INTERFACING USER SOFTWARE WITH THE BYWIRE XGV

This application note explains the process of interfacing your software with the ByWire XGV ("XGV"), simplifying it into a step-by-step process.

WARNING: This application note alone does not contain sufficient information to operate the ByWire XGV. Read the entire XGV User Manual before operating.

Communicating with the XGV

User software interacts with the XGV over Ethernet, sending and receiving binary messages via UDP – the format of these binary messages happens to be JAUS. In this section, we'll explain how to exchange messages with the XGV. Subsequent sections illustrate how to use messages to perform common functions, such as obtaining information from the XGV or moving the steering wheel.

Transport Addresses and Logical Addresses

At the simplest level, a single user software module (a program) running on a user computer exchanges messages with a single firmware module (also a program) running on the XGV controller. The transport addresses involved in this communication are the IP addresses and ports used by the XGV controller and the user computer(s).

Since the XGV is JAUS-interoperable, its capabilities are grouped into individually addressable software entities called "components". A JAUS "component" is the smallest addressable software entity within JAUS; each component is given a 32-bit ID, referred to as its "JAUS ID", or "JAUS address". When sending a message to the XGV, user software must specify the JAUS ID of the component to which the message is destined.

There are six (6) JAUS components on the XGV Controller, which are fully detailed within the XGV User Manual. A Primitive Driver accepts the messages related to open-loop control, a Motion Profile Driver handles closed loop control of speed and curvature, a Velocity State sensor provides access to the on-board speed measurement, a Signals Driver allows control of the audio-visual driving indicators, an Error Manager performs health monitoring and error reporting tasks, and a Node Manager handles message routing.

Messages

In order to communicate to the XGV, your software exchanges JAUS messages with the components on the XGV. Each message contains a message ID indicating the particular message (for example, "Reset", or "Set Motion Profile"), and may also contain additional data. When sending a message, the sender specifies the JAUS ID (the 32-bit ID) of the particular JAUS component on the XGV to which the message is destined.

Additionally, any software communicating with the XGV must also have its own 32-bit JAUS ID, which is also included in each message. Each time the XGV receives a message, the source IP address and source JAUS ID are stored in a table, and outgoing messages to this JAUS ID are sent to the same IP address.

Structure of a JAUS Message

At a minimum, a JAUS message is made up of two parts, as indicated below.

JAUS Header (16 bytes)	Message Data (0 to 4080)

The JAUS header includes such things as the command code (message id), destination JAUS ID, source JAUS ID, sequence number, ack/nak flags, and a number of other items. If the command code specifies additional data, the message data is also included. For example, a "Set Wrench Effort" message would contain additional fields specifying the throttle, brake, and steering.

When a JAUS message is sent over UDP (as on the XGV), an additional JUDP (JAUS over UDP) transport header is added, as shown below. The transport header includes fields such as the JUDP version and message length, and allows multiple JAUS messages to be sent in a single UDP packet.

JUDP Transport Header (5, 8, or 0 bytes)	JAUS Header (16)	Message Data (0 to 4080)

The XGV supports two different transport headers (SAE AS-5669 v1.0 and OPC/ETG JAUS01.0), selectable via the System Configuration Utility. For applications where JAUS-interoperability is less important, a third option to disable the transport header so that UDP packets contain only raw JAUS messages (16-byte header plus data) is also available. For all the examples in this application note, we use the AS-5669 v1.0 transport header.

Steps to Communicate with the XGV

At a minimum, the following steps are necessary to communicate with the XGV. It is assumed that the computer running the user software has a valid IP address and is on the same subnet as the XGV controller.

1. Before starting, assign a 32-bit JAUS ID to your software. This ID must be different from the JAUS IDs of the components running on the XGV. If multiple computers on the network will interface the XGV simultaneously, each needs its own unique JAUS ID.

2. Bind a UDP socket to port 3794, in order to receive messages from the XGV. The XGV sends all outgoing messages to port 3794, the IANA-assigned port for JAUS. The source port used by your software when sending outgoing messages is not important.

