CONVERSATION WITH https://www.blackbox.ai/chat/wIxnGI9

I am trying to learn about a standard called "NMEA 2000". I am using the NMEA 2000 library by Timo Lappalainen. I have built several ESP32 based test devices using mcp2515 canbus modules. One device emulates a temperture sensor, and another emulates an "actisense" serial gateway to the NMEA 2000 network. I am having a bit of trouble understanding how it all fits together. My most basic understanding is that an NMEA 2000 "network" exists as a "backbone" with "devices" connecting to the backbone. I am trying to write a "monitor" that will show me all the data on the network. I am running a program called the "Actisense NMEA Reader" on my laptop and it is receiving information from my monitor and temperature sensor correctly. It is the other direction I am now trying to encode, where the Actisense NMEA Reader sends requests and information to my monitor over a serial port. It is sending a message called "Get Operating Mode", which consists of the hex bytes, in order, of "0x10 0x02 0xa1 0x01 0x11 0x4d 0x10 0x03". I am trying to understand that message and what bytes to send in reply.

NMEA 2000 is a protocol used in marine environments for communication between various devices, such as sensors, displays, and controllers. It operates over a Controller Area Network (CAN) and allows devices to share data in a standardized way.

**Understanding the "Get Operating Mode" Message**

The message you provided, 0x10 0x02 0xa1 0x01 0x11 0x4d 0x10 0x03, can be broken down as follows:

1. **0x10**: This is the **Start of Frame** (SOF) byte, indicating the beginning of a CAN message.
2. **0x02**: This is the **Priority** byte. In NMEA 2000, the priority is typically represented in the first byte after the SOF.
3. **0xa1**: This is the **PGN (Parameter Group Number)**. The PGN 0xA1 corresponds to the "Get Operating Mode" request.
4. **0x01**: This is the **Data Length** byte, indicating that there is 1 byte of data in the message.
5. **0x11**: This is the **Data** byte. In this case, it may represent a specific request or command related to the operating mode.
6. **0x4d**: This is the **Checksum** byte, which is used to verify the integrity of the message.
7. **0x10**: This is the **End of Frame** (EOF) byte, indicating the end of the CAN message.
8. **0x03**: This is the **End of Frame** (EOF) byte, which is often used in serial communication to indicate the end of a message.

**Responding to the "Get Operating Mode" Message**

When your monitor receives the "Get Operating Mode" request, it should respond with a message that includes the current operating mode of your device. The response message will typically follow a similar structure:

1. **Start of Frame (SOF)**: 0x10
2. **Priority**: This can be set based on your device's priority.
3. **PGN**: The PGN for the response should match the request PGN, which is 0xA1.
4. **Data Length**: This should indicate how many bytes of data you are sending back.
5. **Data**: This should contain the actual operating mode information. The specific format of this data will depend on your device's implementation.
6. **Checksum**: This is calculated based on the preceding bytes to ensure data integrity.
7. **End of Frame (EOF)**: 0x10 and 0x03.

**Example Response**

Assuming your device's operating mode is represented by a single byte (e.g., 0x01 for "Normal Mode"), your response might look like this:

* **SOF**: 0x10
* **Priority**: 0x02 (example priority)
* **PGN**: 0xA1
* **Data Length**: 0x01
* **Data**: 0x01 (indicating "Normal Mode")
* **Checksum**: (calculated based on the previous bytes)
* **EOF**: 0x10, 0x03

**Calculating the Checksum**

The checksum is typically calculated using a specific algorithm defined in the NMEA 2000 standard. A common method is to sum all the bytes in the message (excluding the SOF and EOF) and take the two's complement of the least significant byte.

**Summary**

1. Listen for the "Get Operating Mode" message.
2. Construct a response message with the appropriate structure.
3. Calculate the checksum.
4. Send the response back to the Actisense NMEA Reader.

Make sure to refer to the NMEA 2000 documentation for specific details on the PGNs and the expected data formats for your device. This will help ensure that your implementation is compliant with the standard and works correctly with other devices on the network.

What would the value of the checksum byte be in your Example Response?

To calculate the checksum for the example response message, you need to sum all the relevant bytes in the message (excluding the Start of Frame (SOF) and End of Frame (EOF) bytes) and then take the two's complement of the least significant byte of that sum.

