1 BYWIRE™ XGV EMULATOR OVERVIEW

The ByWire™ XGV Emulator is a tool designed to assist in the development of code that will interface with the ByWire™ XGV. The Emulator is an executable program that runs on a host computer and emulates the functionality of the XGV. This allows a user to test code designed to interface with the XGV in a controlled lab environment without the need for an actual XGV. The Emulator DOES NOT perform vehicle dynamics simulation and will NOT provide an indication of the performance of higher-level control algorithms. It will, however, provide a debugging tool to analyze all JAUS message traffic into and out of the XGV, allowing a developer to verify that any message sent to the XGV is in fact parsed, routed, and handled as expected. Furthermore, all the mode control, component status, component control, and safety logic is present in the Emulator, allowing the user to verify the appropriate sequences of messages needed to begin actively controlling and getting feedback from the ByWire™ XGV.

2 INSTALLATION

The installer for XGV Emulator will install the following items:

1. TORC XGV EMULATOR.exe
2. LabVIEW Runtime Engine 8.5.1 (if necessary)
3. Supporting files

The XGV Emulator is run by clicking Start->All Programs->TORC->TORC ByWire XGV Emulator

3 PROGRAM OPERATION

NETWORK SETUP:

Currently, the XGV Emulator must run on a network-enabled computer without any other accompanying JAUS Node Manager. This computer must in essence act as the XGV controller, and communication with this computer must occur over the network. The following diagram illustrates the recommended setup.

NOTE: TORC hopes to provide future versions of the Emulator that do not require the existence of a network, and are able to communicate with JAUS components on the same computer.

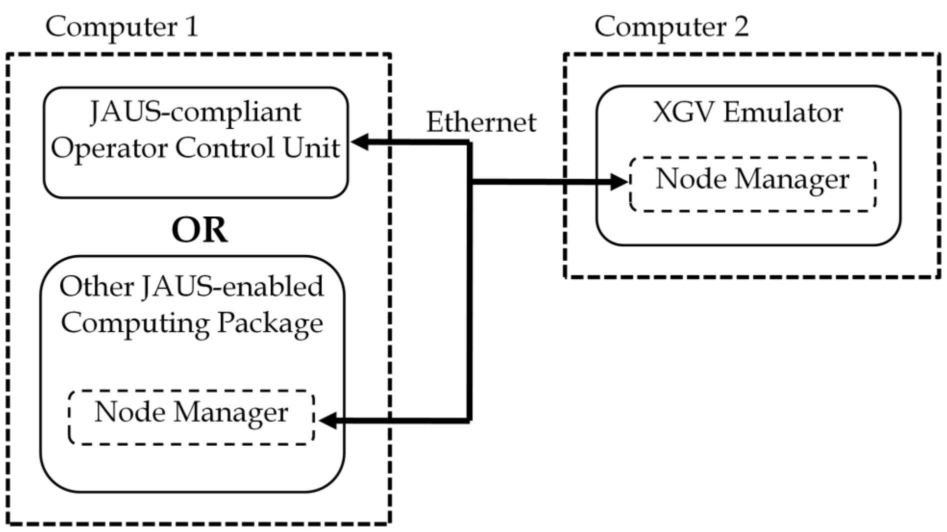
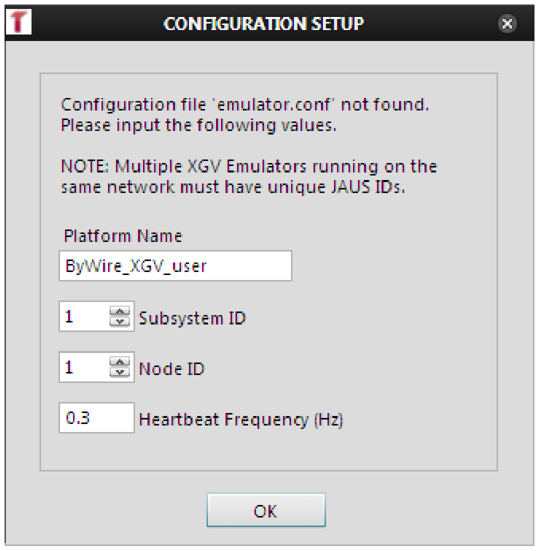

Computer 1
Computer 2
JAUS-compliant Operator Control Unit
OR
Other JAUS-enabled Computing Package
Node Manager
Ethernet
XGV Emulator
Node Manager

Figure 1. Network configuration for interfacing with the XGV Emulator

The first time you run the emulator, you will be prompted to setup a default configuration. The Platform Name, Subsystem ID, Node ID, and Heartbeat Frequency for the Emulator must be defined. If multiple Emulators are running on the same network, they must each have a unique JAUS ID. A unique Platform Name will also help to identify different Emulators running simultaneously.


