IWM2 Serial devices: FunkyGates-DW

MK2 serial protocol

Getting started in .NET

What are the basic principles of the SDK application ?

- The Mainform enables to show a SpringCardIWM2_Serial_Scheduler_Controller (click on the '+'), on a pop-up window
- This controller enables to create and start a SpringCardIWM2_Serial_Scheduler (choose the desired COM port, then “Start”). This object manages a specific COM port to detect all connected readers and maintain communication with each of them. Each detected reader is a SpringCard_Serial_Reader object
- When the user clicks on a detected reader:
  - A SpringCardIWM2_Reader_Controller is created, to control the SpringCard_Serial_Reader object
  - This controller is then added in the Mainform

The interfaces

- There are two 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 controllers

- The SpringCard_IWM2_Serial_Scheduler_Controller controls a SpringCardIWM2_Serial_Scheduler object: it enables to create the scheduler, to start and stop it. This controller also provides callback methods, which are to be called by the scheduler when it stops, or when a reader is found, or dropped.
- 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.
  - As the SpringCardIWM2_Network_Reader class implements the ISpringCardIWM2_Reader interface, it can then be controlled by the SpringCardIWM2_Reader_Controller.
  - It is to be noted that all the callbacks defined in the interface, are not actually implemented in this controller.

The objects (1/2)

There are 3 objects :

- SpringCardIWM2_Serial_Scheduler is an object that manages a COM port.
  - First, it probes all the addresses in the given range, to detect readers. For each responding reader, a SpringCardIWM2_Serial_Reader object is created and added in a list of all installed readers
  - Then, it maintains communication with all the detected readers, polling them one after another: it sends appropriate messages (empty most of the time, or carrying a LED or a buzzer command), and initiates the analysis of what is received from the reader
  - In case of successive communication errors with a reader, the scheduler drops the reader
  - Every 10 seconds, it probes again every address to detect new or dropped readers

- 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.

The objects (2/2)

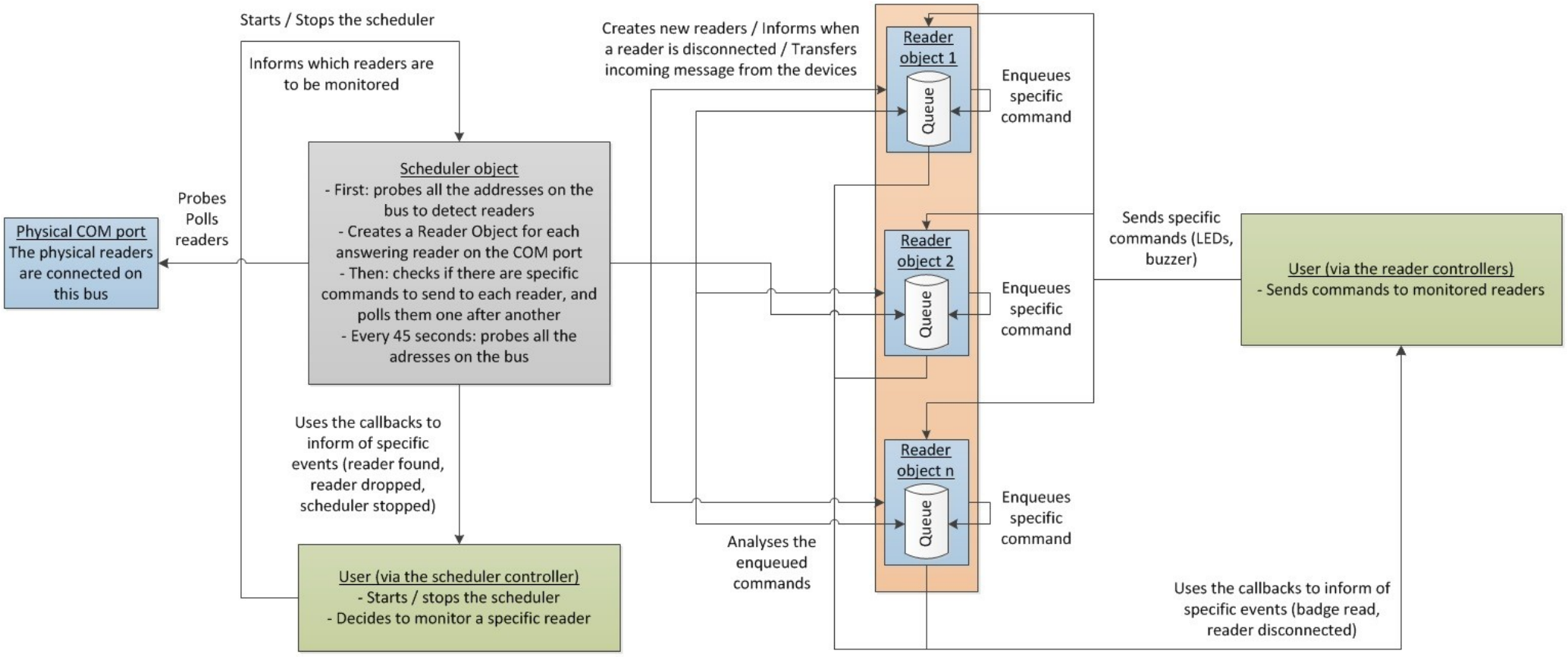
- SpringCardIWM2_Serial_Reader inherits from SpringCardIWM2_Device. As already stated, it also implements the ISpringCardIWM2_Reader interface. It sets the reader-specific callbacks: methods that are called when a badge is read, inserted or removed. Several points are worth mentioning :
  - This object does not perform the communication with the actual device, the communication is performed by the SpringCardIWM2_Serial_Scheduler
  - So, when the user wants to send a command to the reader, the command is sent, by the controller, to the SpringCardIWM_Serial_Reader, which transmits it to the SpringCardIWM2_Serial_Scheduler, through a FIFO queue
  - When the scheduler receives a message from a device on the bus, it sends it to the SpringCardIWM2_Serial_Reader to analyse it and call the appropriate callback method

