AEROSPACE STANDARD

AS6040™	REV. A

Issued 2010-11
Reaffirmed 2015-04
Stabilized 2020-12

Superseding AS6040

JAUS HMI Service Set

RATIONALE

Technology related to this document is not changing; therefore, related interfaces defined in this document are not likely to change.

STABILIZED NOTICE

This document has been declared "Stabilized" by the SAE AS-4JAUS Joint Architecture for Unmanned Systems Committee and will no longer be subjected to periodic reviews for currency. Users are responsible for verifying references and continued suitability of technical requirements. Newer technology may exist.

TO PLACE A DOCUMENT ORDER: **Tel: 877-606-7323 (inside USA and Canada)**
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
SAE WEB ADDRESS: **http://www.sae.org**

For more information on this standard, visit
https://www.sae.org/standards/content/AS6040A/

INTRODUCTION

The primary goal of the JAUS HMI Service Set is to allow elements in an unmanned system to define graphical information to be displayed and controlled from an operator interface such as an Operator Control Unit (OCU), providing a capability of interoperability between the remote asset and operator interface. To this end, each service defines the messages (vocabulary) and protocol (rules) for data exchange. This logical interoperability is independent of the physical transport, and it is expected that a Transport Standard, such as the JAUS/SDP Transport Specification [AS5669], is used in conjunction with this specification.

Each service in the JAUS HMI Service Set can be described using the JAUS Service Interface Definition Language (JSIDL) [AS5684]. JSIDL creates a formal schema that allows for validation of each service definition described herein. Although knowledge of JSIDL is not required to understand or implement this Specification, it is highly recommended for supporting context. For convenience, the JAUS HMI Service Set contains both a text based and XML based representation for each service.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. SCOPE

This document defines a set of standard application layer interfaces called JAUS HMI Services. JAUS Services provide the means for software entities in an unmanned system or system of unmanned systems to communicate and coordinate their activities. The HMI Services represent the platform-independent Human Machine Interface (HMI) capabilities commonly found across all domains and types of unmanned systems. Five services are defined in this document:

- Drawing
- Pointing Device
- Keyboard
- Digital Control
- Analog Control

Each service is described by a JAUS Service Definition (JSD) which specifies the message set and protocol required for compliance. Each JSD is fully compliant with the JAUS Service Interface Definition Language (JSIDL) [AS5684].

1.1 Purpose

The purpose of this document is to provide unmanned systems, subsystems, and payloads a standard message set for the definition of HMI drawing functions and associated control mechanisms. While this standard does not mandate a particular application, a common use allow an unmanned system to provide a consistent user interface across a variety of Operator Control Units (OCUs).

1.2 JAUS Core Service Set

The JAUS Service Definitions defined herein make use of the inheritance functionality provided by JSIDL to incorporate capabilities as specified by the JAUS Core Service Set [AS5710]. These documents must be used together to define a complete service.

It is important to note that details related to 'Message Serialization' and 'Understanding Protocol Descriptions' can be found in the JAUS Core Service Set document, and are not repeated here.

1.3 Compliance

The JAUS HMI Service Set must support compliance assessment. To do so, this specification must be sufficiently precise to enable the "compliant"/"not compliant" distinction to be made independently of the underlying transport mechanism. It is important to note that implementations are considered compliant to individual Service Definitions within this Specification; it is not necessary that a single entity realize each service to be considered compliant.

1.4 Document Organization

The layout of this document is as follows. Section 2 lists external references used throughout the specification. Section 3 specifies common conventions and definitions. Section 4 specifies the JAUS Service Definition for each of the HMI Services, with particular emphasis on the description, assumptions, message set, and protocol behavior. Section 5 describes the message encoding for each message set. Section 6 contains any applicable notes. Finally, APPENDIX A and APPENDIX B contain the complete JSIDL representation for each service and their associated message set.

2. REFERENCES

2.1 Applicable Documents

The following publications form a part of this document to the extent specified herein. The latest issue of SAE publications shall apply. The applicable issue of other publications shall be the issue in effect on the date of the purchase order. In the event of conflict between the text of this document and references cited herein, the text of this document takes precedence. Nothing in this document, however, supersedes applicable laws and regulations unless a specific exemption has been obtained.