CONFIGURATION SETUP

Configuration file 'emulator.conf' not found.
Please input the following values.

NOTE: Multiple XGV Emulators running on the same network must have unique JAUS IDs.

Platform Name: ByWire_XGV_user

Subsystem ID: 1

Node ID: 1

Heartbeat Frequency (Hz): 0.3

Figure 2. Configuration setup screen

The Emulator will then store this configuration as an *emulator.conf* file in the program directory. To later modify the configuration, simply open *emulator.conf* in a basic text editor, change the appropriate values, and restart the Emulator.

THEORY OF OPERATION:

To accurately emulate the behavior and functionality of the XGV, the Emulator runs the majority of code that runs on the XGV embedded controller. Code has been added to provide a user-friendly interface with a variety of debugging tools as well as the ability to 'mimic' important hardware inputs to the XGV control code, such as the in-cab transmission selector, parking brake lever, and SafeStop controls.

Therefore, for the XGV Emulator to begin accepting wrench effort control message the same prerequisites as on the vehicle must be satisfied: the engine must be on, the parking brake must be released, all the doors/liftgate must be closed, the SafeStop must be in Run, no local E-Stop's can be pushed, the Primitive Driver must be set to 'Ready' and be controlled by the commanding component, etc. If any of these conditions are not met, the Error Manager will send out error codes over JAUS just as it would on the XGV.

Unlike the actual XGV the XGV Emulator also provides a visual interface for understanding the internals of the controller without having to receive and parse the outgoing JAUS messages. The JAUS messages are still being sent out, however, providing the developer with the ability to verify that their interpretation of those messages match what is being sent.

The following figure illustrates the XGV Emulator theory of operation.

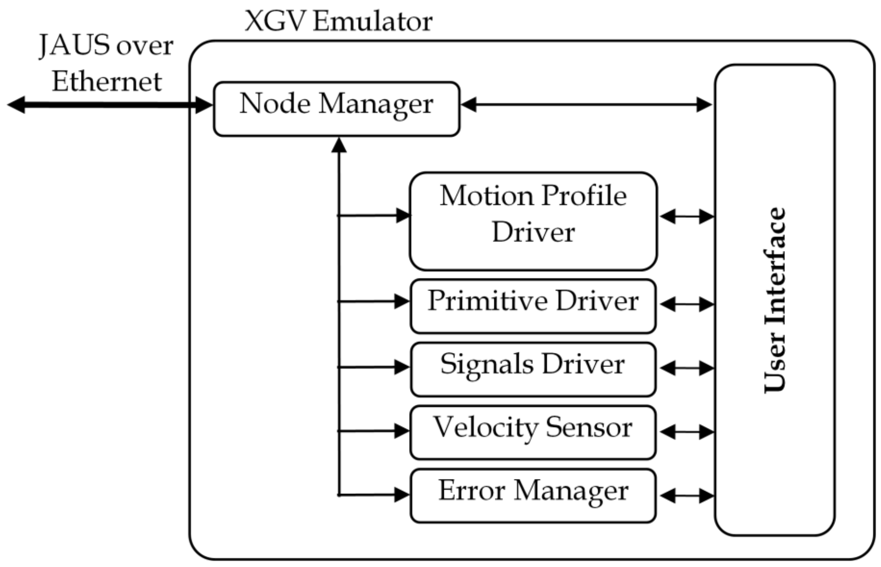


Figure 3. The user can emulate important hardware inputs and monitor the status of each JAUS component via the User Interface.

USER INTERFACE LAYOUT

The Emulator User Interface is composed of three separate areas, the Vehicle page, Message Routing page, and Component Diagnostics page.

The Vehicle Page contains the simulated hardware controls and a status summary of important information about the 'emulated' vehicle. In the *Simulated Controls* area, the user can change:

1. Speed that will be reported in the Report Velocity State message from the Velocity State Sensor
2. Amount of fuel in the tank
3. Actual shifter position in the cab of the vehicle
4. Engine ON/OFF status
5. Parking brake status
6. Door/Liftgate status
7. Emergency Manual Override button
8. Local E-Stop button
9. SafeStop Disable button
10. SafeStop RUN/PAUSE
11. SafeStop link status
12. SafeStop power

NOTE: TORC will be adding a basic simulation package in the future that will provide simple dynamics modeling to approximate vehicle speed.

