

Battery Monitor Project

Guangen Xin

Ming Lei

Zixuan Zhang

1 Introduction

1.1 Context

Laptops are widely used in people's daily life. Not like a desktop, the battery is a significant part of a laptop. It is necessary for most softwares to gain current battery information while running. For example, a media player should be stopped if the power at a low level, or, an update program should not be started if the remaining power is not enough. Thus, it is useful for programmers to know how to get the battery information.

Windows Operating System has its own power management schemes, including the Power Saving, the Performance and the Balanced. Users can choose one of them in the Control Panel to control the power consumption, and get to know the remaining power of the battery through the battery icon on the task bar, but there is no specific information about the battery given to the user. This project is supposed to build an application to monitor the status of battery and get the details of battery information. The program can be run on the Windows Operating System and it supports Windows 8 or 10.

Windows systems can send system messages to user applications. When the power settings of the system change, the system automatically sends related system messages. Such system messages are Power Notifications. User applications can listen to the Power Notifications and get battery information. Additionally, a user application can query a device driver for detailed information related to a device. Control Codes are integers that can be transmitted from an user application directly to a device driver, to ask the device driver to perform certain operations. Power Management Control Codes are used by the project software to request detailed battery information from the battery device. The root header file of the Windows system is Windows.h, and the header contains numerous Application Programming Interfaces (API). These API calls are used to implement the functionalities of the project software.

1.2 Problem Statement

Since desktop machines use power supply rather than battery, the project software is not expected to display any information while it is running on a desktop machine. The software has its focus on monitoring laptop batteries. An example problem is, when a laptop's battery is malfunctioning, the user needs to know the real-time performance of the battery for troubleshooting. To solve this problem, a software that monitors the battery can gather the information about current status of the battery. Should a hardware problem occur in the laptop battery, the performance data displayed by the monitor software can be used to diagnose the battery. Such a software also serves as a mean to notify the user to switch to a new battery in case the current battery's wear is too high. Therefore, the project goal is to build an executable application to collect battery information. There are numerous structures with complex messages in APIs. The challenge part is to correctly handle the Power Notifications.

1.3 Result

The project software is able to collect battery information by parsing Power Notifications, and querying the battery device. Visual Studio, C++ and .NET Framework are used to build this application.

The following battery information can be collected by the project software:

- Remaining Power
- Current Power scheme
- Current Power Source
- Battery Name
- Technology
- Chemistry in Battery
- Design Capacity in mWh
- Full Capacity in mWh
- Current Capacity in mWh
- Battery Wear
- History Cycles
- Battery Status
- Current Voltage
- Current Rate
- Current Temperature
- Relevant Time

1.4 Outline

The rest of this report is structured as follows. Section 2 introduces the relevant APIs in this project and includes how they work. Section 3 describes the obtained battery information in details. The project will be evaluated in Section 4. Section 5 will conclude this battery monitor project and indicate which parts need to be improved.

2 Background Information

The project software uses two approaches to collect the battery information. The first one is listening to the Power Notifications that are broadcasted from the Windows system to user applications (Sections 2.1 to 2.3). The second is letting an application query the Windows system for detailed information related to a device (Sections 2.4 and 2.5).

2.1 System Messages of Windows Systems

In Windows system, the message is used for notifying the system event to the applications. Each message has a value stored in Msg. Msg is used for identifying the system event in the software, by comparing it to the Windows message code. This program is mainly use the message **WM_POWERBROADCAST** and the value is 0x0218. The message structure is assembled in System.Windows.Forms.dll, and it has other properties (reference [1]):

- Hwnd: Gets or sets the window handle of the message.
- Lparam: Specifies the LParam field of the message.

- Wparam: Gets or sets the WParam field of the message.
- Msg: Gets or sets the ID number for the message.
- Result: Specifies the value that is returned to Windows in response to handling the message.

