



System Development Guide

| | |
|--------------------|--|
| Description | AdapTag Development Kit: System Development Guide |
| Date | 2012/12/14 |
| Doc. No. | 8P004-00 |
| Revision | 01 |
| Model Name | S0000AF1T1 |

| | Design Engineering | | |
|--|--------------------|-------|--------|
| | Approval | Check | Design |
| | | | |

龍亭新技股份有限公司 Pervasive Displays, Inc.
 No.71, Delun Rd., Rende Dist., Tainan City 71743, Taiwan (R.O.C.)
 Tel: +886-6-279-5399 / Fax: +886-6-270-5857 / <http://www.pervasivedisplays.com>

Copyright © 2012 by Pervasive Displays, Inc.

Table of Contents

| | |
|--|-----------|
| Revision History | 3 |
| 1. Introduction | 4 |
| 1.1 Overview | 4 |
| 1.2 Kit Contents | 4 |
| 1.3 System Diagram | 5 |
| 2. AdapTag Manager | 6 |
| 2.1 System Architecture | 6 |
| 2.2 The Project of AdapTag Manager | 8 |
| 2.2.1 Development Tools | 8 |
| 2.2.2 Directory Structure | 8 |
| 2.3 The Classes | 9 |
| 2.3.1 Class Block Diagram | 9 |
| 2.4 form_main.cs | 10 |
| 2.4.1 Methods | 10 |
| 2.4.2 Events | 11 |
| 2.5 AdapTag_API.cs | 13 |
| 2.5.1 Structures | 13 |
| 2.5.2 Enumerations | 15 |
| 2.5.3 Classes | 16 |
| 2.6 SerialPort.cs | 20 |
| 2.6.1 Properties | 20 |
| 2.6.2 Methods | 21 |
| 2.6.3 Create Auto Receiving Data Event (Thread) | 22 |
| 2.7 SendSystemPacket() | 23 |
| 2.8 The Functional Flowchart of Sending Image Data | 25 |
| 2.8.1 Start with AdapTag Kit | 25 |
| 2.8.2 Send Image Data to Slave | 26 |
| 3. System Packet | 28 |
| 3.1 System Packet Structure | 28 |
| 3.2 Command Type | 29 |
| 4. REFERENCES | 31 |
| Glossary of Acronyms | 32 |

Revision History

| Version | Date | Page (New) | Section | Description |
|---------|------------|------------|---------|-------------|
| 01 | 2012/12/14 | All | All | First draft |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

1. Introduction





1.1 Overview


The AdapTag Development kit provides a working reference design for software development of sub 1GHz RF applications based on the TI CC430 MCU and the 1.44", 2" and 2.7" EPDs from Pervasive Displays. The AdapTag development kit consists of one AdapTag host connected to an EZ430 USB dongle and three AdapTag slave boards each connected to an EPD.

The purpose of this document is to give an overview of the AdapTag Development kit. This document also serves as an introduction of AdapTag Manager to develop system working with AdapTag development kit. It applies to PDI's 1.44", 2", and 2.7" EPDs. All of the source codes are licensed under the [Apache License, Version 2.0](#) (the "License"). You may not use the code except in compliance with the License.

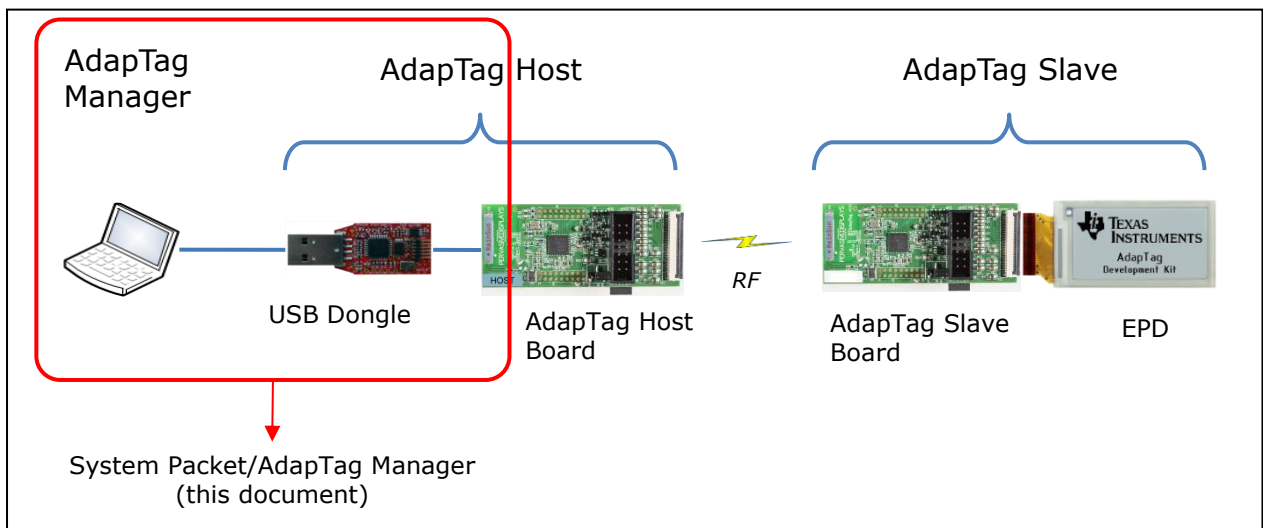
For more introductions on the COG driver, please refer to PDI's "E-paper Display COG Driver Interface Timing" which document number is 4P008-00 [9] and "AdapTag Development Kit COG Driver Programming Guide" which document number is 8P003-00 [5].

1.2 Kit Contents

| Name | Photograph | Note |
|--------------------------|--|---|
| AdapTag Host board |  | 1 piece. marked with blue label "HOST" |
| AdapTag Slave board |  | 3 pieces. marked with white labels with 4 digits number (Slave ID) "1401", "2001" and "2701" |
| eZ430 USB Dongle |  | 1 piece. |
| CR2450 Coin Cell Battery |  | 3 pieces. |

