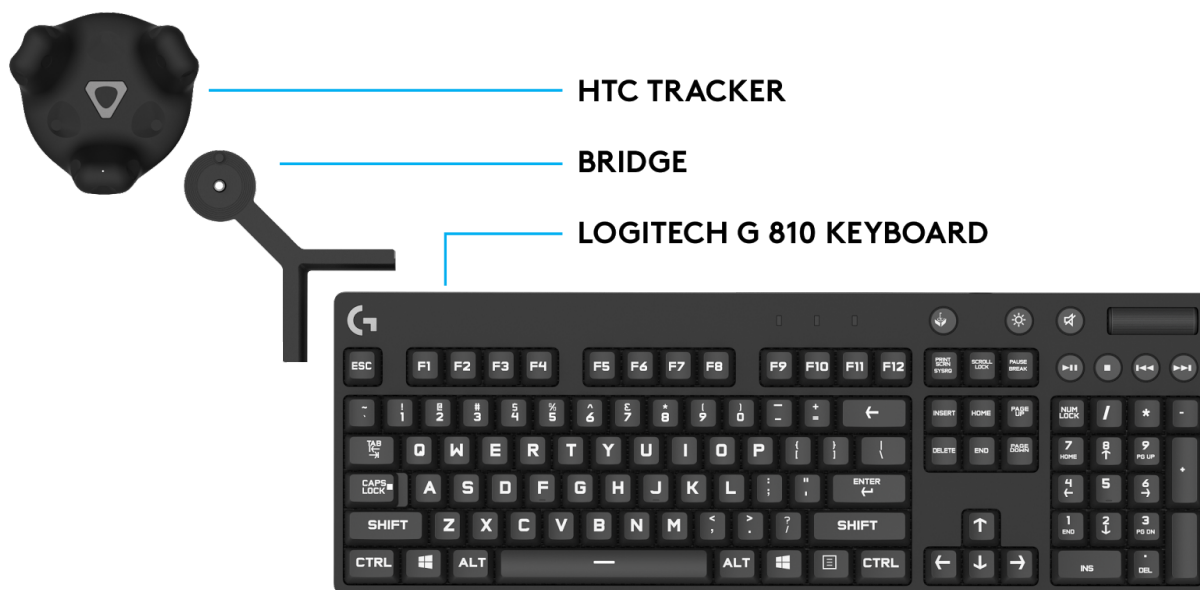
4.Setup instructions:

4.1. Attaching the VIVE Tracker to the Keyboard

1. Ensure that the BRIDGE *Locator Pin (1, labelled below)* 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 (2, labelled below)*.
3. Attach the assembled BRIDGE & VIVE Tracker to the top left corner of the Logitech G810 keyboard. First align BRIDGE to the left side of the keyboard, and then position the other leg on the top of the keyboard and push both sides to make sure is it well secured.
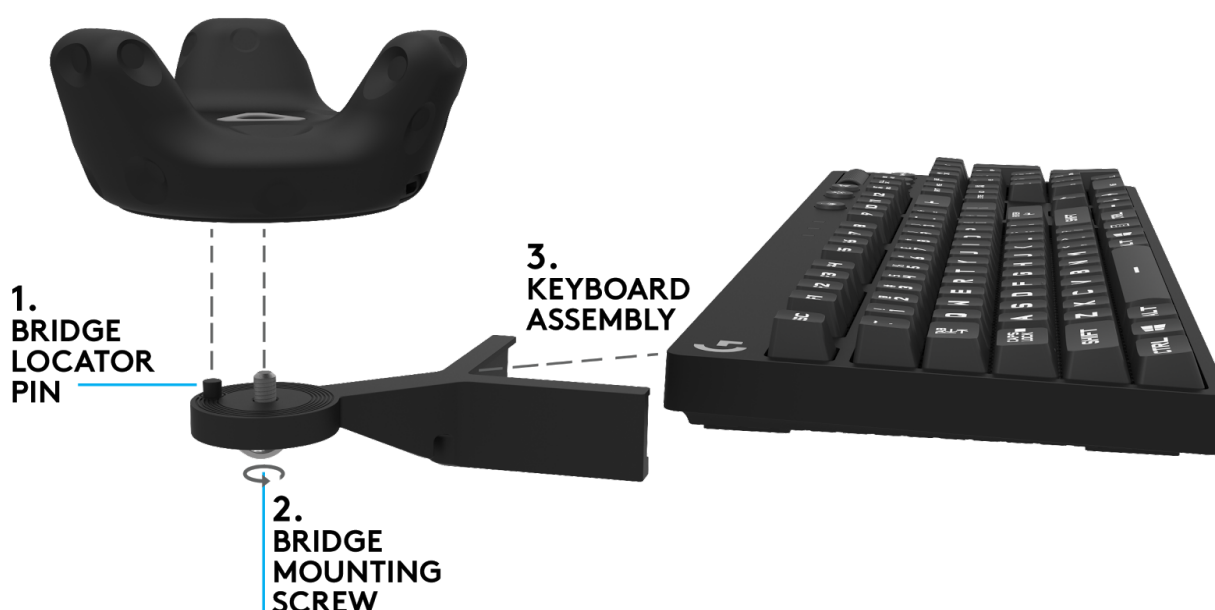
4.2. Install the BRIDGE SW package

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.
- Extract it to a folder of your choice.
- Once extracted, open the main folder and double click on **LogitechG_Bridge.exe** to launch it.

The core SW functionality will run as a service and the main UI is available in the Windows **notification area (tray)** to be accessed whenever needed:

Logitech BRIDGE tray icon
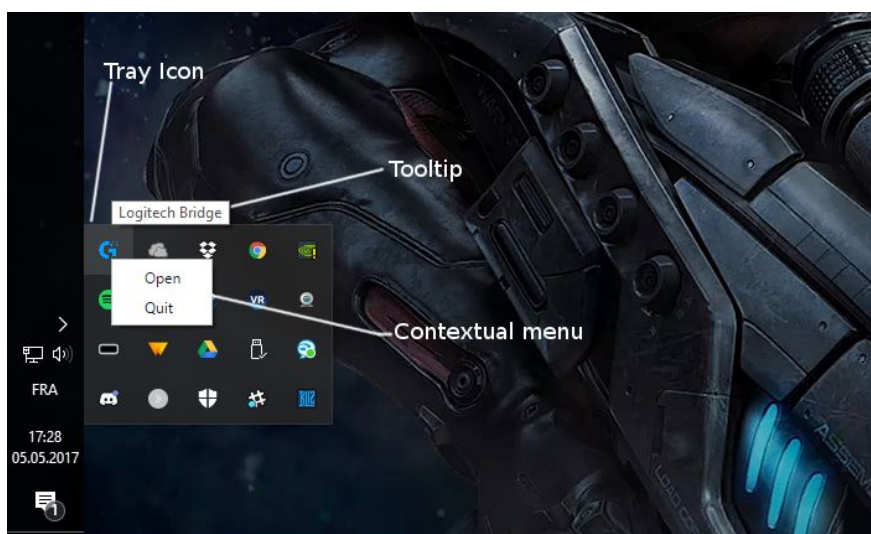
Logitech BRIDGE SDK SW that runs in system tray.