**Example Response Breakdown**

Let's assume the following example response:

* **SOF**: 0x10
* **Priority**: 0x02
* **PGN**: 0xA1
* **Data Length**: 0x01
* **Data**: 0x01 (indicating "Normal Mode")
* **EOF**: 0x10, 0x03 (not included in checksum calculation)

**Relevant Bytes for Checksum Calculation**

The bytes to include in the checksum calculation are:

* 0x02 (Priority)
* 0xA1 (PGN)
* 0x01 (Data Length)
* 0x01 (Data)

**Step-by-Step Calculation**

1. **Sum the bytes**:
   * 0x02 + 0xA1 + 0x01 + 0x01
   * In decimal: 2 + 161 + 1 + 1 = 165
2. **Convert the sum back to hexadecimal**:
   * 165 in decimal is 0xA5 in hexadecimal.
3. **Calculate the two's complement**:
   * To find the two's complement, you subtract the sum from 0xFF (255 in decimal):
   * 0xFF - 0xA5 = 0x5A

**Final Checksum Value**

Thus, the checksum byte for the example response would be 0x5A.

**Complete Example Response**

So, the complete response message would look like this:

* **SOF**: 0x10
* **Priority**: 0x02
* **PGN**: 0xA1
* **Data Length**: 0x01
* **Data**: 0x01
* **Checksum**: 0x5A
* **EOF**: 0x10, 0x03

This response can now be sent back to the Actisense NMEA Reader.

I believe that in the header of the message (0x10, 0x02), the 0x02 is not called the "priority" but is rather termed "STX" (start transmission). The 0x10 seems to be called "DLE" (data link escape). Likewise, the transmission ends with DLE, ETX (0x10, 0x03). Could that be a correct interpretation of the bytes?

Yes, you are correct in your interpretation of the message structure. In the context of NMEA 2000 and similar protocols, the bytes you mentioned have specific roles:

1. **DLE (Data Link Escape)**: The byte 0x10 is used as a Data Link Escape character. It indicates that the following byte is part of a special control sequence. This is a common practice in communication protocols to allow for the transmission of special characters without confusion.
2. **STX (Start of Text)**: The byte 0x02 is indeed referred to as the Start of Text character. It indicates the beginning of the actual data portion of the message.
3. **ETX (End of Text)**: The byte 0x03 is the End of Text character, which signifies the end of the message.

**Revised Breakdown of the Message**

Given your clarification, the breakdown of the message 0x10 0x02 0xa1 0x01 0x11 0x4d 0x10 0x03 would be:

* **0x10**: DLE (Data Link Escape)
* **0x02**: STX (Start of Text)
* **0xA1**: PGN (Parameter Group Number) for the "Get Operating Mode" request
* **0x01**: Data Length (indicating 1 byte of data)
* **0x11**: Data (the specific request or command)
* **0x4D**: Checksum (for error checking)
* **0x10**: DLE (Data Link Escape) again, indicating that the next byte is part of a control sequence
* **0x03**: ETX (End of Text)

**Summary**

Your understanding of the message structure is indeed correct. The DLE and STX/ETX characters are part of the framing used in the protocol to ensure that the data is transmitted correctly and to avoid confusion with the actual data being sent. Thank you for the clarification, and I appreciate your attention to detail!

So, given that, to the best of your knowledge about NMEA 2000 packets and the way checksums for it are calculated, in calculating the checksum, should we, or should we not include the initial DLE and STX in the callculation?

In the context of NMEA 2000 and similar protocols that use DLE (Data Link Escape) and STX (Start of Text) characters, the checksum is typically calculated based on the data portion of the message, excluding the DLE and STX bytes.

**General Rule for Checksum Calculation**

1. **Include Only Relevant Data**: The checksum is usually calculated over the bytes that represent the actual data being transmitted, which means you would exclude the DLE and STX bytes from the checksum calculation.
2. **Checksum Calculation**: You would sum the bytes that are part of the message payload (in this case, the PGN, Data Length, and Data) and then calculate the checksum based on that sum.

**Example Response Breakdown (Revised)**

Given the message structure:

* **DLE**: 0x10 (not included in checksum)
* **STX**: 0x02 (not included in checksum)
* **PGN**: 0xA1
* **Data Length**: 0x01
* **Data**: 0x11 (for example, the operating mode)
* **Checksum**: (to be calculated)
* **DLE**: 0x10 (not included in checksum)
* **ETX**: 0x03 (not included in checksum)

**Checksum Calculation Steps**

1. **Sum the relevant bytes**:
   * 0xA1 + 0x01 + 0x11
   * In decimal: 161 + 1 + 17 = 179
2. **Convert the sum back to hexadecimal**:
   * 179 in decimal is 0xB3 in hexadecimal.
3. **Calculate the two's complement**:
   * To find the two's complement, subtract the sum from 0xFF (255 in decimal):
   * 0xFF - 0xB3 = 0x4C

**Final Checksum Value**

Thus, the checksum byte for the example response would be 0x4C.

