

Example with the Enless Driver

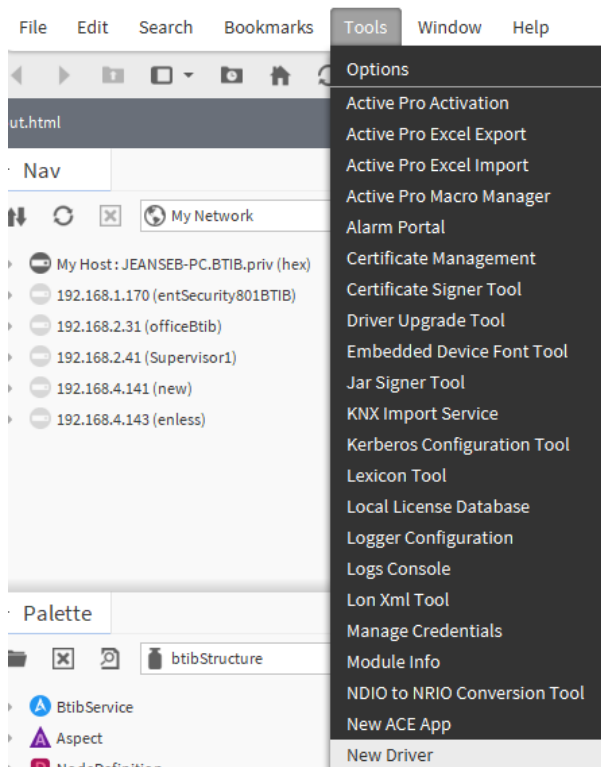
The Enless Driver provides a faster, easier way to interact with the Enless Sensors. You can quickly discover a sensor, configure it and receive its frames. On each sensor, you can also discover points which are already pre-configured to translate the received frames to values (temperature, humidity, CO2, etc).

The sensors work a bit differently than most drivers: they can only receive a frame during their configuration. It means that you can't discover, configure, ping or poll them whenever you want. Each of those functionality has been implemented in a non standard way.

For this reason we will only talk about the processing of a frame received from a sensor.

Step 1: Create the driver

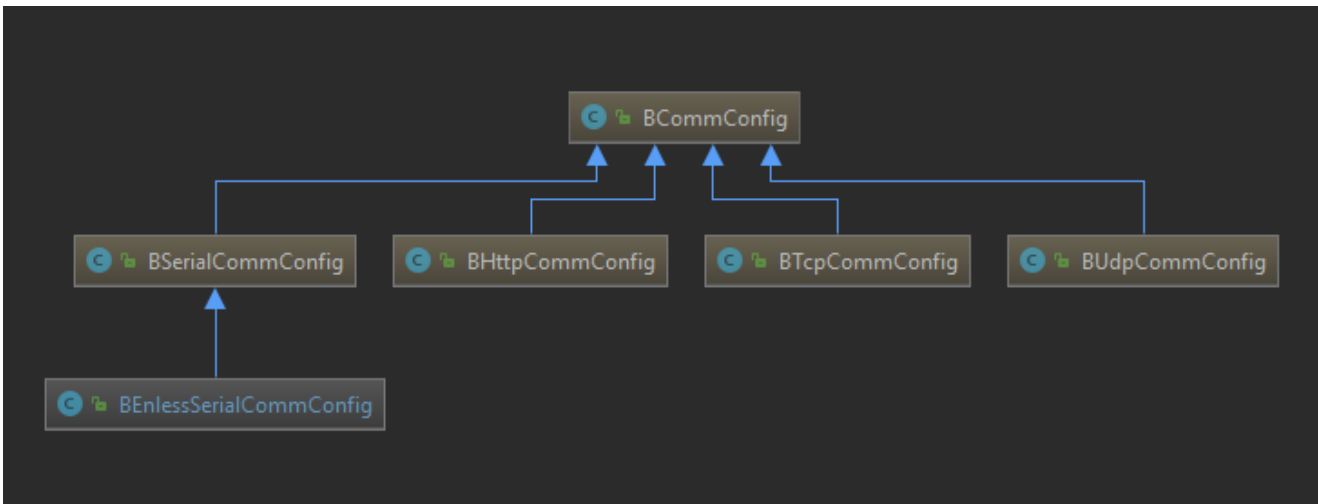
This driver was created using the "New Driver" wizard from the workbench.



It's a NDriver, supporting the Serial Protocol. The wizard will give us all the needed classes to begin our plugin. What's left to do is extend/implement the good classes and interfaces.

Step 2: Extend the BCommConfig class

Our driver is based on the Serial Protocol so we will extend the BSerialCommConfig class. Each protocol has its dedicated class.

