Logitech Gaming G-key SDK
V1.02

Overview and Reference

Contents

Overview 4
User experience 4
SDK basics 4
SDK Package 5
Requirements 5
Interfacing with the SDK 5
Using LogitechGkey.h and LogitechGkey.lib to access LogitechGkey.dll 5
Using LogitechGkey.dll directly 5
Features of Logitech devices with G-keys 5
G710 5
G510 6
G110 6
G19 7
G103 7
G105 7
G105 Call Of Duty 8
G11 8
G13 9
G15 v1 9
G15 v2 10
G35 10
G930 11
Features of Logitech gaming mice with extra buttons 12
G600 12
G300 12
G400 12
Do's and Don'ts 13
Sample usage of the SDK 13
Using header and lib 13
Using DLL directly 14
Reference 16
logiGkeyCBContext Structure 16
Parameters for callback function 16
LogiGkeyInit 17
Parameters 17
Return value 17

LogiGkeyIsMouseButtonPressed....17
Parameters....17
Return value....17
LogiGkeyGetMouseButtonString....17
Parameters....18
Return value....18
LogiGkeyIsKeyboardGkeyPressed....18
Parameters....18
Return value....18
LogiGkeyGetKeyboardGkeyString....18
Parameters....18
Return value....18
LogiGkeyShutdownFunction....18

Overview

The Logitech Gaming G-key SDK enables to get the current state of G-keys and extra mouse buttons for supported Logitech gaming mice and keyboards. It can be used via callback or polling. It only works when the Logitech Gaming Software is running (8.30 or later).

User experience

When the Logitech Gaming Software (LGS) is installed and running (8.30 or later), the user needs to specifically allow their device buttons to report G-keys/buttons directly to the game rather than using macros. If an associated game profile exists in LGS for your application, users can do so by dragging a "G-key" command on top of their G-keys/buttons.

The "G-key" command only appears in LGS for a specific profile after the first time the corresponding application with G-key SDK integration is run. Before that it is not visible anywhere.

If no profile exists when the application is run for the first time, a corresponding profile is created with "G-key" in the commands list.

The SDK will not forward G-key/button events if the user is running an older Logitech Gaming Software (8.20 and earlier).

For mice with on-board modes (G600 and G300), the SDK only works if the mouse is not in on-board mode.

SDK basics

The SDK is a Windows based API for C/C++ programmers. Games based on the Microsoft Win32 API do not access hardware directly. Instead, the Logitech Gaming G-key SDK interacts with supported Logitech devices on behalf of the games.

When you want to enable your application to receive the G-key and mouse button events, you call LogiGkeyInit(), passing in the callback context. After the call to LogiGkeyInit (), whenever the user presses/releases a G-key or extra mouse button in your application on the supported Logitech device, your application's callback function will get called with the proper G-key event message.

Applications will only get G-key/button events from keyboard/mouse if the application is currently in the foreground.

The callback function is executed in the context of a thread within the library. Therefore, take the necessary precautions for thread safety should the callback share resources with other threads in your application.

If you would rather poll every frame than to use the callback you can use the LogiGkeyIsMouseButtonPressed() and LogiGkeyIsKeyboardGkeyPressed() functions. Internally the devices will not get polled every time these functions are called. Instead they simply return the current state based on events received internally. The downside is that a button press could potentially get missed in case a user manages to press and release it within a single game loop.

For convenience when needing to define a string for the G-key or button that was pressed, you can use the LogiGkeyGetMouseButtonString() and LogiGkeyGetKeyboardGkeyString() functions.

When you are done listening to G-key events, call LogiGkeyShutdown () to free up the SDK resources.

For gaming mice, the G-key SDK only reports button events from mouse button 6 and above. Mouse button 1-5 are standard mouse buttons. They are not considered as G-keys.

NOTE: When developing and launching different versions of the same application (debug vs. release) and those different versions have the same executable name, make sure your corresponding profile in LGS points to the same executable. For example if running for the first time with the debug version, a profile will be created in LGS. But then when running a release version, if it has the same executable name, a new profile will not be created, but because the existing profile points to the debug version, things will not work.