In the *Status Information* area the user can get a quick summary of important states on the vehicle. The user can see:

1. Steering, throttle, and brake percentages
2. Actual transmission gear
3. Overall Control Mode
4. Overall Ready Mode
5. Overall Safety Mode
6. Turn signal status
7. Headlights status
8. Hi-Beams status
9. Foglight status
10. Error Manager status

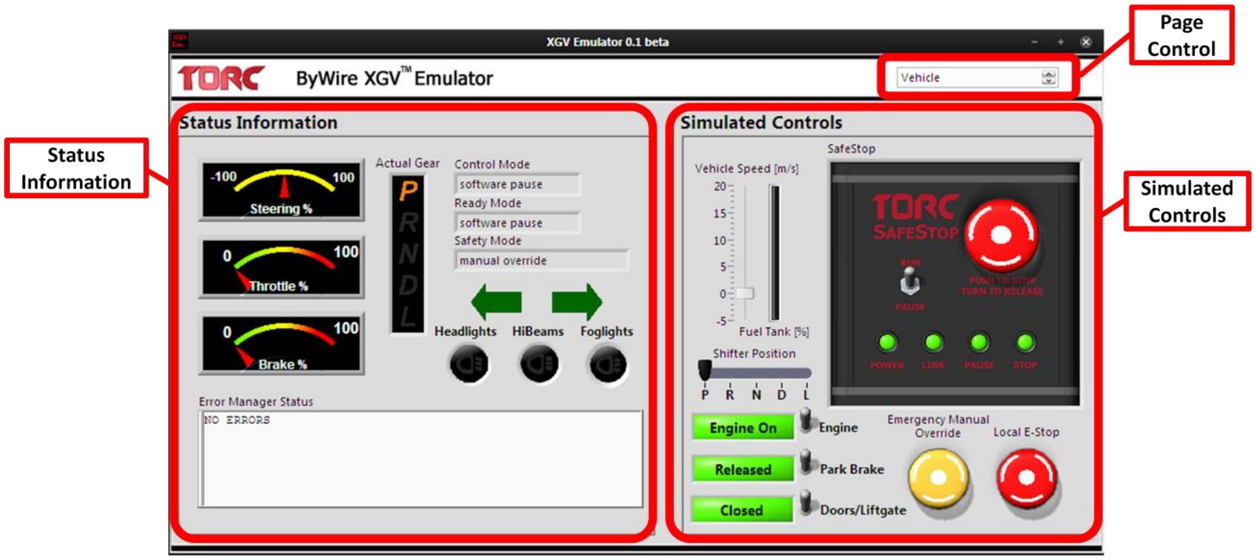
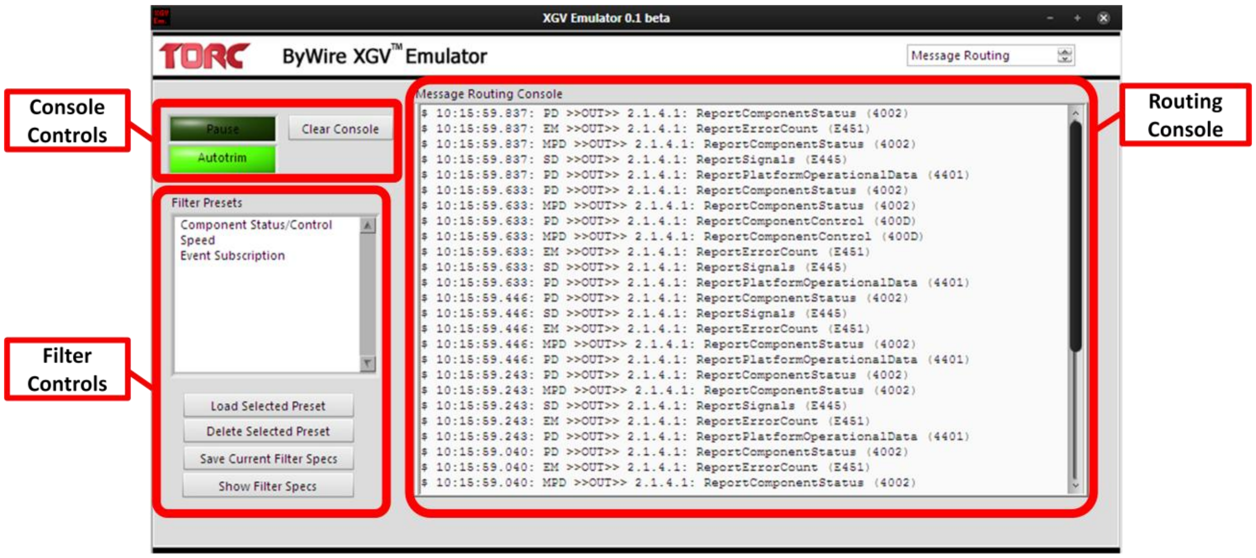