UI overview

The UI allows you to setup the first steps and offers access to various settings to customize your overlay:
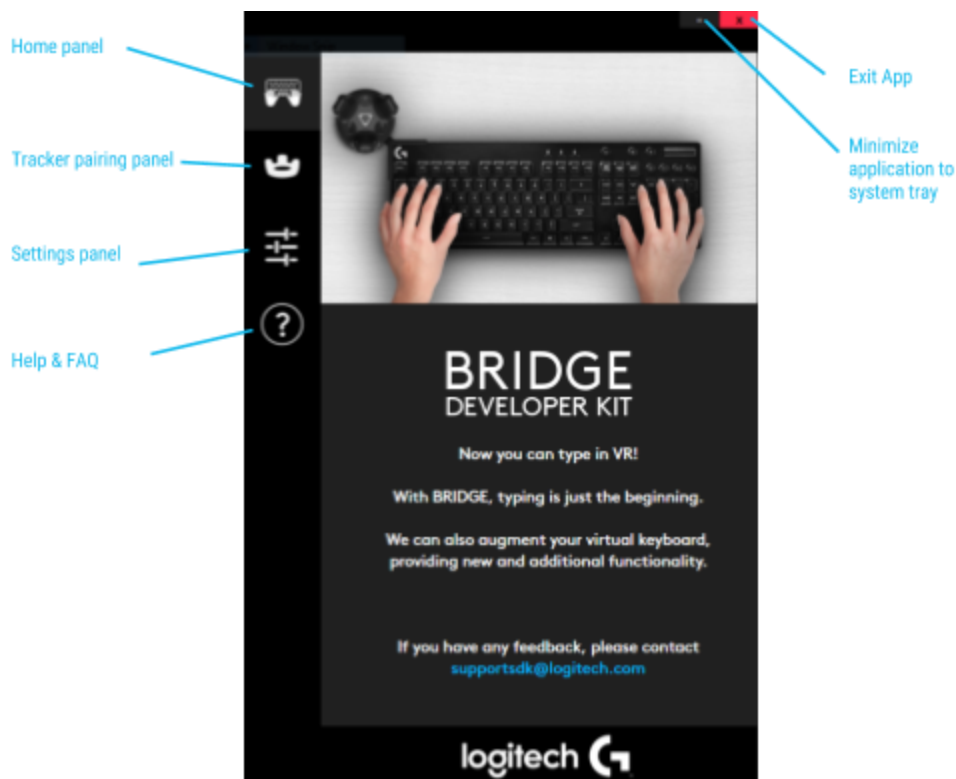
The main UI Home panel
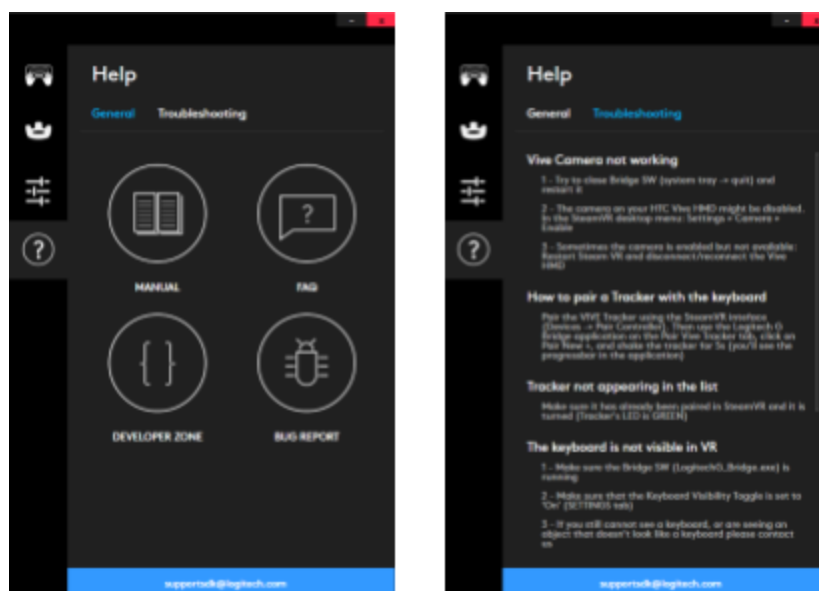
In case you need any help or have questions, head to the Help tab
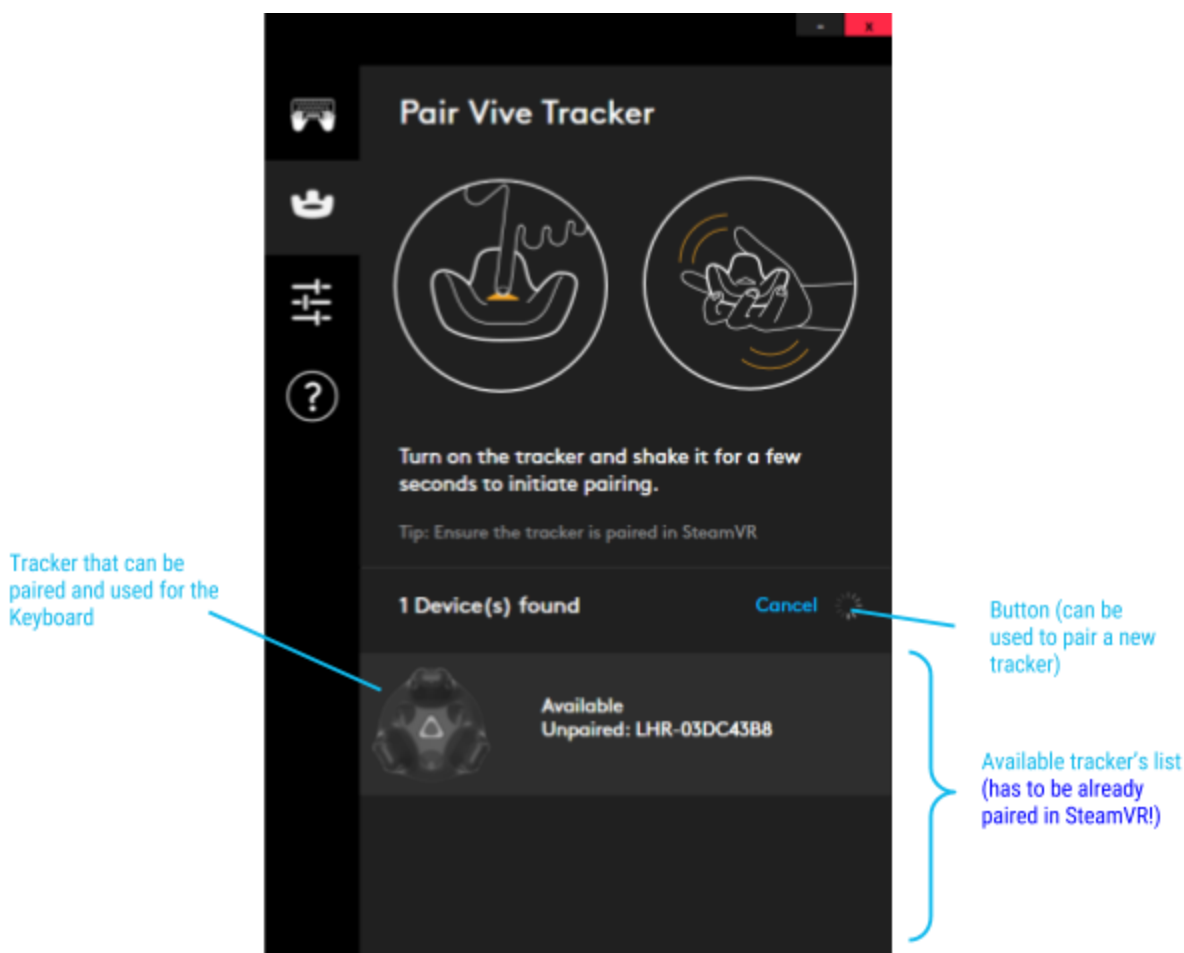
Pairing panel with one tracker assigned to the keyboard

Labels on the image:
- Tracker that can be paired and used for the Keyboard
- Button (can be used to pair a new tracker)
- Available tracker's list (has to be already paired in SteamVR!)

Minimizing the application and bringing it back
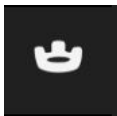
Pressing the minimize button on the Bridge desktop application will hide Bridge in the Windows notification area. In addition to notifications, the **notification area** of the Windows taskbar (sometimes referred to as the "**system tray**") is also used to display icons for system and program features that usually have no presence on the desktop; BRIDGE is one such application.

From the notification area, a single left click on the Bridge software icon will bring it back to the foreground. Using right-click on the tray icon of the Bridge Software, the user can choose to open or exit the app.

4.3. Quick start steps

You might be excited to see how Bridge looks on your system right away, but before you dive in, please follow these steps in order to make sure it runs smoothly:

● **CLOSE any previous instance of the Bridge SW (If any)**	
then	
1. Make sure **SteamVR is open** and running.	
2. Verify your **HTC tracker is paired in Steam VR** *(SteamVR -> Devices -> Pair Controller).* If not, follow steps in Section 4.4, below.	
3. Verify also that the **HTC tracker turned ON** and the led is showing a steady **GREEN**	
4. Check that the **VIVE HMD CAMERA** is enabled and functional *(SteamVR -> Settings -> Camera -> Test Camera Rate,* and follow any instructions from here to ensure it's running).*	
5. Your **HMD is tracking** correctly (led is **GREEN** and there are no errors indicated on the SteamVR window)	
6. Install **Logitech Gaming Software** (LGS) from the Logitech Website http://support.logitech.com/en_us/software/lgs	
then	

1. Run the Logitech G Bridge SW by double clicking on the .exe in the unzipped folder.	
2. Associate the correct tracker with your keyboard, using the Bridge Tracker tab.	
3. Enter the **BRIDGE VR Setup configuration** wizard when prompted, on first opening. You can run it manually from the blue button at the bottom of the SETTINGS tab at any time.	
4. Once in the VR Setup, follow the steps to ensure **your virtual keyboard and your real one are ALIGNED correctly.**	
5. Next, in the VR Setup, follow the steps to make sure you **correctly VISUALIZE your virtual hands.**	
then	
● Launch any VR app and you can start using the keyboard.	