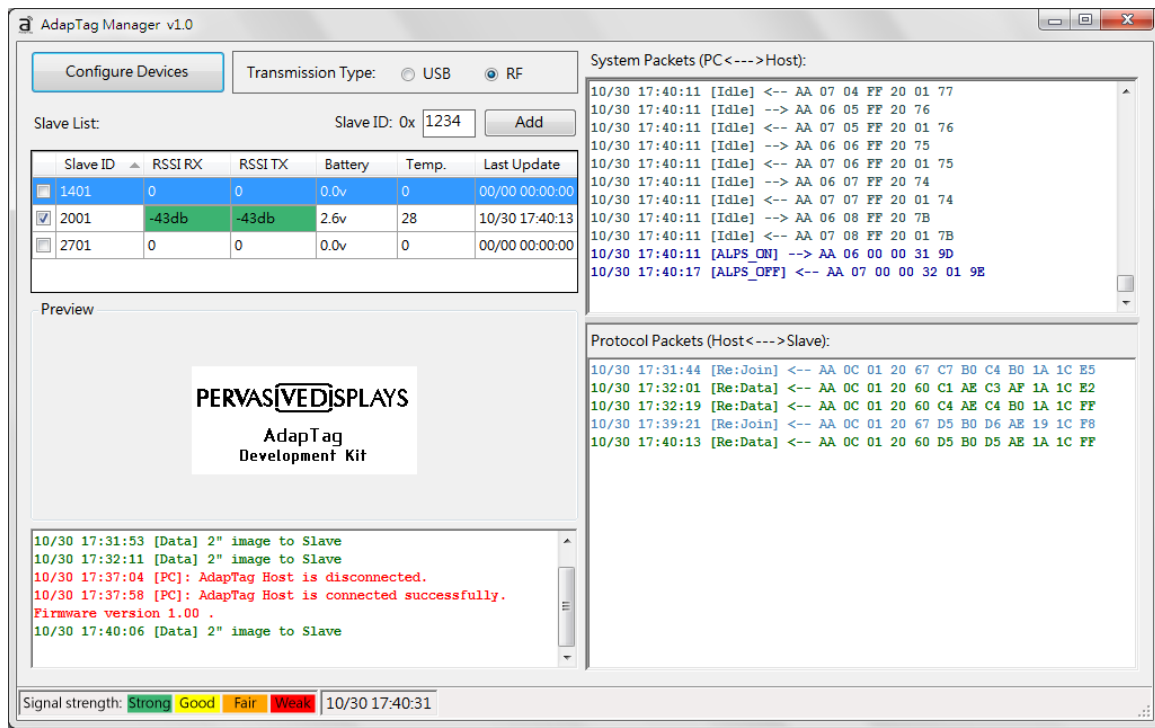
| | | |
|----------------------------------|--|---|
| <p>1.44", 2" and 2.7" EPD</p> |  | <p>3 pieces. 1 piece of each panel size</p> |
| <p>AdapTag Manager installer</p> | <p>Install AdapTag Manager software working with AdapTag kit</p> | |

1.3 System Diagram



2. AdpaTag Manager

This section will introduce the software – AdapTag Manager which is provided source code of AdapTag development kit.

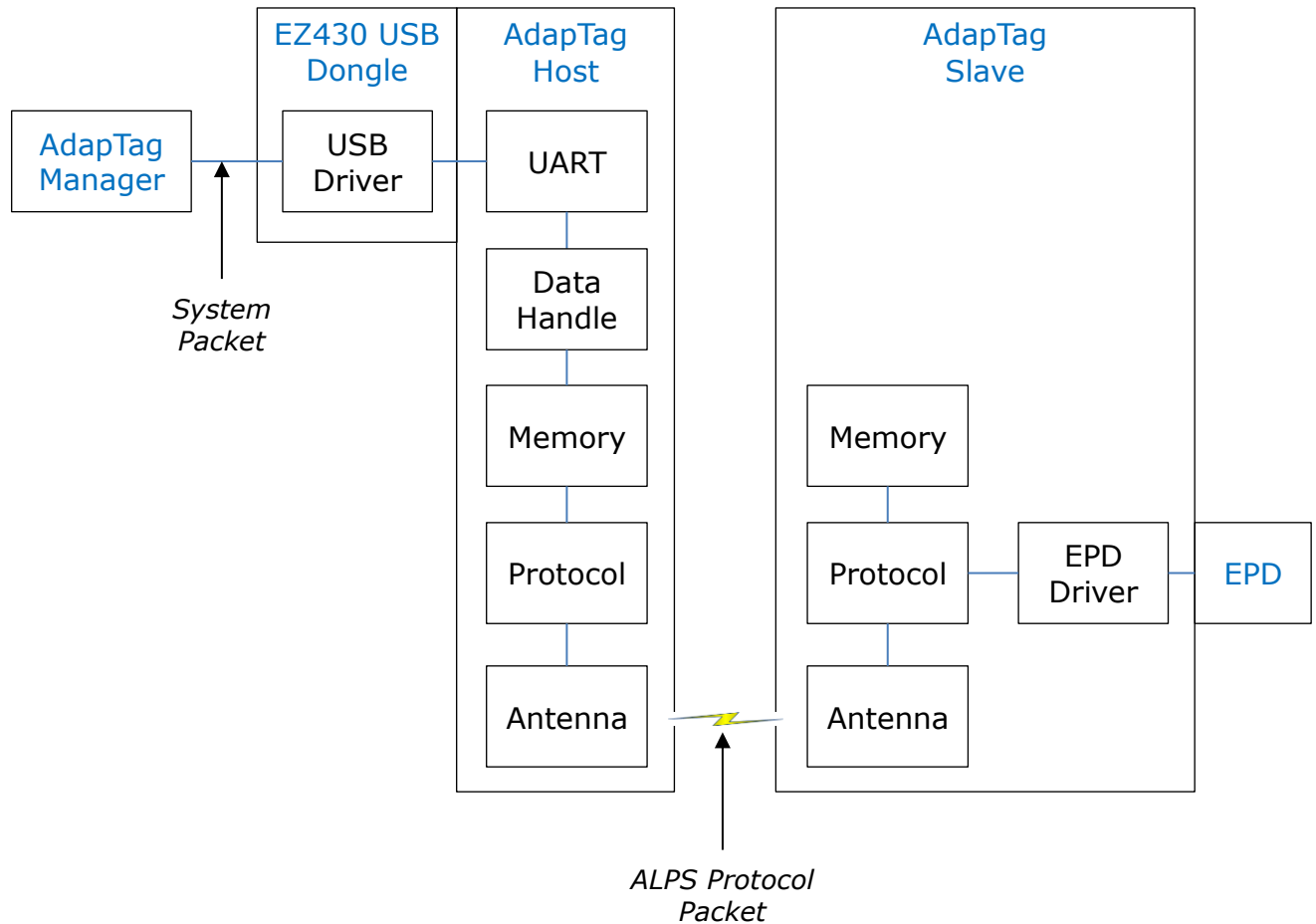


2.1 System Architecture

The diagram below describes the architecture of this kit.

At the host side, the AdapTag Manager sends System Packets to AdapTag host through the EZ430 USB dongle which is the interface from USB to UART. The UART interface has been provided in the firmware to make it possible for a proper machine to write the initialization values into the CC430 securely. When AdapTag Manager sends System Packets to AdapTag host, the Data Handle unit will form the packets as ALPS time slot format and put into memory. The Protocol unit will wait for a complete Frame is established then start transmitting the Frame data in memory to slaves.

At the slave side, the slave continuously operates in a series of Frame Cycle to recognize Protocol Packets. The Protocol unit is based on the received packets to do the defined operation accordingly and returns appropriate packets to host. The EPD Driver includes ADC and COG Driver. When the packets are used to carry image data, AdapTag slave will call EPD driver function to update the EPD.



This document will focus on AdapTag Manager and System Packet that communicates between computer and AdapTag Host.

For more information about how to work with the AdapTag Manager, please refer to AdapTag User Manual [4].

For more information about the protocol or wireless communication, please refer to [7].

2.2 The Project of AdapTag Manager

2.2.1 Development Tools

- [1]** AdapTag Manager:
 Developed by Microsoft® C# language using Microsoft® Visual Studio® version 2010 (VS.NET).
 AdapTag Manager is the GUI working with AdapTag Development Kit. It is also defined System Packet and functions that communicates with AdapTag Host.

