

Logitech Programming Guidelines For G-series Keyboard LCD V1.02

© 2005 Logitech

Confidential

The Logitech LCD SDK, including all accompanying documentation, is protected by intellectual property laws. All use of the Logitech LCD SDK is subject to the License Agreement found in the "ReadMe License Agreement" file and at the end of this document. If you do not agree to the terms and conditions of the License Agreement, you must immediately return any documentation, the accompanying software and all other material provided to you by Logitech. All rights not expressly granted by Logitech are reserved.

Contents

Contents	2
Introduction	3
Connection to LCD library	4
Persistent versus non-persistent	5
AutoStartable versus non-AutoStartable	6
Determining which LCD to use	7
Opening the LCD	8
Displaying content on the LCD	9
Increasing your chances of being displayed on the LCD.....	9
Using high priority screens.....	9
Using IgLcdSetAsLCDForegroundApp() function	10
Other considerations when multiple screens are to be shown.....	11
Cycling through screens using a time constant using a single connection	11
Using multiple connections with a single screen each, all in a single executable	11
Where to install	13
Installation directory	13
Legal Note	14

Introduction

The purpose of this document is to give some advice and guidelines to help ensure a successful implementation of support for the Logitech G-series Keyboard LCD in an application such as a game. This document is meant to be used in conjunction with the "IgLCD.doc" or "IgLCD.pdf" files that have complete details of all the function calls and parameters that are referred to in this document.

The LCD library ships with a developer SDK which includes a library "IgLCD.lib" for applications to link to, along with documentation that describe the complete API available for using the LCD. The LCD library is written to allow for multiple applications using one or more connected LCDs simultaneously.

Connection to LCD library

An application links with the `IgLcd.lib` library. Once linked it can start using the provided functions in the LCD API in order to connect and display information on the LCD.

The first call to the library must be the `IgLcdInit()`. This allows the library to initialize itself. The last call to the library must be the `IgLcdDeInit()`, which frees all the allocated resources by the LCD library.

After initializing the library using `IgLcdInit()`, the application must connect to the library using `IgLcdConnect` function. An application can have multiple connections established with the LCD library. This feature is useful, in order to reduce the number of executables that are running on the user's system. As an example, an application can have a connection established for displaying the weather, while it creates another connection for displaying the time. Each connection will have its own name, hence allowing the user to enable or disable the connection from the LCD library control panel shown later in this document.

A connection is established via a call to the SDK's *IgLcdConnect* function, as shown in the sample code below.

```
//// connect to LCDMon
// set up connection context
IgLcdConnectContext connectContext;
ZeroMemory(&connectContext, sizeof(connectContext));
connectContext.appFriendlyName = _T("simple sample");
connectContext.isAutostartable = FALSE;
connectContext.isPersistent = FALSE;
// we don't have a configuration screen
connectContext.onConfigure.configCallback = NULL;
connectContext.onConfigure.configContext = NULL;
// the "connection" member will be returned upon return
connectContext.connection = LGLCD_INVALID_CONNECTION;
// and connect
res = IgLcdConnect(&connectContext);
```

Each connection is considered as an applet by the LCDMon, which shows the applet's friendly name in the LCD Control Panel (fig. 1). Furthermore, for every applet appearing in the Control Panel, there is a checkbox and a "Configure" button.