4.4. Pairing a tracker

A) in Steam VR

- First, pair the VIVE Tracker as per HTC instructions (http://community.viveport.com/t5/Developer-SDKs-and-Downloads/Vive-Tracker-general-FAQs/ba-p/5465).

- Switch the VIVE Tracker on in PAIRING MODE, indicated by the Tracker LED blinking blue, by long pressing the center button.

- Use the SteamVR drop down menu > DEVICES > PAIR CONTROLLER to pair a new device. Follow the steps there and, when successful, the Vive Tracker light should turn green, and the Tracker icon should appear as below in SteamVR. Be aware that the UI sometimes references Controllers rather than Trackers.
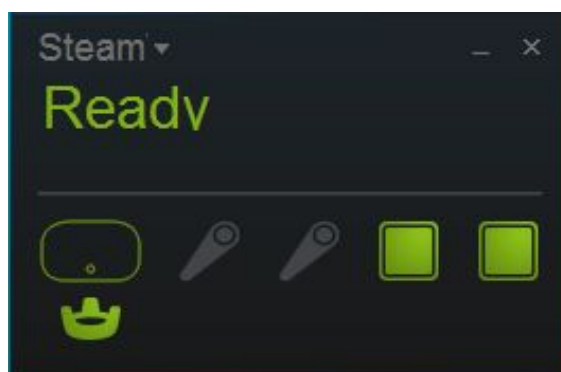
Use the SteamVR menu. Go to DEVICES > PAIR CONTROLLER
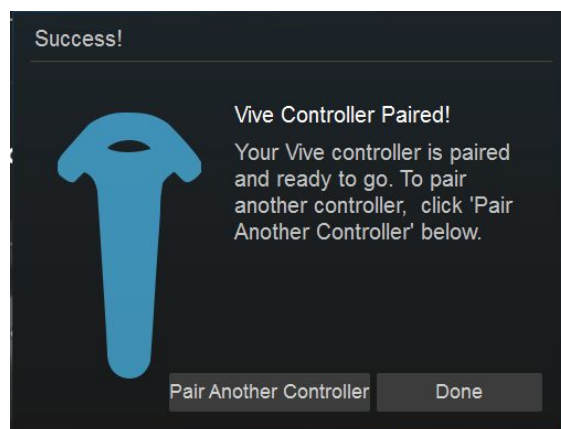
*After successful pairing, the tracker LED should turn green
and you should get this confirmation screen.*

B) in Logitech BRIDGE SW UI

Launch the Logitech BRIDGE Software by right clicking in the system tray and selecting OPEN (if it is not already open).

Pairing panel

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. This panel displays a list of any HTC trackers currently turned on and paired with SteamVR, as from the above step.
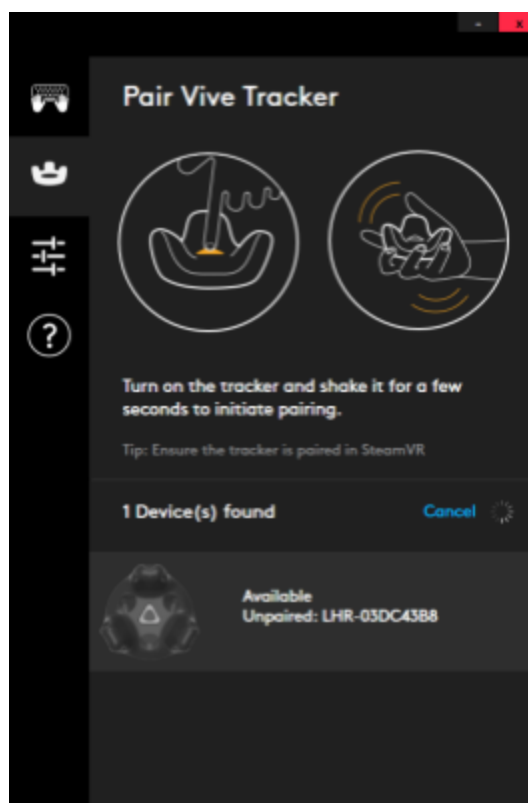
The pairing screen on opening, as long as a tracker is paired to Steam VR

To pair a new tracker with the Bridge keyboard, press the "Pair new +" button. The tracker list will update and include all trackers currently switched on and paired with SteamVR (not just the currently paired one, which will be highlighted in blue).

Shake it!

In order to identify the correct tracker associated with the keyboard, the user then has to shake their selected tracker for ~5s. As they shake the tracker, a progress bar will appear at the bottom of the application to show it is being recognised.
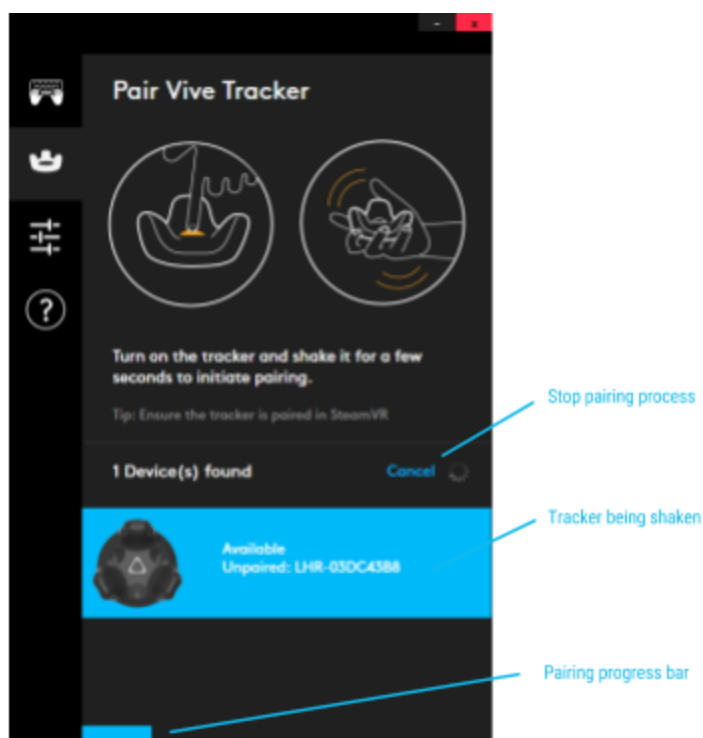
While the corresponding tracker is being "shaken"
it will highlight and show a progress bar.

Note: If a second tracker is moved during this process, 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.
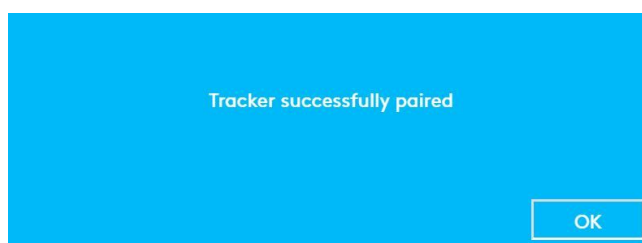
Confirmation message after the "shaking".

Note: if a tracker is not clearly seen by the HTC Vive Lighthouses, it may be look like it's drifting, and causing an automatic pairing as it appears to be moving. Ideally, 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 **PAIRED**. If a paired tracker is not detected or turned off, it will appear as **INACTIVE or OFF** in the following screen:
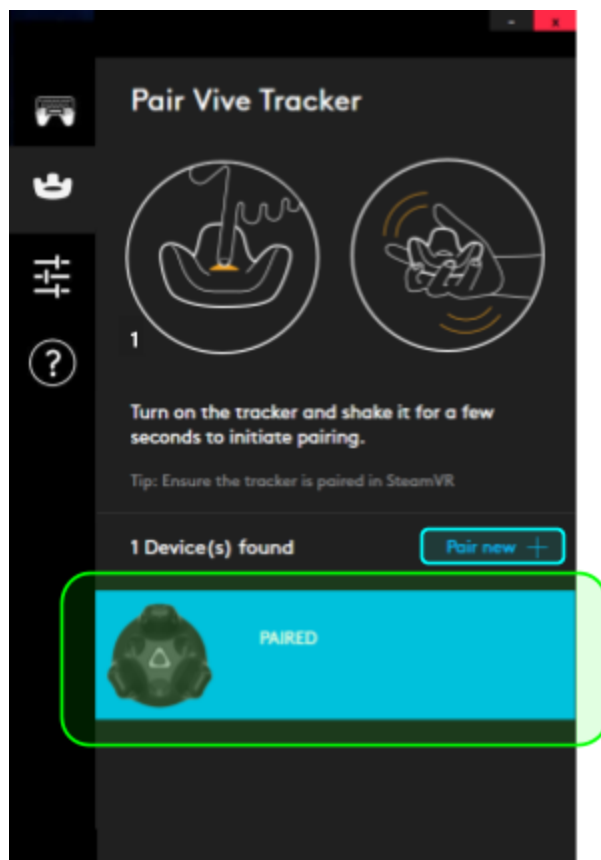
A successful pairing: one tracker should be set as PAIRED.