2.2.2 Directory Structure

Switch to the project folder in VS.NET. You will find the directory structure as below:

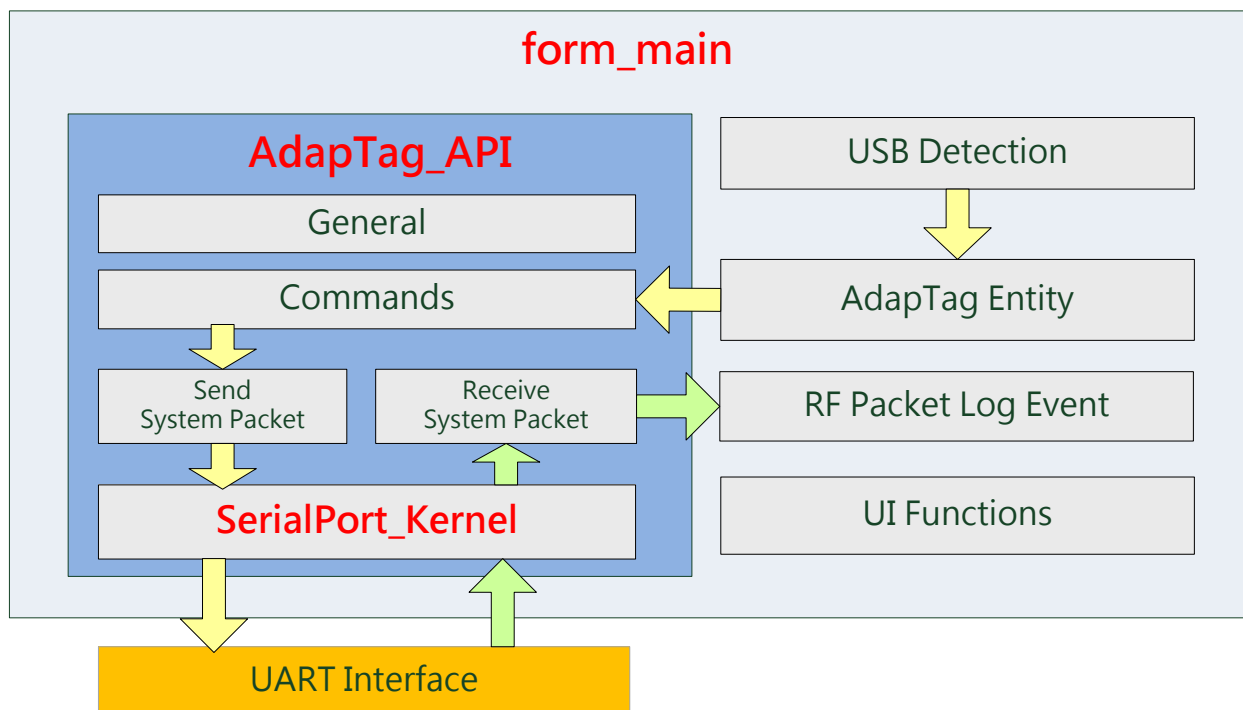
| | | |
|--|----------------------------|--|
| | bin directory | The bin directory is the directory that final output binaries and any dependencies or other deployable files will be written to. |
| | obj directory | The obj directory is for intermediate object files and other transient data files that are generated by the compiler or build system during a build. |
| | AdapTag Manager.sln | This is a structure for organizing projects in Visual Studio. <u>Please double click on this file to start this project.</u> You can also start Visual Studio, select [File]/[Open Project] from top menu and switch to this path selecting this file. |
| | form_main | The main form of this project. |
| | form_report | The report form of this project to show the test results if enable debug. |
| | form_update | The "Configure Devices" form of this project |
| | program.cs | The main entry point of the application |
| | SerialPort.cs | The library of serial port communication with host using windows kernel32.dll |
| | AdapTag_API .cs | AdapTag_API.cs contains a collection of software library which provides users implement AdapTag RF Protocol and some functions. |

2.3 The Classes

The name space of AdapTag Manager project is "AdapTag". There are three important files form the AdapTag Manager software which are:

1. **form_main.cs**: the main graphic user interface. Its class name is form_main.
2. **AdapTag_API.cs**: includes AdapTag kit system packet protocol and function library. Its class name is AdapTag_API.
3. **SerialPort.cs**: the communication interface between AdapTag API and UART interface of host/slave. Its class name is SerialPort_Kernel.

2.3.1 Class Block Diagram



When an AdapTag device connects with computer's USB port successfully, an AdapTag Entity is created and starts communicating with the AdapTag device by UART interface. In the AdapTag_API, it provides system commands to send address, poll, data or others. It also receives the packets to trigger the corresponding events and logs. All of the packet traffic goes through the SerialPort_Kernel unit.

The System Packets and Commands are predefined in AdapTag Manager. User can base on defined System Packet and command to develop own control software without modifying the firmware of AdapTag kit. If user wants to change or add any of the packet formats or command types, user must have the software source code of AdapTag Manager and the firmware source code of AdapTag host and slave to modify together. The packet format between System Packet and Protocol Packet must correspond.

2.4 form_main.cs

2.4.1 Methods

| Method | Return | Input Argument | Description |
|----------------------------|--------|--------------------|--|
| <i>Connect</i> | - | - | Use AdapTag API to establish the connection with AdapTag device and creates related events |
| <i>Disconnect</i> | - | - | Disconnect with AdapTag device |
| <i>OnData</i> | - | - | Finished sending data packets to host. Always follows ALPS_ON or HostShow. |
| <i>OnReadDeviceID</i> | - | structSystemPacket | Get device's ID |
| <i>OnReadVersion</i> | - | structSystemPacket | Get device's firmware version |
| <i>OnReadSlotTime</i> | - | structSystemPacket | Get the duration of time slot. |
| <i>OnHostShowON</i> | - | - | Update content on EPD by host via USB |
| <i>OnALPS_ON</i> | - | - | Start ALPS radio communication |
| <i>OnALPS_OFF</i> | - | - | Check whether retry or repeat |
| <i>OnAddressClear</i> | - | - | Send Poll or Data to host |
| <i>OnRespondPoll</i> | - | structSystemPacket | Update slave's health |
| <i>OnDataFinished</i> | - | structSystemPacket | Update slave's health |
| <i>OnRespondJoinPacket</i> | - | structSystemPacket | Check managed or unknown slave. Update slave's health |
| <i>OnSlaveHealth</i> | - | structSystemPacket | Update slave's health |
| <i>OnAddressCopyPacket</i> | - | - | Copy one slave's image |

| | | | |
|------------------------|-------------|------------------|--|
| | | | and EPD size |
| <i>SendDummyPacket</i> | - | int count | Send dummy (Idle) packet |
| <i>SendSlaveSub</i> | <i>Bool</i> | string imageFile | Decide sending Idle or Address packet to host or slave via RF or USB |