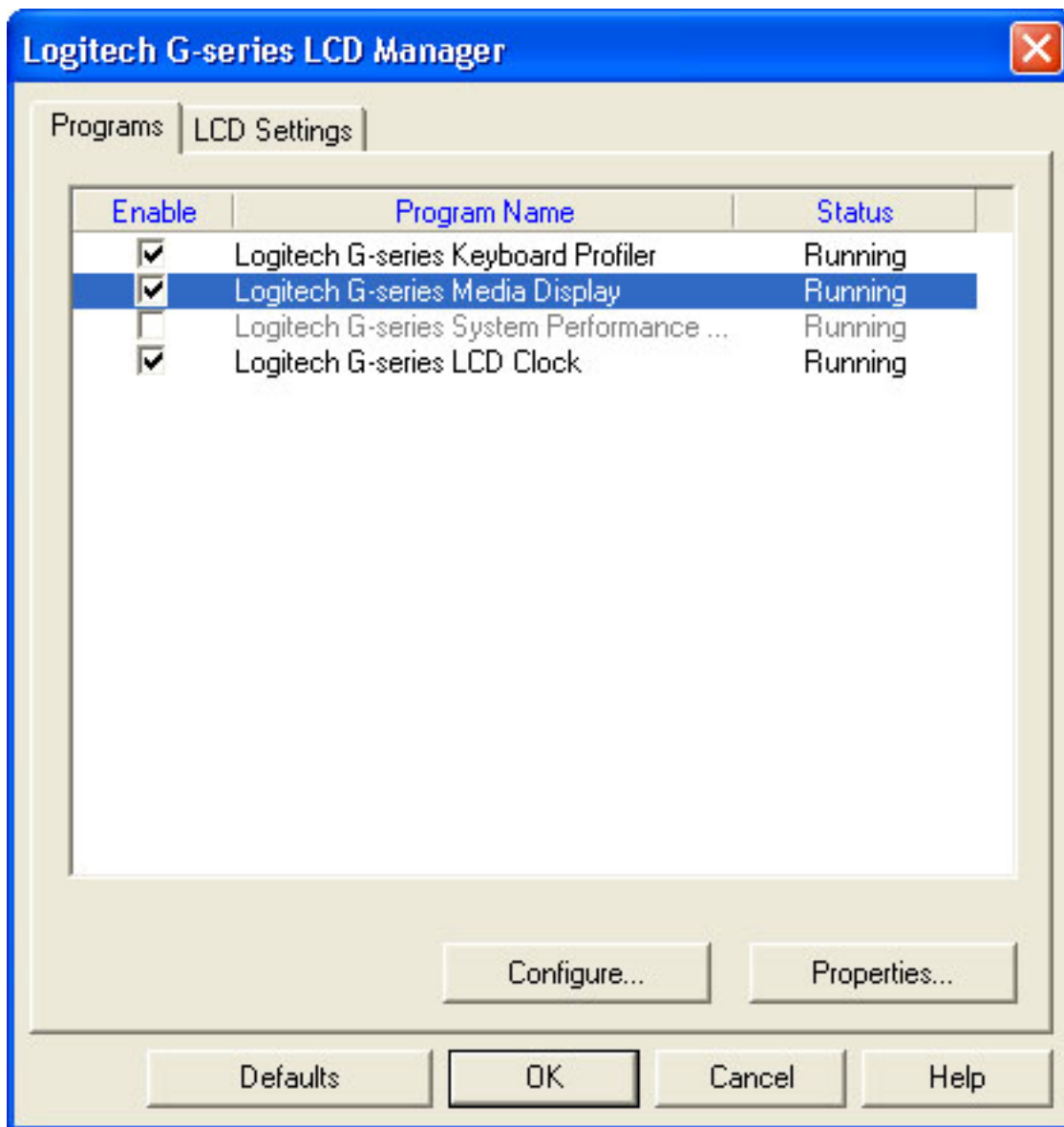


Figure 1

The checkbox allows a user to disable or enable each applet separately. Enabled applets are then given screen time according to the user's settings in the "LCD Settings" tab (fig. 2). Disabled applets are not shown on the LCD and don't get any screen time allocated.

The "Configure" button enables the application to display its own configuration screen. This is useful for simple applets that will not have any standalone user interface.

An application must terminate all connections using the `IgLcdDisconnect()` function provided.

Persistent versus non-persistent

The persistence is set inside the `IgLcdConnectContext` structure. If persistent is set to true, then the applet will show up in the LCD Control Panel even if it is not running. If persistent is set to false, then the applet shows up only while it's running.

A game should typically be non-persistent, and should not make usage of the Configure button in the LCD control panel. Instead it should use in-game menus for LCD configurations.

Good candidates for being persistent are applets that may run in the background anytime, and that may be started through the LCD control panel anytime. A weather applet or a messenger applet could be examples of persistent applets.

AutoStartable versus non-AutoStartable

The `isAutostartable` flag is set inside the *IgLcdConnectContext* structure. If it is set to true, then the applet will be started every time LCD library is started, which happens usually during a system start up. This is useful for applets that need to run and can run on their own in the background anytime. This flag should not be set for applications such as games that want to use the LCD.