SDK Package

The following files are included:

- LogitechGkey.h: C/C++ header file containing function prototypes
- LogitechGkey.lib: companion lib file to access DLL exported functions (32 and 64 bit)
- LogitechGkey.dll: library of SDK functions (32 and 64 bit)

Requirements

The Logitech Gaming G-key SDK can be used on the following platforms:

- Windows XP SP2 (32-bit and 64-bit)
- Windows Vista (32-bit and 64-bit)
- Windows 7 (32-bit and 64-bit)
- Windows 8 (32-bit and 64-bit)

The Logitech Gaming G-key SDK is a C based interface and is designed for use by C/C++ programmers. Familiarity with Windows programming is required.

Interfacing with the SDK

Using LogitechGkey.h and LogitechGkey.lib to access LogitechGkey.dll

The application can include LogitechGkey.h and link to LogitechGkey.lib (see "Sample usage of the SDK" further below or sample program in Samples folder). Installation folder for the DLL needs to be the same as the main executable, or needs to be part of the Path in the system environment.

Using LogitechGkey.dll directly

Alternatively the game can use the DLL directly by loading it via LoadLibrary, and accessing its functions using GetProcAddress (see "Sample usage of the SDK" further below or sample program in Samples folder).

Features of Logitech devices with G-keys

G710

G-keys: 1 to 6
Number of modes: 3

G510

G-keys: 1 to 18
Number of modes: 3

G110

G-keys: 1 to 12
Number of modes: 3

G19

G-keys: 1 to 12
Number of modes: 3

G103

G-keys: 1 to 6
Number of modes: 3

G105

G-keys: 1 to 6
Number of modes: 3

G105 Call Of Duty

G-keys: 1 to 6
Number of modes: 3

G11

G-keys: 1 to 18
Number of modes: 3

G13

The SDK treats this device as a keyboard.

G-keys: 1 to 29
Number of modes: 3

G15 v1

G-keys: 1 to 18
Number of modes: 3

G15 v2

G-keys: 1 to 6
Number of modes: 3

G35

G-keys: 1 to 3
Number of modes: 1

G930

G-keys: 1 to 3
Number of modes: 1

Features of Logitech gaming mice with extra buttons

G600

Extra buttons: 6 to 20

G300

Extra buttons: 6 to 9

G400

Extra buttons: 6 to 8

Do's and Don'ts

These are a few guidelines that may help you implement 'better' support in your game:

- If using the callback works for you, use it rather than polling. When polling if a user clicks and releases a button within a single game loop the button press will not be seen.

Sample usage of the SDK

Using header and lib

```
#include "LogitechGkey.h"

…

void __cdecl GkeySDKCallback(GkeyCode gkeyCode, wchar_t* gkeyOrButtonString, void*
/*context*/)
{
    // Look at gkeyCode to figure out which G-key/mode or button has been pressed or
released
    …
}

…

logiGkeyCBContext gkeyContext;
ZeroMemory(&gkeyContext, sizeof(gkeyContext));
gkeyContext.gkeyCallBack = (logiGKeyCB)GkeySDKCallback;
gkeyContext.gkeyContext = NULL;

// If polling instead of callback use NULL as argument
LogiGkeyInit(&gkeyContext);

…

// If not using callback, check all G-keys to see if they are being pressed
for (int index = 6; index <= LOGITECH_MAX_MOUSE_BUTTONS; index++)
    {
```

```
    if (LogiGkeyIsMouseButtonPressed(index))
    {
        …
    }
}

for (int index = 1; index <= LOGITECH_MAX_GKEYS; index++)
{
    for (int mKeyIndex = 1; mKeyIndex <= LOGITECH_MAX_M_STATES; mKeyIndex++)
    {
        if (LogiGkeyIsKeyboardGkeyPressed(index, mKeyIndex))
        {
            …
        }
    }
}

…

LogiGkeyShutdown();
```