**Complete Example Response (Revised)**

So, the complete response message would look like this:

* **DLE**: 0x10
* **STX**: 0x02
* **PGN**: 0xA1
* **Data Length**: 0x01
* **Data**: 0x11
* **Checksum**: 0x4C
* **DLE**: 0x10
* **ETX**: 0x03

**Summary**

In summary, you should **not** include the DLE and STX bytes in the checksum calculation for NMEA 2000 messages. The checksum is calculated based on the actual data being transmitted, which typically includes the PGN, Data Length, and Data bytes. Thank you for your patience, and I hope this clarifies the checksum calculation process!

So, then, what would be the checksum for the 4 relevant bytes 0xa1, 0x01, and 0x01. Later I will encode the algorithm, but you can save me time by calculating it for me in this one case until I get an indication that I am sending a correct packet to the NMEA reader. With the previous checksum that did not happen.

Let's calculate the checksum for the relevant bytes you provided: 0xA1, 0x01, and 0x01.

**Step-by-Step Checksum Calculation**

1. **Sum the relevant bytes**:
   * 0xA1 + 0x01 + 0x01
   * In decimal:
     + 0xA1 = 161
     + 0x01 = 1
     + 0x01 = 1
   * Total: 161 + 1 + 1 = 163
2. **Convert the sum back to hexadecimal**:
   * 163 in decimal is 0xA3 in hexadecimal.
3. **Calculate the two's complement**:
   * To find the two's complement, subtract the sum from 0xFF (255 in decimal):
   * 0xFF - 0xA3 = 0x5C

**Final Checksum Value**

Thus, the checksum byte for the relevant bytes 0xA1, 0x01, and 0x01 would be 0x5C.

**Summary**

* **Relevant Bytes**: 0xA1, 0x01, 0x01
* **Checksum**: 0x5C

You can use this checksum in your packet to ensure that it is correctly formatted for the NMEA reader. If you have any further questions or need additional calculations, feel free to ask!

undefined

It seems like your message may have been sent accidentally without any content. If you have a question or need further assistance, please feel free to provide more details, and I'll be happy to help!

Using any and all publically available sources that you have access to, what can you tell me about general NMEA 2000 network and protocol, particularly with how devices shoiuld boot and/or join the network, and the role and protocol of the 60928 "address claim" PGN.

NMEA 2000 is a marine networking protocol that allows various marine electronic devices to communicate with each other. It is based on the Controller Area Network (CAN) protocol and is designed to facilitate the integration of devices such as GPS, chart plotters, autopilots, and other marine sensors. Here’s an overview of how devices boot and join the network, along with details about the Address Claim PGN (Parameter Group Number) 60928.

**General Overview of NMEA 2000**

1. **Network Structure**:
   * NMEA 2000 uses a multi-master architecture, meaning that multiple devices can communicate on the network without a central controller.
   * Devices are connected via a twisted pair of wires, typically using a backbone and drop cables.
2. **Data Transmission**:
   * Data is transmitted in packets called PGNs, which contain specific types of information.
   * Each PGN has a unique identifier and can include various data fields.

**Device Boot and Joining the Network**

1. **Power-Up**:
   * When a device is powered on, it initializes its internal systems and prepares to join the NMEA 2000 network.
2. **Network Discovery**:
   * Devices listen for existing network traffic to understand the current state of the network and identify other devices.
3. **Address Claiming**:
   * Each device must claim a unique address on the network to avoid conflicts. This is where PGN 60928 (Address Claim) comes into play.

