XFS4IoT SP-Dev Workgroup

18 May 2021

To add others from your company

Please email us at:

xfs4iot_sp-dev_info@kal.com

How can workgroup members contribute?

- Ideas for contributions:
  - **New SPs using the framework**
  - SP simulators using the framework
  - Test scripts, test harnesses, etc. for SP testing
  - Sample code for new XFS4 applications
  - GitHub “pull requests” for changes to the SP-Dev code

XFS4IoT framework available: Card Reader

- The first SP-Dev framework was made **available** on GitHub on 4 May 2021
  - Card reader class
- *Members can create a GitHub fork and start developing card reader SPs*

The next step in XFS4IoT SP-Dev: Cash Dispenser

- The second SP-Dev framework is targeted to be **available** on GitHub 6 July 2021
  - Cash Dispenser class
- The initial release will not include
  - End-to-end security
  - Cash recycling

Reviewed XFS4IoT – device classes

- Card Reader (IDC) / Card Dispenser (CRD)
- Printer (PTR)
- Cash Dispenser (CDM)
- Cash Acceptor (CIM)
- PIN pad (PIN)
- Sensors/Indicators (SIU)
- Biometrics (BIO)
- Check Readers and Scanners (CHK)
- Scanner (IPM)
- Barcode Readers (BCR)
- Camera (CAM)
- Alarms (ALM)
- Text Terminal (TTU)