2.1.1 SAE Publications

Available from SAE, 400 Commonwealth Drive, Warrendale, PA 15096-0001, Telephone: 877-606-7323 (inside USA and Canada) or 724-776-4970 (outside USA), Web address: http://www.sae.org

AS5710 JAUS Core Service Set, Revision A

AS5669 JAUS/SDP Transport Specification, Revision A

AS5684 JAUS Service Interface Definition Language, Revision A

2.2 Acronyms

AS: Aerospace Standard

ASCII: American Standard Code for Information Interchange

DCM: Digital Control Message

HMI: Human Machine Interface

ID: Identifier

JAUS: Joint Architecture for Unmanned Systems

JSD: JAUS Service Definition

JSIDL: JAUS Service Interface Definition Language

OCU: Operator Control Unit

URN: Uniform Resource Name

XML: Extensible Markup Language

3. COMMON CONVENTIONS

3.1 Graphics Viewport

For the following defined Drawing and Pointing Device Service definitions, the operator interface will be required to create a Graphics Viewport to interpret and display the Drawing Definition messages from the remote asset, in addition to allow cursor movements and control actions to be returned to the remote asset.

The Graphics Viewport is an area of the operator interface used to display Drawing Definitions that have been requested from a remote asset. The remote asset implementing the Drawing Service specifies the dimensions of the Graphics Viewport using the page graphics command (section 5.3.1.1.1) that the operator interface scales to the available physical area. Remote assets shall assume that the aspect ratio of the Graphics Viewport is square when designing the graphical drawing. In case of the actual Graphics Viewport being rectangular, it is the responsibility of the operator interface to manipulate the drawing process to compensate.

The minimum X and Y coordinates are in the lower left corner of the screen, and by default start with an origin of (0, 0). Positive X is horizontal right and positive Y is vertically up.

All coordinates for drawing and pointing device messages are specified in a device-independent coordinate system called User Space, which is mapped to the region defined by the remote asset to specify the dimensions of the Graphics Viewport.

Using the Drawing Service, a remote asset creates a Drawing Definition using the defined Graphics Commands (section 5.3.1). The operator interface uses the received Drawing Definition generated by the remote asset to render a graphical display on the Graphics Viewport. The operator interface interprets the Graphics Commands in the order they appear in the Drawing Definition message.

Graphics commands may be placed within conditional blocks (section 5.3.1.1.28). Remote assets shall use no more than 8 levels of conditional blocks.

The Graphics Viewport transform is held in a stack. The current transform can be copied to the top of the stack using a Push command (section 5.3.1.1.26). The current view can then be transformed and the user can return to the original view by using a Pop command (section 5.3.1.1.27). Remote assets shall use no more than 8 levels of Graphics Viewport stack manipulation.

The operator interface shall implement a mechanism that allows the simultaneous display of up to 20 digital controls, such as buttons. The 20 digital controls may be displayed concurrently or through a menu structure, though further functions above 20 may be utilised by remapping the digital controls. The digital controls shall be separated from the Graphics Viewport and referenced as four groups of five digital controls. Figure 1 shows a recommended screen format with five digital controls along each edge forming a digital control group. A menu structure can be created within the Drawing Definition to cater for additional functionality to be mapped to the 20 digital controls.

The Graphics Viewport can be adapted to allow the use of hardware digital controls. Where hardware digital controls are used, it is highly recommended that the operator interface shall display the labels of the digital control on the screen next to the physical hardware digital control.

An operator interface alternatively may create interfaces that have no Graphics Viewport representation, for example an operator interface may offer a mechanism to display remote assets found where a button only interface may be constructed. For example, a remote asset may construct a Drawing Definition dynamically that could include digital controls to select individual payloads depending upon the vehicles configuration at deployment, providing support for a plug and play style functionality. In this case, the operator interface may initially consist of a button only interface allowing the operator to select one of the payloads. Once the payload has been selected, the Drawing Definition would then be requested and presented for the that payload. Alternatively, the remote asset may construct a single Drawing Definition to present to the operator interface by combining several Drawing Definitions from other components into a single Drawing Definition or provide different display configurations to access individual Drawing Definitons though a digital control interface.