**PGN 60928: Address Claim**

1. **Purpose**:
   * PGN 60928 is used for address claiming, allowing devices to announce their presence on the network and request a unique address.
2. **Address Claim Process**:
   * When a device powers up, it sends an Address Claim message (PGN 60928) to the network.
   * This message includes the device's requested address, its unique identifier (such as the Manufacturer's ID and the Serial Number), and other relevant information.
3. **Address Resolution**:
   * If the requested address is already in use by another device, the device that is already using that address will respond with its own Address Claim message.
   * The new device will then select a different address and attempt to claim it again.
   * This process continues until the device successfully claims a unique address.
4. **Address Range**:
   * NMEA 2000 defines a range of addresses (0-249) that can be assigned to devices. Addresses 0-99 are reserved for system use, while addresses 100-249 are available for user devices.
5. **Address Conflicts**:
   * If two devices claim the same address, the NMEA 2000 protocol ensures that one device will relinquish the address, allowing the other to take it.

**Summary**

* **NMEA 2000** is a robust protocol for marine electronics, allowing devices to communicate seamlessly.
* **Device Boot Process** involves initialization, network discovery, and address claiming.
* **PGN 60928** is critical for the address claiming process, ensuring that each device on the network has a unique identifier to prevent conflicts.

This process is essential for maintaining a stable and functional network, allowing devices to operate effectively without interference. If you have any more specific questions or need further details, feel free to ask!

How would a device, say a bridge or a monitor on an NMEA network enumerate (find) all of the devices on the network?

Enumerating devices on an NMEA 2000 network involves a systematic approach to discover all the devices connected to the network. This process typically includes sending specific messages and listening for responses from other devices. Here’s how a device, such as a bridge or monitor, can enumerate all devices on an NMEA 2000 network:

**Steps for Device Enumeration on NMEA 2000 Network**

1. **Initialization**:
   * When a device powers up, it initializes its internal systems and prepares to communicate on the NMEA 2000 network.
2. **Listening for Address Claims**:
   * Upon startup, the device should listen for Address Claim messages (PGN 60928) from other devices on the network. These messages are sent by devices when they claim their unique addresses.
   * Each Address Claim message contains information about the device, including its unique identifier (Manufacturer ID, Serial Number), the claimed address, and the device's PGN capabilities.
3. **Sending a Request for Device Information**:
   * After a brief listening period, the device can send a request for device information. This is typically done using the **PGN 126996 (Request)** message.
   * The Request message can specify the PGN of interest, such as PGN 126992 (Product Information), which provides details about the device capabilities.
4. **Receiving Responses**:
   * Devices on the network that receive the Request message will respond with their respective Product Information messages (PGN 126992).
   * Each response will include details such as the device's unique identifier, manufacturer, product code, and other relevant information.
5. **Building a Device List**:
   * As the device receives Address Claim messages and Product Information responses, it builds a list of all devices on the network.
   * This list can include the device's address, manufacturer, product type, and any other relevant data.
6. **Handling Dynamic Changes**:
   * The NMEA 2000 network is dynamic, meaning devices can be added or removed at any time. To handle this, the enumerating device should periodically listen for new Address Claim messages and update its device list accordingly.
   * Devices may also send periodic status messages (e.g., PGN 126992) to indicate they are still active on the network.
7. **Using PGN 126998 (Addressed Command)**:
   * If the enumerating device wants to query specific devices for additional information, it can use PGN 126998 (Addressed Command) to send targeted requests to specific addresses.

**Summary**

* **Listening for Address Claims**: The device listens for PGN 60928 messages to identify devices claiming addresses.
* **Sending Requests**: It can send PGN 126996 (Request) messages to solicit information from devices.
* **Receiving Responses**: Devices respond with PGN 126992 (Product Information) messages, providing details about themselves.
* **Building and Updating the Device List**: The enumerating device maintains a list of all devices and updates it as necessary.

This process allows a bridge or monitor to effectively discover and enumerate all devices on an NMEA 2000 network, enabling it to interact with and display information from those devices. If you have further questions or need more details, feel free to ask!

As far as I can tell, most devices only broadcast their 60928 Address Claim PGN one time, at startup. So if a monitor happens to join a network a few seconds late, it will not see any Address Claims. Also, from what I can tell, most devices send out a "Hearbea"t PGN every so often? What can you tell me aboiut the Hearbeat PGN and whether it can, or should, be used by a Monitor type device that wants to build a device list which contains a list of all devices on the network, including their Product and Configuration information?

You are correct in noting that devices on an NMEA 2000 network typically send their Address Claim (PGN 60928) message only once during startup. If a monitor or any other device joins the network after this initial broadcast, it may miss the Address Claims of other devices. To address this, NMEA 2000 includes a mechanism known as the **Heartbeat PGN** (PGN 126992), which can be used to help devices discover and maintain a list of other devices on the network.