Figure 4. Vehicle Page

The Message Routing Page provides the ability to monitor every single JAUS message entering or leaving the vehicle in time sequence. On the *Message Routing Console,* every JAUS message being received or sent by a component is displayed with the timestamp the message was sent or received, the destination and source with an indication of whether or not the message was coming into or out of the emulator, the message name, and the message code.



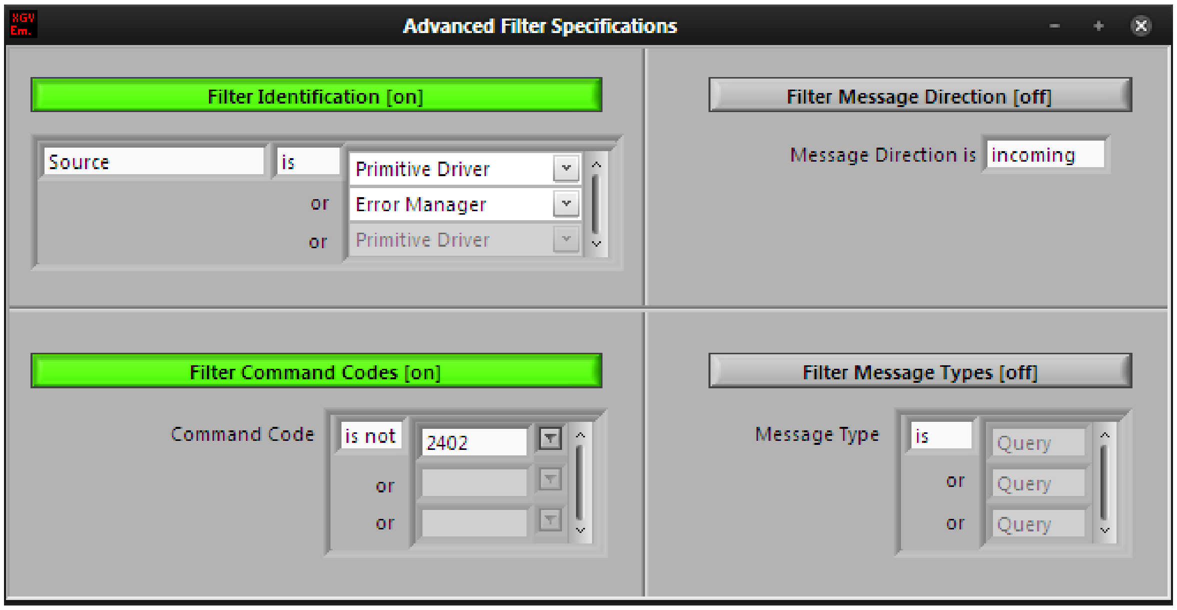
```
$ 10:15:59.837: PD >>OUT>> 2.1.4.1: ReportComponentStatus (4002)
$ 10:15:59.837: EM >>OUT>> 2.1.4.1: ReportErrorCount (E451)
$ 10:15:59.837: MPD >>OUT>> 2.1.4.1: ReportComponentStatus (4002)
$ 10:15:59.837: SD >>OUT>> 2.1.4.1: ReportSignals (E445)
$ 10:15:59.837: PD >>OUT>> 2.1.4.1: ReportPlatformOperationalData (4401)
$ 10:15:59.633: PD >>OUT>> 2.1.4.1: ReportComponentStatus (4002)
$ 10:15:59.633: MPD >>OUT>> 2.1.4.1: ReportComponentStatus (4002)
$ 10:15:59.633: PD >>OUT>> 2.1.4.1: ReportComponentControl (400D)
$ 10:15:59.633: MPD >>OUT>> 2.1.4.1: ReportComponentControl (400D)
$ 10:15:59.633: EM >>OUT>> 2.1.4.1: ReportErrorCount (E451)
$ 10:15:59.633: SD >>OUT>> 2.1.4.1: ReportSignals (E445)
$ 10:15:59.633: PD >>OUT>> 2.1.4.1: ReportPlatformOperationalData (4401)
$ 10:15:59.446: PD >>OUT>> 2.1.4.1: ReportComponentStatus (4002)
$ 10:15:59.446: SD >>OUT>> 2.1.4.1: ReportSignals (E445)
$ 10:15:59.446: EM >>OUT>> 2.1.4.1: ReportErrorCount (E451)
$ 10:15:59.446: MPD >>OUT>> 2.1.4.1: ReportComponentStatus (4002)
$ 10:15:59.446: PD >>OUT>> 2.1.4.1: ReportPlatformOperationalData (4401)
$ 10:15:59.243: PD >>OUT>> 2.1.4.1: ReportComponentStatus (4002)
$ 10:15:59.243: MPD >>OUT>> 2.1.4.1: ReportComponentStatus (4002)
$ 10:15:59.243: SD >>OUT>> 2.1.4.1: ReportSignals (E445)
$ 10:15:59.243: EM >>OUT>> 2.1.4.1: ReportErrorCount (E451)
$ 10:15:59.243: PD >>OUT>> 2.1.4.1: ReportPlatformOperationalData (4401)
$ 10:15:59.040: PD >>OUT>> 2.1.4.1: ReportComponentStatus (4002)
$ 10:15:59.040: EM >>OUT>> 2.1.4.1: ReportErrorCount (E451)
$ 10:15:59.040: MPD >>OUT>> 2.1.4.1: ReportComponentStatus (4002)
```

