The following was extracted from;

"Electric Brakes ABS Device Profile & Test Plan"

This document provides an overview of the communications interfaces utilizes by the EABS Controller designed by Real Time Objects & Systems, LLC (RTOS) and is provided as an example of a portion of the contents of an Extensible Realtime Application Environment (XRAE) "Device Profile". A Device Profile is the core document supporting any control device, which contains; requirements, hardware operational descriptions, User Application Programs (UAP) operational descriptions, object descriptions, system operational descriptions, test plans, test results and references other documents such as hardware schematics, UAP drawings, test tools, test log files and so forth. The User Application Program drawings are "software schematics" of each application program where the executable code is a one to one mapping of function icons to function code.

The RTOS Process intentionally deviates from traditional product development documentation as XRAE assets are maintained throughout the development process and "match the implementation" when product is released and thus utilized for product support purposes as well as reuse of these assets for other new product developments. XRAE improves product quality and significantly reduces product development time by utilizing a common architecture (design through implementation), implementation rules and massive reuse of "drag and drop" assets due to the XRAE architecture. Objects defined and utilized in products created in 2004 are still utilized today, although supporting documentation has improved through the years and a few methods have been added to a few of the initial object classes.

The contents of this document shall demonstrate some of the naming convention rules of XRAE that facilitate reuse of said assets.

6 Networking

XRAE control devices communicate with other control devices via the Trailer CAN (TCAN) network utilizing various Real Time Objects & Systems, LLC (RTOS) protocols. Specifically the EABS Controller currently communicates with User Interfaces (UI), the tow vehicle and personal computers.

Relative to the EABS Controller communicating with UI devices the following types of messages are supported;

- Configuration messages for tuning EABS Controller and enabling/disabling various functionality
- Monitor and test messages used to diagnose the operation of sensors, brake magnets, brake controller, power system, EABS Controller operation and so forth.

Communication interfaces utilize various standardized Extensible Realtime Application Environment (XRAE) protocols;

- Explicit/Interpreted messaging protocols which interact with instances of public (network accessible) object classes.
- IO message protocols are messages where the CAN Identifier value determines the content of the message where the data at a specific offset within the message has a fixed meaning including;
    - a protocol for parameter value editing,
    - protocols for status/data monitoring screens,
    - protocols for debug/diagnostic messages,
    - protocols for tow vehicle communications,
    - protocols for sharing data with other trailer control devices.

The IO Message protocols are defined for each control device, where three of the following standardized protocols are used across the majority of XRAE vendors' control devices to simplify product support.

- The "Command Message" application protocols are used by UI and test devices to select the production of specific IO message data. The request message contains the "requested screen number" and an incrementing

transaction identifier (XID), where upon receipt the EABS Controller produces the respective Command Response Message. See "Display.h" for declaration values of the Screen Numbers for a specific control device. Thus when a new screen is selected, the screen number in the command request message changes and the EABS Controller immediately changes the contents of the command response message being sent and often imitates cyclic production of a Debug Message.

- "Parameter Editing Message" protocols are used to edit the values of instances of the Parameter Object Class by a UI device, utilized to configure/calibrate the operational behavior of a control device. See "UAP_Parameter.h" for parameter instances supported for a specific control device.
- "Fault Queue Message" protocols are defined to retrieve any active fault codes, unique to each control device, in a common way. See "System_Defines.h" for a list of fault codes for a specific control device.
- "Debug Message" protocols are determined by value contained within a specific instance of a Parameter Object class, where Debug Messages are produced cyclically.
- "Vehicle Message" protocols are defined to consume messages sent from a control device in the tow vehicle, typically plugged into the OBDII connector.

Although the UAP that supports the respective IO message can provide any desired communications behavior, data monitoring IO messages are generally produced cyclically at an interval dependent upon the value contained in the TIMERELAY_Preset property of the respective On Delay Timed Relay object instance within the respective communications UAP. The following sub-sections describe the UAPs that support these selected protocols.

NOTE: XRAE utilizes its object classes and UAPs to provide functionality unique to a specific control device as well as all other internal functionality. For example; UAP_FAULTS.c adds/removes fault codes from a FIFO_Queue object where a Command Message then provides a standardized interface to retrieve these fault codes from any control device, from any vendor, in any industry. By viewing UAP_FAULTS.doc shows the condition(s) that set and remove each fault code (PLC logic on steroids).