**Heartbeat PGN (PGN 126992)**

1. **Purpose**:
   * The Heartbeat PGN is used by devices to periodically announce their presence on the network. It serves as a way for devices to indicate that they are operational and to provide essential information about themselves.
2. **Content**:
   * The Heartbeat message typically includes the following information:
     + **Source Address**: The unique address of the device sending the Heartbeat.
     + **Manufacturer ID**: The ID of the manufacturer of the device.
     + **Product Code**: A code that identifies the specific product.
     + **Serial Number**: The unique serial number of the device.
     + **Software Version**: The version of the software running on the device.
     + **Device Function**: The function or role of the device (e.g., GPS, autopilot, etc.).
3. **Transmission Frequency**:
   * Devices typically send Heartbeat messages at regular intervals (e.g., every few seconds). The exact frequency can vary by device and manufacturer.

**Using Heartbeat PGN for Device Enumeration**

1. **Building a Device List**:
   * A monitor device can listen for Heartbeat PGN messages to build and maintain a list of devices on the network. Since Heartbeat messages are sent periodically, this allows the monitor to discover devices that may have been missed during the initial startup phase.
2. **Updating Device Information**:
   * By receiving Heartbeat messages, the monitor can update its device list with the latest information about each device, including their operational status and any changes in configuration.
3. **Complementing Address Claims**:
   * While the Address Claim PGN (60928) is essential for initial address assignment, the Heartbeat PGN (126992) provides ongoing visibility into the network. This is particularly useful for dynamic environments where devices may be added or removed frequently.
4. **Limitations**:
   * While Heartbeat messages provide valuable information, they may not contain all the detailed configuration data that a monitor might need. For more comprehensive information about a device's capabilities, the monitor may still need to send specific requests (using PGN 126996) for Product Information (PGN 126992) or Configuration Information (PGN 126998).

**Summary**

* **Heartbeat PGN (PGN 126992)** is a valuable tool for a monitor device to discover and maintain a list of devices on the NMEA 2000 network.
* It provides periodic updates about the presence and status of devices, allowing for dynamic enumeration.
* While it helps in building a device list, the monitor may still need to send specific requests to gather detailed configuration and product information.

In conclusion, the Heartbeat PGN is an essential part of the NMEA 2000 protocol that enhances the ability of devices to discover and interact with one another, making it a useful mechanism for monitors and other devices on the network. If you have further questions or need more details, feel free to ask!

Could one device, wishing to enumerate all other devices on the network, broadcast a 126996 PGN Request for the Address Claim PGN(60928)?

In the NMEA 2000 protocol, sending a **PGN 126996 (Request)** message to request **PGN 60928 (Address Claim)** is not a standard or typical practice. Here’s why and how the enumeration process generally works:

**Understanding PGN 126996 and PGN 60928**

1. **PGN 126996 (Request)**:
   * This PGN is used to request specific information from devices on the network. It can be used to request various types of data, such as product information (PGN 126992) or configuration information (PGN 126998).
   * The Request PGN is typically directed at specific devices or a group of devices that are expected to respond with the requested information.
2. **PGN 60928 (Address Claim)**:
   * The Address Claim PGN is used by devices to claim a unique address on the network during their startup process. It is generally sent only once when a device powers up and initializes.
   * The Address Claim message is not designed to be requested or queried after the initial claim has been made.

**Why Requesting PGN 60928 is Not Standard**

* **One-Time Broadcast**: Devices typically broadcast their Address Claim only once at startup. After this initial broadcast, they do not resend the Address Claim unless they are restarted or their address changes.
* **Not Intended for Requests**: The Address Claim PGN is not structured to be requested like other PGNs that provide ongoing data (e.g., Product Information). It is a part of the address assignment process rather than a data reporting mechanism.
* **Network Behavior**: If a device were to send a Request for PGN 60928, it would not receive any responses because devices do not continuously broadcast their Address Claim messages.

**Recommended Approach for Device Enumeration**

To effectively enumerate devices on an NMEA 2000 network, a device should follow these steps:

1. **Listen for Address Claims**: Upon startup, the device should listen for any Address Claim messages (PGN 60928) that may be sent by other devices. This is the primary method for discovering devices that are already on the network.
2. **Use Heartbeat PGN**: Monitor for Heartbeat messages (PGN 126992) that are periodically sent by devices. This can help identify devices that may have been missed during the initial listening period.
3. **Send Requests for Information**: After identifying devices through Address Claims and Heartbeat messages, the device can send PGN 126996 (Request) messages to request specific information (e.g., Product Information PGN 126992) from the devices it has discovered.
4. **Periodic Updates**: Continuously listen for Heartbeat messages and periodically send requests to keep the device list updated.