2.2 Register and Unregister an Application for Receiving System Messages

The method **RegisterPowerSettingNotification** is used for registering the application to the system power message service, so that the application can listen to the messages broadcasted from the system (reference [2]).

```
457 BatteryProject::BatteryMonitor form; //create a BatteryMonitor object called "form"
458 HWND hwnd = static_cast<HWND>(form.Handle.ToPointer()); //get the window handle of that object
459 BatteryPowerNotifyHandle = RegisterPowerSettingNotification(hwnd, &GUID_BATTERY_PERCENTAGE_REMAINING, DEVICE_NOTIFY_WINDOW_HANDLE);
460 BatterySchemeNotifyHandle = RegisterPowerSettingNotification(hwnd, &GUID_POWERSCHEME_PERSONALITY, DEVICE_NOTIFY_WINDOW_HANDLE);
461 BatterySourceNotifyHandle = RegisterPowerSettingNotification(hwnd, &GUID_ACDC_POWER_SOURCE, DEVICE_NOTIFY_WINDOW_HANDLE);
```

Figure 2-1 Register the Application

The project software is implemented as a class called **BatteryMonitor**, which inherits from Windows Form class. Line 457 instantiate a **BatteryMonitor** object. The object's handle is statically casted and then used as a handle to the application.

At each Line from Line 459 to 461, **RegisterPowerSettingNotification** function is invoked with three arguments :

- **hwnd** - is a handle to a window, which is defined in WinDef.h. The windows system gives every windows application a unique hwnd handle to mark them.
- A pointer to a Power Setting GUID struct. Power Setting GUID structs are constants defined in WinNT.h, and are used to specify the type of power messages that the application want to receive (reference [3]). There are three types in this case
 1. **GUID_BATTERY_PERCENTAGE_REMAINING**
 2. **GUID_POWERSCHEME_PERSONALITY**
 3. **GUID_ACDC_POWER_SOURCE**
- **DEVICE_NOTIFY_WINDOW_HANDLE** - the flag of hwnd

The software has three pvoid (defined as **HPOWERNOTIFY**) pointers to store the returned notification by RegisterPowerSettingNotification method. The returned value will be null if the registration failed.

The software should unregister the message service before the software is exited. The program uses **UnregisterPowerSettingNotification** method. This method has one parameter which is the **HPOWERNOTIFY** variable as shown in Figure 2-2.

```
64 void BatteryMonitor::OnFormClosing(FormClosingEventArgs^ e) {
65     cout << "quit application\n" << endl;
66     UnregisterPowerSettingNotification(BatteryPowerNotifyHandle);
67     UnregisterPowerSettingNotification(BatterySchemeNotifyHandle);
68     UnregisterPowerSettingNotification(BatterySourceNotifyHandle);
69     Form::OnFormClosing(e);
70 }
```