CEN XFS – new class process

↓CEN/XFS Committee

↓Proposal

↓Discussion

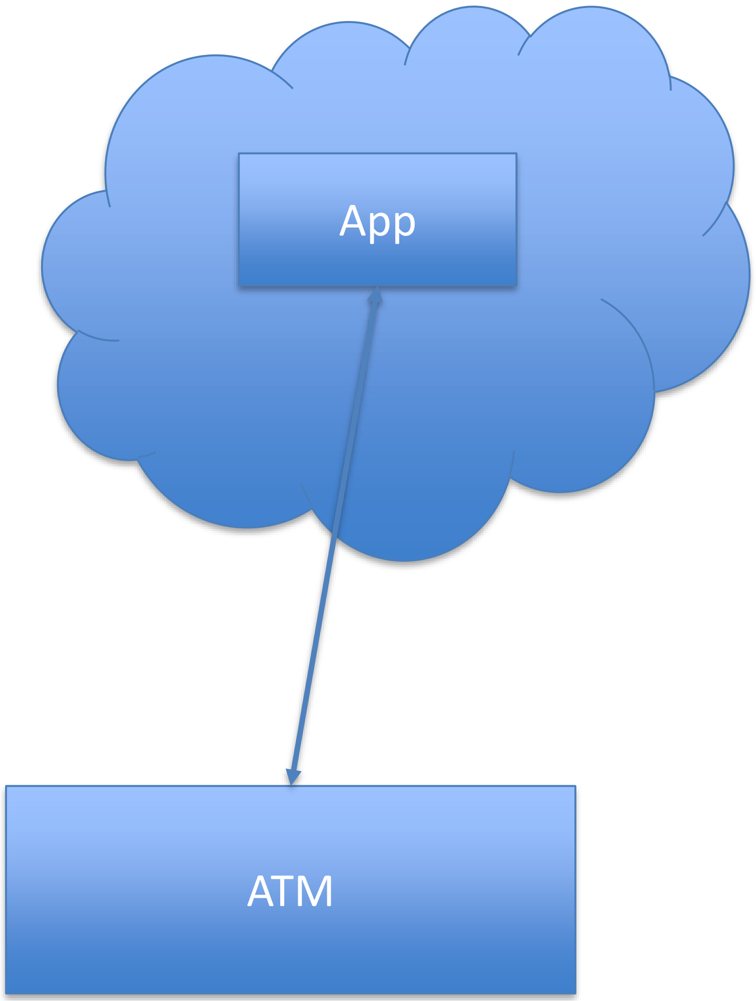
↓Consensus between members

↓Add to specification

- *Publication by CEN*

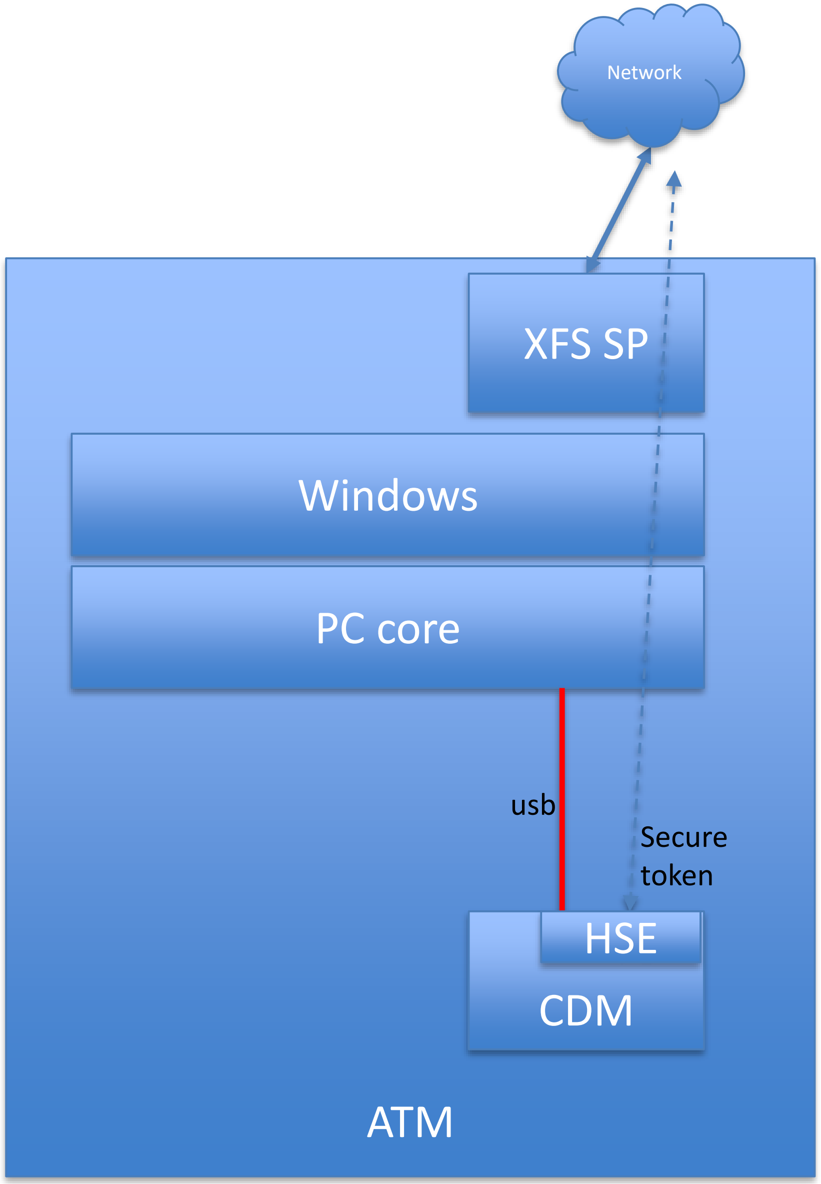
XFS4 Security – implementation issues to consider

- XFS4 security defines three types of secure connectivity between the SP and the world:
  - — TLS with certificates
  - — WebSockets over TLS for bi-directional communication
  - — Security tokens with TR34, TR31, HMAC
- The connection should end “inside the ATM”, but where exactly?



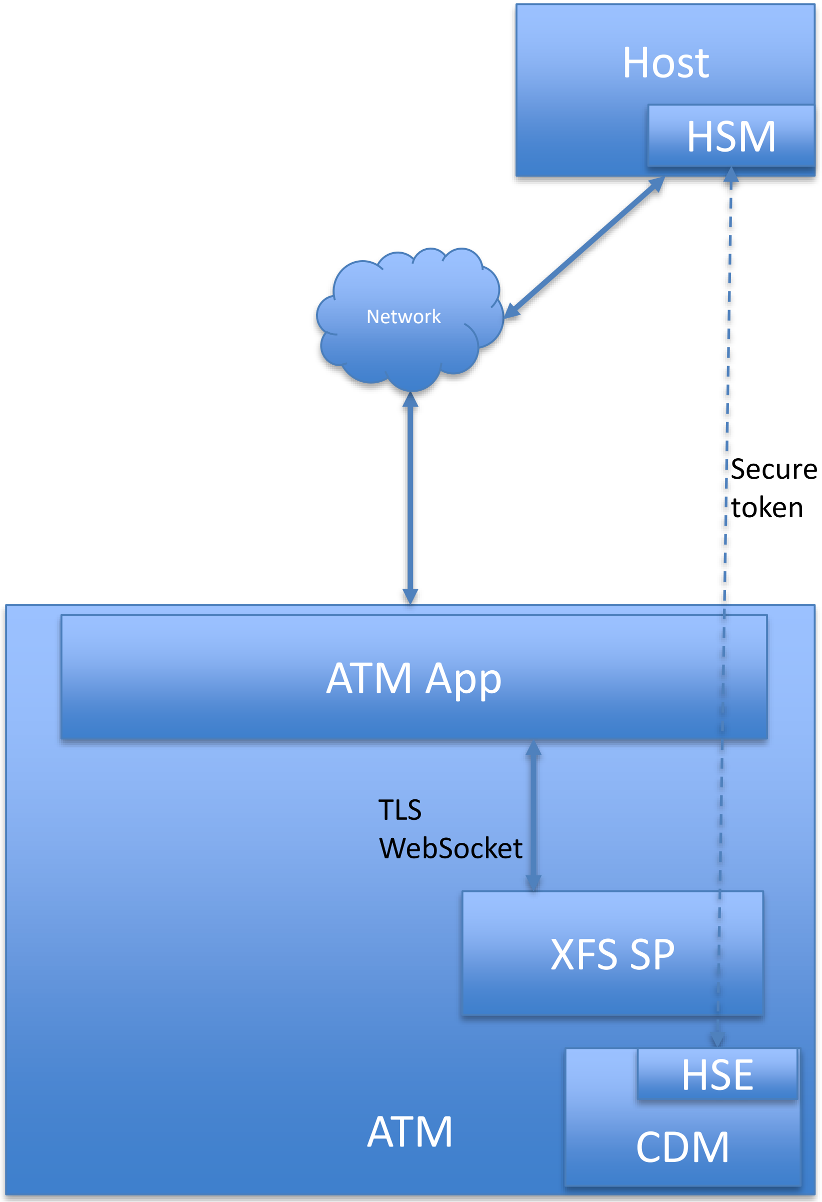
Where is the "End" inside the ATM?

