IWM2 TCP/IP devices: HandyDrummers and FunkyGates-IP
Getting started in .NET

What are the basic principles of the SDK application ?

- The Mainform enables to add TCP devices (click on the green '+')
- The device's characteristics must be specified in the form that shows up
- Once all the device's characteristics are specified, an object is created :
  - a SpringCardIWM2_Network_Reader object enables to communicate with a FunkyGate IP
  - a SpringCardIWM2_Network_GPIOs object enables to communicate with a HandyDrummer
- Dedicated UserControls are created and added in the Mainform :
  - SpringCardIWM2_Reader_Controller controls the SpringCardIWM2_Network_Reader object
  - SpringCardIWM2_GPIOs_Controller controls the SpringCardIWM2_Network_GPIOs object

The interfaces

- There are three interfaces :
  - The ISpringCardIWM2_Device interface defines all methods relevant to all IWM2 SpringCard devices (both TCP and serial): methods to call (=callbacks) when the device's status changes or when an unknown TLV is received from the device, commands to get the global status, reset the device, etc ...
  - The ISPringCardIWM2_Reader interface inherits from ISpringCardIWM2_Device. It adds the methods related to the badge, LEDs and buzzer management. All methods in this interface are relevant to all FunkyGate readers (both TCP and serial)
  - The ISpringCardIWM2_GPIO interface inherits from ISpringCardIWM2_Device. It adds the methods related to the inputs and the outputs. All methods in this interface are relevant to all HandyDrummer devices (at the time of this writing, all the HandyDrummer devices communicate through a TCP link)

The controllers

- The SpringCardIWM2_Reader_Controller controls a ISpringCardIWM2_Reader interface. This means that any object, that implements that interface, may be controlled with this controller: this way, it is usable with both TCP and serial readers (the SpringCardIWM2_Network_Reader class implements the ISpringCardIWM2_Reader interface).
- The SpringCardIWM2_GPIOs_Controller controls a ISpringCardIWM2_GPIOs interface. This means that any object, that implements that interface, may be controlled with this controller: this way, it is usable with TCP HandyDrummer, and with other HandyDrummer devices in the future (the SpringCardIWM2_Network_GPIOs class implements the ISpringCardIWM2_GPIOs interface)
- It is to be noted that all the callbacks defined in the interfaces, are not actually implemented in the controllers.

The objects (1/2)

There are 4 objects :

- SpringCardIWM2_Device is an abstract class that implements the “Application layer” TLV commands (get global status, clear LEDs, etc …). Both “TCP” and “serial” objects inherit from this class, because those commands are the same for all.

- SpringCardIWM2_Network_Device inherits from SpringCardIWM2_Device. This object performs all the communication tasks with the actual TCP devices (connect, send/receive frames, disconnect, etc …). It also implements the ISpringCardIWM2_Device interface.

The objects (2/2)

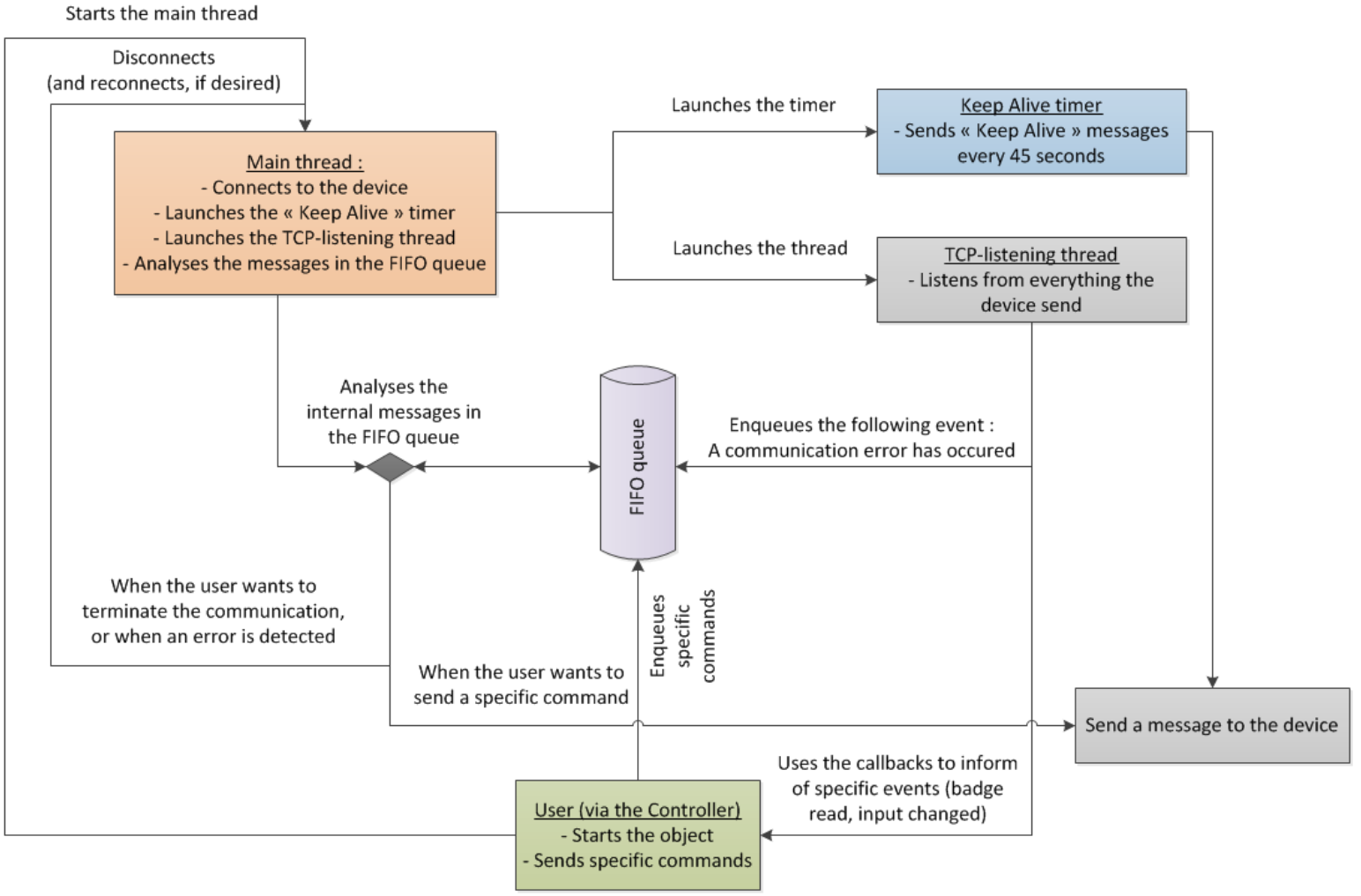
- SpringCardIWM2_Network_Reader inherits from SpringCardIWM2_Network_Device. As already stated, it also implements the ISpringCardIWM2_Reader interface. It sets the reader-specific callbacks: methods that are called when a badge is read, inserted or removed.