Figure 2-2 Unregister the Application

2.3 Filter System Messages and Obtain System Data

The function **WndProc** is used for processing the power message, as shown in figure 2-3.

```
20 void BatteryMonitor::WndProc(Message& m) { //message handler function
21     if (m.Msg == WM_POWERBROADCAST) { //identify the message
22         cout << "Message id:" << m.Msg << endl;
23         int idevent = (int)m.WParam;
24         cout << "Event id:" << idevent << endl;
25         if (idevent == PBT_POWERSETTINGCHANGE) { //identify the event
26             POWERBROADCAST_SETTING* ppstruct = (POWERBROADCAST_SETTING*)m.LParam.ToPointer();
27             if (sizeof(int) == ppstruct->DataLength && ppstruct->PowerSetting == GUID_BATTERY_PERCENTAGE_REMAINING) { //obtained remaining power info
28                 cout << "Received data about remaining battery power" << endl;
29                 int percentageremaining = *(int*)(DWORD_PTR)ppstruct->Data;
30                 this->label2->Text = System::Convert::ToString(percentageremaining) + "%";
31             } else if (sizeof(int) == ppstruct->DataLength && ppstruct->PowerSetting == GUID_ACDC_POWER_SOURCE) { //obtained power source info
32                 cout << "Received data about type of power supply" << endl;
33                 int powerSource = *(int*)(DWORD_PTR)ppstruct->Data;
34                 if (powerSource == 1) {
35                     this->label6->Text = "Battery";
36                 } else {
37                     this->label6->Text = "AC Power";
38                 }
39             } else if (sizeof(GUID) == ppstruct->DataLength && ppstruct->PowerSetting == GUID_POWERSCHEME_PERSONALITY) { //obtained power scheme info
40                 cout << "Received data about power scheme" << endl;
41                 GUID newScheme = *(GUID*)(DWORD_PTR)ppstruct->Data;
42                 if (newScheme == GUID_MAX_POWER_SAVINGS) {
43                     this->label4->Text = "Power Saving";
44                 } else if (newScheme == GUID_MIN_POWER_SAVINGS) {
45                     this->label4->Text = "Performance";
46                 } else if (newScheme == GUID_TYPICAL_POWER_SAVINGS) {
47                     this->label4->Text = "Balanced";
48                 } else {
49                     this->label4->Text = "Balanced";
50                 }
51             }
52         }
53     }
54 }
55
56
57
58
59 if (m.Msg == WM_LBUTTONDOWN && batteryexist) { //load Battery info upon mouse click
60     this->loadBatteryInfo();
61 }
62 Form::WndProc(m);
63 }
```

Figure 2-3 Handling System Messages

First, this method needs to identify whether the received message, parameter **Message& m**, is power message, by checking if (**m.Msg == WM_POWERBROADCAST**). If the message type is **WM_POWERBROADCAST**, the **Wparam** will store the an event id related to this message type. A **WM_POWERBROADCAST** message can have many event ids (reference [4]). This project uses the **PBT_POWERSETTINGCHANGE** event only so the program needs to filter out other events by getting the event id at line 23 and checking if it is a **PBT_POWERSETTINGCHANGE** event at line 25. The numerical value of **PBT_POWERSETTINGCHANGE** is 0x8013.

The **POWERBROADCAST_SETTING** is a system struct shown in figure 2-4 (reference [5]).

```
typedef struct {
    GUID PowerSetting;
    DWORD DataLength;
    UCHAR Data[1];
} POWERBROADCAST_SETTING;
```

Figure 2-4 The POWERBROADCAST_SETTING Struct

This power setting struct is used for storing the data of the power message. Line 39 to 55 shows how to obtain the data in the power setting struct. At line 39 the **DataLength** member is compared with expected length, which is the size of a GUID in this case. Also the **PowerSetting** member is compared with **GUID_POWERSCHEME_PERSONALITY**. If these two comparisons are true then the **Data** member is confirmed to be about power schemes, and can be extracted at line 41 and further processed. Data about remaining power and current power source

are obtained in similar ways. This software will use three GUIDs to identify the data in the power messages (reference [3]):

- **GUID_BATTERY_PERCENTAGE_REMAINING**
 - The application will receive this message if the computer is using battery
 - The data stores the value of battery percentage remaining
- **GUID_ACDC_POWER_SOURCE**
 - The data value will be 1 if the computer is using battery
 - The data value will be 2 if the computer is using AC power source
- **GUID_POWERSCHEME_PERSONALITY**
 - The data will be GUID
 - The Windows system has three default power scheme: power savings, balance, and performance