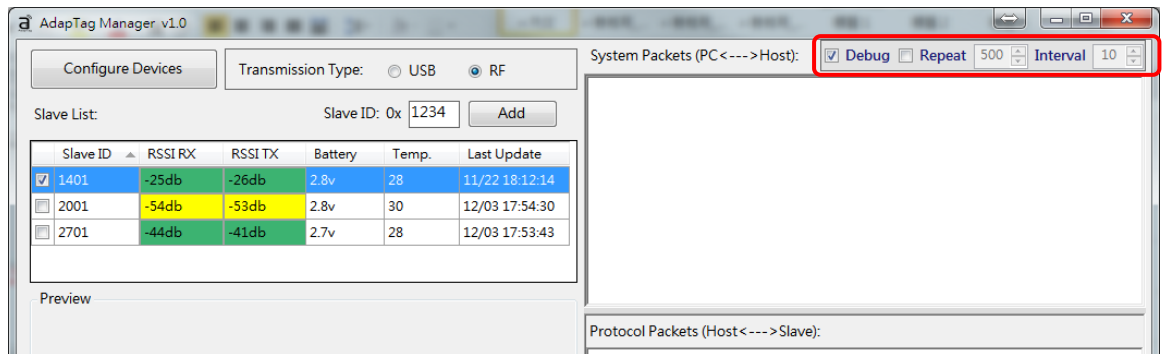
✧ Detecting USB Dongle

| Method | Return | Input Argument | Description |
|--------------------------------|---------------------------|----------------|--|
| <i>WndProc</i> | - | Ref Message m | Overriding the WndProc method to handle OS messages identified in the Message structure which is used for detecting connect or disconnect USB dongle |
| <i>OnDeviceChange</i> | - | Ref Message m | Synchronize the device message and current thread |
| <i>OnDeviceChangeSub</i> | - | Int wParam | Get message and number to handle the plug in or poll out operation |
| <i>RegisterHidNotification</i> | - | - | Registers for getting notification for new messages. |
| <i>FindDongles</i> | <i>List<String></i> | - | Read PID and VID value of the Config.xml file (at AdapTag Manager execution folder) to find AdapTag device |

2.4.2 Events

| Event | Return | Input Argument | Description |
|-----------------------|--------|----------------|--|
| <i>OnConnected</i> | - | Object sender | The event when connected successfully |
| <i>OnDisconnected</i> | - | Object sender | The event when disconnected successfully |
| <i>OnRFPacketLog</i> | - | Object sender | When any packet returns from host or slave, this |

| | | | |
|-----------------------|---|----------------|--|
| | | | event will be triggered to get the system packets |
| <i>OnPC_Host</i> | - | Object Message | The event is triggered on the communication message between computer and host |
| <i>OnHost_Slave</i> | - | Object Message | The event is triggered on the communication message between host and slave |
| <i>OnErrorHandler</i> | - | Object Message | The error handler event when using hotkey "Alt+R" and checked the "Debug" box. It uses for debugging message and repeating sending image data. See the screen capture below. |



2.5 AdapTag_API.cs

2.5.1 Structures

This section consists of the following parts:

| Structure | Description |
|--------------------------------------|--|
| <code>structSystemPacket</code> | The structure of system packet |
| <code>structImageInfo</code> | The structure of image header which is inserted at the beginning of image data packets |
| <code>structCopyAddressPacket</code> | The structure of copying one slave's ID to show the same image |
| <code>StructCOM_Info</code> | The Vendor ID (VID) and Product ID (PID) of COM device |

✧ **structSystemPacket** Structure

For more information on System Packets, please refer to Section 3: System Packet.

```
public struct structSystemPacket
{
    public byte Header;
    public byte PacketLength;
    public UInt16 Address;
    public byte CommandType;
    public byte[] Data;
    public byte CRC;
}
```

- Parameter Details

| Name | Type | Description |
|---------------------|--------|--|
| <i>Header</i> | byte | Header of system packet |
| <i>PacketLength</i> | byte | The length of system packet |
| <i>Address</i> | UInt16 | Target slave ID |
| <i>CommandType</i> | byte | Command type |
| <i>Data</i> | byte[] | Payload for carrying address or image data |
| <i>CRC</i> | byte | Checksum |

✧ **structImageInfo** Structure

For more information on Image Header, please refer to [5] the Insert Image Header section.

```
public struct structImageInfo
{
    public UInt16 SlaveID;
    public byte State;
    public byte PanelType;
    public UInt16 HORIZONTAL;
    public UInt16 VERTICAL;
}
```

- Parameter Details

| Name | Type | Description |
|-------------------|--------|--|
| <i>SlaveID</i> | UInt16 | Target slave ID |
| <i>State</i> | byte | The image state, default is 0xFF means new image |
| <i>PanelType</i> | byte | EPD size. Refer to <code>enumPanelSize</code> 0=1.44", 1=2", 2=2.7" |
| <i>HORIZONTAL</i> | UInt16 | The width of image data |
| <i>VERTICAL</i> | UInt16 | The height of image data |

◇ **structCopyAddressPacket** Structure

This is used for there is an image was sent to host and the other slaves are going to show the same image. It will copy the slave's ID and load the stored image to update EPD.

```
public struct structCopyAddressPacket
{
    public UInt16 SlaveID;
    public byte State;
    public byte PanelType;
    public UInt16 CopyID;
}
```

- Parameter Details

| Name | Type | Description |
|------------------|--------|--|
| <i>SlaveID</i> | UInt16 | Target slave ID |
| <i>State</i> | byte | The image state, default is 0xFF means new image |
| <i>PanelType</i> | byte | EPD size. Refer to <code>enumPanelSize</code> |

| | | |
|---------------|--------|--|
| | | 0=1.44", 1=2", 2=2.7" |
| <i>CopyID</i> | UInt16 | The slave ID of the image that will update |

✧ **StructCOM Info Structure**

To recognize the connected device is AdapTag host, slave or not.

```
public struct structCOM_Info
{
    public string VID;
    public string PID;
}
```

- Parameter Details

| Name | Type | Description |
|------------|--------|------------------------------|
| <i>VID</i> | String | The Vender ID of COM device |
| <i>PID</i> | String | The Product ID of COM device |

2.5.2 Enumerations

This section consists of the following parts:

| Enumeration | Description |
|----------------------------|---|
| <code>enumPanelSize</code> | List EPD sizes |
| <code>enumDirection</code> | List the direction of communication message |

✧ **enumPanelSize Enumeration**

```
public enum enumDeviceSize
{
    EPD144,
    EPD20,
    EPD27
};
```

- Parameter Details

| Value | Description |
|---------------|---------------|
| <i>EPD144</i> | 1.44 inch EPD |
| <i>EPD20</i> | 2 inch EPD |
| <i>EPD27</i> | 2.7 inch EPD |

✧ **enumDirection** Enumeration

```
public enum enumDirection
{
    PC_Host,
    Host_PC,
    Host_Slave,
    Slave_Host
};
```

- Parameter Details

| Value | Description |
|-------------------|---------------------------------------|
| <i>PC_Host</i> | Message from computer to host (P → H) |
| <i>Host_PC</i> | Message from host to computer (P ← H) |
| <i>Host_Slave</i> | Message from host to slave (H → S) |
| <i>Slave_Host</i> | Message from slave to host (H ← S) |

2.5.3 Classes

This section consists of the following parts:

| Class | Description |
|---------------|--|
| AdapTag_API | The major class of this project in AdapTag_API.cs |
| Protocol | The command type of System Packet protocol |
| ProtocolAlias | The alias name of System Packet command type |
| Win32 | Define the constants and function of kernel32.dll to detect plug-in or poll-out the USB dongle |

✧ **AdapTag API Class**

- Properties

| Property | Type | Description |
|----------------|--------|--------------------------------|
| <i>COMport</i> | string | Serial (COM) port information |
| <i>IsOpen</i> | bool | Check the state of serial port |