Figure 5. Message Route Page

Typically, the Emulator will be routing a large number of messages in and out. This can make it difficult to monitor for a specific sequence of messages or messages having to do with a specific mode of control. Therefore the Message Routing Page utilizes a customizable filter. Controls for the filter are to the left of the Message Routing Console. To view and change the filter specifications, click on the *Show Filter Specs* button. This will pop-up the Advanced Filter Specifications page. There are 4 different filter types which can be combined.

1. Filter by Identification (ID) – Uses the Destination and Source ID of the message
2. Filter by Command Code – Uses the message Command Code in hexadecimal format
3. Filter by Message Direction – Incoming/Outgoing based on whether or not the destination/source is on the vehicle
4. Filter by Message Types – Uses different categories or groups of messages

NOTE: Filter by Message Types is not currently supported but will be added at a future date.


Advanced Filter Specifications
Filter Identification [on]
Filter Message Direction [off]
Filter Command Codes [on]
Filter Message Types [off]

Source is Primitive Driver
or Error Manager
or Primitive Driver

Message Direction is incoming

Command Code is not 2402
or
or

Message Type is Query
or Query
or Query

Figure 6. Advanced Filter Specification window with two filters activated

Different filter specifications can be saved, loaded, and deleted as Presets via the filter controls on the Message Routing Page. These Presets are saved in the filter_presets.pre file in the program directory and are automatically loaded on startup.

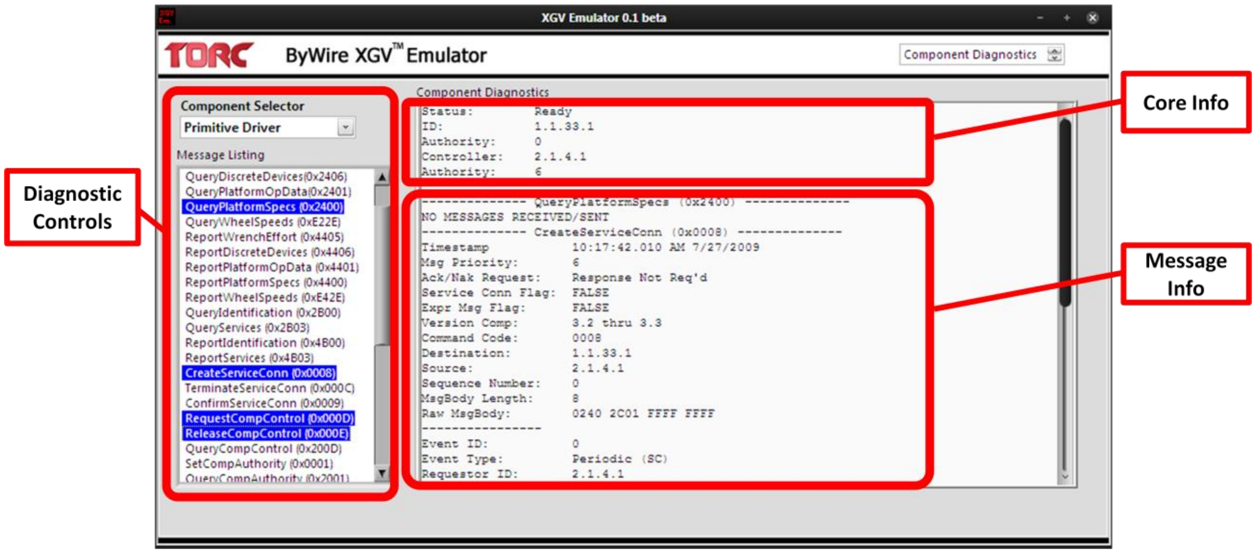
The console controls provide management of the Message Routing Console. The user can clear the console at any time, and can Pause the console to examine routing information at their leisure.