2.4 Query the Battery Device Using the Device Management Control Codes

DeviceIoControl function can send a Control Code directly to a device driver, and ask the device driver to perform certain operation. In the project, this function is used to query battery information from the battery device by letting the battery device driver fetch the information. The function is invoked with 8 arguments (reference [6]):

```

326 TCHAR outname[100]; //null-terminated unicode string is an array of TCHAR
327 bqi.InformationLevel = BatteryDeviceName; //now bqi can be used to query battery name
328 if (DeviceIoControl(hBattery,
329     IOCTL_BATTERY_QUERY_INFORMATION,
330     &bqi,
331     sizeof(bqi),
332     outname,
333     sizeof(outname),
334     &dwOut,
335     NULL)) {
336     wstring tmp = outname; //convert the TCHAR array to a wstring object
337     string tmp2(tmp.begin(),tmp.end()); //a wstring object can be converted to a string object
338     this->label29->Text = gcnew String(tmp2.c_str()); //convert to the System::String^ at last
339 }
```

Figure 2-5 Query the Battery Device for Battery Name

1. The handle of a device. **hBattery** is a handle to the battery device. It is similar to a file descriptor and lets the function know which device is to be queried.
2. The Control Code, an integer that specifies what operation the device driver should perform. In this case, **IOCTL_BATTERY_QUERY_INFORMATION** is a Device Management Control code, meaning the device driver should do a query operation.
3. The pointer to an input buffer, a **BATTERY_QUERY_INFORMATION** struct. This struct has a parameter **InformationLevel** and is used to specify what information needs to be fetched in the query operation (reference [7]). At line 327, this parameter is set to **BatteryDeviceName**, meaning the data returned will be name of the battery.
4. The size of the input buffer. The function needs to know about this size in bytes.
5. The output buffer. This output buffer is used to store the returned data. An array of **TCHAR** declared on line 326 is used to store the battery name string.

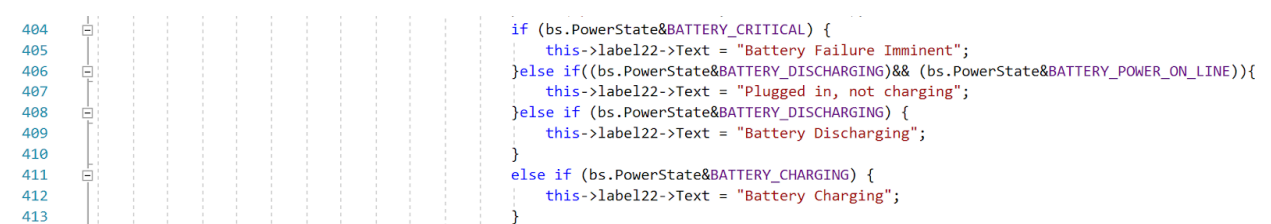
6. The size of the output buffer in bytes.
7. The pointer to an integer. This integer is used to store how many bytes of data are returned into the output buffer.
8. The last parameter is related to overlapped operations, and is not used in the project, so it is left as **NULL**.

The function returns true on successful, so all the data loaded into the output buffer can be processed and displayed on the GUI of the software. Line 336 to 338 converted the data from the output buffer into a string object so that the data type matches what the software can display. This example shows how to query the battery device for the name of the battery. The functionalities to query the battery device for other information are implemented in similar ways. With **bqi.InformationLevel** set to **BatteryInformation** and the output buffer set to a **Battery_Information** struct (reference [8]), the function can query the battery device for information such as type, capacity and history cycles. With the Control Code set to **IOCTL_BATTERY_QUERY_STATUS**, and the output buffer set to a **Battery_STATUS** struct (reference [9]), the function can query the battery device for power status, voltage, and charging rate.

2.5 Analyze Data from Power Management Structures

The **Battery_Information** struct and **BATTERY_STATUS** struct are Power Management structures. They are used as output buffers for the **DeviceIoControl** function. The **Battery_Information** struct stores data such as battery name, chemical type, capacities and history cycles. The **BATTERY_STATUS** struct stores data such as battery power status, charging rate, voltage. If the data is in type of **TCHAR** array, **UCHAR** array, or integers, they are converted to string objects and then displayed on the GUI. If the data is a hexadecimal value, the bitfields in the value needs to be read, to determine what to display.