- Events

| Event | Description |
|-------|-------------|
|-------|-------------|

| | |
|-----------------------|--|
| <i>OnRFPacketLog</i> | When any packet returns from slave, the host will trigger this event to get the system packets |
| <i>OnConnected</i> | The event is triggered on success connecting AdapTag_API |
| <i>OnDisconnected</i> | The event is triggered on success disconnecting AdapTag_API |
| <i>OnPC_Host</i> | The event is triggered on the communication message between computer and host |
| <i>OnHost_Slave</i> | The event is triggered on the communication message between host and slave |
| <i>OnErrorHandler</i> | The event is triggered on the internal error of AdapTag_API |

- Methods

| Method | Return | Input Argument | Description |
|-------------------------------|---------------|--|--|
| <i>DetectUSB_COMport</i> | <i>string</i> | StructCOM_Info comInfo | Get the name of serial ports |
| <i>ConnectCOMport</i> | <i>bool</i> | string COMname | Connect with the selected COM port |
| <i>DisconnectCOMport</i> | - | - | Disconnect with the selected COM port |
| <i>SendIdlePacket</i> | - | UInt16 SlaveID | Send Idle packet by given slave ID |
| <i>SendPollPacket</i> | - | UInt16 SlaveID | Send Poll packet by given slave ID |
| <i>SendDataPacket</i> | - | UInt16 SlaveID | Start sending image data to host |
| <i>SendAddressPacket</i> | - | UInt16 SlaveID string ImageFile enumPanelSize size | Send data by given slave ID, image file and EPD size for address packets |
| <i>SendCopyAddressPacket</i> | - | UInt16 SlaveID UInt16 CopyID enumPanelSize size | Send image data to given slave ID, copy one slave's image and EPD size |
| <i>SendClearAddressPacket</i> | - | - | Clear address list on host |

| | | | |
|------------------------------|---------------|---|--|
| <i>EnableDebug</i> | - | bool enable | Show detail protocol packet information |
| <i>StartALPS</i> | - | - | Start ALPS radio communication after commands are sent to host |
| <i>HostShow</i> | - | - | Update content on EPD by host via USB |
| <i>ResetUSB</i> | - | - | Reset the COM Device |
| <i>ConvertUshort</i> | <i>ushort</i> | byte Hbyte byte Lbyte | Combine two Bytes to an Unsigned Integer |
| <i>ConvertPacketToString</i> | <i>string</i> | structPacket packet | Convert Packet data to String |
| <i>ConvertStringToUshort</i> | <i>ushort</i> | string sourceString | Convert String to Unsigned Integer |
| <i>ConvertUshortToString</i> | <i>string</i> | ushort sourceUshort | Convert Unsigned Integer to String |
| <i>ConvertImageTo1bit</i> | <i>Bitmap</i> | string ImageFilePath string saveFilePath enumPanelSize Size | Convert any image to 1 bit bitmap format by given file path and EPD size |
| <i>ReadImage</i> | <i>Image</i> | string imageFile | Read an image to memory |
| <i>ReadNetworkID</i> | - | - | Read network ID from device |
| <i>WriteNetworkID</i> | - | ushort networkID | Write network ID to device |
| <i>ReadSlaveID</i> | - | - | Read slave ID from device |
| <i>WriteSlaveID</i> | - | ushort slaveID | Write slave ID to device |
| <i>ReadChannel</i> | - | - | Read channel ID from device |
| <i>WriteChannel</i> | - | byte channel | Write channel ID to device |
| <i>ReadTX_Power</i> | - | - | Read Tx power from device |
| <i>WriteTX_Power</i> | - | byte power | Write Tx power from 0 to 3 to device where 0=-6, 1=0, 2=5 and 3=10 |
| <i>ReadEncryptKey</i> | - | - | Read encryption key from device |

| | | | |
|------------------------|---|------------|---|
| <i>WriteEncryptKey</i> | - | string key | Write encryption key to device |
| <i>ReadSlotTime</i> | - | - | Read the duration of time slot from device |
| <i>WriteSlotTime</i> | - | byte time | Write the duration of time slot from 0 to 3 to device where 0=0.5, 1=1, 2=2 and 3=4 |
| <i>ReadFW_Version</i> | - | - | Read firmware version from device |

✧ **Protocol and ProtocolAlias Class**

Please refer to section 3: System Packet.

✧ **Win32 Class**

```
public const int WM_DEVICECHANGE = 0x0219;
public const int DBT_DEVICEARRIVAL = 0x8000, DBT_DEVICEREMOVECOMPLETE = 0x8004;
public const int DEVICE_NOTIFY_WINDOW_HANDLE = 0;
public const int DEVICE_NOTIFY_SERVICE_HANDLE = 1;
public const int DBT_DEVTYP_DEVICEINTERFACE = 5;
public const int DBT_DEVNODES_CHANGED = 7;

public static Guid GUID_DEVINTERFACE_HID = new Guid("4D1E55B2-F16F-11CF-88CB-00111000030");
public static Guid GUID_DEVINTERFACE_USB_DEVICE = new
Guid("A5DCBF10-6530-11D2-901F-00C04FB951ED");
[StructLayout(LayoutKind.Sequential)]
public class DEV_BROADCAST_DEVICEINTERFACE
{
    public int dbcc_size;
    public int dbcc_devicetype;
    public int dbcc_reserved;
    public Guid dbcc_classguid;
    public short dbcc_name;
}
[DllImport("user32.dll", SetLastError = true)]
public static extern IntPtr RegisterDeviceNotification(
IntPtr hRecipient,
IntPtr NotificationFilter,
Int32 Flags);
[DllImport("kernel32.dll")]
public static extern int GetLastError();
```

2.6 SerialPort.cs

For the serial port communication, the AdapTag Manger doesn't use .NET framework library: "System.IO.Ports.SerialPort". It uses Windows Kernel32.dll which is the 32-bit dynamic link library found in the Windows operating system kernel. It handles memory management, input/output operations, and interrupts. When Windows boots up, kernel32.dll is loaded into a protected memory space so other applications do not take that space over. It will reduce the unknown hardware error issues when using .NET framework library: "System.IO.Ports.SerialPort".

This section introduces the SerialPort_Kernel class, properties, methods and how to use it.

For more information on the constants, variables and functions of Kernel32.dll, please refer to the code #region Kernel32.dll in SerialPort.cs file.

2.6.1 Properties

| Property | Type | Description |
|-----------------------|------|---|
| <i>IsOpen</i> | bool | Indicating the open or closed status of the SerialPort object (COM port) |
| <i>RtsEnable</i> | bool | Whether the Request to Send (RTS) signal is enabled during serial communication |
| <i>DtrEnable</i> | bool | Enables the Data Terminal Ready (DTR) signal during serial communication |
| <i>DiscardNull</i> | bool | Indicating whether null bytes are ignored when transmitted between the port and the receive buffer |
| <i>DataBits</i> | byte | The standard length of data bits per byte (default is 8 bits) |
| <i>Parity</i> | byte | The length of parity-checking protocol per byte (default is 0, 0-4=no, odd, even, mark and space) |
| <i>StopBits</i> | byte | The standard number of stop bits per byte (default is 0, 0,1,2 = 1, 1.5, 2) |
| <i>BaudRate</i> | int | The serial baud rate (default is 9600) |
| <i>ReadBufferSize</i> | int | The size of the SerialPort input buffer (default is 4096) |
| <i>ReadTimeout</i> | int | The number of milliseconds before a time-out occurs when a read operation does not finish (default is 10) |

| | | |
|------------------------|--------|--|
| <i>WriteBufferSize</i> | int | The size of the serial port output buffer (default is 2048) |
| <i>WriteTimeout</i> | int | The number of milliseconds before a time-out occurs when a write operation does not finish (default is 2000) |
| <i>PortName</i> | string | The port for communications, including but not limited to all available COM ports |