Determining which LCD to use

After a connection is established, the application can retrieve the capabilities of all the attached LCDs that are supported. To enumerate the attached devices, call `IgLcdEnumerate()` and pass in 0 as index parameter. On subsequent calls, increase the index parameter by 1 until the function returns `ERROR_NO_MORE_ITEMS`. For every device that is found at a given index, its capabilities are returned in the `IgLcdDeviceDesc` structure provided by the application in the function parameter.

Opening the LCD

Once a device is found on a particular index using the `IgLcdEnumerate()` function, the application uses the `IgLcdOpen()` function to open the device at that index. The library returns the `Id` for the device that it will use to reference the device in the `IgLcdOpenContext` structure device parameter.

```
m_ctx.index = m_nIndex; // Chosen using IgLcdEnumerate
// OnSoftButtonCB is a member function that will be called during
// callbacks by the LCD library.
m_ctx.onSoftbuttonsChanged.softbuttonsChangedCallback =
                                OnSoftbuttonsCB;
m_ctx.onSoftbuttonsChanged.softbuttonsChangedContext = this;

DWORD dwRes = IgLcdOpen(&m_ctx);
if (ERROR_SUCCESS != dwRes)
{
    // something has gone wrong with the open call.
    HandleLgLcdError(dwRes);
}
```


Displaying content on the LCD

Once an LCD device is opened using the supplied calls in the LCD API, the application can provide bitmaps to be displayed on the LCD using the `IgLcdUpdateBitmap` function. Each application connected to the LCD library has a virtual LCD that it displays into. The library uses the algorithm chosen by the user to determine which virtual LCD will be displayed next on the LCD. There are three ways an application can update its content on the LCD using the `IgLcdUpdateBitmap` call. The first is by writing asynchronously to the LCD. This is the fastest mode, and the application only delivers the bitmap to the virtual LCD. The delivered bitmap is then later displayed by the LCD library when the application is scheduled to be displayed on the LCD. Using the synchronous mode of operation takes approximately 30 ms per call, and the call will return after the frame time is exhausted. Since there are multiple applications that can be using the LCD at the same time, it is not guaranteed that a frame delivered in synchronous fashion will actually be displayed by the library. During each frame time (30 ms), the library examines all the synchronous requests that are pending, and will display one and respond back to all the others with `ERROR_SUCCESS`, even though they were not displayed. This is done in order not to lock out any application.

Version 1.02 of the SDK has added the mode of `SYNC_COMPLETE_WITHIN_FRAME`. This mode will return an error code of `ERROR_ACCESS_DENIED` for every `IgLcdUpdateBitmap` call that is made using this parameter that does not get displayed on the LCD within the current frame. The frames that are displayed on the LCD will get back `ERROR_SUCCESS` as their return code.

```
int priority = m_PriorityCurrentlyChosen;
if (m_bSyncUpdatesCompleteWithinFrame)
{
    priority = LGLCD_SYNC_COMPLETE_WITHIN_FRAME(priority);
}
else if (m_bSyncUpdates)
{
    priority = LGLCD_SYNC_UPDATE(priority);
}
else
{
    priority = LGLCD_ASYNC_UPDATE(priority);
}

DWORD dwRes = IgLcdUpdateBitmap(m_ctx.device, &m_Bitmap, priority);
if (ERROR_SUCCESS != dwRes)
{
    HandleLgLcdError(dwRes);
}
```

Increasing your chances of being displayed on the LCD

Using high priority screens

An application can set the priority for each bitmap sent via `IgLcdUpdateBitmap`. If for example an application has a base screen, but wants to display another screen as an alert screen, it can display that other screen with a higher priority, which will result in it being more likely to be displayed, if the LCD Settings is in the Klever-VU rotate mode. Do not over-use the high priority or it will result in LCDMon automatically lowering the application's overall priority for a period of time in order to prevent a single application of using all of the LCD's time.

The user changes the algorithm that the library uses by using the LCD Settings tab in the LCD control panel as shown in Figure 2. There are four algorithms to choose from. In Manual algorithm, the user changes the

applications on the LCD by using the control button of the LCD, known as the reserved button. If “Immediate Display Of High Priority Items” is checked and the “Manual” mode is selected, then an application will be shown on the LCD, when it delivers a high priority bitmap to the library. Switching is still left to the user using the LCD controls.