Using DLL directly

```
#define LOGITECH_MAX_MOUSE_BUTTONS 20
#define LOGITECH_MAX_GKEYS 29
#define LOGITECH_MAX_M_STATES 3

typedef struct
{
    unsigned int keyIdx         : 8;        // index of the G key or mouse button, for
example, 6 for G6 or Button 6
    unsigned int keyDown        : 1;        // key up or down, 1 is down, 0 is up
    unsigned int mState         : 2;        // mState (1, 2 or 3 for M1, M2 and M3)
    unsigned int mouse          : 1;        // indicate if the Event comes from a mouse,
1 is yes, 0 is no.
    unsigned int reserved1      : 4;        // reserved1
    unsigned int reserved2      : 16;       // reserved2
} GkeyCode;

// Callback used to allow client to react to the Gkey events. It is called in the context
of another thread.
typedef void (WINAPI *logiGkeyCB)(GkeyCode gkeyCode, const wchar_t* gkeyOrButtonString,
void* context);

typedef struct
{
    logiGkeyCB    gkeyCallBack;
    void*       gkeyContext;
}logiGkeyCBContext;

typedef BOOL (* LPFNDLLGKEYINIT)(logiGkeyCBContext*);
typedef BOOL (* LPFNDLLGKEYISMOUSEBUTTONPRESSED)(int);
typedef wchar_t* (* LPFNDLLGKEYGETMOUSEBUTTONSTRING)(int);
typedef BOOL (* LPFNDLLGKEYISKEYBOARDGKEYBUTTONPRESSED)(int, int);
typedef wchar_t* (* LPFNDLLGKEYGETKEYBOARDGKEYSTRING)(int, int);
typedef void (* LPFNDLLGKEYSHUTDOWN)();
```

```
LPFNDLLGKEYINIT g_lpfnDllGkeyInit = NULL;
LPFNDLLGKEYISMOUSEBUTTONPRESSED g_lpfnDllGkeyIsMouseButtonPressed = NULL;
LPFNDLLGKEYGETMOUSEBUTTONSTRING g_lpfnDllGkeyGetMouseButtonString = NULL;
LPFNDLLGKEYISKEYBOARDGKEYBUTTONPRESSED g_lpfnDllGkeyIsKeyboardGkeyPressed = NULL;
LPFNDLLGKEYGETKEYBOARDGKEYSTRING g_lpfnDllGkeyGetKeyboardGkeyString = NULL;
LPFNDLLGKEYSHUTDOWN g_lpfnDllGkeyShutdown = NULL;

…

void __cdecl GkeySDKCallback(GkeyCode gkeyCode, wchar_t* gkeyOrButtonString, void*
/*context*/)
{
    // Look at gkeyCode to figure out which G-key/mode or button has been pressed or
released
    …
}

…

HINSTANCE logiDllHandle = LoadLibrary(L"LogitechGkey.dll");
if (logiDllHandle != NULL)
{
    g_lpfnDllGkeyInit = (LPFNDLLGKEYINIT)GetProcAddress(logiDllHandle, "LogiGkeyInit");
    g_lpfnDllGkeyIsMouseButtonPressed =
(LPFNDLLGKEYISMOUSEBUTTONPRESSED)GetProcAddress(logiDllHandle,
"LogiGkeyIsMouseButtonPressed");
    g_lpfnDllGkeyGetMouseButtonString =
(LPFNDLLGKEYGETMOUSEBUTTONSTRING)GetProcAddress(logiDllHandle,
"LogiGkeyGetMouseButtonString");
    g_lpfnDllGkeyIsKeyboardGkeyPressed =
(LPFNDLLGKEYISKEYBOARDGKEYBUTTONPRESSED)GetProcAddress(logiDllHandle,
"LogiGkeyIsKeyboardGkeyPressed");
    g_lpfnDllGkeyGetKeyboardGkeyString =
(LPFNDLLGKEYGETKEYBOARDGKEYSTRING)GetProcAddress(logiDllHandle,
"LogiGkeyGetKeyboardGkeyString");
    g_lpfnDllGkeyShutdown = (LPFNDLLGKEYSHUTDOWN)GetProcAddress(logiDllHandle,
"LogiGkeyShutdown");

    if (g_lpfnDllGkeyInit)
    {
        g_lpfnDllGkeyInit(&gkeyContext);
    }
}

…

for (int index = 6; index <= LOGITECH_MAX_MOUSE_BUTTONS; index++)
{
    if (g_lpfnDllGkeyIsMouseButtonPressed(index))
    {
        …
    }
}

for (int index = 1; index <= LOGITECH_MAX_GKEYS; index++)
{
```

```
    for (int mKeyIndex = 1; mKeyIndex <= LOGITECH_MAX_M_STATES; mKeyIndex++)
    {
        if (g_lpfnDllGkeyIsKeyboardGkeyPressed(index, mKeyIndex))
        {
            …
        }
    }
}

…

g_lpfnDllGkeyShutdown();
```