- Assume a traditional architecture with an XFS SP running in Windows 10
  - The TLS connection *can end* at the SP
  - The WebSocket connection *can end* at the SP
- But the security token *must* be created / verified inside the CDM



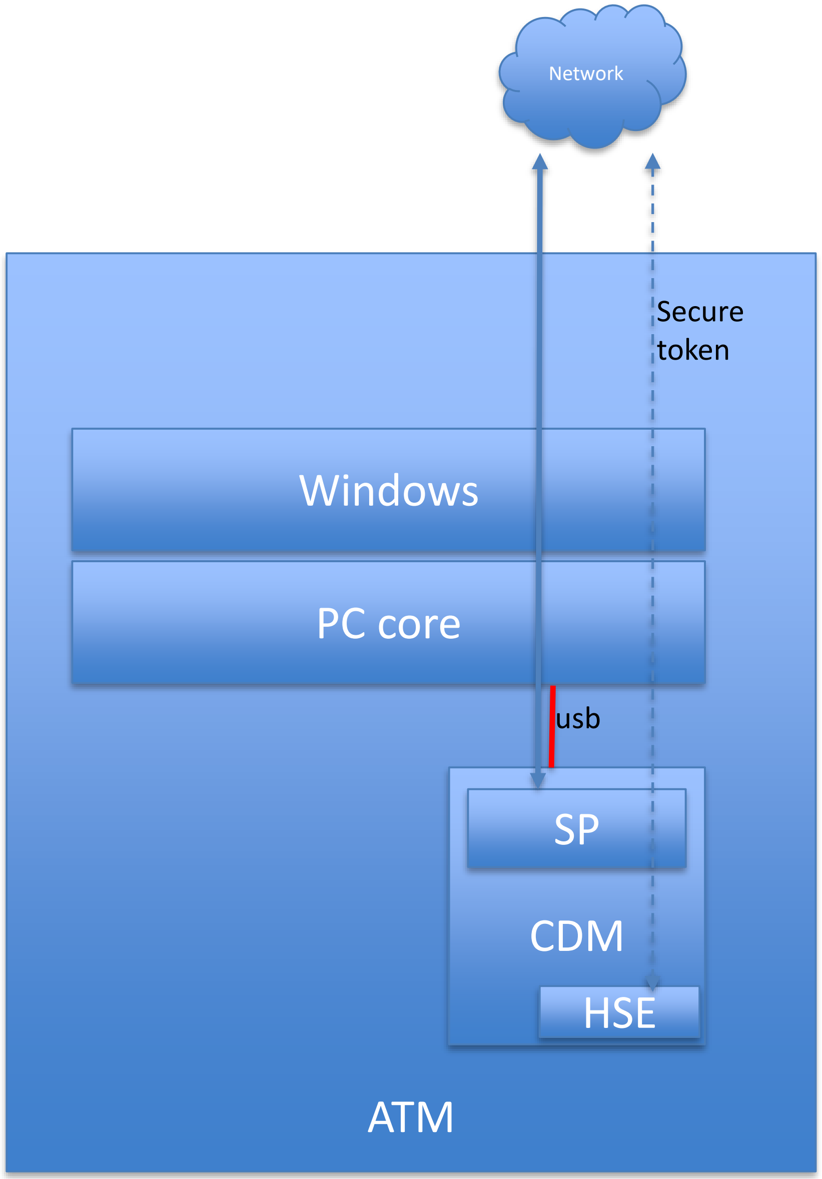
Where is the other "End"?