The **PowerState** member of the **Battery_Status** struct is a hexadecimal value (reference [9]). The 0th bit of **PowerState** indicates if the battery is connected to charger. The 1st bit of **PowerState** indicates if the battery is discharging. The 2nd bit of **PowerState** indicates if the battery is charging. The 3rd bit of the **PowerState** indicates if the battery is having hardware problem. As shown in the following figure, the **BATTERY_POWER_ON_LINE**, the **BATTERY_DISCHARGING**, the **BATTERY_CHARGING**, the **BATTERY_CRITICAL** are constants. Each of these constants is a bitmask for one of the four aforementioned bitfields.



```

404 if (bs.PowerState & BATTERY_CRITICAL) {
405     this->label22->Text = "Battery Failure Imminent";
406 } else if ((bs.PowerState & BATTERY_DISCHARGING) && (bs.PowerState & BATTERY_POWER_ON_LINE)) {
407     this->label22->Text = "Plugged in, not charging";
408 } else if (bs.PowerState & BATTERY_DISCHARGING) {
409     this->label22->Text = "Battery Discharging";
410 }
411 else if (bs.PowerState & BATTERY_CHARGING) {
412     this->label22->Text = "Battery Charging";
413 }

```

Figure 2-6 Identify Battery Power State

Therefore, the bitwise operator **&** is used to read the 3rd bit of **PowerState** at line 404, to check if a battery has hardware problems. Then, similarly, the 1st and the 0th bit are checked to see if the battery is in a "Plugged in, not charging" state. Thereafter, at line 408, if the 1st bit of **PowerState** is set, the battery is in normal discharging state. If all other battery states are not caught, then the normal charging state is caught on line 411 by checking the 2nd bit.

3 Result:

The expectation of this software is to display the information related to the battery, including relevant system power information, battery status data, and battery performance data. The running software is shown on the left side of the following figure.