Assigning 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 click on "Pair New +" button (blue text).

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

5. Functionality:

5.1. 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
- Needs the keyboard layout in the Operating System to be US English (for best results)
- Uses Open VR API's
- Compatible with all apps that are developed based on Steam VR (©Valve)
- For some enhanced functionality we suggest to install the **Logitech Gaming Software**.

Recommended: install Logitech Gaming Software (LGS)

For some enhanced functionality we suggest to install the **Logitech Gaming Software** that will allow our sw to automatically manage the keyboard's backlight. This is useful in some advanced situations.

Please follow those steps:

1. Install Logitech Gaming Software (LGS) from this website:
   http://support.logitech.com/en_us/software/lgs

2. Launch it (make sure the G810 keyboard is connected with the USB of the PC)

3. And in the options panel, make sure that the "**Allow Games to control illumination**" is CHECKED

5.2. 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 SW in order to control the keyboard's appearance, skins, layout and other elements.

5.3. Toggle the visibility (keyboard & hands)

Toggling the keyboard in VR allows the user to show or hide the keyboard in the VR world. Once an HTC tracker is paired and is tracked correctly by the Lighthouses:

Use the SETTINGS TAB to enable the KEYBOARD and HANDS visualisation in VR:
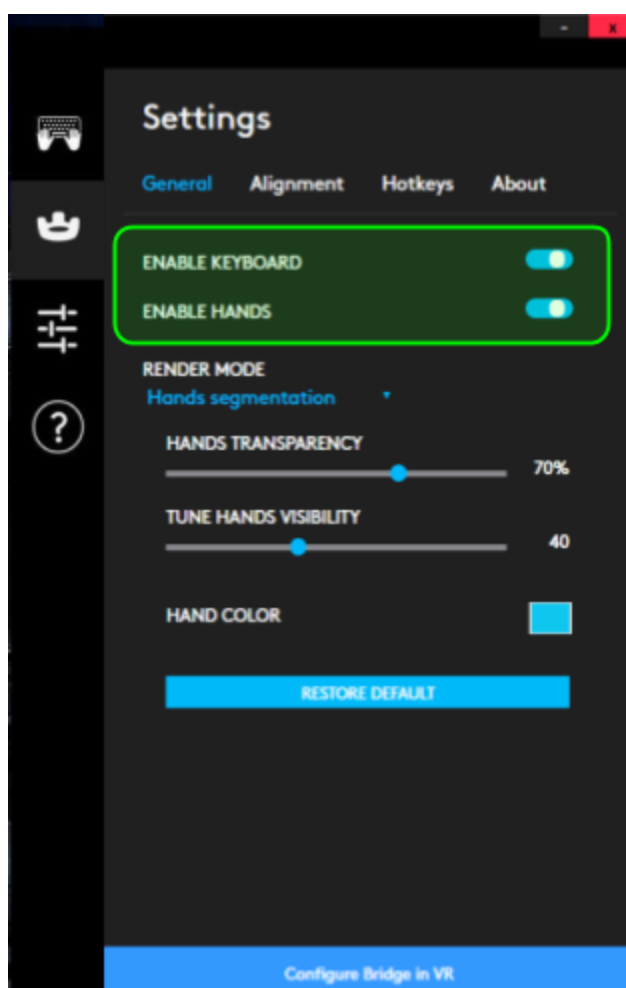
5.4. Hands support via Vive front-facing Camera

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 :
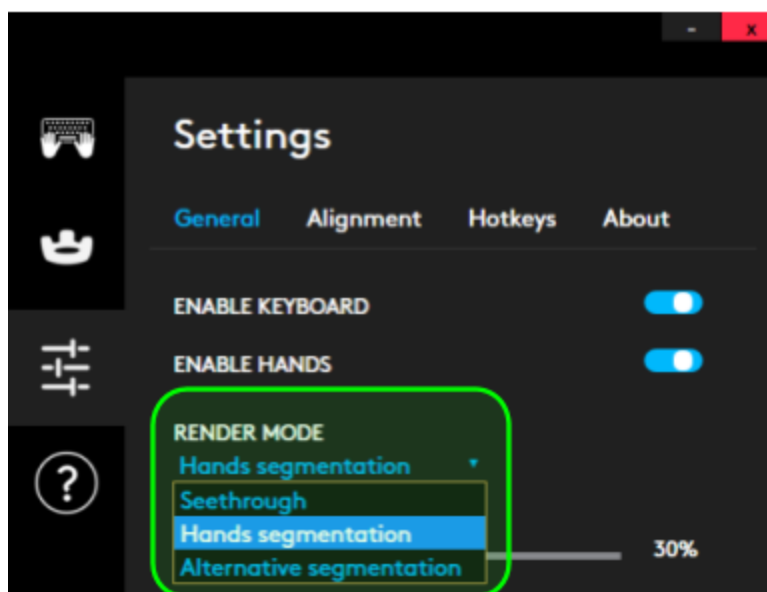
The settings panel and render mode for hands

See-through

It is a "window into reality": the see-through hands mode is a frontal projection of the camera feed limited to the section where the keyboard (ie the tracker) actually is (aka Region Of Interest). Therefore the user sees their real keyboard and hands with no post-processing. This is intended as a demonstration/reference mode only.
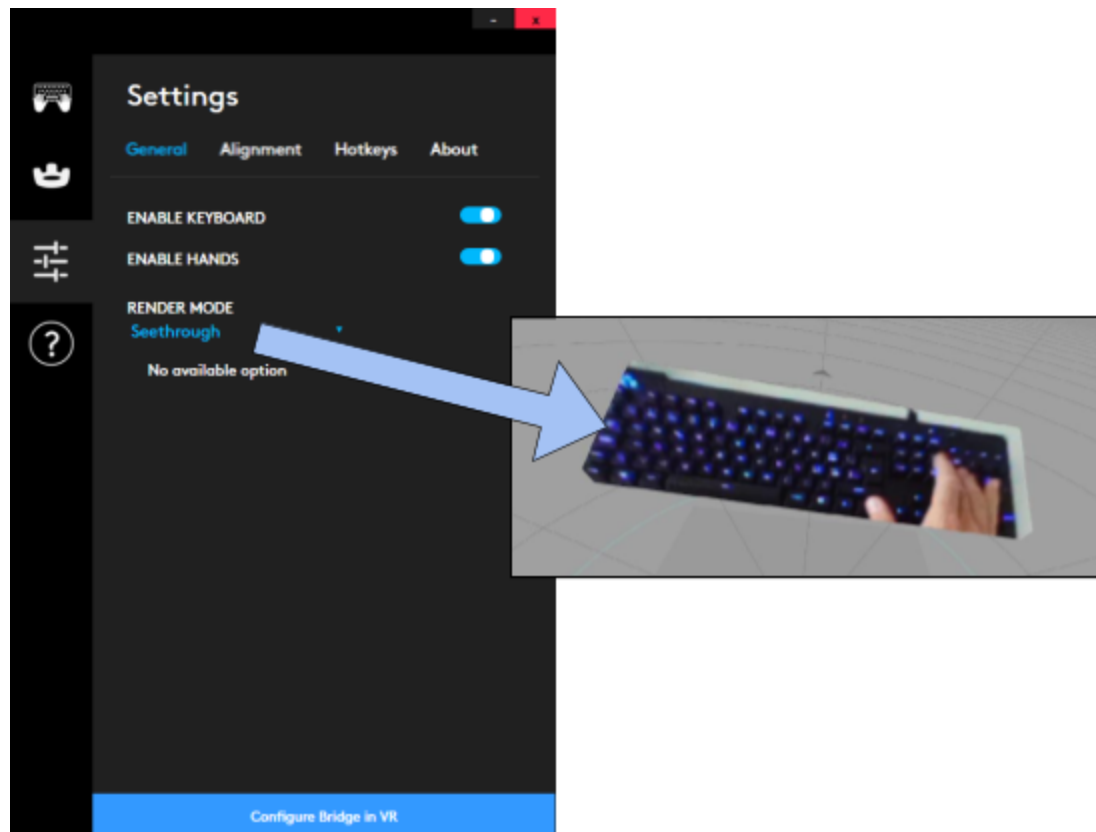
Note: due to the low resolution of the HTC Vive camera, the keyboard will be difficult to read as the keycaps prints will look blurred.

