QSI CORPORATION

Application Note

Qlarity® USB Serial Communications for Windows®

Introduction

The QTERM-G58 and QTERM-G56 are USB composite devices, containing two USB functions: a mass storage device and a serial communications device. The mass storage function uses standard USB device drivers and should automatically mount and appear as a Removable Disk in the Windows Explorer, where files can be read and written normally. The serial communications function requires the installation of a custom QSI USB communications driver, which allow USB communication between the G58/G56 terminal and a Windows PC. This includes firmware and application download using Qlarity Foundry®, and communications between your custom application running on a Windows PC and the Qlarity application running on the terminal. The serial communications function supports multiple logical connections (called "destinations"). Each destination has its own handle and is treated as a separate serial connection. It is much like having several serial ports available on the G58/G56.

When the G58/G56 USB cable is first plugged into a PC, the PC USB host only sees the mass storage function. This allows the mass storage function to be used on any PC, without nagging the user for QSI drivers. However, if you want to use the USB serial communications function then you must load the QSI drivers.

To install the drivers, merely launch the QSI USB Driver install executable and follow the dialog prompts. Actually two drivers are installed: a filter driver (USBQSICTRL) and the serial function driver (USBQSICOM). The filter driver allows applications that are aware of the QSI USB communications function (such as Foundry) to enable/disable the communications function. The mass storage function is available in either case, but the G58/G56 must re-enumerate when the communications function is enabled or disabled.

USB Communications with Qlarity Foundry

After the drivers are installed, a new tab (labeled USB) will appear in the Qlarity Foundry download dialog. A list of connected QTERM-G58/G56 terminals appears below the tab (disconnected terminals are grayed out). The desired device may be selected from the list, then an application or firmware image may be downloaded. Make sure that the terminal is in the Application Loader before you attempt a download. (NOTE: In Qlarity Foundry 2.60, the results window is for standard serial communications only, and will not show any feedback from the terminal during USB downloads Also, Qlarity Foundry 2.60 does not support USB communications in Simulation Mode).

USB Communications with a Qlarity application

From a Qlarity perspective, USB communications are very similar to the existing (UART-based) serial and network communications. A USB communications channel is obtained by calling USBGetCommChannel() and released by calling USBCloseCommChannel (see documentation below). Once a communication channel is obtained, the application is informed that the USB host has connected to the associated destination via the MSG_USBD_CONNECTION message. Subsequent disconnections by the host are also passed to the application with this message.

Data is received from the USB host via MSG_COMM_RECEIVE messages and data is transmitted to the host with calls to the Transmit() or Send() API functions. Remember to register the object to receive data (with a MSG_COMM_RECEIVE handler) by calling RegisterMsgHandler().

QlarityUSBComm Library

QSI provides a C++ library that contains functions to access the USB serial communications with your Windows application. The library is called *QlarityUSBComm.lib* and comes with the header file *QlarityUSBComm.h*. There are several builds of the library; the one to use will depend on your development environment (i.e. Microsoft Visual Studio 2003 vs. 2005 and different runtime libraries used). Please contact QSI Support to obtain the correct version for your development environment. Sample code that demonstrates use of the library is also available. Some guidelines on using the library are included below.

First, you should ensure that the QSIUSBCom drivers are installed with this function:

```
bool IsDriverInstalled();
```

After confirming that the drivers are installed you need to get a list of connected QSI USB Communication devices. This also gives you the internalName of the device which you will use in other function calls to select a particular device.

```
bool GetDevicesNames(std::vector<DeviceInfo> &names, bool autoMakeReady, bool block);
```

parameters:

names (OUT) This vector will receive information about any QSI devices that are attached via USB. Any data in this vector when this function is called will be erased and replaced with the current information

autoMakeReady Indicates whether we should try to make any unread devices ready. When GetDeviceNames is searching for connected devices, if it detects a device which is attached but the serial communication mode is not enabled, then if this parameter is set to true it will enable serial communication on that device.

block If set to true then function will not return until the device is ready. This is only used if autoMakeReady is set to true

To get a list of destinations that are open on a device call the following function:

```
bool GetDestinations(const QlarityUSBWideString &internalName, std::vector<DestInfo> &names);
```

parameters:

internalName The internal name for the device that was returned from the GetDeviceNames function.

names A list of destination numbers and destination names that are available on the device.

Once you have the internal name and destination for the channel you want to use, get a handle for the destination by calling the OpenFileHandle() function. This handle can then be used with standard file I/O APIs: WriteFile() to transmit data and ReadFile() to receive data. Be sure to call CloseHandle() when you are finished communicating with the device.

```
HANDLE OpenFileHandle(const QlarityUSBWideString &internalName, USHORT dest, DWORD &maxPacketSize);
```

parameters:

internalName The internal name for the device that was returned from the GetDeviceNames function.

dest The destination that you want to use as your serial connection. A list of available destinations is determined by calling the GetDestinations function.

maxPacketSize (Out). Receives the maximum size, in bytes, for single read or write transfers.

Here is an example of opening a destination on a USB Comm device and receiving some data. For the sake of simplicity all error checking has been removed.

```
UCHAR buffer[50];
std::vector <DeviceInfo> DeviceNames;
std::vector <DestInfo> Destinations;
DWORD returnSize;
HANDLE handle;

QlarityUSB::IsDriverInstalled();
QlarityUSB::GetDeviceNames(DeviceNames, true, false);
if (DeviceNames.size() > 0) {
    QlarityUSB::GetDestinations(DeviceNames[0].internalName, Destinations);
    handle = QlarityUSB::OpenFileHandle(DeviceNames[0].internalName, Destinations[0].destNum, 8192);
    ReadFile(handle, buffer, 50, returnSize, NULL);
    CloseHandle(handle);
}
```

Some additional functions are defined. RenameDevice() is used to assign a friendly name to a device. GetLastErrors() can be used to display what the last error was. IsDeviceValidAndReady() is used to verify that the device is ready to open a file handle. GetDriverVersions() can be used to display the versions of the QSI USB Comm drivers. These should be self explanatory from comments in the header file.