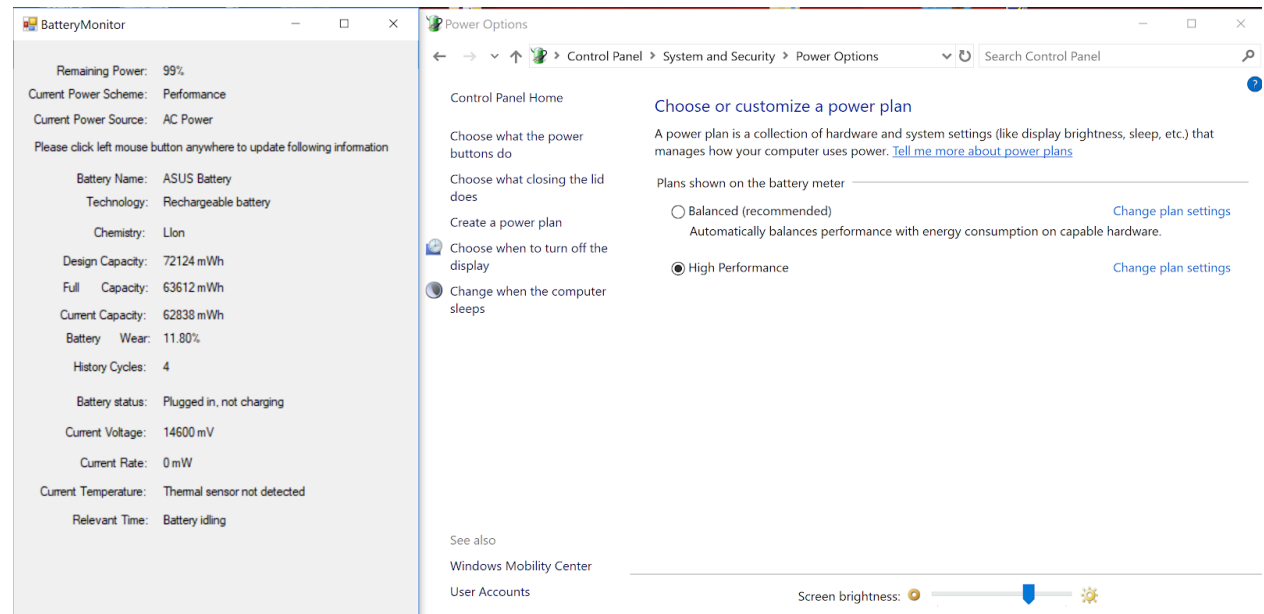


Figure 3-1 Test Run of the Software

Power source:

There are two power sources for every computer: AC (alternating current) and DC (direct current). AC is for the desktop and/or for the laptop. DC is for the laptop when the power source is from battery. The software will receive the system power notifications and display the power source correctly. For the laptop, the power source will be changed if user switch the power source. (i.e. AC for plugging in the charger and AD for plugging off the charger).

Power Scheme:

There are three default power schemes in windows 10: balanced, power saving and performance. The user can change add new power plans via Control Panel\All Control Panel Items\Power Options. This project should identify and display the default currently using power plan. Also, it should be switched if user changed the power plan through the control panel.

Remaining power:

The software will display the remaining power percentage. This will be “this system has no battery” in desktop machines.

Battery Name:

The name of the battery.

Technology:

Either rechargeable or non-rechargeable.

Chemistry:

The type of the battery (reference [8]). The software can identify if a battery is Lead Acid(**PbAc**), Lithium Ion(**LION**), Nickel Cadmium(**NiCd**), Nickel Metal Hydride(**NiMH**), Nickel Zinc(**NiZn**), and Alkaline-Manganese(**RAM**).

Design Capacity:

The capacity that a battery is designed for, with unit in mWh.

Full Capacity:

The capacity of the battery when the current battery is fully charged. This is different from the Design Capacity because the longer time a battery is used, the more unusable capacity it would have. In other words, the usage of a battery may reduce its health, and shrink its Full Capacity. The unit is in mWh.

Current Capacity:

The capacity of the current battery in mWh.

Battery Wear:

The wear of a battery is the ratio of unusable capacity to the Design Capacity of a battery. A laptop may need a new battery if current battery wear is high.

History Cycles:

The number of charge-discharge cycles in history.

Battery Status:

- "Battery Charging" indicates the battery is performing normal charging operation.
- "Battery Discharging" indicates the battery is performing normal discharging operation.
- "Plugged in, not charging" indicates that the battery is not charging despite the fact that AC power is connected to the laptop.
- "Battery Failure Imminent" indicates the battery is not working correctly due to hardware problems.

Current Voltage:

The current voltage supplied to the battery in mV.

Current Rate:

The charging rate or the discharging rate in mW. The discharging rate is negative.

Current Temperature:

The battery temperature is shown in Celsius, if a thermal sensor is available.

Relevant Time:

Display the time to full charge based on current charging rate, or the time to empty based on the current discharging rate.

4 Evaluation and Quality Assurance

4.1 A General View

The battery monitor software is able to display various information about the battery in an laptop, as shown in Figure 4-1 below. Since the software has no complex algorithms, or complex data structures that consume lots of CPU power, the software can run smoothly with minimal CPU usage, usually below 1% CPU usage.