Hands Segmentation

This is the **default hands mode.** The HANDS SEGMENTATION mode is a <u>brightness-based</u> 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 "transparent" hands visible on top.
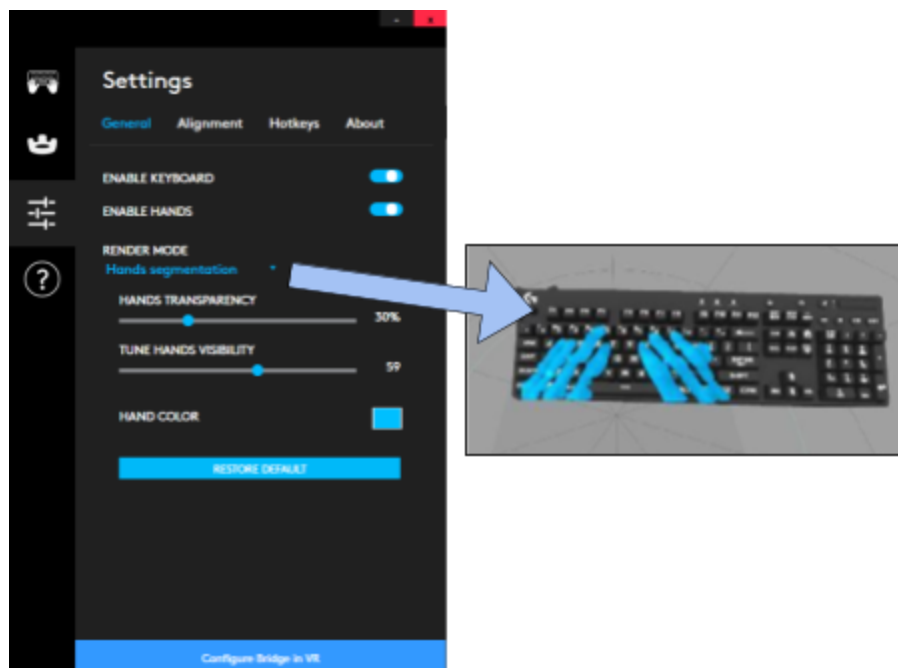
Slider: Transparency

This settings allows you to decide the amount of transparency of your hands layer (you can actually see through your hands). Some people use 15% transparency as a good tradeoff and this allows them to better see the animation of the keys when pressed.
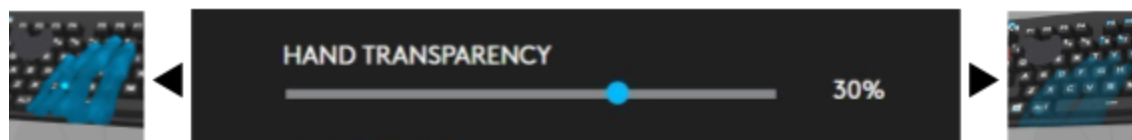
Slider: Tune visibility

This setting (also see "Setup Hands" chapter below) allows you to change how much of your hands will be captured and cut out from the background (ie the keyboard) in the camera image feed.
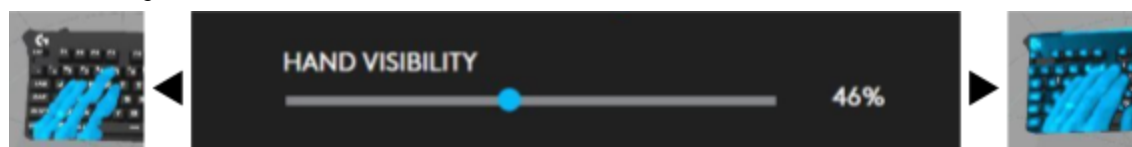
Alternative Segmentation

This is an EXPERIMENTAL mode because the lighting conditions as well as the difference of contrast between the user's hands and the keyboard are sometimes not enough to yield a good brightness segmentation (see previous mode). This also is an attempt to have better hand segmentation in poor lighting conditions, where there is not a strong contrast between the keyboard and skin tones.
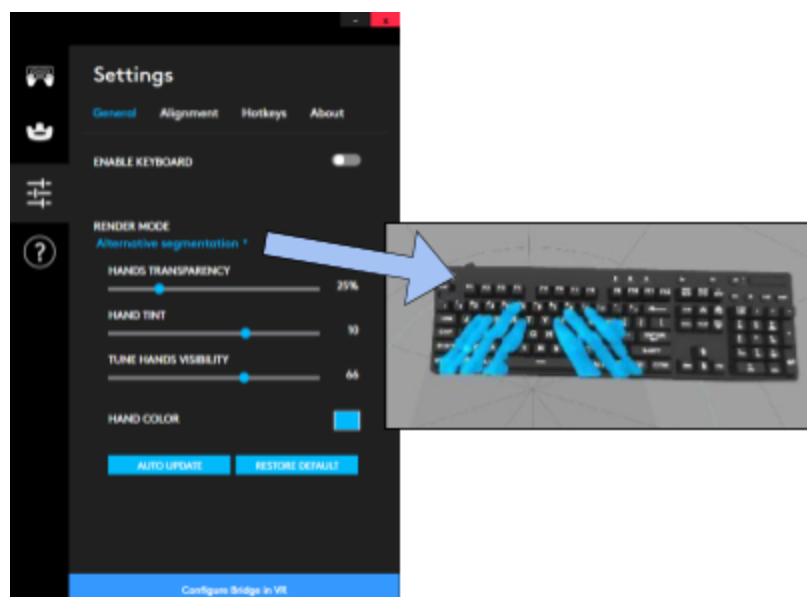
Slider: Hand Tint

This setting will allow you to select the tint of your hand on the HUE wheel. On the left it will be more of a cold-blueish tint, on the right it will be more warm-yellow tint.

Slider: Tune Hand Visibility

This setting will allow you to select how much of side tones you will allow to select around the tint selected with the above slider (HAND TINT). Less permissive on the left or more permissive on the right.
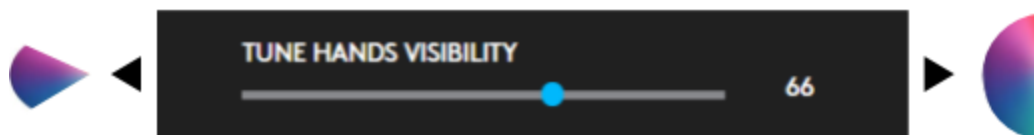
Setup (your) Hands

Bridge Guided setup in VR

The **easiest way is to launch the VR guided setup** that is launched automatically at the first start of the app, or by clicking on the button at the bottom of the UI in the SETTINGS tab.
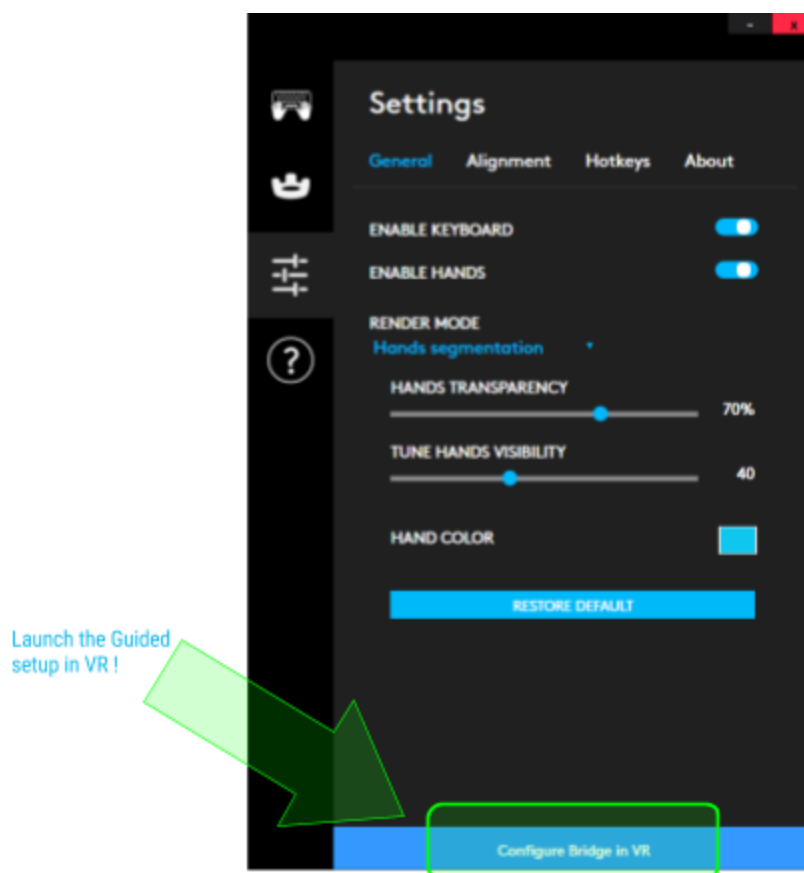
When you click this button a specific application will be launched and you will need to put your HMD on and follow the instructions from there.