Reference

logiGkeyCBContext Structure

The logiGkeyCBContext is used to give the SDK enough information to allow the G-key events to be sent back to your application. The registered callback is called when the user presses/releases the G-key/mouse buttons, and the SDK client is currently in the foreground.

```
typedef struct
{
    logiGkeyCB    gkeyCallBack;
    void*       gkeyContext;
} logiGkeyCBContext;
```

```
typedef struct
{
    unsigned int keyIdx         : 8;        // index of the G key or mouse button, for
example, 6 for G6 or Button 6
    unsigned int keyDown        : 1;        // key up or down, 1 is down, 0 is up
    unsigned int mState         : 2;        // mState (1, 2 or 3 for M1, M2 and M3)
    unsigned int mouse          : 1;        // indicate if the Event comes from a mouse,
1 is yes, 0 is no.
    unsigned int reserved1      : 4;        // reserved1
    unsigned int reserved2      : 16;       // reserved2
} GkeyCode;
```

The callback function logiGkeyCB is defined as follows:

```
typedef void (__cdecl *logiGkeyCB)(GkeyCode gkeyCode, const wchar_t* gkeyOrButtonString,
void* context);
```

Parameters for callback function

- gkeyCode
    - This parameter tells you all the information about a G-key event, including the index of the G-key or mouse button which generates the event, the direction of the event (up or down), the M states of the G-key (1, 2 or 3), and if the event comes from a mouse.
- gkeyOrButtonString
    - Friendly string for the G-key or button event received. Can be used anywhere the name of the G-key/button needs to be displayed to the user. An example for keyboards is “G3/M1”, and for mice “Mouse Btn 7”.

- context
    - This specifies an arbitrary context value of the application that is passed back to the client in the event that the registered gkeyCallback function is invoked.

Remarks
Note that the callback function is executed in the context of a thread within the library. Therefore, take the necessary precautions for thread safety should the callback share resources with other threads in your application.

LogiGkeyInit
The **LogiGkeyInit**() function initializes the G-key SDK. It must be called before your application can see G-key/button events.

```
BOOL LogiGkeyInit(logiGkeyCBContext* gkeyCBContext);
```

Parameters
- gkeyCBContext: context for callback. See sample code above or sample program in Samples folder. This value can be set to NULL if the game wants to use the polling functions instead of a callback.

Return value
If the function succeeds, it returns TRUE. Otherwise FALSE.

Remarks
Use this initialization if working with any application that is not built using Unreal Engine or Unity game engine. For these two game engines use appropriates function as follows :
- Unreal Engine -> LogiGkeyInitWithoutCallback()
- Unity -> LogiGkeyInitWithoutContext

See the examples in the relative documentation to see how to get those functions to work.

LogiGkeyIsMouseButtonPressed
The **LogiGkeyIsMouseButtonPressed**() function indicates whether a mouse button is currently being pressed.

```
BOOL LogiGkeyIsMouseButtonPressed(int buttonNumber);
```

Parameters
- buttonNumber: number of the button to check (for example between 6 and 20 for G600).

Return value
TRUE if the specified button is currently being pressed, FALSE otherwise.

LogiGkeyGetMouseButtonString
The **LogiGkeyGetMouseButtonString** () function returns a button-specific friendly string.

```
wchar_t* LogiGkeyGetMouseButtonString(int buttonNumber);
```

Parameters

- buttonNumber: number of the button to check (for example between 6 and 20 for G600).

Return value

Friendly string for specified button number. For example "Mouse Btn 8".

LogiGkeyIsKeyboardGkeyPressed

The **LogiGkeyIsKeyboardGkeyPressed**() function indicates whether a keyboard G-key is currently being pressed.

```
BOOL LogiGkeyIsKeyboardGkeyPressed(int gkeyNumber, int modeNumber);
```

Parameters

- gkeyNumber: number of the G-key to check (for example between 1 and 6 for G710).
- modeNumber: number of the mode currently selected (1, 2 or 3)

Return value

TRUE if the specified G-key for the specified Mode is currently being pressed, FALSE otherwise.

LogiGkeyGetKeyboardGkeyString

The **LogiGkeyGetKeyboardGkeyString** () function returns a G-key-specific friendly string.

```
wchar_t* LogiGkeyGetKeyboardGkeyString(int gkeyNumber, int modeNumber);
```

Parameters

- gkeyNumber: number of the G-key to check (for example between 1 and 6 for G710).
- modeNumber: number of the mode currently selected (1, 2 or 3)

Return value

Friendly string for specified G-key and Mode number. For example "G5/M1".

LogiGkeyShutdownFunction

The **LogiGkeyShutdown** function unloads the corresponding DLL and frees up any allocated resources.

```
void LogiGkeyShutdown();
```