2.6.2 Methods

| Method | Return | Input Argument | Description |
|-------------------------|----------|---|---|
| <i>Open</i> | - | - | Opens a new serial port connection |
| <i>Close</i> | - | - | Closes the port connection |
| <i>Dispose</i> | - | - | The same as Close() |
| <i>Read</i> | byte [] | - | Reads data from the serial port input buffer |
| <i>Write</i> | int | byte[] buffer int offset, int count | Writes data to the serial port output buffer and assign offset and quantity of the buffer |
| <i>Write</i> | int | byte[] WriteBytes | Use the PID and VID in Config.xml file to find AdapTag device |
| <i>DiscardInBuffer</i> | - | - | Discards data from the serial driver's receive buffer |
| <i>DiscardOutBuffer</i> | - | - | Discards data from the serial driver's transmit buffer |
| <i>DiscardBuffer</i> | - | - | Discards data from all the serial driver's buffer |
| <i>Reset</i> | bool | - | Reset the connected COM port |
| <i>GetPortNames</i> | string[] | - | Gets an array of serial port names for the current computer |
| <i>GetPortNames</i> | string[] | string VID string PID | Gets an array of serial port names for the current computer by VID and PID |

2.6.3 Create Auto Receiving Data Event (Thread)

✧ Declare Events

```
public delegate void SerialDataReceivedEventHandler(object receiveData);
public delegate void SerialPortErrorEventHandler(object errorMessage);
public event SerialDataReceivedEventHandler DataReceived;
public event SerialPortErrorEventHandler ErrorReceived;
```

SerialDataReceivedEventHandler: Represents the method that will handle the DataReceived event of a SerialPort object

SerialPortErrorEventHandler: Represents the method that will handle the error event of a SerialPort object

✧ Declare Thread and Interrupt

```
private Thread threadRead = null;
private bool IsRead = false;
```

✧ Create Thread

```
private void InitThread()
{
    try
    {
        IsRead = true;
        threadRead = new Thread(new ThreadStart(threadReadSub));
        threadRead.IsBackground = true;
        threadRead.Priority = ThreadPriority.Lowest;
        threadRead.Start();
    }
    catch (Exception exp) { if (ErrorReceived != null)
ErrorReceived(exp.Message); }
}

private void threadReadSub()
{
    while (IsRead)
    {
        try
        {
            byte[] data = Read();
            if (data != null && data.Length > 0)
            {
                if (DataReceived != null) { DataReceived(data); }
            }
        }
        catch (Exception exp) { if (ErrorReceived != null)
ErrorReceived(exp.Message); }
    }
}
```

2.7 SendSystemPacket()

The `SendSystemPacket()` method (in `AdapTag_API.cs` file) includes sending Idle packet, Poll packet, Address packet, Data packet and other command packet.

```
void SendSystemPacket (UInt16 Address, byte cmd, byte[] payload, bool Start)
```

| Input Argument | Description |
|----------------|--|
| Address | Slave ID. 2 bytes. If the following command is without Slave ID, the input value is 0. |
| cmd | System Packet's Command Type. 1 byte. For more information on the command type, please refer to section 3.2: Command Type. The input argument is command type ID which is defined in "Protocol" class in <code>AdapTag_API.cs</code> |
| payload | The data of System Packets. 6 - 64 bytes. If the command type is without data, the input argument is Null. The payload length will set to 6. |
| Start | Start thread to send System Packet or not |

The `SendSystemPacket()` will convert the input arguments to byte array as System Packet structure (bs) (section 3.1) and add the byte array to queue. Use Lock to make sure the data is completed.

```
lock (queSend.SyncRoot) queSend.Enqueue (bs);
//If the thread is Sleep or Join, the thread is blocked. ThreadReceive.Interrupt()
to unblock.
if ( (ThreadSend.ThreadState & System.Threading.ThreadState.WaitSleepJoin) ==
System.Threading.ThreadState.WaitSleepJoin)
{
    if (Start) ThreadSend.Interrupt();
}
```

✧ SendAddressPacket

```
public struct ImageInfo SendAddressPacket (UInt16 SlaveID, string ImageFile, enum PanelSize size)
```

In this method, it will convert `ImageFile` to 1 bit bitmap image format first. The `GenSendBytes(string BMPfile)` function provides the approach to generate the image data bytes that COG needs. (Refer to [5] section 2 for understanding how to generate the image data bytes.)

All of the image data bytes will save to `SendMatrix` byte array. This byte array will use for `SendDataPacket` method later to send image data follows Address packet.

When sending Address packet in System Packet structure, the two bytes Slave ID

will use for the amount of image data bytes.

```
SendSystemPacket((ushort)SendMatrix.Count, Protocol.Address, headbytes, true);
```

The Image Header (see [5] section 2.5) will be got and returned (`imageInfo`) in this method including Slave ID and panel size information in order to have the target slave updating on correct EPD content. The `StructSerialize()` will convert the image header to byte array as `headbytes`.

✧ SendCopyAddressPacket

```
public void SendCopyAddressPacket(UInt16 SlaveID, UInt16 CopyID, enumPanelSize size)
```

This method uses for a Slave (`SlaveID`) to update EPD screen but use assigned Slave's (`CopyID`) image data. Any Slave uses this methods will show the same image as copied Slave on EPD.