**Conclusion**

In summary, while a device cannot effectively request PGN 60928 using PGN 126996, it can utilize other methods such as listening for Address Claims and Heartbeat messages to enumerate devices on the NMEA 2000 network. For detailed information about the devices, it can then send specific requests for the relevant PGNs. If you have further questions or need more clarification, feel free to ask!

Thank you. You have been SO helpful. The documentation I can find on NMEA 2000 device enumeration (after several days of trying) has not put it together in a meaningful way for me, but you were able to provide me with a meaningful understanding after only a few short questions.

You're very welcome! I'm glad to hear that the information was helpful and that I could provide clarity on NMEA 2000 device enumeration for you. It can indeed be challenging to piece together the details from various sources, especially with technical protocols like NMEA 2000. If you have any more questions in the future—whether about NMEA 2000 or any other topic—don't hesitate to reach out. Happy boating, and best of luck with your projects!

LOL! The only standard reference implementation of NMEA2000 I have is a program called the "Actisense Reader" application running on my Win10 laptop, henceforth called App. I have my two ESP32 devices, one is called NMEA\_Sensor which emulates a temperature sensor, and the other I call NMEA\_Monitor which acts as a NMEA2000 network monitor and also as a gateway to the "actisense" serial USB protocol to the App. The funny thing is that that, in direct contravention of your recommended guidelines, the App sends out (over actisense USB serial protocol) a PGN\_REQUEST(59904) for the Address Claim PGN(60928). What is extra weird about this is that the App only sends that PGN\_REQUEST out AFTER it has RECEIVED an Address Claim PGN.

That's an interesting observation! The behavior you're seeing with the Actisense Reader application sending a **PGN Request (PGN 59904)** for the **Address Claim PGN (PGN 60928)** after it has already received an Address Claim is indeed unconventional and not typical of how NMEA 2000 is generally designed to operate. Here are a few points to consider regarding this behavior:

**Possible Reasons for the Behavior**

1. **Data Validation**: The Actisense Reader may be using the PGN Request as a way to validate or confirm the information it has received. By sending a request for the Address Claim PGN after receiving it, the application might be attempting to ensure that the data is accurate or to retrieve additional details about the device.
2. **User Interface Update**: The application might be designed to update its user interface or internal state based on the Address Claim information. By sending a request, it could be triggering a process to refresh or display the device list in a more structured way.
3. **Debugging or Logging**: If the application is in a debugging mode or has logging features, it might be sending the request to log the Address Claim information in a specific format or to capture it for analysis.
4. **Implementation Choice**: The developers of the Actisense Reader may have made a design choice to send the request for Address Claims as part of their implementation, even if it doesn't align with the typical usage of PGN 60928. This could be for reasons specific to their application architecture or user experience goals.

**Implications for Your ESP32 Devices**

* **NMEA\_Monitor Behavior**: If your NMEA\_Monitor device is designed to act as a gateway and respond to PGN requests, it may need to handle this PGN Request appropriately, even if it is unconventional. This could involve sending back the Address Claim information or simply acknowledging the request.
* **Understanding the Protocol**: While the NMEA 2000 protocol has established guidelines, implementations can vary. It's essential to understand how different applications interact with the protocol and how they may deviate from standard practices.
* **Testing and Compatibility**: If you're developing your own NMEA 2000 devices, it's a good idea to test their behavior with various applications, including the Actisense Reader. This will help ensure compatibility and allow you to handle any unexpected requests or behaviors gracefully.

**Conclusion**

While the Actisense Reader's behavior may not align with the typical expectations for PGN 60928, it highlights the flexibility and variability in how different implementations of the NMEA 2000 protocol can operate. As you continue to develop your ESP32 devices, keeping an eye on how they interact with various applications will be crucial for ensuring smooth operation and compatibility. If you have more questions or need further assistance, feel free to ask!