- Assume a traditional architecture with the application running inside ATM
  - — The XFS4 TLS connection *can end* at the local app
  - — The XFS4 WebSocket connection *can end* at the local app
- But the security token *must* be created and verified in an external HSM



Back now to the ATM “End”

- What if the SP is inside the CDM?
  - — The TLS connection *ends* at the SP
  - — The WebSocket connection *ends* at the SP
- The security token is delivered inside the CDM and is verified by the HSE
- The PC core is now just a “router” – ie part of the network



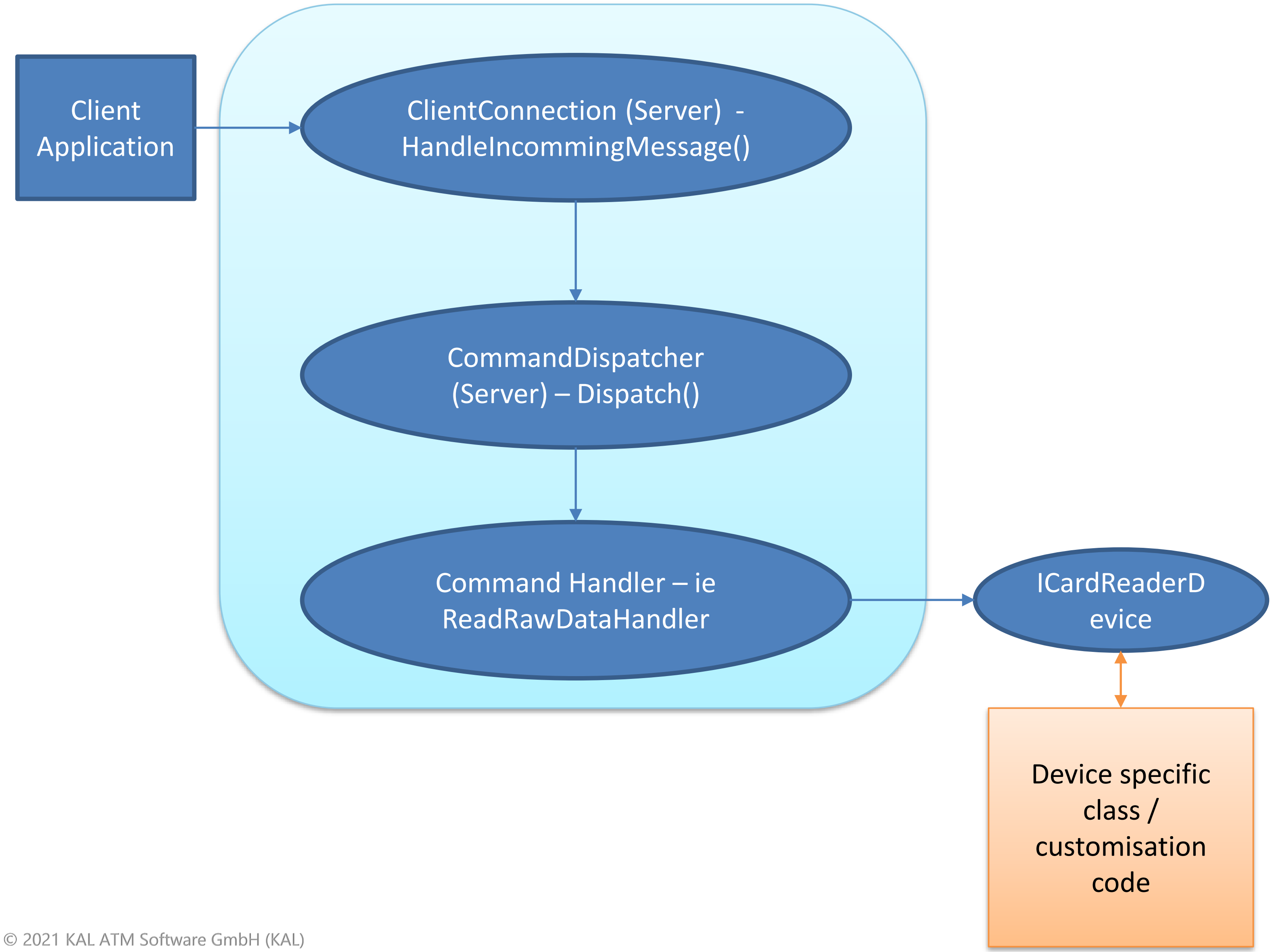
How to use the Card Reader framework

- The Card Reader framework looks after all the XFS specific details:
  - — Processing commands
  - — Sending command responses
  - — Sending events