✧ SendDataPacket

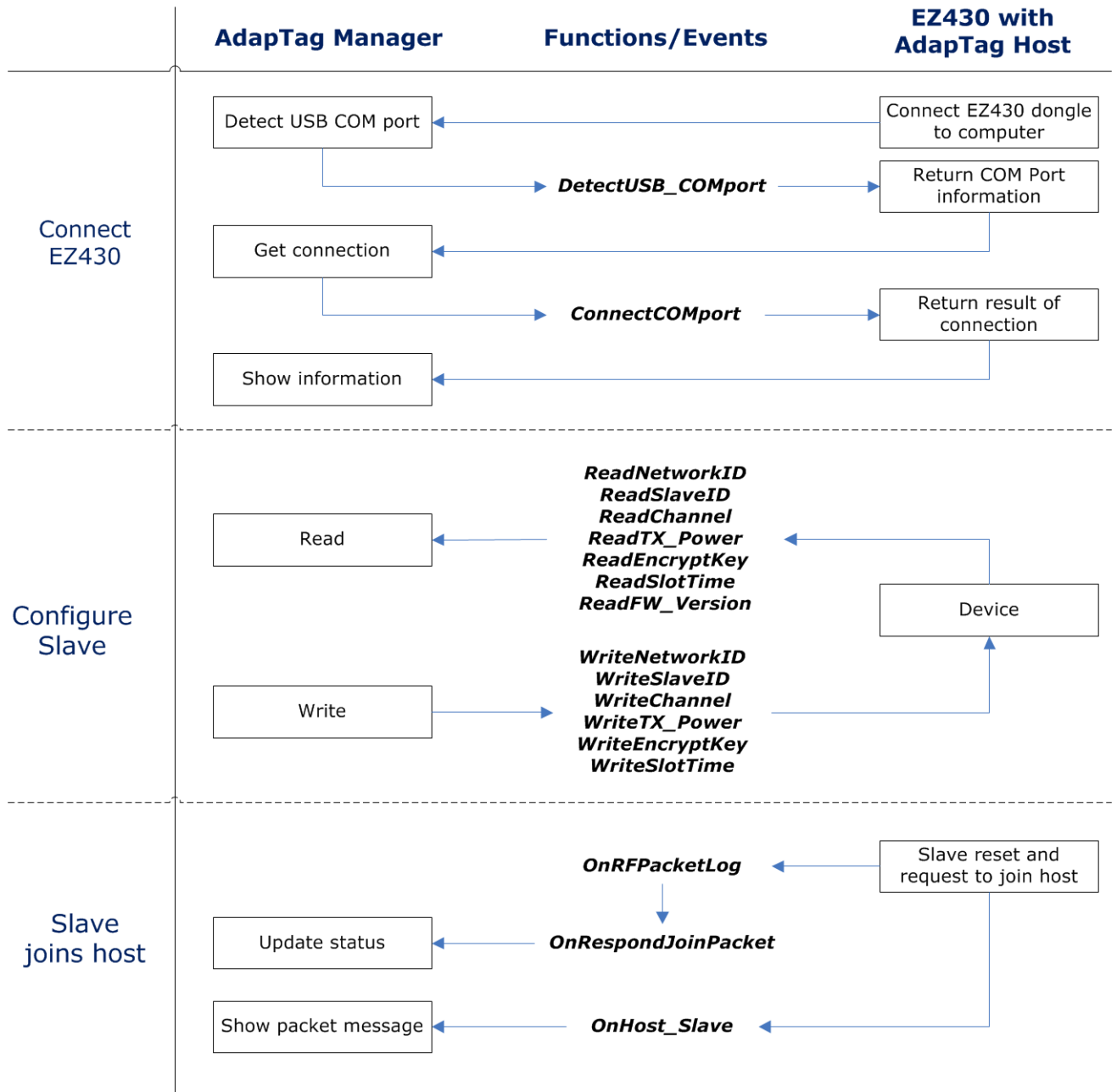
```
public void SendDataPacket(UInt16 SlaveID)
```

This method will send each image data byte in `SendMatrix` byte array.

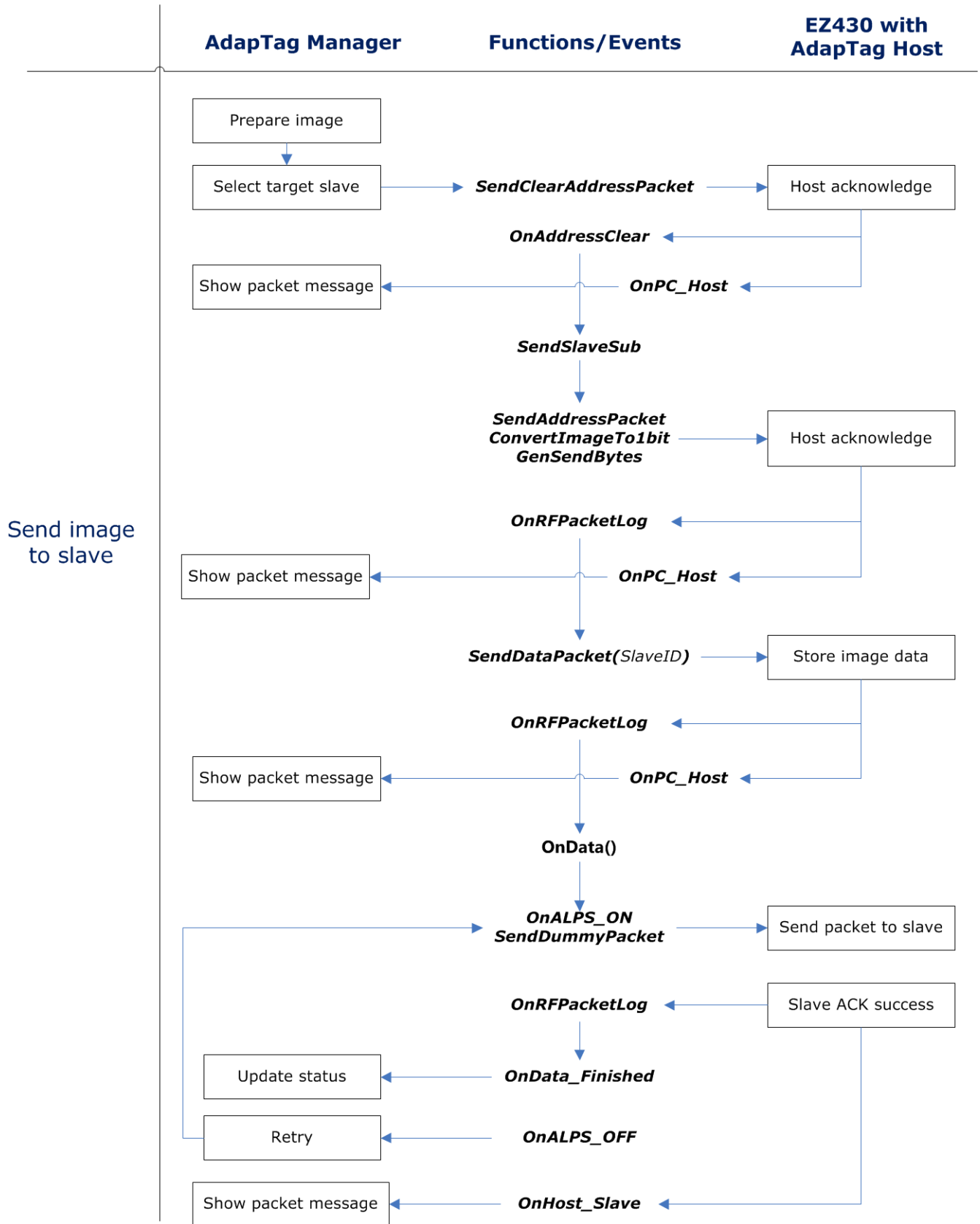
2.8 The Functional Flowchart of Sending Image Data

This section lists the functional flowcharts that will use what functions and are triggered by which events when connecting eZ430 USB dongle, configuring device and sending image data to slave. User could refer to source code and see the operation flow as below.

2.8.1 Start with AdapTag Kit



2.8.2 Send Image Data to Slave



- To poll slave or send image to slave, the first command is to clear address on Host. After AdapTag Host acknowledge, the `OnAddressClear` will send poll packet (`SendPollPacket`) if user selected poll slave command, or send image data (`SendSlaveSub`) via RF or USB if user chose send image command.
- In `SendSlaveSub` function, AdapTag Manager will send the address packet (`SendAddressPacket`) to AdapTag Host. The `SendAddressPacket` function converts image file to 1 bit bitmap image format (`ConvertImageTo1bit`) and generates the image data bytes (`GenSendBytes`). The image data is prepared in this function as `SendMartix[]` byte array in order to send data packet later.
- After AdapTag Host acknowledges by address packet, the `SendDataPacket` follows to send the image data `SendMartix[]` to Host. The AdapTag Host will base on the received data packet stores as Protocol Packet in flash memory.
- If AdapTag Host acknowledges storing image data, the `OnData` will start radio (`OnALPS_ON`) and the AdapTag Host starts sending packets to target AdapTag Slave.
- Any packet returns from Host will trigger on `OnRFPacketLog` event. The `OnRFPacketLogSub` will base on the received packet and its command type to do corresponding event or function.
- The `SendDummyPacket` will add 4 seconds dummy time slots (idle packet) at the end of frame cycle for slave is capable to update EPD and ready for next coming frame cycle. The 4 seconds here: if time slot=0.5s, there are 8 dummy time slots. If time slot=1s, there are 4.
- If the target Slave acknowledges success, the `OnDataFinished` function will update the target slave's health information on list. Otherwise, the time out will trigger `OnALPS_OFF` to proceed retry command.