- SpringCardIWM2_Network_GPIOs inherits from SpringCardIWM2_Network_Device. As already stated, it also implements the ISpringCardIWM2_GPIOs interface. It sets the method that is called when a HandyDrummer input changes

Short summary

- To sum up :
  - All the intelligence related to the TCP connection is in the SpringCardIWM2_Network_Device class
  - As the SpringCardIWM2_Network_Reader and the SpringCardIWM2_Network_GPIOs inherit from SpringCardIWM2_Network_Device, they inherit all this intelligence, and add their distinctive features (indicate an input has changed, or a badge has been inserted, etc ...)
  - When an event occurs (a badge is read, the device is connected, etc…), the controllers show them on the screen
  - When the user wants to interact with a device (clear a LED, set an output, etc ...), the corresponding method is called by the controller.

SpringCardIWM_Network_Device: block diagram



The SpringCardIWM2_Network_Device object (1/2)

- The object needs the IP address, the port, the communication mode (plain, secured with operation key, secured with administration key) and the key value
- To launch the communication, the “Start()” method is used.
  - This method creates a background thread, that tries to connect to the specified device, in the desired communication mode – we'll call this the main thread
  - The use of a background thread is mandatory, in order not to freeze the application's main form
  - Once the communication is established:
    - The main thread creates a timer, to generate and send the “Keep Alive” messages every 45 seconds, via the PlanifyKeepAlives() method
    - The main thread creates another thread, to listen to incoming messages from the device, via the LaunchTCPListenThread() method

The SpringCardIWM2_Network_Device object (2/2)

- The main thread then waits for incoming messages in a blocking thread-safe FIFO queue, to take appropriate action(s)
- Messages in the blocking thread-safe FIFO queue may come from different parts of the application :
  - When the thread, that listens from incoming messages from the device, detects a communication error, it enqueues a reconnection command – see OnCommunicationError()
  - When the user desires to terminate the communication, or send a command to the device, the specific message or TLV is enqueued – see for example GetGlobalStatus() method in SpringCardIWM2_Device class
- When the thread, that listens from incoming messages from the device, receives a frame, its content is analysed, and the appropriate method is called back (a badge has been read on a FunkyGate reader or an input has changed on the HandyDrummer)

How to build your own application (1/2)

- Copy the following files in your project:
  - Interface ISpringCardIWM2_Device.cs
  - Interface ISpringCardIWM2_GPIO.cs (if you work with HandyDrummers)
  - Interface ISpringCardIWM2_Reader.cs (if you work with FunkyGates)
  - Class SpringCardIWM2_Device.cs
  - Class SpringCardIWM2_Network_Device.cs
  - Class SpringCardIWM2_Network_GPIOs.cs (if you work with HandyDrummers)
  - Class SpringCardIWM2_Network_Reader.cs (if you work with FunkyGates)
  - Class SystemConsole.cs (if you're not building a Command Line Application)

- Use the appropriate namespaces : “SpringCard.IWM2” and “SpringCard.LibCs” (this last one: only if you're not building a Command Line Application)

- Only if you're using a Command Line Application, and you didn't add SystemConsole.cs :
  - You can log the debug messages in the console, by replacing the “SystemConsole.Verbose(...)” by “Console.WriteLine(...)” in file “SpringCardIWM2_Network_Device.cs”, method “protected override void log(string msg)”

How to build your own application (2/2)

- Create a SpringCardIWM2_Network_Reader, or a SpringCardIWM2_Network_GPIOs object, for each device you want communicate with
- Define the callbacks you need in your own “controllers”. A good starting point is to define methods that will be called when :
  - The device's status changes
  - A badge has been read, or inserted/removed, depending on the configuration of your FunkyGate (if you use a FunkyGate reader)
  - An input has changed on a HandyDrummer (if you use a HandyDrummer)
- Use the Start() method, to connect:
  - Each monitored event will then trigger the call of the specified callback method
  - Each time a command has to be sent, use the public methods. For example:
    - ClearOutput(x) for a HandyDrummer
    - SetLeds(x, y) for a FunkyGate
  - To print all the cryptographic operations, use the public method: “SetShowCrypto(true)”