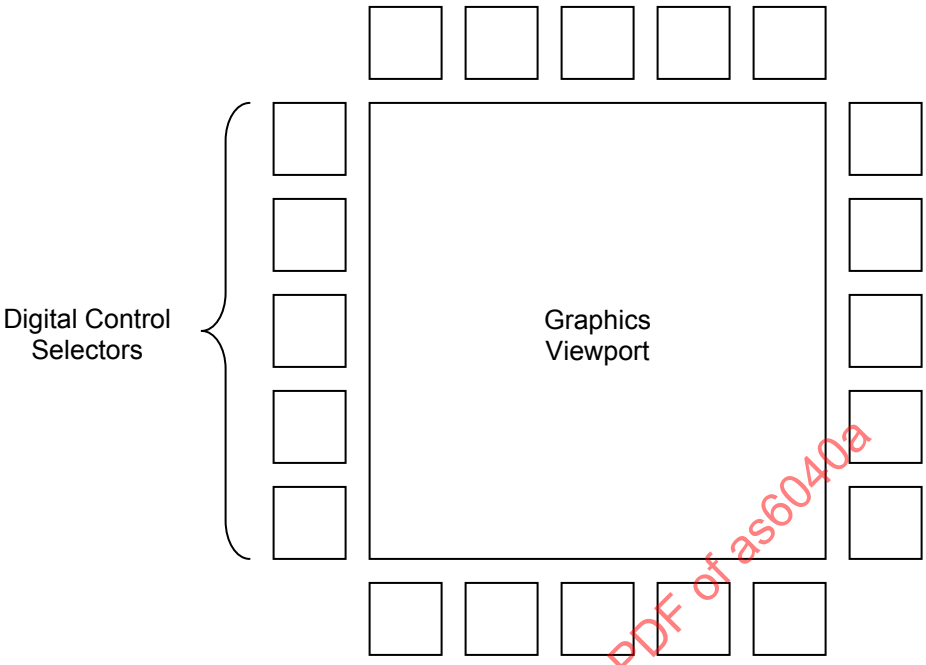


FIGURE 1 - RECOMMENDED SCREEN LAYOUT

3.2 Interpreting a Drawing Definition

The Report Drawing Definition message forms the foundation of HMI services. This message allows the remote asset to specify the layout of a graphical user interface on the client's viewport. To achieve this goal, a Report Drawing Definition message contains a list of primitive operations, such as defining the pen or brush color, drawing a shape, or manipulating stack transformations.

Upon receipt of a Drawing Definition message, the operator interface must interpret the Drawing Definition in sequence, as each operation is modal. For example, setting a pen color impacts all shapes drawn until the pen color is changed again. The following pseudo-code provides a sample of a sequence which draws two shapes on the Graphics Viewport:

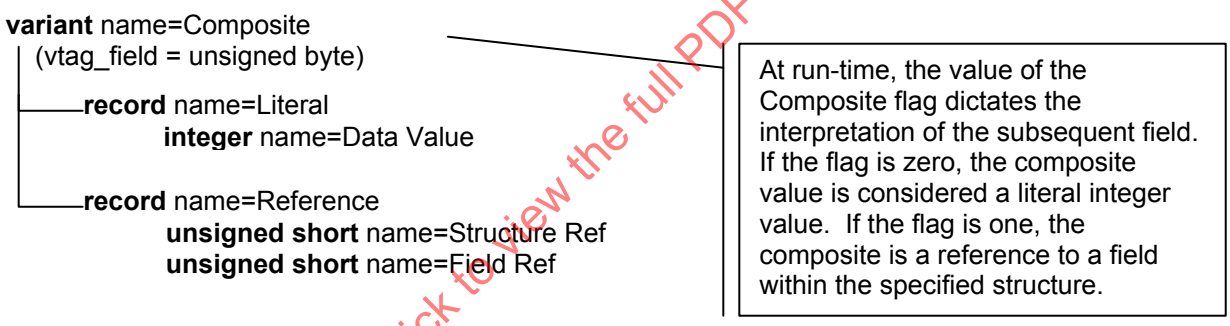
```
Page( Minimum X = 0, Minimum Y = 0, Maximum X = 100, Maximum Y = 100 )
Background( Red = 255, Green = 255, Blue = 255 )
Pen( Red = 255, Green = 0, Blue = 0 )
PenWidth( Width = 3 )
Brush( Red = 0, Green = 255, Blue = 0 )
FilledRectangle( X = 0, Y = 0, Width = 10, Height = 20 )
Push()
   Translate( TX = 50, RX = 50 )
   Pen( Red=0, Green = 0, Blue = 255 )
   FilledCircle( X = 0, Y = 0, Radius = 10 )
Pop()
Label( X = 0, Y = 0, Alignment = TopRight, Size = 8, Text = "Back at corner" )
```