6.1 Explicit Message Protocol

The Explicit network interface provides access to selected object classes within a control device where EABS Controller consumes explicit request messages at the default CAN_ID = 0x540 + Address = 0x54B and responds to these messages at CAN_ID = 0x500 + Address = 0x50B. The request and response protocols are shown in Table 6-1 and Table 6-2 where the following values of the variables below are the same for all XRAE control devices within all vendors products in all industries;

- Class Identifier values are unique to the object class. See "SystemDefines.h" values reserved by the System Architecture (XRAE).
- Service Identifier values are reserved for all object classes. See "SystemDefines.h" values reserved by XRAE.
- Instance Identifiers are unique to the specific control device except for a few instances that are reserved by XRAE to uniquely identify a control device. See "UAP_*ClassName*.h" for device specific values.
- Attribute/Property Identifiers are unique to the object class where consistency across object classes for property identifiers is encouraged for "programmable devices". See "*ClassName*.h" for values reserved by XRAE.

NOTE: XRAE utilizes standardized names when implementing various User Application Programs (UAP) common to most control devices, solely for the purpose of simplifying product support/re-use where an engineer not familiar with a specific control device intuitively knows where to look to make modifications to a product they have never supported. Most new products reuse and modify an existing UAP to provide similar functionality when the actual logic within said UAPs are similar. This includes the ability to reuse and edit existing test plans, test code, UAP operational descriptions and other support assets reducing development time and thus providing consistent behavior across all products. XRAE defines a development process, control architecture, control objects, network protocols, naming standards, coding standards and so forth all which complement one another to facilitate asset re-use, improve product quality, reduce development time and thus reduce development time.

XRAE control systems seldom utilize explicit message protocols for control purposes but they are utilized by support tools (personal computer software) to configure, program and/or test a control device. The explicit protocol is supported

by "UAP_CAN_EXPLICIT_RTOS.c" exactly consistent with logic drawing "UAP_CAN_EXPLICIT_RTOS.doc".

Byte	Data Type	Description	Value	DESCRIPTION
x	UINT16	CAN Identifier	0x54B	PC → EABS Controller
x		Length	8	
0	UINT8	Class Identifier		See "SystemDefines.h"
1	UINT8	Service Identifier		See "SystemDefines.h"
2	UINT8	Instance Identifier		See "UAP_*ClassName*.h"
3	UINT8	--Service specific --		Often Property Identifier
4	UINT8	--Service specific --		
5	UINT8	--Service specific --		
6	UINT8	--Service specific --		
7	UINT8	--Service specific --		

Table 6-1: Explicit Message Request Protocol

NOTE: The ENGINE_CAN_ClassParser() method in the Engine Object in all XRAE control devices consume explicit request messages where this method then calls the class parser for the selected object classes. For example, if the Class Identifier in the Explicit Request Message is CID_RTOS_TIME_RELAY [0x03] this method then calls TIMERELAY_Parser() which generates an Explicit Response Message, if the Class Identifier in Explicit Request Message contains the value CID_RTOS_DATA_BRANCH [0x19] ENGINE_CAN_ClassParser() then calls DATABRANCH_Parser(), and so on. If a vendor chooses to support this public interface to an object class a call to the respective class parser is included in ENGINE_CAN_ClassParser() and if external access is not provided a call is not included and the respective code for the *objectclass*_Parser() is commented out. Seldom is external access provided to all object classes except in externally programmable control devices. Similarly if a new UAP drawings includes an icon for an object class not currently supported by a control device, the engineer "drags and drops" three files into the project to support it. For example, assume a selector switch object icon is used in a new UAP the following files are then dropped into the project; "Selector.c" (object function conde and declarations), "Selector.h" (object class declarations) and "UAP_Selector.h" (instance declarations referenced in UAPs). See "realtimeobjects.net/training" for training materials for creating new control devices.

Byte	Data Type	Description	Value	DESCRIPTION
x	UINT16	CAN Identifier	0x50B	EABS Controller → PC
x		Length		
0	UINT8	Response Service Identifier		See "SystemDefines.h"
1	UINT8	--Service specific --		
2	UINT8	--Service specific --		
3	UINT8	--Service specific --		
4	UINT8	--Service specific --		
5	UINT8	--Service specific --		
6	UINT8	--Service specific --		
7	UINT8	--Service specific --		

Table 6-2: Explicit Message Response Protocol

6.2 Command Message Protocols

Upon consumption of a Command Message produced by another control device the EABS Controller shall produce its response message data determined by the received "Screen Number". Consumption of the command message protocol is supported by "UAP_CAN_CONS_CMD_MSG.c" and the response message production protocols are supported by "UAP_CAN_PROD_CMD_MSG.c" where the declarations of the Screen Number values are contained within "Display.h".