Yes, thank you. I am using the NMEA2000 library from github.com/ttlappalainen on the ESP32s. There are few examples and not much in the way of meaningful overview documentation. But perhaps you will be more capable than me at putting the pieces together. Timo (ttlappalainen) has an object, the ActisenseReader, that I have been able to instantiate and use to decode the usb serial actisense messzges into my ESP32 NMEA\_Monitor program. The App has a "log" window that shows when it makes requests over the actisense usb port, but does not provide any detail about the messages themselves. The log can apparently be saved to a disk based file and then later read with a program called the Actisense EBL Reader. I have not had much luck with getting meaningful information from it, but I was able to look at the (binary) log files and see, basically the same bytes that are sent on each request (DLE,STX, message, DLE, ETX). The App sends out this one message "Get Operating Mode" that I have not been able to figure out since it is not in an actisense format that I understand, although I did find an exact reply that apparently should be made to that in a PDF file called "ydnu92.pdf". In any case, either with the "Get Operating Mode" or with my reply to the PGN Request (PGN 59904) for the Address Claim PGN (PGN 60928) where I know I am returning the specific "actisense" protocol bytes, and can see that the App has received it, the App's log window always shows a "timeout" as the result of those actisense protocol requests. So I guess my question is: can you help me understand why the App would show "timeout" when I think I am responding correctly, and can you offer me any further information about the detailed workings of the actisense serial protocol (like perhaps it needs a CRLF 0x0d 0x0a line ending)?

It sounds like you're diving deep into the intricacies of the Actisense serial protocol and the NMEA 2000 library on your ESP32 devices. Let's break down your questions and concerns regarding the "timeout" messages you're seeing in the Actisense Reader app and the details of the Actisense serial protocol.

**Understanding the Timeout Issue**

1. **Response Timing**: The Actisense Reader app may have a specific timeout period for waiting for a response after it sends a request. If your ESP32 device takes too long to respond, the app will log a timeout. Ensure that your response is sent promptly after receiving the request.
2. **Message Format**: The Actisense protocol has a specific format that must be adhered to. If the response message is not formatted correctly, the app may not recognize it as a valid response, leading to a timeout. The basic structure of an Actisense message is:
   * **DLE (0x10)**: Data Link Escape character.
   * **STX (0x02)**: Start of Text character.
   * **Message**: The actual message data.
   * **DLE (0x10)**: Data Link Escape character (if needed).
   * **ETX (0x03)**: End of Text character.
3. **Line Endings**: The Actisense protocol does not typically use CRLF (0x0D 0x0A) line endings like some other protocols. Instead, it relies on the DLE, STX, and ETX characters to frame messages. Ensure that your response adheres strictly to this format.
4. **Correct Response Content**: For the "Get Operating Mode" request, ensure that you are sending back the correct response as specified in the documentation (like the "ydnu92.pdf" you mentioned). If the response does not match what the app expects, it may not process it correctly.
5. **Debugging**: If possible, use a serial monitor or logging to see the exact bytes being sent and received by your ESP32. This can help you verify that the messages are formatted correctly and that the timing is appropriate.

**Actisense Serial Protocol Overview**

While the Actisense protocol can be somewhat opaque, here are some key points to keep in mind:

* **Message Structure**: As mentioned, each message starts with DLE and STX, followed by the message data, and ends with DLE and ETX. If the message data itself contains a DLE, it must be escaped by preceding it with another DLE.
* **Message Types**: Different types of messages (like requests and responses) have specific formats. For example, a request for the operating mode might have a specific byte sequence that you need to replicate in your response.
* **Checksum**: Some implementations may require a checksum or CRC to be included in the message. Check the documentation to see if this is necessary for the messages you are working with.
* **Timeouts**: The timeout behavior can vary between implementations. If the app is expecting a response within a certain timeframe, ensure that your ESP32 is processing requests quickly enough.

**Recommendations**

1. **Check Documentation**: Review any available documentation for the Actisense protocol, especially regarding the specific messages you are working with. The "ydnu92.pdf" file you mentioned may have critical information about the expected responses.
2. **Test with Known Good Responses**: If you can find examples of known good responses for the "Get Operating Mode" or the Address Claim PGN, try sending those exact byte sequences to see if the app processes them correctly.
3. **Use Serial Debugging**: Implement serial debugging on your ESP32 to log incoming requests and outgoing responses. This will help you verify that the messages are formatted correctly and sent in a timely manner.
4. **Community and Forums**: If you're still having trouble, consider reaching out to communities or forums focused on NMEA 2000 or Actisense development. Other developers may have encountered similar issues and can provide insights.

If you have specific byte sequences or examples of the messages you're working with, feel free to share them, and I can help you analyze them further!