If “Rotate Between Programs” is selected, then each application will get a fixed interval of time in seconds to display its bitmaps. The time interval is determined by the user, using the slider control shown in figure 2. If “Use Klever-VU Technology” is enabled as well, then the library will use the priority field in the IgLcdUpdateBitmap to determine which client will be shown on the LCD.

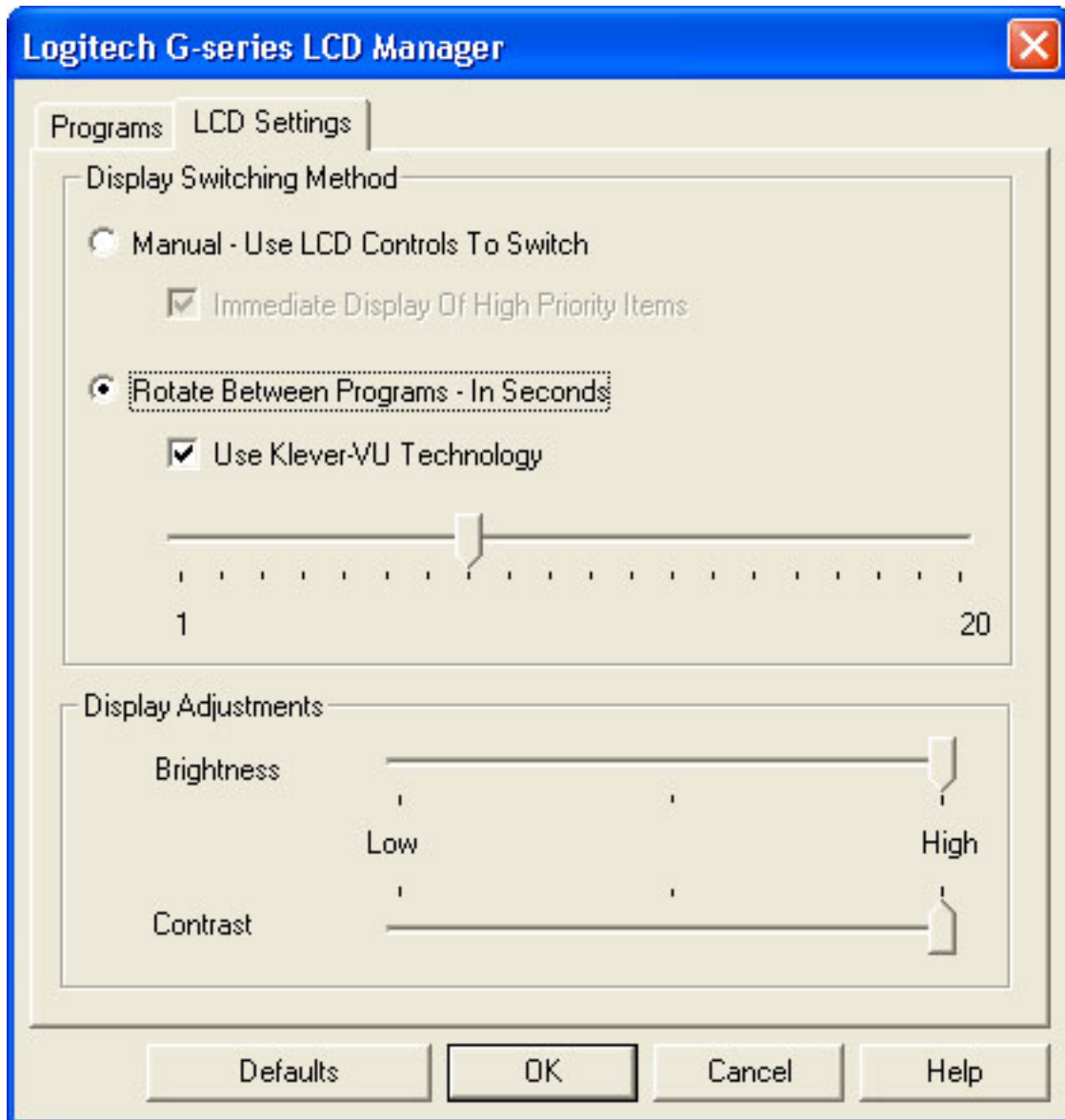


Figure 2

Using IgLcdSetAsLCDForegroundApp() function

Applications such as games that want to be on the LCD and not be swapped out, can use the IgLcdSetAsLCDForegroundApp() call, with the foregroundYesNoFlag set to “YES”.

```
DWORD dwRes = IgLcdSetAsLCDForegroundApp(m_ctx.device,
                                           LGLCD_LCD_FOREGROUND_APP_YES);

if (ERROR_SUCCESS != dwRes)
{
```

```

    HandleLgLcdError(dwRes);
}