In the above example the Page() command defines the user space scale range. It is the responsibility of the opertaor interface to map the actual resolution of the display area to the user space range, which in this case is 100 by 100. After creating a white background, the subsequent 4 commands define a filled rectange with color attributes in the lower left hand corner. The rectangle has a red border of width 3 pixels and a green fill color. A Push() command is then used to add the current graphics transform to a stack to allow an easy return to this position following the subsequent Translate() function that shifts the origin of the coordinate frame to the center of the viewport. The result of the Translate() command results in the circle drawn in the center of the viewport rather than the lower left corner. The circle uses the new blue pen color, but retains the green fill color. The Pop() command decreases the graphics transform stack, returning to the transformation defined prior to the Push() command, removing the effect of the Translate(). The text label is then created in the lower-left hand corner on top of the rectangle, displayed in 8 point blue font.

Providing this modal behavior, combined with multiple levels of stack manipulation and conditional expressions, sophisticated user interfaces are possible.

3.3 Composite Data Type

The Drawing Service provides a set of graphical commands to construct a Drawing Definition that will be interpreted and displayed on an operator interface. To allow the displayed Drawing Definition to be updated dynamically, many of the data types used within each graphical element use the Composite data type. The use of the Composite data type allows the elements within the Drawing Definition to be updated with new values when sent by the remote asset. For example, a rotation command within a Drawing Definition that sets the angle of a manipulator joint may be updated with a new angle as and when the manipulator joint is moved on the remote asset. By using the Composite data type the value used to set the angle on the Graphics Viewport is obtained from a data structure (section 5.3.3) defined by the remote asset. The contents of the data structure is updated by the remote asset and sent to the operator interface for updating the display to the operator. This may be particulary useful for generating dynamic information messages to be presented to the operator or variable status information from the remote asset. The Composite data type is a 5 byte value with a flag used to determine if the value to be used in the Drawing Definition command is a literal value, or a reference to a data structure element. If the characterize flag is set to 0 then the data value will be taken as a literal data type and used directly within the Drawing Definition graphical command. If the characterize flag is set to 1 then the data value will be a reference to a field in a data structure and the data defined in the structure should be used within the Drawing Definition graphical command. The Composite data type is defined using the *variant* type [AS5684]:



4. SERVICE DEFINITIONS

The following subsections provide a textual definition for each Service Definition in the JAUS HMI Service Set. Corresponding JSIDL definitions are offered in the Appendix. Additional information on interpreting the Service Definition elements may be found in [AS5684].

