Logitech Programming Guidelines
For G15, Z-10 and G19 LCDs
V3.01

For use with the
Logitech LCD SDK
for Microsoft® Windows®
(lglcd)
V3.01

© 2006-2010 Logitech

Contents

Introduction

The purpose of this document is to give some advice and guidelines to help ensure a successful implementation of support for a Logitech LCD such as the one found on the G15 and G19 keyboards or the Z-10 speakers, in an application such as a game. This document is meant to be used in conjunction with the "lgLcd.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 "lgLcd.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 lgLcd.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 *lgLcdInit*(). This allows the library to initialize itself. The last call to the library must be the *lgLcdDeInit*(), which frees all the allocated resources by the LCD library.

After initializing the library using *lgLcdInit*(), the application must connect to the library using *lcLcdConnect*() or *lgLcdConnectEx*() 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 *lgLcdConnect()* function, as shown in the sample code below.

```
// connect to LCD Manager

// set up connection context

lgLcdConnectContext 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 = lgLcdConnect(&connectContext);
```

An applet can also connect using *lgLcdConnectEx()*. Using this function enables the applet to indicate what type of device it supports, as well as enables the notification functionality, which can be used to optimize the resource usage of an applet (and dynamically react to device plug/unplug).

Each connection is considered to be an applet by the LCD Manager, 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 if you have user configurable items and you enable this callback function.
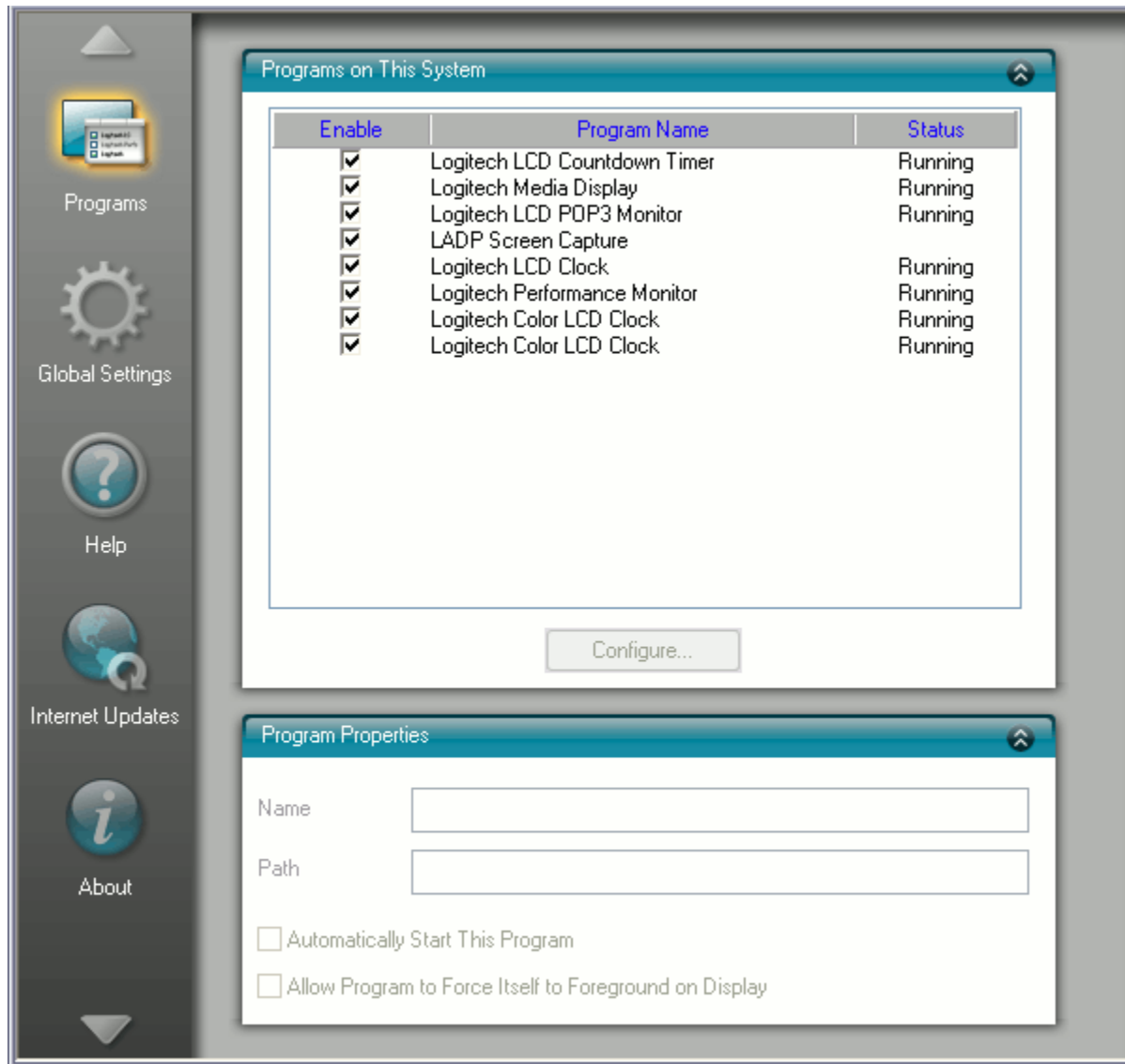
Figure 1