NOTE: Pausing the Message Routing Console does NOT pause the actual sending or receiving of messages in the Emulator, only the display itself. While paused, the Emulator will continue receiving and responding to messages like normal.

The Component Diagnostics Page provides a component-centric debugging interface. All of the XGV software components are listed in the Component Selector. Selecting a component automatically populates the Message Listing with all the messages that component supports.

This includes both the JAUS core messages and the component specific messages. Selecting a component also displays that component's core dynamic information at the top of the Component Diagnostics window. This includes the component's status, JAUS ID, authority, controller JAUS ID if present, and the controller's authority.

The user can select any combination of messages to inspect. For each message selected the Components Diagnostics window will display information about the last message that was either sent or received. This includes all the JAUS message header information, the raw message body in hexadecimal, and the parsed, human-readable information in the message body.



QueryDiscreteDevices(0x2406)
QueryPlatformOpData(0x2401)
QueryPlatformSpecs (0x2400)
QueryWheelSpeeds (0xE22E)
ReportWrenchEffort (0x4405)
ReportDiscreteDevices (0x4406)
ReportPlatformOpData (0x4401)
ReportPlatformSpecs (0x4400)
ReportWheelSpeeds (0xE42E)
QueryIdentification (0x2B00)
QueryServices (0x2B03)
ReportIdentification (0x4B00)
ReportServices (0x4B03)
CreateServiceConn (0x0008)
TerminateServiceConn (0x000C)
ConfirmServiceConn (0x0009)
RequestCompControl (0x000D)
ReleaseCompControl (0x000E)
QueryCompControl (0x200D)
SetCompAuthority (0x0001)
QueryCompAuthority (0x2001)

```
Status:        Ready
ID:            1.1.33.1
Authority:     0
Controller:    2.1.4.1
Authority:     6
```

```
-------------- QueryPlatformSpecs (0x2400) --------------
NO MESSAGES RECEIVED/SENT
-------------- CreateServiceConn (0x0008) --------------
Timestamp             10:17:42.010 AM 7/27/2009
Msg Priority:         6
Ack/Nak Request:      Response Not Req'd
Service Conn Flag:    FALSE
Expr Msg Flag:        FALSE
Version Comp:         3.2 thru 3.3
Command Code:         0008
Destination:          1.1.33.1
Source:               2.1.4.1
Sequence Number:      0
MsgBody Length:       8
Raw MsgBody:          0240 2C01 FFFF FFFF
----------------
Event ID:             0
Event Type:           Periodic (SC)
Requestor ID:         2.1.4.1
```

Figure 7. Component Diagnostics Page with 4 messages selected for display

In figure 7 the Primitive Driver has been selected, and from the Component Diagnostics window we can see that it is in the Ready State and is being controlled by a component with JAUS ID 2.1.4.1 which happens to be the XGV OCU. Four different messages have been selected for display, but only two are currently visible in the Component Diagnostics window without scrolling down. The first message, QueryPlatformSpecs (0x2400) has yet to be sent or received, so no information is present to be displayed. The second message, CreateServiceConn (0x0008) has been received by the Primitive Driver from the XGV OCU and the parsed information such as the Event ID and Type as well as the raw message body is shown.

4 QUESTIONS/FEEDBACK?

The XGV Emulator is a tool designed to make YOUR life as an XGV developer easier. While still in beta form, the Emulator should provide some useful tools for integrating with an actual XGV. As a beta tester, please feel free to compile comments/questions/feature requests and send them to support@torctech.com. Found a bug? Report it!