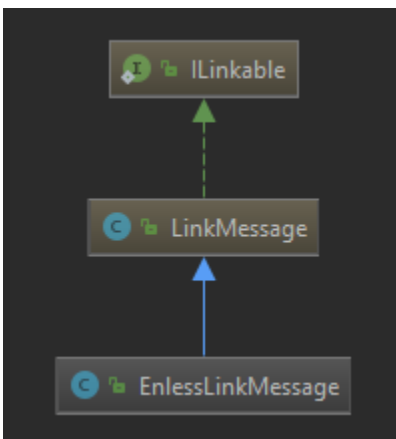


This class will create the correct LinkLayer and will determine how the frames received will be translated to a LinkMessage then to a custom NMessage.

We receive frames and we have to translate them so there are two important methods to override here:

- *protected NLinkMessageFactory makeLinkMessageFactory()*
create a custom message Factory with our custom LinkMessage (see step 3).
- *protected IMessageFactory makeMessageFactory()*
return our custom MessageFactory (see step 4).

Step 3: Extend the LinkMessage class

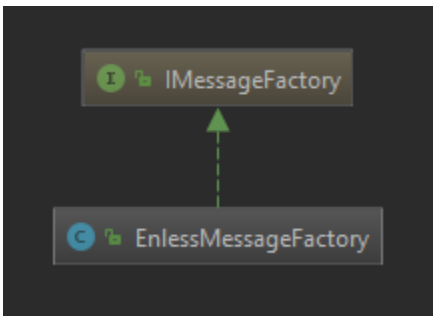


This is the class transforming a frame to a LinkMessage. There is one method to override:

public boolean receive(InputStream in) throws Exception

- handle the beginning and the ending of messages in the received input stream.
- validate the content and store it in the buffer inherited from the LinkMessage class.

Step 4: Implement the IMessageFactory interface



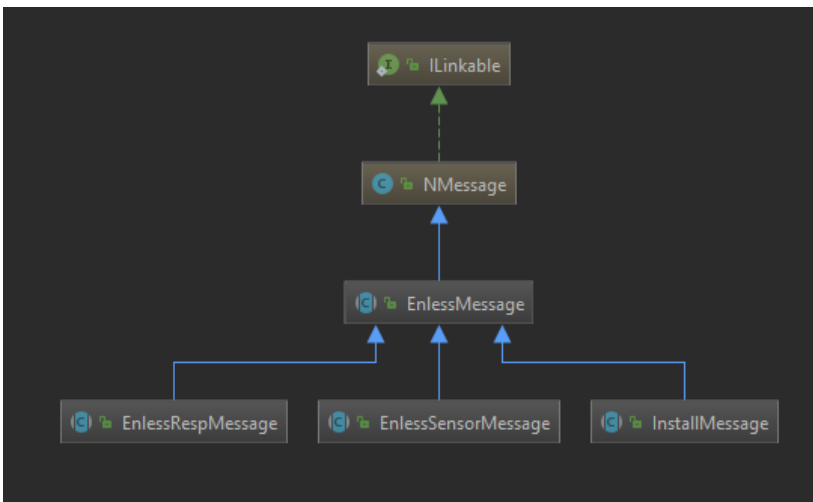
The factory receives the previously created `LinkMessages` and converts them to `NMessages`.

To send data, each Enless sensors has its own type of frame. The configuration of a sensor also need the exchange of some specific frames, so in our case it was necessary to translate each possible frame to a standard `NMessage`. There is one method to override:

public NMessage makeMessage(LinkMessage lm) throws Exception

create the appropriate `NMessage`

Step 5: Extend the `NMessage` class



(for simplicity's sake, only the core abstract class are shown, the driver has around 20 custom `NMessages`)

There are two type of `NMessages`: incoming and outgoing.

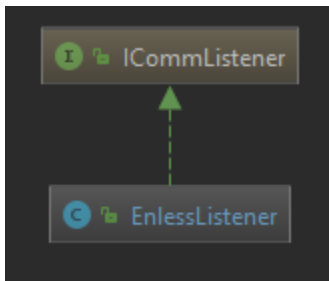
The incoming messages must override *public void fromInputStream(InputStream in) throws Exception*:

translate and store the data contained in the `InputStream`

The outgoing messages must override *public boolean toOutputStream(OutputStream out) throws Exception*:

write the data stored in the `NMessage` in the format expected by the driver.

Step 6: Implement the `ICommListener` interface



Now your received frames are correctly identified, translated and stored, what's left to do is to process them. There is one method left to override, *public void receiveMessage(NMessage nMessage)*:

process the message, update values in the workbench, respond with you own NMessage, etc.