… Enjoy the "ride" in VR!

Manual setup

If you prefer you can also do that manually. There are a few steps outlined below which will help you to get this working for your particular VR setup.

Step 1: Use the correct Backlighting for the keyboard

- To have better performance of the Hands Segmentation, it is recommended to **TURN OFF** the keyboard BACKLIGHT.
- The easiest way to achieve that is to use the TOGGLE ILLUMINATION button on the keyboard itself (button with the "sun" icon on the top of the keyboard).

Turn OFF the backlight !

Step 2: Activate Vive Front-facing Camera

- Before starting, ensure that the Vive's HMD "Chaperone" camera is enabled.
- SteamVR app->Settings->Camera->Enable Camera
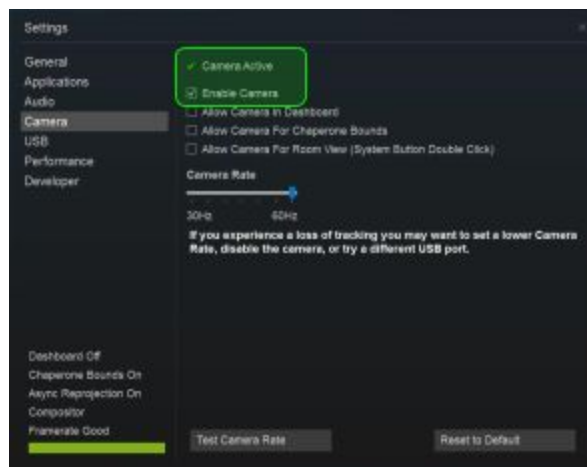
SteamVR Settings panel, Camera settings.

- You can click on Test Camera Rate to check if it works correctly
- For better performance, setup the camera rate to **60Hz.**
  - to **avoid possible USB issues** use 30-45Hz instead.
- 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 3: Experiment to find a Suitable *TUNE HAND VISIBILITY* slider setting

- Make sure you have **decent and uniform lighting conditions** on your desk (ie avoid reflections and direct hard surn light on your keyboard).
- In the Bridge SW, if you open the SETTINGS -> GENERAL tab
- Make sure the ENABLE HANDS is turned on.
- Make sure your left hand is on the keyboard (as if you would type).
- Select the "HANDS SEGMENTATION" RENDER MODE and use the TUNE HANDS VISIBILITY slider
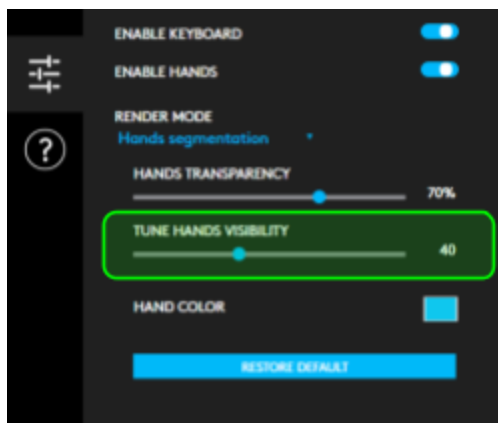
- you can adjust a slider to change the **HANDS VISIBILITY** value. Adjusting this value to higher values should help give you a clearer image of your hands but your should stop when you see too many "other" portions of the keyboard being shown.
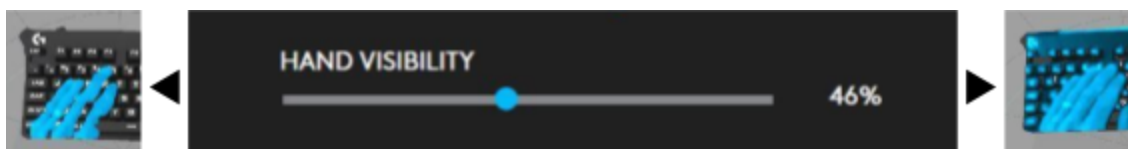
Note: There is no one-size-fits-all universal setting for the selected value: Factors such as the light level in your room, the source, direction and uniformity of lighting, the color of the keyboard and contrast with your hands may all have an impact, so you may have to spend a few minutes tweaking this for your specific setup.

- You can also personalize other settings, as the **HANDS TRANSPARENCY,** that will allow you to see through your hands (superpowers anyone?)

- Or play with your favourite **HAND COLOR** (want to feel like hulk or a smurf?).

Step 4: Alignment

This is a **very important** step: the image of your hands from the camera feed <u>must</u> <u>be</u> <u>aligned</u> <u>with</u> <u>the</u> <u>keyboard</u> model, so that what you see (your hands and the keyboard model) matches with what you touch (the real keyboard).
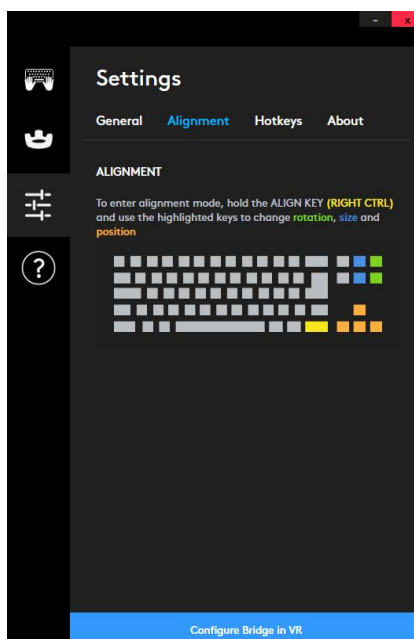
*The SETTINGS / ALIGN tab: recaps the
key bindings used for alignment*

*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.*

A) <u>Enter the **ALIGN MODE**</u>

   ○   Enter the ALIGN MODE by **holding the RIGHT CONTROL** (in VR it is labelled ALIGN) key:
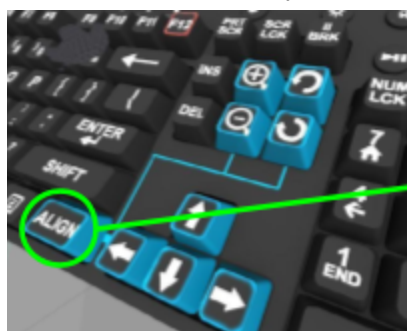
*Enter the ALIGN mode by pressing (and
holding) the ALIGN key (RIGHT CTRL)*

Alignment mode representation in VR:
-   ***Real Keyboard*** *(from camera)*
-   ***Virtual keyboard*** *(blue) outline*

B) First adjust the **ANGLE**

- ○ try to align the edges of the real keyboard with the blue outline of the "virtual keyboard"
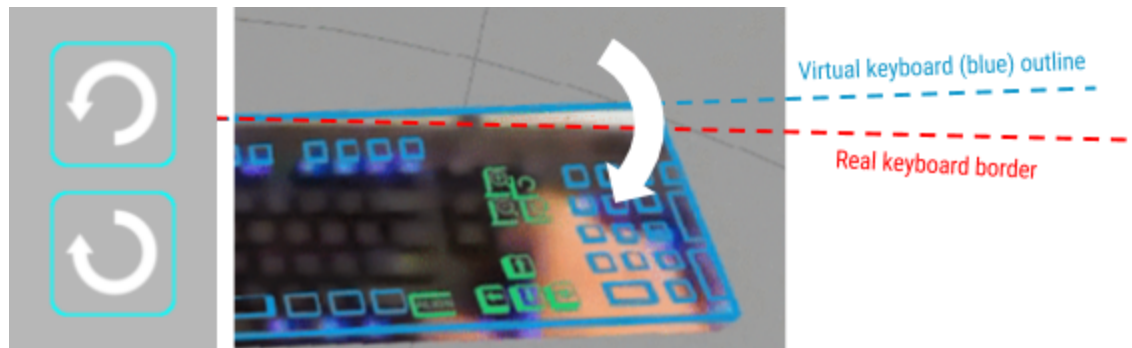
- ○ use **PAGE UP/DOWN** keys to rotate the visualisation until they are aligned.