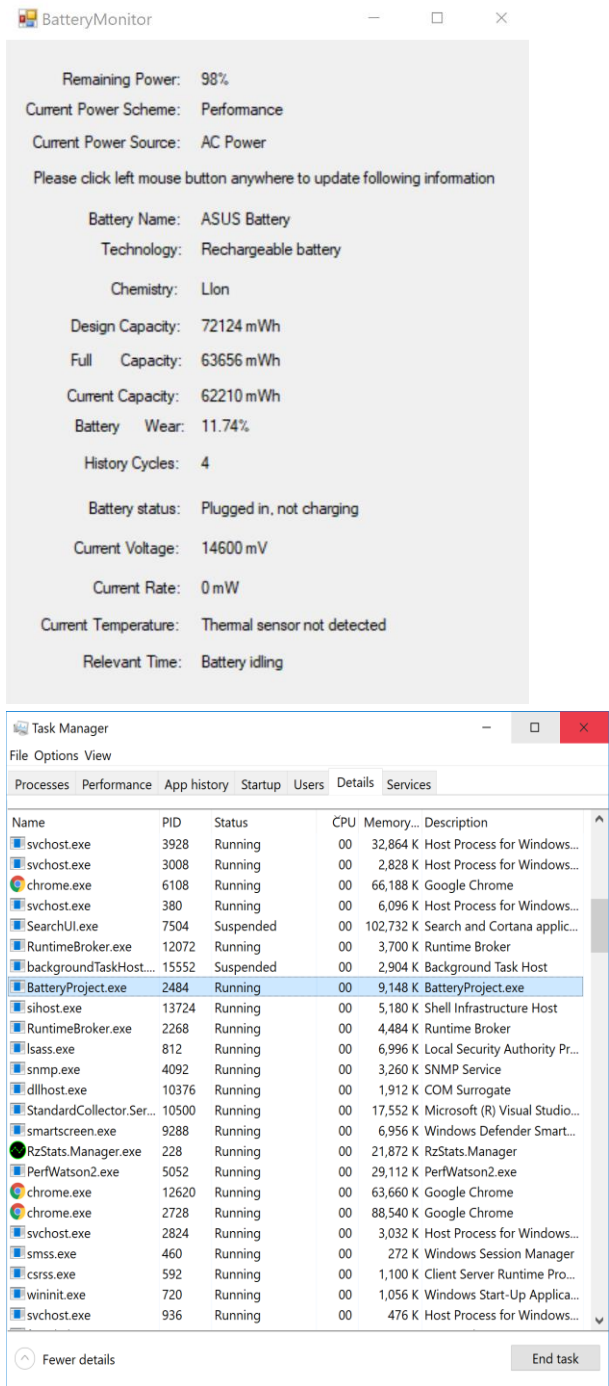


Figure 4-1 Software Test Run

Figure 4-2 CPU Usage

The information displayed by the software has been tested against the inherent features of windows system to ensure its accuracy, including the Battery Power Icon on the right side corner of the Windows Taskbar, the Power Options in the Windows Control Panel, and the Windows Battery Report which can be generated by running this command in a Windows Powershell (reference [10]):
`powercfg /batteryreport /output "C:\battery_report.html"`

The information displayed by the software is real time, however, some information require a user's action to update. The information regarding remaining power, current power scheme, and current power source are updated automatically because they are implemented by processing the incoming system messages. The other information on the GUI are updated manually at the cost of clicking the left mouse button, because they are implemented by querying the battery device of the system. The main thread can passively process the system messages when the messages are received, but cannot constantly query the battery device without blocking the inputs from users. Future work may be building a second thread that can update these information automatically.

The software is able to display accurate information with minimal CPU usage, but some information needs to be updated manually via a mouse click. Next sections will evaluate the software from the perspectives of System Compatibility and Memory Management.

4.2 System Compatibility and Dependencies

The software was tested on 64-bit Windows 10 and Windows 8 systems and no compatibility issue occurred.

The use of dynamically linked library lets the compiled executable have a small size, but this also means the software cannot run without its dependency. The executable is dependent on VC++ redistributable runtime library (reference [11]).

4.3 Memory Management

.NET Framework 4.6.2 was used in the project. The framework provides a garbage collector that automatically cleans up allocated memory in the heap (reference [12]).