4.1 Drawing Service

name=Drawing
id=urn:jaus:jss:HMI:Drawing
version=1.0

inherits-from Events
  name=Events
  id=urn:jaus:jss:core:Events
  version=1.0

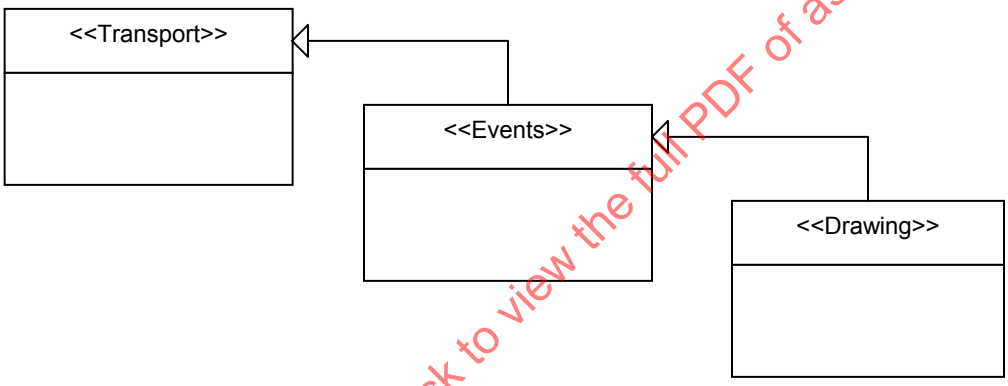


FIGURE 2 - DRAWING SERVICE

4.1.1 Description

The Drawing Service allows remote assets to define a graphical output to an operator interface. A remote asset creates a Drawing Definition that it shall send to the operator interface. The Drawing Definition message consists of a series of commands that allow the remote asset to create a graphical representation and offer mechanisms for its control.

Variable data can be represented in a Drawing Definition to allow the remote asset to update aspects of the display on the operator interface using a mechanism of Data Definition messages. Data Definition messages allow a remote asset to define a data structure that can be updated independent of the Drawing Definition. This may be used to update information such as joint angles reflecting the correct position and orientation displayed on the operator interface. Variable data is indicated within a Drawing Definition by the use of the Composite data type (section 3.3). A reference to a data structure element is substituted for the literal value if the Composite type has been set as variable data. A remote asset can then send Data messages that correspond with the Data Definition reference, allowing the operator interface to read the data and update the display accordingly.

It is highly recommended that the operator interface creates an Event for the remote asset to send Report Drawing Definition and Report Data messages upon a change to receive any updates that may occur.

4.1.2 Assumptions

Messages may be delayed, lost or reordered.

4.1.3 Vocabulary

The table below lists the vocabulary of the Drawing Service.

TABLE 1 - DRAWING SERVICE VOCABULARY

Message ID (hex)	Name	isCommand?
Input Set		
2700	QueryDrawingDefinition	False
2701	QueryDataDefinition	False
2702	QueryData	False
Output Set		
4700	ReportDrawingDefinition	False
4701	ReportDataDefinition	False
4702	ReportData	False

4.1.4 Protocol Behavior

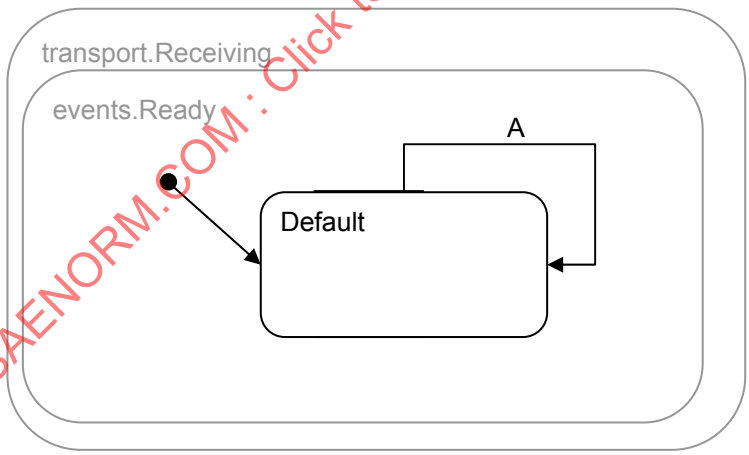


FIGURE 3 - DRAWING SERVICE PROTOCOL BEHAVIOR

TABLE 2 - DRAWING SERVICE STATE TRANSITION TABLE

Label	Trigger	Guard	Actions
A	QueryDrawingDefinition		sendReportDrawingDefinition
	QueryDataDefinition		sendReportDataDefinition
	QueryData		sendReportData

TABLE 3 - DRAWING SERVICE TRANSITION ACTIONS

Action	Interpretation
sendReportDrawingDefinition	Send Report Drawing Definition message to the operator interface that sent the query
sendReportDataDefinition	Send Report Data Definition message to the operator interface that sent the query
sendReportData	Send Report Data message to the operator interface that sent the query