- The SP developer only needs to focus on the device-specific interface

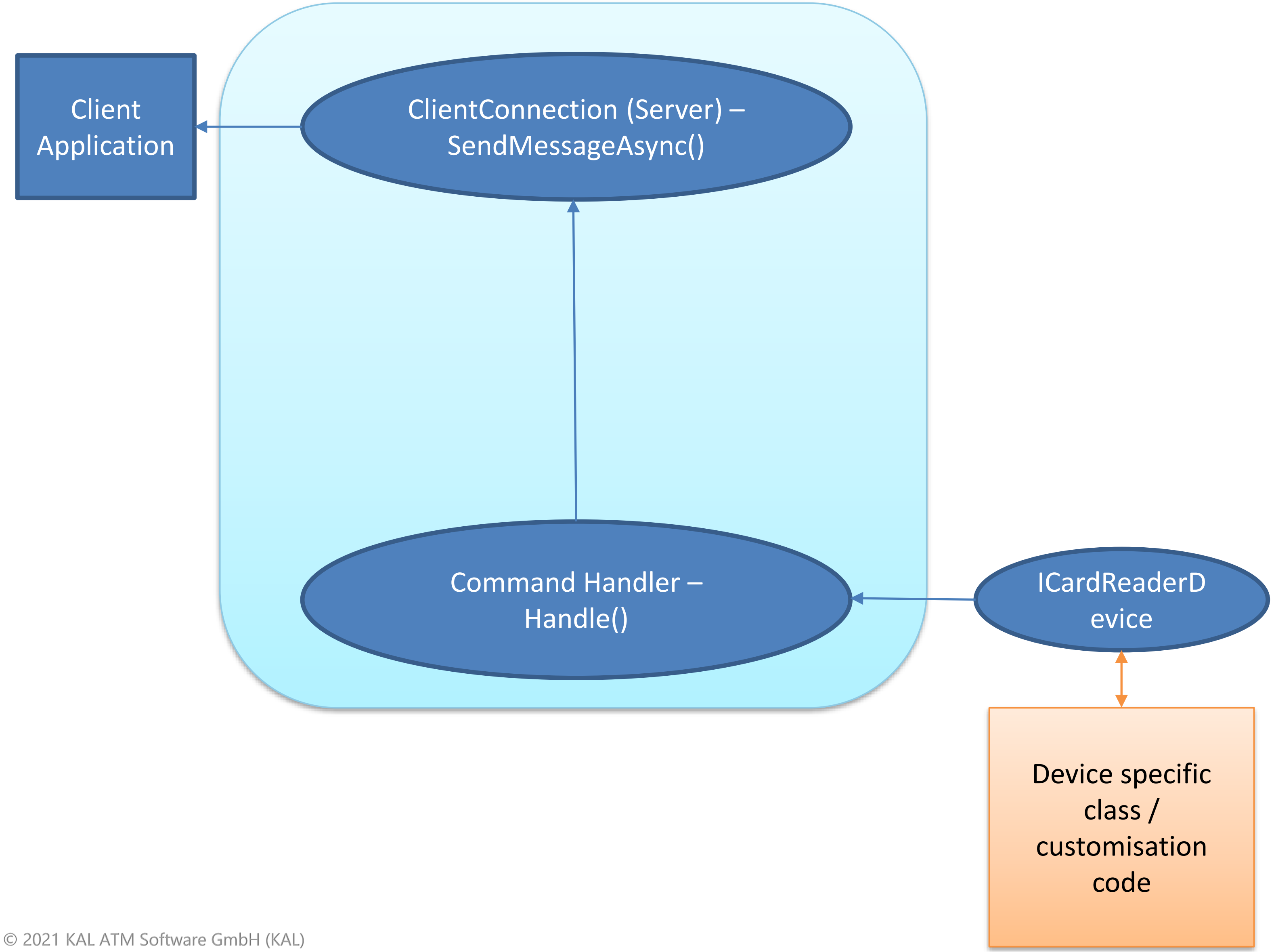
- We are first going to see the general mechanisms of the framework and the main classes it uses, then we will look at how to develop a sample SP

Processing commands


Client Application
ClientConnection (Server) - HandleIncommingMessage()
CommandDispatcher (Server) – Dispatch()
Command Handler – ie ReadRawDataHandler
ICardReaderDevice
Device specific class / customisation code