Additionally, Visual Leak Detector, a tool available on the Visual Studio Marketplace (reference [13]), has been used to ensure there is no memory leak with the software. The content in the following figure was output from Visual Studio Debugger when test running the software in the debugger.

```
No memory leaks detected.  
Visual Leak Detector is now exiting.  
The program '[8344] BatteryProject.exe' has exited with code 0 (0x0).
```

Figure 4-3 Debugger Output

5 Conclusion

5.1 Summary

The main achievements of this project is creating a software to collect the battery information, and display them to users. The software is capable of parsing system power information from system messages, and querying the battery device for detailed battery information.

5.2 Relevance

This project aligns with the topics such as Inter Process Communication, System Messages, and Devices. The project software is a System Monitor Tool that can receive system messages and query the battery device.

5.3 Future Work

Future work may be using a different thread to update the battery information, to let the software be used more conveniently. Aside from that, a functionality can be added, to create a log file about the battery performance over a certain period.

Contributions of Team Members

Guangen Xin:

- Researched about how to handle Windows Messages
- Researched about how to query a battery device
- Prepared a prototype software that can display battery power
- Added the methods to query the battery device for detailed information

Ming Lei:

- Researched about how to handle Windows Messages
- Researched about how to query a battery device
- Added methods to identify and display the power scheme

Zixuan Zhang:

- Researched about how to register/unregister an application for listening to Windows Messages
- Added methods to identify and display the power source

Reference

[1] "Message Structure Properties" Accessed: Apr.8, 2018
[https://msdn.microsoft.com/en-us/library/system.windows.forms.message\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.windows.forms.message(v=vs.110).aspx)

[2] "RegisterPowerSettingNotification function" Accessed: Apr.9, 2018
[https://msdn.microsoft.com/en-us/library/windows/desktop/aa373196\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa373196(v=vs.85).aspx)

[3] "Power Setting GUIDs" Accessed: Apr.9, 2018
[https://msdn.microsoft.com/en-us/library/windows/desktop/hh448380\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/hh448380(v=vs.85).aspx)

[4] "WM_POWERBROADCAST message" Accessed: Apr.9, 2018
[https://msdn.microsoft.com/en-us/library/aa373247\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa373247(v=vs.85).aspx)

[5] "POWERBROADCAST_SETTING structure" Accessed: Apr.8, 2018
[https://msdn.microsoft.com/en-us/library/windows/desktop/aa372723\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa372723(v=vs.85).aspx)

[6] "IOCTL_BATTERY_QUERY_STATUS control code" Accessed: Apr.8, 2018
[https://msdn.microsoft.com/en-us/library/windows/desktop/aa372699\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa372699(v=vs.85).aspx)

[7] "BATTERY_QUERY_INFORMATION structure" Accessed: Apr.8, 2018
[https://msdn.microsoft.com/en-us/library/windows/desktop/aa372667\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa372667(v=vs.85).aspx)

[8] "BATTERY_INFORMATION structure" Accessed: Apr.9, 2018
[https://msdn.microsoft.com/en-us/library/windows/desktop/aa372661\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa372661(v=vs.85).aspx)

[9] "BATTERY_STATUS structure" Accessed: Apr.9, 2018
[https://msdn.microsoft.com/en-us/library/windows/desktop/aa372671\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa372671(v=vs.85).aspx)

[10] D. RUBINO, "How to generate a Battery Report in Windows 10" Accessed: Apr.9, 2018
<https://www.windowscentral.com/generate-battery-report-windows-10>

[11] "The latest supported Visual C++ downloads" Accessed: Apr.1, 2018
<https://support.microsoft.com/en-ca/help/2977003/the-latest-supported-visual-c-downloads>

[12] R. Petrusha, "Fundamentals of Garbage Collection" Accessed: Apr.1, 2018
<https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals>

[13] A. Shapkin, "Visual Leak Detector for Visual C++" Accessed: Mar.31, 2018
<https://marketplace.visualstudio.com/items?itemName=ArkadyShapkin.VisualStudioLeakDetectorforVisualC>