3. Your software will begin receiving a periodic JAUS message from the XGV, sent via UDP broadcast. This message, "Report Heartbeat Pulse", contains the JAUS ID of the Node Manager component as its source and is simply a beacon that proclaims "I am alive". The JUDP transport header included with this message can be specified via the System Configuration Utility.

4. Assemble the JAUS message to send, specifying the message ID (referred to in JAUS as the "command code"), message data if applicable, the source JAUS ID you assigned your software, and the destination JAUS ID of the component on the XGV for which the message is destined.

5. Serialize the message, add the transport header, and send it to the IP address of the XGV, to port 3794. Note that your software can auto-detect the IP address of the XGV from the periodic Report Heartbeat Pulse received from the XGV.

6. Once the XGV receives the message, it will store your software's JAUS ID and IP address in a table. If an outgoing response message is generated, it will be sent to the last known IP address for the relevant JAUS ID.

Example: Obtaining Periodic Data from the XGV

In this example, we will demonstrate how user software can subscribe for periodic updates from the XGV. In JAUS, subscriptions for periodic data are called "service connections".

In short, this will amount to requesting a periodic update for a particular message, from a particular component. A table of the available messages for each component is provided in the XGV User Manual, but suppose for this example that we require periodic updates of the vehicle speed. The Velocity State Sensor component supports service connections for the Report Velocity State message, which contains the necessary data.

As illustrated below, a service connection is initiated with a "Create Service Connection" message specifying the desired message, and the desired periodic update rate. The provider will respond with a "Confirm Service Connection" message indicating whether the request was allowed or denied, and the confirmed periodic rate. The message is then transmitted to the requestor until a "Terminate Service Connection" message is received.

Let's now work through this process for our example.

1. We'll assume you've already assigned a valid 32-bit JAUS ID to your software as previously described, and opened a UDP socket on port 3794 to receive incoming messages from the XGV.

2. Build a "Create Service Connection" message, command code 0x0008. This message has three data fields: (a) the command code for the message that will flow on the service connection, (b) the desired update rate, and (c) the presence vector, for messages with optional fields. Using the values 0x4404 (Report Velocity State), 20 Hz, and 0x0001

respectively, the message data is: 0444 AF04 0100 0000, expressed in Hex. The value of the presence vector is determined by the specification for the Report Velocity State message. Since we want the Velocity X field, we set bit 0, for a presence vector of 0x0001.

3. Add the 16-byte JAUS header. If we set the message priority to "6", the ack/nak request to "response not required", the service connection flag to "true", the experimental message flag to "false", the JAUS version to "3.2 thru 3.3", the sequence number to "0", the destination ID to subsystem 1, node 1, component 42, instance 1, and the source ID to subsystem 2, node 1, component 2, instance 1, the message is now: 4602 0800 012A 0101 0102 0102 0800 0000 0444 AF04 0100 0000 in Hex.

4. Add the 5-byte AS-5669 transport header for JUDP, version 1. The message now looks like this: 0100 0000 1846 0208 0001 2A01 0101 0201 0208 0000 0004 44AF 0401 0000 00 in Hex. Note: Unlike every other field in JAUS, which is encoded Little Endian, the message length encoded in the AS-5669 header is Big Endian.

5. Send this message via UDP, to the IP address of the XGV, to port 3794.

6. The XGV's Velocity State Sensor will respond with a Confirm Service Connection message (command code 0x0009). Your software should receive this message and parse it. If the service connection was successfully created, this confirmation message will contain a service connection ID (called the "instance ID"). Your software should store this ID as it will be used later to terminate the service connection. The Confirm Service Connection message (with 16-byte and 5-byte headers) might look like this, if the service connection was confirmed at 20 Hz, with instance ID "1": 0100 0000 1646 0209 0001 0201 0201 2A01 0106 0000 0004 4401 B104 00 in Hex.