```

Using this call will cause the LCD library to switch its algorithm to Manual (no rotation) and display the application that called this function last. Keep in mind that multiple applications can call this function and the library will display the last calling application's bitmaps on the LCD. When an application that is the foreground application on the LCD leaves this mode (calls the function with the "NO" parameter) the library will switch to the previous, if any, application that had called this function. Once the last application leaves the foreground mode, the library will return to whatever algorithm was chosen by the user, prior to the first application entering the foreground mode.

```

DWORD dwRes = lgLcdSetAsLCDForegroundApp(m_ctx.device,
                                           LGLCD_LCD_FOREGROUND_APP_NO);

if (ERROR_SUCCESS != dwRes)
{
    HandleLgLcdError(dwRes);
}

```

Other considerations when multiple screens are to be shown

The application may have multiple screens sent via `IgLcdUpdateBitmap`. This situation applies to applications that have multiple screens to display that are similar in nature. These multiple screens could be dealt with in a number of ways, which have their pros and cons:

Cycling through screens using a time constant using a single connection

The application could simply be cycling through its different screens, using a time constant. The problem with this solution is that the user has no control over this internal cycling using the LCD Control Panel. So for example if the user set the overall cycling to 5 seconds, and the application has 3 screens cycling at 2 seconds each, the 3rd screen will be cut off. If now the user decided to shorten the overall cycling down to 2 seconds, he will see one or 2 of the application's screens somewhat randomly. The application could address the issue by offering a configuration screen of its own enabling the user to disable some of its internal screens or to change the method of internal screen switching.

Using multiple connections with a single screen each, all in a single executable

This method should be best for applications that have multiple screens that are quite distinct in what they display. The way to create multiple connections is to call `IgLcdConnect` multiple times, as shown in the sample code below. Each screen in this case will show up as a separate applet in the LCD Control Panel. The advantage is that each screen is guaranteed to be displayed if its checkbox is selected in the LCD Control Panel. The disadvantage however is that if there are a lot of screens the LCD Control Panel will show a lot of applets for the same application, which may result in cluttering.

```

lgLcdConnectContext connectContext1;
ZeroMemory(&connectContext1, sizeof(connectContext1));
ConnectContext1.appFriendlyName = _T("simple sample applet #1");
...
res = lgLcdConnect(&connectContext1);

lgLcdConnectContext connectContext2;
ZeroMemory(&connectContext2, sizeof(connectContext2));
ConnectContext2.appFriendlyName = _T("simple sample applet #2");
...

```

```
res = lgLcdConnect(&connectContext2);
```

Where to install

You can choose to install your LCD application (applet) in your own directory by default or in the directory where the Logitech G series software is installed. All applets that are installed here, are automatically started by the LCD library once, and if they are auto-startable will be started every time the LCD library is started subsequently.

Installation directory

The registry key below gives the installed directory location of the LCD library.

HKEY_LOCAL_MACHINE\SOFTWARE\Logitech\G-series Software\LCD\CurrentVersion

The value is: **"InstallPath"**

"InstallPath" contains the installation path of the LCD library software (i.e. "**C:\Program Files\Logitech\G-series Software**"). Append the **"Applets"** folder onto it. This is where all the applets shipped from Logitech are stored. You must create a unique folder for your applet under the "Applets" folder, and store it along with any other files that it needs in there. An example is shown below:

"C:\Program Files\Logitech\G-series Software\Applets\SampleInc\SampleIncApplet.exe"

Make sure that the folder name you choose for your applet is not common; otherwise you risk being overwritten by a similar applet.

Legal Note

The Logitech LCD SDK, including all accompanying documentation, is protected by intellectual property laws. All use of the Logitech LCD SDK is subject to the License Agreement found in the "ReadMe License Agreement" file and in the Reference Manual. All rights not expressly granted by Logitech are reserved.