Mesa2-3 RFID Source Code Explanation

Assumptions:

- You have already signed the RFID INDEMNIFICATION AGREEMENT
- You intend to write your own RFID application for the Windows Mesa2 or Mesa3
- Your programming language is c#

Your responsibilities:

- Regulatory compliance – This is the reason you signed the agreement, to control transmit power levels, operate in the correct frequency (region), and possibly put cellular into airplane mode.
- Control power during sleep – Your application is responsible for putting the RFID expansion into a low power mode during sleep (i.e. kill power to it and re-connect after resume). Otherwise, your battery life will suffer.

RFID Configuration:

Each RFID Mesa is pre-configured with a configuration. This configuration is stored as an SMBIOS key (similar to how the device serial number is stored). It can be obtained with a DLL that is built into the Mesa Operating System. This configuration consists of three items.

1) **The region**, such as NA2, EU3, or AU (for North America, Europe, or Australia). This corresponds to Jadak's regions that are set using the following function.

```
RfidReader.ParamSet("/reader/region/id", configuredRegion);
```

2) **The maximum average power level**. The M6e Nano module is capable of running at 27 dBm, but due to regulatory reasons, you cannot operate at this power level continuously. For example, in North America, this power level is 20 dBm (2000 centi-dBm), and so if the RFID module is going to be operated continuously, 20 dBm is the highest power setting you can use. If, however, you are going to be scanning intermittently, you just need to ensure that the average power is kept below 20 dBm by preventing scans for a certain amount of "off time" following a scan (see "Scanning" section below).
3) **Disable Cellular**. If the disable cellular flag is set, then cellular communications are not allowed at the same time as RFID scanning. To scan for RFID, first you must disable cellular, then scan RFID, and then re-enable cellular.

Here are two example RFID Configurations:

1) NA2, 2000, 0 – This is configured for North America (using NA2), has a maximum average power level of 2000 centi-dBm (20 dBm), and does not need to disable cellular during RFID scans.
2) EU3, 2400, 1 – This is configured for Europe (using EU3), has a maximum average power level of 2400 centi-dBm (24 dBm), and needs to disable cellular during RFID scans.

Software Components used:

There are two software components you will need to interface with. Both of which are built into the Mesa Operating System. First is the information DLL "`M2DllInfo.dll`" or "`M3DllInfo.dll`" for Mesa2 and Mesa3 respectively. This is how we obtain the RFID Configuration with the functions `GetMesa2RfidRegion() or GetRfidRegion()`. The second is the expansion driver. This is how power is turned on or off to the RFID module; and cellular is disabled or re-enabled.

More about disabling cellular:

There is a GPIO that connects the main processor to the cell module that, when asserted, shuts off the RF section of the cell module (basically puts it into airplane mode, although you won't see the airplane mode icon in Windows task tray). If cellular is disabled for a short enough time (perhaps less than 2 seconds), then the network has a good chance of remaining connected. Otherwise, the cellular network will disconnect, and will take roughly 15 seconds to reconnect after cellular is re-enabled.

Source Code:

The DriverAccess.cs file should be copied in its entirety to your project. This contains everything needed to interface to the expansion driver. The rest of the code, you will probably only want to copy functions or other code snippets from. Most of these are in the RfidModel.cs file.

Application Startup

When this application starts up, the InitializeRfid() function is called in RfidModel.cs. This function gets the configuration items, gets a handle to the expansion driver, checks to see if there is an RFID module found, and then connects to the RFID module. Some of the code that is common between first time startup, and re-connecting after sleep/resume is wrapped up in the StartupRfid() function. Before calling StartupRfid(), power is ensured to be disabled. Then this function gets a list of COM ports, applies power to the RFID module, and then gets a new list of COM ports. The new port should be the RFID module, and it connects to it using RfidReader.Connect(). Next the configuration items are compared to what the module supports, and then a read-once plan is setup. Last, the module is put into MINSAVE power mode until the user initiates a scan.

Handling Sleep/Resume

In MainWindow.xaml.cs, we register for suspend/resume notifications. See functions Window_Loaded/Window_Unloaded, and WndProc. When the Mesa enters sleep, it is preferable to

turn power off to the RFID module. This means that after the Mesa resumes, that most of the startup code must be ran again. One note about the suspend notification: sometimes this notification does not come immediately after the screen goes off. It can take a couple of minutes. But the notification does come when the processor finally enters its lowest power state for sleep.

Scanning and handling "Off Time"

See the ScanThread function in RfidModel.cs. Here, when we are ready to scan, we check to see if we are allowed to scan yet or not. If the last scan was more powerful than the maximum average power level allowed, we might have to wait for so much "off time" before we can allow a scan again. Next, if the Disable Cellular flag was set in the configuration, then we disable cellular before starting the scan. The scan time is timed so we can calculate how much off time we must enforce for next time. See also the calculateOffTime() function. Then there is a lot of keyboard simulating code you probably won't need for your app, and last we put the module back to MINSAVE and possibly re-enable cellular.

Scanning from a physical button

The easiest way to do this for an application that will remain in the foreground is to configure P1, P2, or P3 (physical buttons on the Mesa) to one of the "F" keys (such as F7) in the Keypad control panel application, and then handle that F key within your application.