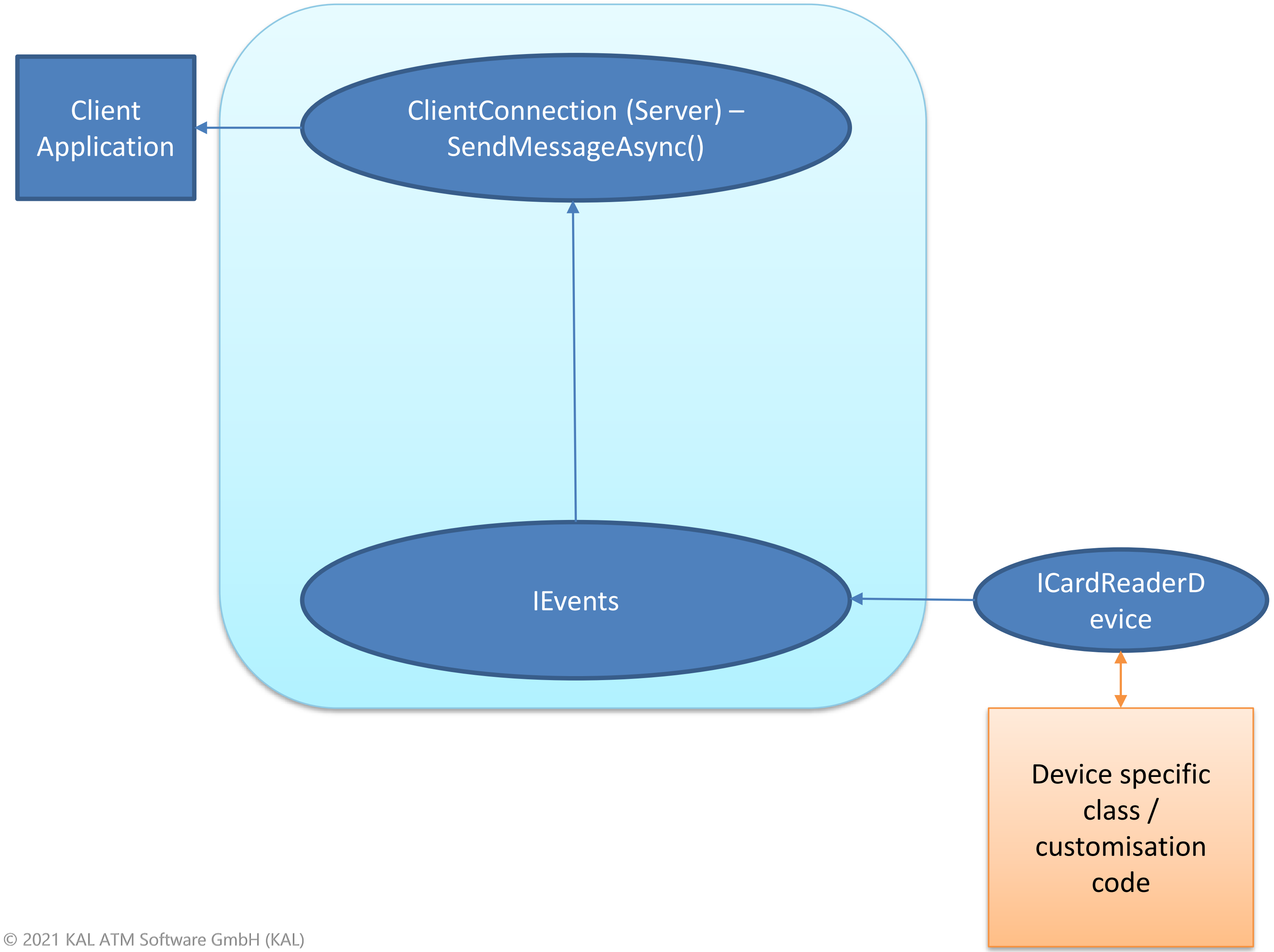
- The SP sits in a loop waiting for incoming command messages
- The framework uses reflection to parse the command
- The command dispatcher calls a function of the device-specific class

Command responses


Client Application
ClientConnection (Server) – SendMessageAsync()
Command Handler – Handle()
ICardReaderDevice
Device specific class / customisation code

- The command handler waits from a completion event from the device-specific class
- The framework uses asynchronous messages to send the completion event

Sending events



Client Application
ClientConnection (Server) – SendMessageAsync()
IEvents
ICardReaderDevice
Device specific class / customisation code

- The framework uses an IEvents class to allow sending unsolicited events (events that are not the result of a command call)
- The framework uses asynchronous messages to send the events

Using the framework to develop a Card Reader SP

The SP developer needs to create two components:

- An executable hosting one or multiple service provider classes. It will be used as the class factory for the device-specific class.