4.2 Pointing Device Service

name=PointingDevice
id=urn:jaus:jss:HMI:PointingDevice
version=1.0

inherits-from Events
  name=Events
  id=urn:jaus:jss:core:Events
  version=1.0

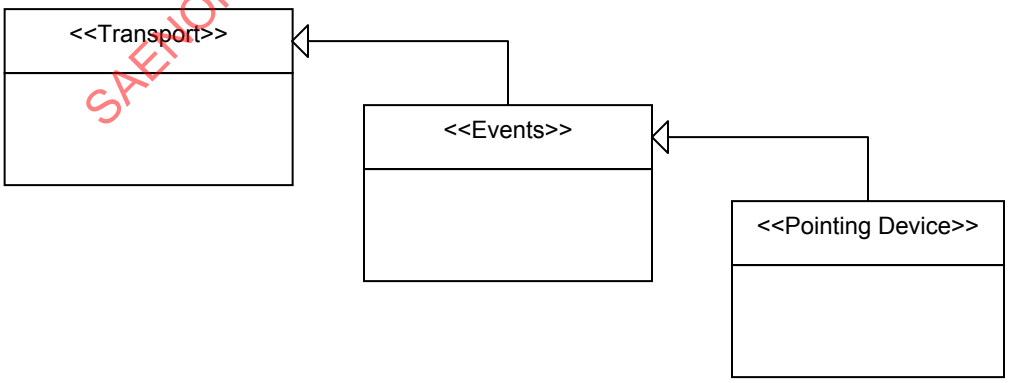


FIGURE 4 - POINTING DEVICE SERVICE

4.2.1 Description

The Pointing Device Service is used to detect mouse, or other similar pointing device coordinate position on the Graphics Viewport. The Pointing Device Service is used in conjunction with the Drawing Service.

The remote asset may create an Event for the operator interface to send Report Pointing Device Movement messages. This Event can either be upon a change or at a periodic frequency. Upon receipt of this Event creation, the operator interface shall return Report Pointing Device Movement messages at the requested interval.

The remote asset may create an Event for the operator interface to send Report Pointing Device Action messages. This Event can be configured to send either "on change" or at a periodic frequency. Upon receipt of this Event creation, the operator interface shall return Report Pointing Device Action messages at the requested interval.

4.2.2 Assumptions

Messages may be delayed, lost or reordered.

4.2.3 Vocabulary

The table below lists the vocabulary of the Pointing Device Service.

TABLE 4 - POINTING DEVICE SERVICE VOCABULARY

Message ID (hex)	Name	isCommand?
Input Set		
2703	QueryPointingDeviceMovement	False
2704	QueryPointingDeviceAction	False
Output Set		
4703	ReportPointingDeviceMovement	False
4704	ReportPointingDeviceAction	False

4.2.4 Protocol Behavior

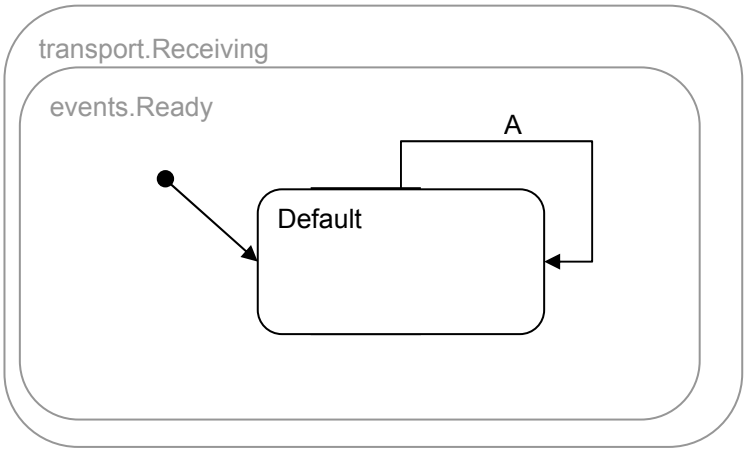


FIGURE 5 - POINTING DEVICE SERVICE PROTOCOL BEHAVIOR

TABLE 5 - POINTING DEVICE SERVICE STATE TRANSITION TABLE

Label	Trigger	Guard	Actions
A	QueryPointingDeviceMovement	*pointingDeviceExists*	sendReportPointingDeviceMovement
	QueryPointingDeviceAction	*pointingDeviceExists*	sendReportPointingDeviceAction

