How To Develop Driver, Part 2

In the second part of the lesson about Driver development, we will start with the actual programming, and take a closer look at the Link module.

As mentioned in the first part of the lesson, the internal architecture of the Driver contains a Link and a Packet Processor modules, which represent the communication layer ensuring connection between the Driver and the device. They are part of the SDK called Framework 4, provided by Gamanet.

In Framework 4, rules concerning overall behavior of the Driver during communication breakdown and recovery are implemented to both Link and Packet Processor. If a developer uses this SDK, he doesn't need to deal with implementation of these rules. However, if he decides to use his own communication layer, he must ensure these rules by himself.

The Driver Context ensures sharing of common data objects among different parts of the code of the Driver and Device Logic.

Initialization of Device Logic and Driver is pregenerated within the source codes created on the Portal Services in mySDK section.

Plugin Runner, as a host of the initialized instance of the Driver, provides it with the information about the ID of the Bus Controller which it should operate. At the same time, an instance of the already initialized Simple Client is provided for communication with the C4 System, along with the Cancellation Token in case there's a request to terminate the Driver's operation. Developers should use the Cancellation Token in all calls, to be able to terminate the entire Driver's activity as quickly as possible.

As part of the initialization, Simple Client needs to have defined the time zone within which the device operates. By default, UTC time is defined, but the Driver should be set to the time zone configured within its parameters in the C4 System settings.

Subsequently, a Driver Context is created and initialized within the generated code. The system retrieves the entire tree from the C4 Server. Based on this tree, it generates a complete list of object instances which were prepared by the developer for individual subsystem types.

Now, let's take a closer look at the Link module.

Link is a module of Framework 4, whose main task is to establish connection with the device. It monitors whether the connection has broken, and if so, it attempts to reconnect every 30 seconds.

In case of any changes in the connection, the Link ensures notification to other parts of the Driver, as well as informs the C4 System about it through logs and statuses.

Another task of the Link is to ensure the physical sending of data from the Driver to the device and receiving them back in the Logic of the particular communication protocol. Since the device protocols can be of different types, the Link ensures conversion of the information managed by the Driver into the protocol that the particular device understands.

Since the Link ensures communication with the device, it must always correspond to the type of connection enabled by the specific device.

In general, there are two categories of Link. The first one are Links that handle binary protocols, whether it's a network protocol based on IP (TCP Client Link, TCP Server Link, UDP Link), or physical device connections like RS232 or USB connection to the computer.

The second category are Links that handle remote function calls, such as a REST API Link or SQL Link.

Within the SDK, Gamanet provides all the basic types of Links commonly used for connection to external devices. In the case of a special connection type, it is possible to develop your own custom type of Link.

This lesson focuses on Links that handle binary protocols. In our case, we will develop Simple Driver based on the TCP Client.

In mySDK section of Portal Services, a developer defines a list of Links through which it is possible to connect to the particular device. Subsequently, in the configuration of the C4 System, the user will set the respective Link according to the type of connection of the device to the C4 System in the specific installation. Based on this setting, the Link selected by the user will already be defined in the LinkEnum property within BusControllerGenerated.

This Link is then initialized in the object of type Driver in the OnStart section, and inserted to the Packet Processor.

Internal architecture of the binary Link consists of several submodules.

The first one is a Binary Convertor which ensures transformation of data from Driver's internal objects of type Packet to a binary form, and subsequently sending it to the device.

Then it contains a Transport Layer, which is a dedicated thread used for asynchronous sending and receiving of data from the device.

Then it contains two submodules, which must be programmed by a developer - Packet Parser and Encryptor.

Packet Parser ensures back conversion of binary data from the device to the packets. It is a specific module with implemented Logic and binary packet structure defined in the documentation of the communication protocol provided by the device manufacturer. Data coming from the computer network may not arrive as a whole; they can come in parts, especially when the device is remote. Packet Parser works as a buffer where the bytes are collected, and once a complete packet is assembled, they are transformed into an object of type Packet. Subsequently, this packet containing the received data runs in the Device Logic as a single object.

Encryptor is an optional module which serves for encryption and decryption of data, in case that the connection is encrypted.

Let's now take a look at how the Link is created and implemented. Here we can see that a TCP Link has been created and initialized through Driver Context. We can also see here a Bus Controller ID and SimpleDevicePacketParser which was inserted there. This object serves for conversion of input data into packets and must be programmed by a developer.

To test how the Link works in practice, we need to implement Packet Parser and a few input and output packets.

Different categories and types of packets, as well as the way how they are used will be explained in the next part of the lesson.