- A .Net Class Library implementing the device-specific class
  - — Derived from the ICommonDevice interface to have all the general XFS support in place
  - — Derived from the ICardReader interface to expose all the specific commands and events of a Card Reader SP

Getting started

- Go to the KAL_XFS4IoT_SP-Dev code repository on GitHub
- Pull the code from there. You can open it from the GitHub desktop or use another compiler.
- In the following example we will be using Visual Studio, but any other C# compiler can be used
- We are going to look at the ServerHostSample and CardReaderSample projects that are part of the SPs solution

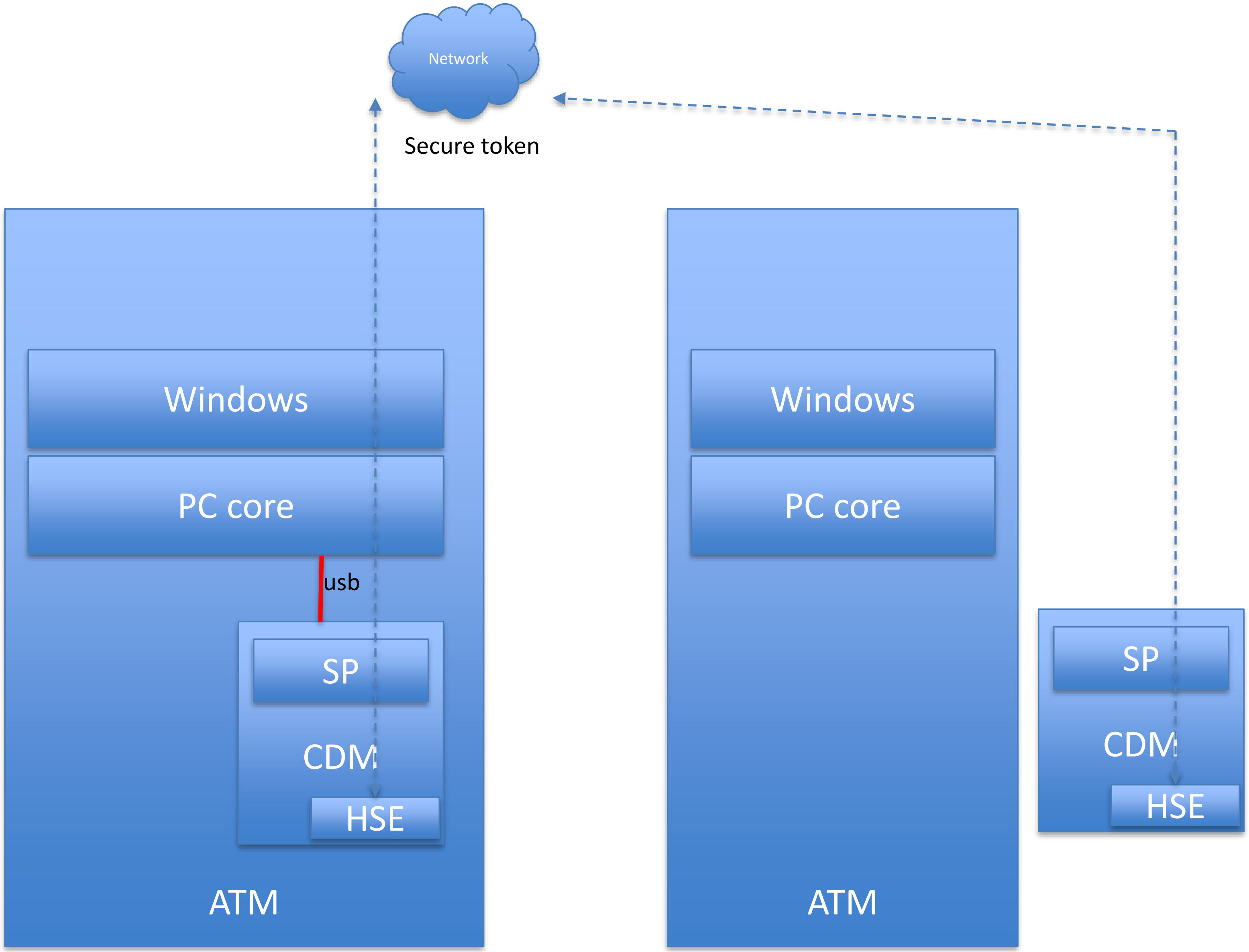
The ServerHost component

- Creating a Logger class derived from the ILogger interface
- Creating an instance of the device-specific class
- Creating a Card Reader SP endpoint
- Publishing it
- Running the main loop

The device-specific class library

- AcceptCardAsync
- ReadCardAsync
- ChipIOAsync
- EjectCardAsync and managing the Card Taken event
- Maintaining status properties

Framework support for small devices



- Consider the scenario with the SP inside the CDM
- The CDM itself inside the ATM, or maybe physically separated
- The SP now runs in “firmware”

Framework support for small devices

- Is it possible to use .NET in a small footprint device?