B) adjust the **SCALE** of the keyboard

Note: it is usually not needed to adjust that. Check before making any changes !, but If you feel you need to adjust this parameter:

- ○ While keeping **ALIGN** pressed …

- ○ … use the **HOME/END** arrow keys to make sure that the size of the keyboard fits the virtual outline.

C) fine tune the finger's **POSITION** (XY)

○ While keeping **ALIGN** pressed …

○ … use the **UP/DOWN/LEFT/RIGHT** arrow keys to realign the key of the keyboard (3D model) with the image seen by the camera (blue overlay).

IMPORTANT: It is more important to align the center of the keyboard (Space bar, C, V, B, D, F, G, …) keys than the peripheral ones, because that is where your hands mostly are.

NOTE: be sure to keep the keys pressed long enough (while RIGHT CTRL is pressed) to allow the offset to be noticeable since it granularity is very small.

5.5. Skins

As of now, the SDK integrates some pre-defined keyboard skins. We plan on releasing new ones, depending on the need, as well as allowing developers play with that as well (stay tuned for that!).

How to Change Skins

Manually

You can switch between skins (ie circle through them sequentially) using the F1 key on you keyboard at any time.

Press F1 to change (toggle) skins

Programmatically

You can use the supplied API to load a specific skin at a certain time, for example when you app starts, or when a specific mode is entered in you app. Use those functions:

- *GetSupportedKeyboards(SupportedKeyboards* sk)*
  In order to retrieve how many keyboards (only a single one is supported right now) and all the skins associated with it.

- *SetSkin(string skinName)*
  To set the desired skin by its string/name.

In case you want to add one, have a look at the JSON file `user.settings` in the LogitechGBridge/Ressources folder.

```
"skins":
[
        {"name": "Logitech"},
        {"name": "Battered"},
        {"name": "ScifiTech" },
        {"name": "VirtualDesktop"},
        {"name": "Align"},
        {"name": "SkinSwitch"},
        {"name": "CrashTest"}
]
```

Skins

Generic G810 skin (Logitech) / Identifier: "Logitech"
This one is representing the original G810 Keyboard, with slightly different and larger font in order to optimize readability.

Rusty Blue Battered / Identifier: "Battered"
This one is showing how a fully colored and textured can look like in VR.

SciFi "gaming" Tech / Identifier: "SciFiTech"
This one has a particular highlight on WASD keys as well as letter keys that are greyer than the rest. It could be used as a "gaming" skin.

Virtual Desktop / Identifier: "VirtualDesktop"

> This one has been created to match virtual desktop SW and suggest some of the available shorcuts that the SW allows (mostly F keys).

Bridge Setup - Align keys / Identifier: "Align"

> This skin has been crafted to highlight the alignment method that makes use of some particular key bindings to help you align the virtual layer with the real keyboard.

Bridge Setup - Skin Switch / Identifier: "SkinSwitch"

> This skin has been crafted to show you which key (F1) can be used to change skins.

Crash Test / Identifier: "CrashTest"

This is another example to show what a lighter-colored skin (with black letters) can be. See how differently colored key groups are used to differentiate different functionalities on the keyboard.

5.6. Keyboard shortcuts

Here are liste the keyboard shortcuts accessible in the app:
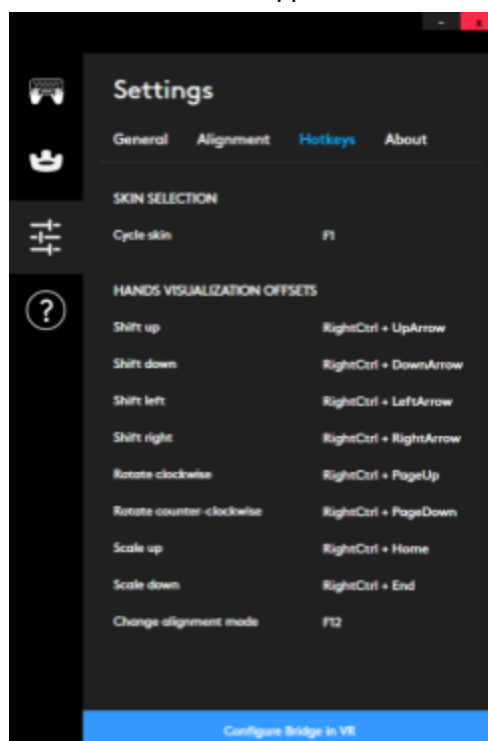
shortcut	Keys	notes
Toggle Keyboard visibility	TBD	Not yet implemented
Go to next skin	F1	(it cycles back at the end)
Enable Align mode	hold RIGHT-CTRL *(labelled "**ALIGN**" in VR)*	Hold right control at any time to enter align mode *(visualisation of keyboard changes)*
<u>**rotate**</u> **hands layer over model**	(hold ***ALIGN***) + PAGE DOWN/UP	Rotate the hands layer clockwise / counterclockwise.
<u>**scale**</u> **hands layer over model**	(hold ***ALIGN***) + HOME/END	scale the hands layer bigger/smaller.
<u>**align**</u> **hands over model**	(hold ***ALIGN***) + ARROW KEYS	Move the hands layer alignment vs the keyboard model.
change the align mode	F12	toggles the alignment mode between seethrough and (experimental)"tron".

6. API

The Bridge SDK* (*starting from version 1.0.X) 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 in **API v1.1.0:**

Generic functions:
- **Init**
- **Shutdown**

Keyboard layer functions:
- **SetKeyboardVisibility**
- **GetKeyboardStatus**
- **GetSupportedKeyboards**
- **SetSkin**

Hands layer functions:
- **SetHandsVisibility**
- **SetHandsColor**
- **SetHandsRepresentationMode**
- **SetHandsSegmentationThreshold**
- **SetHandsOpacity**
- **GetHandsStatus**

6.1. Foreword on API usage

Background architecture

The SDK is still under development and in the coming releases we will expose a notification mechanism allowing your application to know when some setting has been changed as well as letting the other clients know about changes happening from your app. For now polling the keyboard and hands status has to be done manually by your application. In other terms we encourage you to update (via GetHandsStatus and GetKeyboardStatus) your internal state variables before issuing any API call.

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.

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.

6.2. API functions

Init

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

Description: To be called (<u>MANDATORY</u>) 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

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)

```
struct KeyboardStatus
{
    bool isVisible = false;
    string pairedTrackerID = "";}
;
```

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 (applies to C++ interface).
- FAILURE

Get Supported Keyboards

Function prototype: *EGetSupportedKeyboardsErrorCode* GetSupportedKeyboards(*SupportedKeyboards* sk*)

Description: Returns a struct that contains list of currently supported keyboards and a list of their respectively available skins' names.

```
struct Skin
{
  string name;
};

struct Keyboard
{
  string name;
  vector<Skin> skins;
};

struct SupportedKeyboards
{
  vector<Keyboard> keyboards;
};
```

Returns one of the following *EGetSupportedKeyboardsErrorCode* values:

- SUCCESS
- INIT_REQUIRED
- NO_CONNECTION
- INVALID_SERVER_RESPONSE
- FAILURE

Set Skin

Function prototype: *ESetSkinErrorCode* SetSkin(*string skinName*)

Description: Tries to set the keyboard skin given the name you provided.
The list of available skins with their identifier (i.e. unique skin name) is available in section 5.5.

Returns one of the following ESetSkinErrorCode values:

- SUCCESS
- INIT_REQUIRED
- NO_CONNECTION
- INVALID_SERVER_RESPONSE
- INVALID_INPUT: The name you provided did not correspond to any available skins.
- 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 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

Set Hands Representation Mode

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

Description: sets a predefined visibility mode for the hands from the EHandsRepresentationMode enum:
- SEETHRU: full see through, no hand segmentation
- HANDS_SEGMENTATION: shaded (i.e. tinted) view of segmented hands
- ALTERNATIVE_SEGMENTATION: Perform a segmentation of the hands using another method (experimental).