Byte	Data Type	Description	Value	Values
x	UINT16	CAN Identifier	0x44B	UI → EABS
x		Length	0-8	
0	UINT8	SCREEN NUMBER	**0-255**	Values unique to each control device are defined within the "Display.h" for each product.
1	UINT8	XID	0-255	Increments for every message sent
2-7		Screen dependent		

Table 6-3: Command Request Message (UI → EABS)

Receipt of a Command Request Message often initiates cyclic production of a Debug Message after the Command Response Message is sent by the EABS Controller. Consistent with most CAN networks, the CAN Identifier used to produce a message identifies the content of the produced message. In the case of the Screen Messages, when *synchronization* is required (subnet hops) between the control device sending the Command Request Message and the EABS Controller producing the selected Screen Message, two standardized variables are optionally contained with the command response message. The first optional byte (byte 0) confirms receipt of the Command Request Message by echoing the Screen Number and the second byte (byte 1) optionally echoes the transaction identifier (XID) of the last consumed Command Request Message. The EABS Controller thus produces Command Response Messages containing the Screen Number and XID values. See "UAP_CAN_PROD_CMD_MSG.c" for specific message code or UAP_CAN_PROD_CMD_MSG.doc" for logic drawings showing the Command Response Message contents based upon consumed Screen Number values declared in "Display.h" and optionally Debug Messages produced for a selected screen number.

Byte	Data Type	Description	Value	Values
x	UINT16	CAN Identifier	0x40B	EABS → DISPLAY
x		Length	0-8	

Byte	Data Type	Description	Value	Values
0	UINT8	SCREEN NUMBER (optional)	0-255	Contains the "response screen number"..See "Display.h" for screen declaration values.
1	UINT8	REMOTE XID (optional)		
2-7		Screen dependent		Screen Number dependent data

Table 6-4: Command Response Message (EABS → other)

Byte	Data Type	Description	Value	Values
x	UINT16	CAN Identifier	0x40B	EABS → DISPLAY
x		Length	0-8	
0-7		Screen dependent		Screen Number dependent data

Table 6-5: Command Response Message (EABS → other)

6.3 Debug Message Protocols

Debug Messages come in various forms within XRAE control devices, where like all other messages these produced messages contain properties from various object classes to test the operation of specific functionality within a control device.

> EXAMPLE: "UAP_PID_LF.c" in the EABS Controller modulates the left front brake magnet based upon various criteria, where monitoring selected values within this UAP during product testing are utilized to verify its operation meets or exceeds the design criteria. In practice a specific Debug Message Number is selected, where object properties are produced at an extremely high rate on TCAN, whose contents are captured in a file by a TCAN monitoring tool, where the captured file is then imported into an excel spreadsheet where the values are plotted for review. A trailer simulation module is connected to EABS module which simulates the operation of the braking system, including generating wheel sensor pulses, blue wire pulses, varying battery voltages and so forth based upon IO messages received from PC test software. When operational defects are identified by viewing the excel plots, the UAP within EABS Controller are modified and tests rerun to verify elimination of the identified defect. Debug messages are often captured during road testing to identify UAP defects as well. When defects are detected actual trailer wheel sensor and blue wire values captured in road test files and are then applied to the simulator to generate the exact same conditions captured during trailer testing and thus used to isolate a newly identified defect and eliminate it. The simulator module utilizes XRAE objects and UAPs and PC tools interact with these objects using debug messages and explicit messages.

See "UAP_CAN_PROD_DEBUG.c" code and "UAP_CAN_PROD_DEBUG.doc" drawing for the various produced debug messages. The specific debug message produced, and when it is produced, is determined by a value contained within a specific instance of the parameter object class, where "UAP_PARAMETER.h" contains the declaration of the various instances for a control device where within the EABS Controller instance "PARAM_DEBUG_MSG" sets the debug message number produced. When this parameter is set to a supported value production of the respective message begins, where the default parameter value is zero, resulting in no message being produced. Thus supporting Debug Messages does not require an instance of the Consumer Object and associated UAP code space to initiate message production.

NOTE: New debug messages are often added to “UAP_CAN_PROD_DEBUG.c” during system testing, commenting out other debug messages when additional code space is needed, making these messages available for reuse during final product acceptance testing. Some of these debig messages may remain in product release code should they be useful for actual trailer testing by qualified service personnel.

Further details relative to the use of various messages are described in the associated UAP drawing, UAP operational descriptions in the Device Profile.