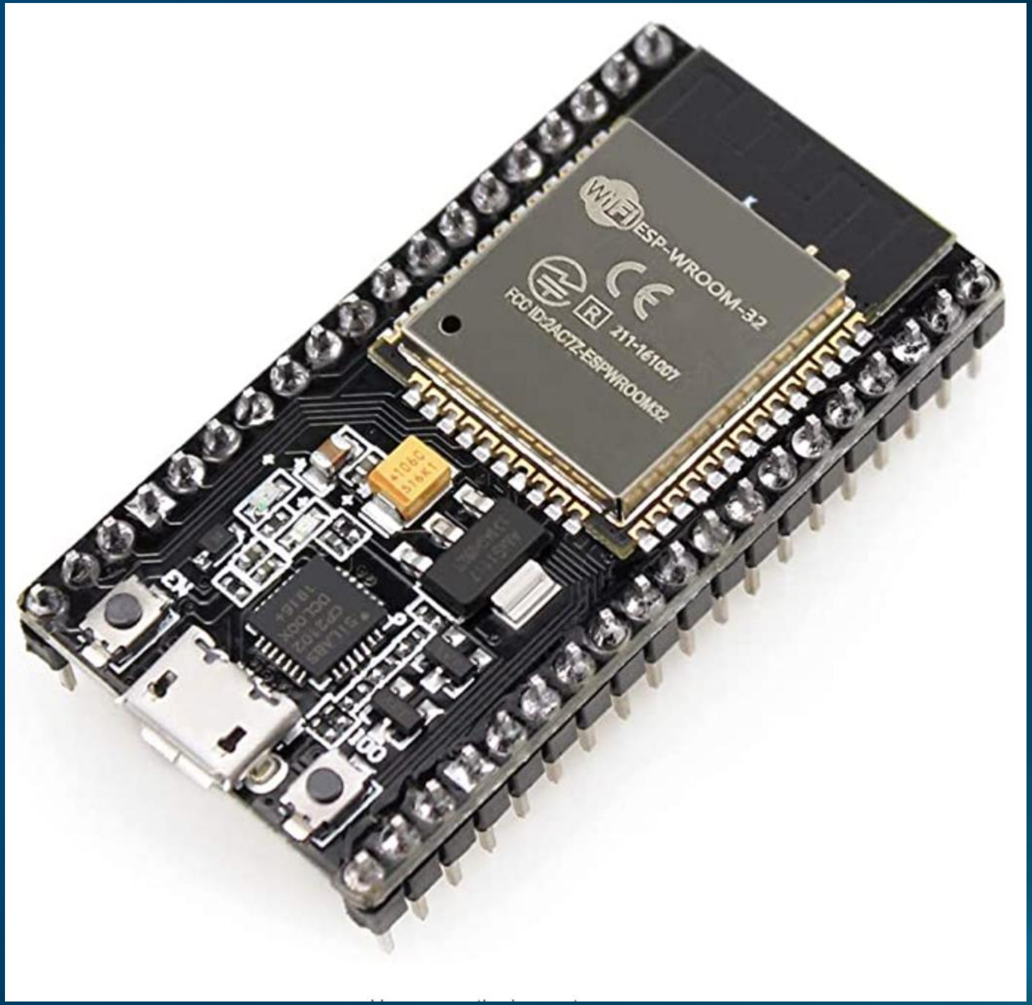
- There are several .NET versions for small devices:
  - — Nanoframework (was NETMF from Microsoft)
  - — Wilderness Meadow .NET

- These frameworks do not have the full functionality of .NET

Framework support for small devices

- Nanoframework can run on small devices

- $15 retail
- WiFi and Bluetooth
- Crypto processor
- ESP32 CPU
- 0.5MB SRAM
- 0.45MB Flash



Framework support for small devices

- Should KAL:
  - — Create one SP framework using .NET and Nanoframework but use “lowest functionality” to fit both environments?
  - — Create two SP-Dev frameworks one for .NET and one for small devices?
  - — If we create a separate “small device framework” should we use a .NET and C# or just use C/C++?

Next call

MS Teams

Video calls every two weeks:
Tuesdays at 1300 UK time

(we will reduce to calls once a month from June onwards)

Next call: 1st June 2021, 1300 UK, 0800 US EST, 2100 Tokyo time