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 Segmentation Threshold

Function prototype: *ESetHandsSegmentationThresholdErrorCode*
SetHandsSegmentationThreshold(*EHandsRepresentationMode* mode, *float* segmentationThreshold)

Description: Set the value of the threshold we use segmenting the hands for the specified visualization mode. It refers to the "hands visibility" slider in our GUI, but the scale is inverted. The larger the segmentation threshold, the less visible the hands.
Note that not all modes will make use of this parameter (e.g. passthru).

Parameters: segmentationThreshold: Goes from 0.0 (lets everything pass through) to 1.0 (most aggressive segmentation)

Returns one of the following ESetHandsSegmentationThresholdErrorCode 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 the threshold is not within the [0-1] range.
- INVALID_SERVER_RESPONSE
- FAILURE

Set Hands Opacity

Function prototype: *ESetHandsOpacityErrorCode* SetHandsOpacity(*EHandsRepresentationMode* mode, *float* opacityLevel)

Description: Set the opacity level of the hands for the specified visualization mode. It refers to the "hands transparency" slider in our GUI, but the scale is inverted. The larger the opacity level, the more opaque the hands.

Parameters:
opacityLevel: Goes from 0.0 (0% opaque) to 1.0 (100% opaque)

Note that not all modes will make use of this parameter (e.g. passthru).

Returns one of the following ESetHandsOpacityErrorCode 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 the opacity level is not within the [0-1] range.
- INVALID_SERVER_RESPONSE
- 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 within the [0-255] range)
- opacityLevel (float within the [0-1] range)
- segmentationThreshold (float within the [0-1] range)
- handTintOffset (int within the [-90, 90] range, this field should only be read when the handsMode variable is set to ALTERNATIVE_SEGMENTATION).

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 (applies to C++ interface).
- FAILURE

7. Sample projects

The "samples" folder on Github contains a Visual Studio 2015 C++ project which loads the developer DLL and performs on-demand example API calls. We also provide a C# wrapper made for Unity (Unity project and package file).

Both these samples are client applications that connect to the Bridge software. Therefore you must have Bridge running prior to executing the samples

Our developer DLL is called "BridgeOverlay_SDK.dll" and 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, and call the Shutdown function as well when your application exits. Both code samples illustrate automated init and shutdown.*

- *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.

```
// -------------------------------------------------------------------------------
// Logitech Bridge SDK: Command line sample application
// Copyrights Logitech - 2017

// This is a sample project which loads the Bridge Developer DLL
// Press 'h' to display the shortcuts list
// Press 'x' to stop the application
// -------------------------------------------------------------------------------

>> Connection to Bridge Runtime worked, type in a command:

>> Available shortcuts:

        'x' : Stop the application.
        'q' : Display the keyboard.
        'w' : Hide the keyboard.
        'e' : Set the hands representation mode to 'HANDS_SEGMENTATION'.
        'r' : Set the hands representation mode to 'SEETHRU'.
        't' : Set the hands color in 'HANDS_SEGMENTATION' mode to RGB values (0, 140, 245).
        'u' : Set the hands segmentation threshold to 30%.
        'i' : Set the hands segmentation threshold to 10%.
        'o' : Set the hands tint offset to +10.
        'p' : Set the hands tint offset to -30.
        'a' : Update the KeyboardStatus instance you passed as a parameter.
        's' : Update the HandsStatus instance you passed as a parameter.
        'f' : Set opacity level to 70% ('HANDS_SEGMENTATION' mode).
        'g' : Get supported keyboards.  'j' : Send 'AUTO UPDATE' IPMessage.
        'k' : Send 'LOAD DEFAULT' IPMessage.
        'c' : Set skin.
```

Command line C++ sample preview

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:
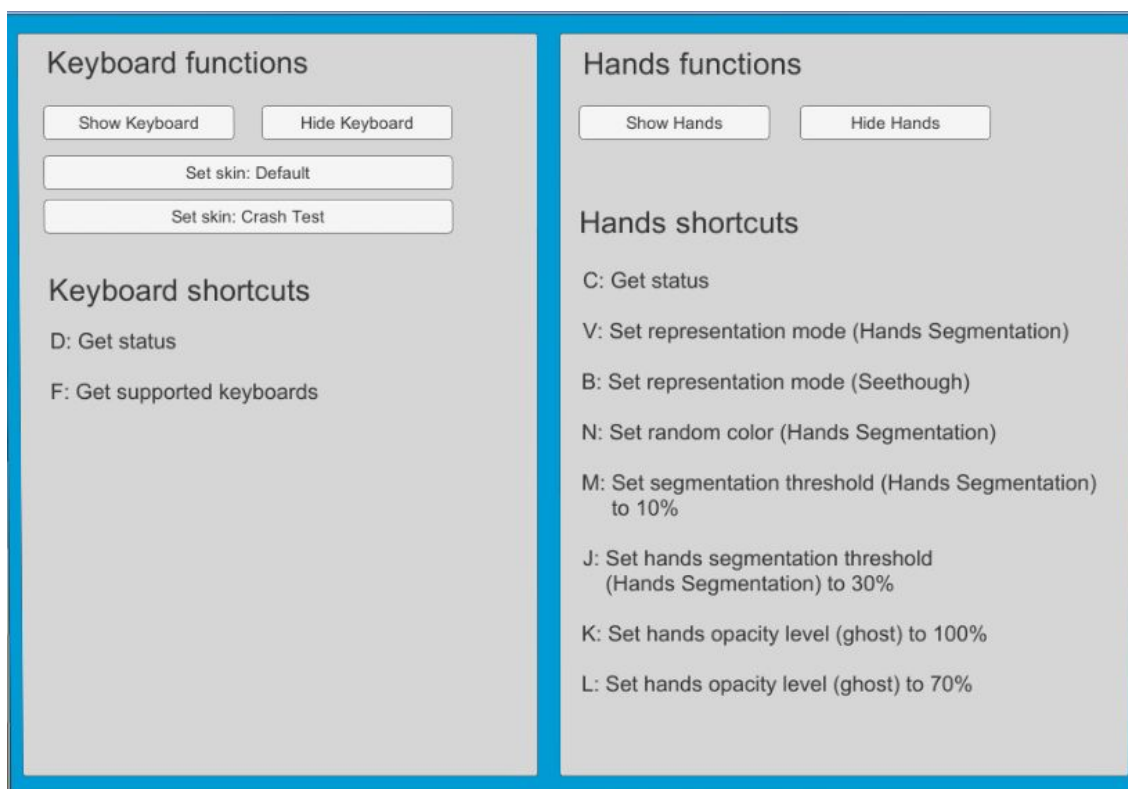
Screen in the sample Unity scene

Note: the shortcuts will output their respective results in the console.

The scene runs the Bridge SDK in a singleton object that handles connecting to and disconnecting from the Bridge runtime during Unity's Awake(), respectively OnApplicationQuit() functions. Make sure the Bridge runtime is running prior to launching the Unity scene. If the connection fails, a message will appear in the console.

The example uses the "BridgeManager" object to attach the BridgeSDK to. From there, any game object with a script component can access functions, as shown in the script attached to the "SampleController" object.
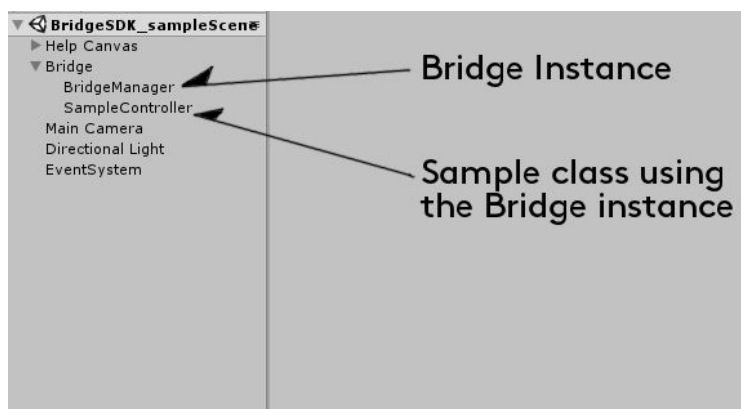
Sample scene hierarchy

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

NB: The sample project does not include any third party VR plugin or software (SteamVR not included). More information is available in the README.txt at the root of the sample.

8. Feedback & Bug report procedure:

DISCLAIMER: Please be aware this is an <u>BETA version</u> of this SDK and it mainly meant as a POC and to spark discussion and feedback from your side. <u>You can expect seeing bugs and robustness issues</u>, 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 supportsdk@logitech.com.