7. The XGV will then begin sending the Report Velocity State message (command code 0x4404) at 20 Hz, to the JAUS ID that requested the service connection. Your software should receive this message and parse it to obtain the vehicle speed.

8. When the service connection is no longer desired, your software should send a "Terminate Service Connection" message (command code 0x000C) to the XGV Primitive Driver. If the instance ID is "1", and the command code is "0x4404", the message looks like this (with 16-byte and 5-byte headers): 0100 0000 1346 020C 0001 2A01 0101 0201 0203 0000 0004 4401 in Hex.

9. At this point, the XGV will discontinue sending the Report Velocity State message.

Example: Send a command to move the steering wheel in open-loop mode

In this example, we will demonstrate how to put the XGV into open-loop mode and command the steering wheel to move.

Recall from the XGV User Manual that there are multiple vehicle modes. The first mode, the control mode, determines what type of commands messages the XGV should execute; this mode is commanded by the user via JAUS commands. The second mode, the ready mode, is a software-controlled pause, also controlled by the user via JAUS commands. The final mode, the safety mode, is determined based on the status of the SafeStop, the manual override, internal health monitoring, and other safety devices.

As illustrated below, we will first command the XGV into "wrench effort" control mode, by activating the Primitive Driver. We will then enable the Primitive Driver by setting its component status to "ready" (setting the ready mode). Next, we will send the Set Wrench Effort message (containing throttle, braking, and steering efforts) to the XGV Primitive Driver. Finally, the last step cannot be accomplished by software – we must disengage the manual override, and place the SafeStop wireless emergency stop in either "bypass" or "run".

Let's now work through this process for our example.

1. We'll assume you've already assigned a valid 32-bit JAUS ID to your software as previously described, and opened a UDP socket on port 3794 to receive incoming messages from the XGV.

2. First, build a "Request Component Control" message destined for the Primitive Driver. This message contains the authority level of the requesting component – that's the authority level of your user software. For this example, we'll set the authority to "3", and set the message priority to "6", the ack/nak request to "no response required", the service connection flag to "false", the experimental message flag to "false", the JAUS version to "3.2 thru 3.3", the sequence number to "0", the destination ID to subsystem 1, node 1, component 33, instance 1, and the source ID to subsystem 2, node 1, component 2, instance 1. Our message, with 16-byte header will look like this: 0602 0D00 0121 0101 0102 0102 0100 0000 03 in Hex.

3. Add the 5-byte AS-5669 transport header, version 1. Our message looks like this: 0100 0000 1106 020D 0001 2101 0101 0201 0201 0000 0003 in Hex. Note: Unlike every other field in JAUS, which is encoded Little Endian, the message length encoded in the AS-5669 header is Big Endian.

4. Send the message via UDP, to the IP address of the XGV, to port 3794.

5. The XGV will respond with the "Confirm Component Control" message (command code 0x000D), indicating whether control was accepted or denied. If control was accepted, the message (with 16-byte and 5-byte headers) might look like this: 0100 0000 1106 020F 0001 0201 0201 2101 0101 0000 0000 in Hex.

6. If control was accepted, the XGV is now in Wrench Efforts control mode. We can now start sending Set Wrench Effort messages (command code 0x0405) to control the throttle, braking, and steering, however a software pause will still be in effect.

7. To clear the software pause, send a “Resume” message (command code 0x0004) to the Primitive Driver to set the ready mode to “ready”. The XGV is now ready to execute the incoming Set Wrench Effort commands, however the manual override and emergency stop may be overriding the safety mode.

8. To enable drive-by-wire operation, we must finally disengage the manual override and place the SafeStop in “run”. After the safety mode changes to “drive-by-wire”, the XGV will begin executing the commanded throttle, braking, and steering commands in the Set Wrench Effort messages.

An example Set Wrench Effort message setting throttle to 0%, steering to -25%, and brake to 50% might look like this (with 5-byte and 16-byte headers): 0100 0000 1706 0205 0401 2101 0101 0201 0207 0000 0061 0000 0000 E080 in Hex.