3. System Packet

The structure below defines the System Packet to communicate with AdapTag Host. The communication packets between computer and AdapTag Host are formatted as the following System Packet Structure. The AdapTag Host will base on the received System Packets to restructure the packet data as Protocol Packet in memory and wait for command sending to target AdapTag Slave.

3.1 System Packet Structure

| Byte | 0 | 1 | 2~3 | 4 | 5~62 | 63 |
|------|--------|---------------|---------|--------------|------|-----|
| Name | Header | Packet Length | Address | Command Type | Data | CRC |

- ✧ Header:
 - 0xAA: Initial header, the beginning of the System Packet
- ✧ Packet Length:
 - 6~64 (0x06~0x40)
- ✧ Address:
 - Slave ID: 1~65,535(0x0001~0xFFFF), where 0 (0x0000) is for AdapTag Host
- ✧ Command Type:
 - See next section
- ✧ Data:
 - 58 bytes maximum. The payload.
 - When Command Type is Idle or Poll, the Packet Length is 6 (0x06) without data packet.
 - The Data format is 1 bit bitmap image. For more information about preparing bitmap image, please refer to [5] section 2: Preparing Image Data.
- ✧ CRC:
 - Error-detecting check
 - Header XOR Packet Length XOR Address XOR Command Type XOR Data = CRC

3.2 Command Type

The data flow direction: P=Computer (AdapTag Manager), H=Host, S=Slave

| Type | Command Name | Alias name (Show on AdapTag Manager) | Direction | Description |
|------|--------------------|--|-----------|---|
| 0x10 | Address | Address | P ↔ H | The following data bytes are address list |
| 0x11 | Address_Clear | Clear_Addr | P ↔ H | Clear address list |
| 0x12 | Address_Copy | Copy_Addr | P ↔ H | Copy one slave's address to show the same image |
| 0x20 | Idle | Idle | P ↔ H | Send Idle command |
| 0x30 | Poll | Poll | P ↔ H | Send Poll command |
| 0x31 | ALPS_ON | ALPS_ON | P → H | Start to send packets to slave |
| 0x32 | ALPS_OFF | ALPS_OFF | P ← H | Respond for stopping ALPS |
| 0x40 | Data | Data | P ↔ H | The following data bytes are image data |
| 0x41 | Host_Show | Host:Show | P ↔ H | Update EPD by AdapTag host board via USB |
| 0x50 | Enable_Debug | Enable_Debug | P ↔ H | Start showing detail packet information. (where data byte=0x00=Disable(default), 0x01=Enable) |
| 0x51 | Host_Config | Host:Config | H → S | List detailed configuration packets |
| 0x52 | Host_Address | Host:Address | H → S | List detailed address packets |
| 0x53 | Host_Data | Host:Data | H → S | List detailed data packets |
| 0x54 | Host_Signature | Host:Signature | H → S | List detailed signature packets |
| 0x55 | Host_Poll | Host:Poll | H → S | List detailed poll packets |
| 0x56 | Host_Idle | Host:Idle | H → S | List detailed Idle packets |
| 0x60 | Data_Finished | Re:Data | H ← S | Reply finished sending data |
| 0x66 | RespondPolling | Re:Poll | H ← S | Reply for Polling command |
| 0x67 | RespondJoinChannel | Re:Join | H ← S | Reply when slave joins channel |

| | | | | |
|------|------------------|--|-------|--|
| 0x70 | Write_NetworkID | | P ↔ H | Write Network ID to device |
| 0x71 | Read_NetworkID | | P ↔ H | Read Network ID from device |
| 0x72 | Write_SlaveID | | P ↔ H | Write Slave ID to device |
| 0x73 | Read_SlaveID | | P ↔ H | Read Slave ID from device |
| 0x74 | Write_Channel | | P ↔ H | Write Channel ID to device |
| 0x75 | Read_Channel | | P ↔ H | Read Channel ID from device |
| 0x76 | Write_TX_Power | | P ↔ H | Write TX Power to device |
| 0x77 | Read_TX_Power | | P ↔ H | Read TX Power from device |
| 0x78 | Write_EncryptKey | | P ↔ H | Write Encryption Key to device |
| 0x79 | Read_EncryptKey | | P ↔ H | Read Encryption Key from device |
| 0x7A | Write_SlotTime | | P ↔ H | Write the duration of time slot to device |
| 0x7B | Read_SlotTime | | P ↔ H | Read the duration of time slot from device |
| 0x7C | Read_Version | | P ↔ H | Read current firmware version |

If the direction has “←” and “→” below, which means not only send out the command but also return data or acknowledgement. You will find the return packets in System Packet window on AdapTag Manager.

The communication packet between P and H will be shown in System Packet window on AdapTag Manager. If the communication packet is between H and S, the packet data will be shown in Protocol Packet window on AdapTag Manager. The printing packet information is defined at `PrintSendPackets` and `PrintReceivePackets` in `AdapTag_API.cs`

The communication packet of Command type from 0x70 to 0x7C will not show on AdapTag Manager. They are about configuring AdapTag Host or Slave.

4. REFERENCES

The following references provide additional information on the AdapTag development kit, the Texas Instruments ALPS protocol, and the kit specification in general.

- [1] The Microsoft® Visual Studio® project and source code of AdapTag Manager is available for download on customer zone of PDI website
- [2] The Code Composer Studio™ project and firmware source code of AdapTag host and slave is available for download on customer zone of PDI website
- [3] AdapTag Development Kit Quick Start Guide. (Printed copy included in the box with the kit)
- [4] AdapTag Development Kit User Manual (Doc No. 8P002-00)
- [5] AdapTag Development Kit COG Driver Programming Guide (Doc No. 8P003-00)
- [6] AdapTag Development Kit System Development Guide (Doc No. 8P004-00) (This document)
- [7] AdapTag Development Kit Protocol and Communication Guide (Doc No. 8P005-00)
- [8] Code Composer Studio™ (CCStudio) Integrated Development Environment (IDE)
URL: <http://www.ti.com/tool/ccstudio>
- [9] E-paper Display COG Driver Interface Timing (Doc No. 4P008-00 rev.02)
- [10] Texas Instruments CC430F5137, 16-Bit Ultra-Low-Power MCU.
<http://www.ti.com/product/cc430f5137>

Glossary of Acronyms

| | |
|---------------|---|
| EPD | Electrophoretic Display (e-Paper Display) |
| EPD Panel | EPD |
| TCon | Timing Controller |
| MCU | Microcontroller Unit |
| SPI | Serial Peripheral Interface |
| COG | Chip on Glass |
| PDI, PDi | Pervasive Displays Incorporated |
| RF | Radio Frequency |
| ALPS | Advanced Low Power Star network protocol |
| CCS, CCStudio | Texas Instruments' Code Composer Studio™ IDE development tool |
| RSSI | Received Signal Strength Indicator |