Short summary

- All the intelligence related to the serial communication with the device(s) is in the SpringCardIWM2_Serial_Scheduler class

- The SpringCardIWM2_Serial_Reader is used as a “command enqueuer” (when a command is received from a user), and a “callback caller” (when an event is received from the device)

Block Diagram



The SpringCardIWM2_Serial_Scheduler object (1/2)

- The object needs the COM port, and its configuration (by default, 38400 bps, no parity, 8 bits of data and one stop bit)
- Once the object is created, it is started via the public “Start(int x, int y)” method: the parameters are the minimum and maximum addresses on the bus
- This method creates a background thread, that first probes all specified addresses on the bus : see “Enumerate(...)” method.
- The use of a background thread is mandatory, in order not to freeze the application's main form

The SpringCardIWM2_Serial_Scheduler object (2/2)

- After the first probing loop :
  - If no reader is detected, the thread ends: the appropriate callback is called
  - If at least one reader answers a probing request:
    - For each detected reader, a SpringCardIWM2_Serial_Reader object is created and added in the "InstalledReaders" list
    - A timer is created, to indicate when the next probing loop ("Enumerate(…)") must occur: see the "PlanifyNextEnumerations()" method and the "EnumerateNow" boolean
    - The thread then polls all detected readers – see TalkToReaders() method : for each reader, it first checks, in the reader's internal FIFO queue, whether there is a specific command to send (this may be an Acknowledge, or a LED/buzzer command), it then sends the appropriate command if needed, or an empty I-Block, to maintain the communication active
    - Every 10 seconds ,it switches back to the probing loop, to detect new or dropped readers
- When information is received from a device (a badge has been read), its content is analysed in the ISpringCardIWM2_Serial_Reader, and the appropriate method is called back

How to build your own application (1/2)

- Copy the following files in your project:
  - Interface ISpringCardIWM2_Device.cs
  - Interface ISpringCardIWM2_Reader.cs
  - Class SpringCardIWM2_Device.cs
  - Class SpringCardIWM2_Serial_Reader.cs
  - Class SpringCardIWM2_Serial_Scheduler.cs
  - 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 “SpringCard.LibCs.SystemConsole.Verbose(...)” by “Console.WriteLine(...)” in file “SpringCardIWM2_Serial_Scheduler.cs”, method “public void LogString(String s)”

How to build your own application (2/2)

- Create a SpringCardIWM2_Serial_Scheduler object, to take care of all the communications on the bus, and retrieve the addresses of all installed reader.
- Create or reuse the scheduler's “controller”, depending on your needs. Define all the needed callbacks, notably when:
  - The scheduler stops / starts
  - A reader has been found, on a particular address
  - A reader has been dropped
- Start the scheduler, using the “Start(x, y)” method, and let the callbacks be called whenever a event occurs
- To print all the sent/received frames on the bus, use the public method: SetShowFrames(true)
- Retrieve each installed SpringCardIWM2_Serial_Reader object, using SpringCardIWM2_Serial_Scheduler.GetInstalledReader(address) – this can be done in the callback used when a reader has been found
- Create your own “controllers” for the reader, and define the callbacks. 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
- Each time a command has to be sent to a reader, use its public methods. For example: SetLeds(x, y)