The "enable" checkbox to the left of each application 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 Configuration" property sheet. 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. The configure button is only enabled if you have a handler for the button and you have registered the callback function using *onConfigure* members of the connect context structure during connection to the LCD Manager.   The following code fragment shows how you would initalize your applet to handle the configuration button

```
m_connectContext.onConfigure.configCallback = LcdOnConfigureCB;
m_connectContext.onConfigure.configContext = this;

// the "connection" member will be returned upon return
m_connectContext.connection = LGLCD_INVALID_CONNECTION;

// and connect
rc = lgLcdConnect(&m_connectContext);
```

5

The function that you defined as your callback needs to do some special processing to initialize you configuration page and bring the page to the foreground. Keep in mind that in general your applet will need to hide the main window (normally the configuration window) at startup and this window is then shown when the configuration callback is executed. Here is an example of a possible callback function for MyAppletDlg. The function tries to place the window in the center of the display and then show it to the user.

```
DWORD CMyAppletDlg::LcdOnConfigureCB(int connection, const PVOID pContext)
{
    // NOTE: This callback may occur in the LCD Manager's thread context
    // try and bring ourselves to the foreground centered in the window

    CMyAppletDlg* pThis = (CMyAppletDlg *)pContext;

    pThis->MoveWindow(pThis->getOrigRect(),TRUE);
    pThis->SetWindowPos(&wndTopMost, -1, -1, 0, 0,
        SWP_NOMOVE|SWP_NOSIZE|SWP_SHOWWINDOW);
    pThis->SetWindowPos(&wndNoTopMost, -1, -1, 0, 0,
        SWP_NOMOVE|SWP_NOSIZE|SWP_SHOWWINDOW);
    pThis->ShowWindow(SW_SHOWNORMAL);

    return ERROR_SUCCESS;
}
```

An application must terminate all connections using the *lgLcdDisconnect*() function provided.

Persistent versus non-persistent

Previous versions of the LCD Manager API supported the concept of persistent applets. In as of Version 3.x of the LCD Manager and API this functionality has been deprecated.

AutoStartable versus non-AutoStartable

The *isAutostartable* flag is set inside the *lgLcdConnectContext* 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.

I'm Connected, now what?

Once a connection to the LCD Manger is established, there is one further step that must be completed before you can begin sending data to an LCD Device. This step is:

- Opening the device with the desired display capability.

Opening the LCD

An applet typically will display its data in either B/W or color (dual-mode applets are described later in this document). The following code fragment illustrates how to open a desired device:

```
m_ctx.devicetype = LGLCD_DEVICE_BW;

// 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 = lgLcdOpenByType(&m_ctx);

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

The library will return a device handle in the *lgLcdOpenByTypeContext.device* member:

```
int nDevice = m_ctx.device;
```

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 *lgLcdUpdateBitmap*() 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 *lgLcdUpdateBitmap*() 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 *lgLcdUpdateBitmap*() 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 = lgLcdUpdateBitmap(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 *lgLcdUpdateBitmap*(). 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 LCD Manager 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 Configuration property page on 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 *lgLcdUpdateBitmap*() to determine which client will be shown on the LCD.
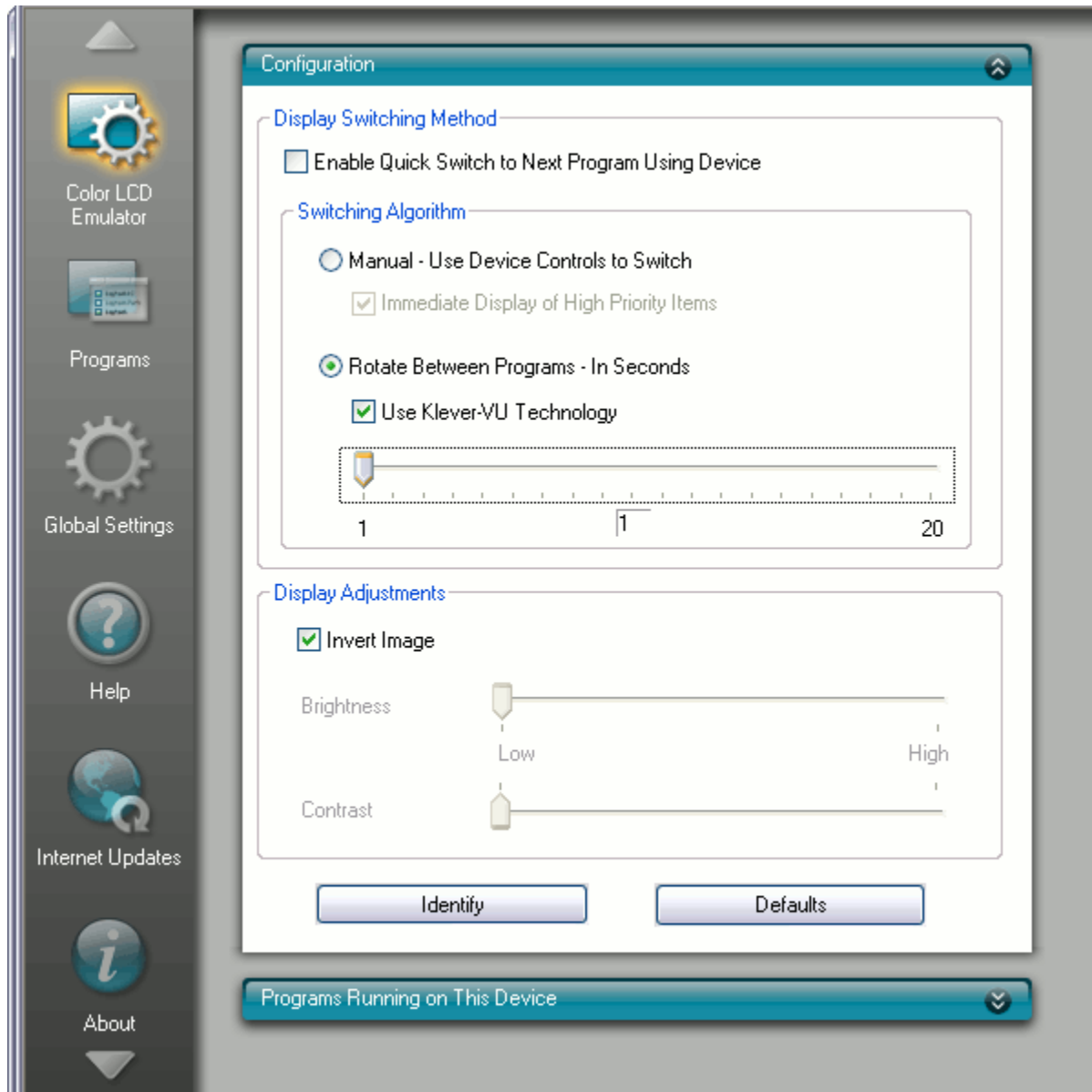
Figure 2

Using lgLcdSetAsLCDForegroundApp() function

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

```
DWORD dwRes = lgLcdSetAsLCDForegroundApp (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

10

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 *lgLcdUpdateBitmap*(). 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 two 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 *lgLcdConnect*() 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);
```

Developing Dual Mode Applets

Some applet developers will be interested in packaging a single executable that has support for both color and black and white devices. This approach will give your users the best overall experience when using your applet, but it does require you to keep track of two distinct devices: one that is to be used with black and white data, and the second to be used with color data.

The following code fragment illustrates how to open two types of devices:

```
lgLcdConnectContextEx connectContextEx;

ZeroMemory(&connectContextEx, sizeof (connectContextEx));

connectContextEx.appFriendlyName = _T("My Dual Mode Applet");
connectContextEx.dwAppletCapabilitiesSupported =
    LGLCD_DEVICE_BW | LGLCD_DEVICE_QVGA;

res = lgLcdConnectEx(&connectContextEx);

// Open the color and b/w devices

int colorDevice = LGLCD_INVALID_DEVICE;
int bwDevice = LGLCD_INVALID_DEVICE;
lgLcdOpenByTypeContext ctx;

ZeroMemory(&ctx, sizeof(ctx));

ctx.connection = m_connectContextEx.connection;
ctx.deviceType = LGLCD_DEVICE_QVGA;

res = lgLcdOpenByType(&ctx);

if(ERROR_SUCCESS == res)
{
    colorDevice = ctx.device;
}

ctx.deviceType = LGLCD_DEVICE_BW;

res = lgLcdOpenByType(&ctx);

if(ERROR_SUCCESS == res)
{
    bwDevice = ctx.device;
}

// render bitmaps and send to appropriate devices...
```


Important Programming Note

Keep in mind all of these operations will be performed against a single connection context. Since the applet configuration callback is tied to a connection, your configuration page must be either be written in a generic fashion such that the configuration information is device independent, or you must include some mechanism on the configuration page that allows the user to configure all display types using the

same configuration screen. This can be accomplished using a set of tabs for each device format or other mechanism, but you should consider this when designing the configuration user interface for your applet.

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.

H KEY_LOCAL_MACHINE\SOFTWARE\Logitech\LCD Software\LCD Manager\CurrentVersion The value is: "InstallPath"

"InstallPath" contains the installation path of the LCD library software (i.e. 'C:\Program Files\Common Files\Logitech\LCD Software\LCD Manager'). 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\ Common Files\Logitech\LCD Software\LCD Manager \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 "Logitech LCD SDK License Agreement" file and in the Reference Manual. All rights not expressly granted by Logitech are reserved.