TABLE 6 - POINTING DEVICE SERVICE CONDITIONS TABLE

Guard	Interpretation
pointingDeviceExists	True if a pointing device exists on the client interface

TABLE 7 - POINTING DEVICE SERVICE TRANSITION ACTIONS

Action	Interpretation
sendReportPointingDeviceMovement	Send Report Pointing Device Movement message to the remote asset that sent the query
sendReportPointingDeviceAction	Send Report Pointing Device Action message to the remote asset that sent the query

4.3 Keyboard Service

name=Keyboard
id=urn:jaus:jss:HMI:Keyboard
version=1.0

inherits-from Events
  name=Events
  id=urn:jaus:jss:core:Events
  version=1.0

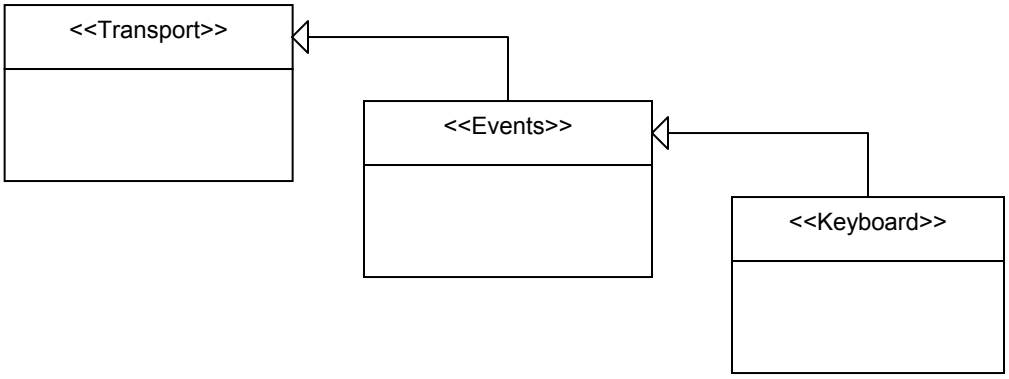


FIGURE 6 - KEYBOARD SERVICE

4.3.1 Description

An operator interface that has a keyboard device attached may implement the Keyboard Service to return key strokes to the remote asset.

The remote asset may create an Event for the operator interface to send Report Keyboard Action messages. It is highly recommended that this Event is setup for "on change" and not a periodic frequency.

4.3.2 Assumptions

Messages may be delayed, lost or reordered.

4.3.3 Vocabulary

The table below lists the vocabulary of the Keyboard Service.

TABLE 8 - KEYBOARD SERVICE VOCABULARY

Message ID (hex)	**Name**	**isCommand?**
Input Set		
2705	QueryKeyboardAction	False
Output Set		
4705	ReportKeyboardAction	False

4.3.4 Protocol Behavior

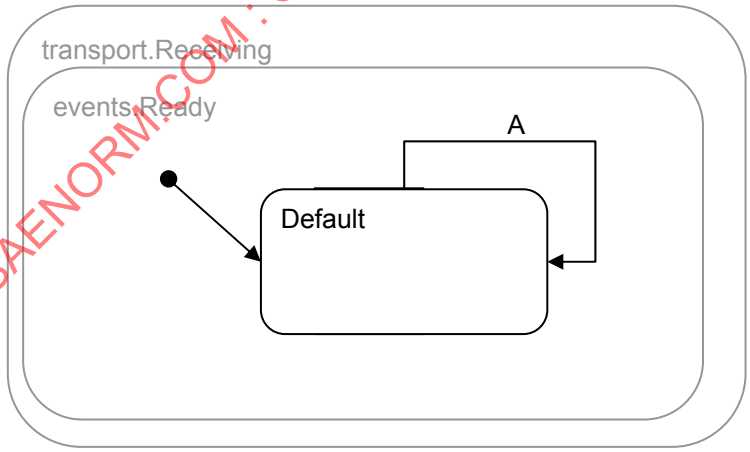


FIGURE 7 - KEYBOARD SERVICE PROTOCOL BEHAVIOR

TABLE 9 - KEYBOARD SERVICE STATE TRANSITION TABLE

Label	Trigger	Guard	Actions
A	QueryKeyboardAction	*keyboardExists*	sendReportKeyboardAction

TABLE 10 - KEYBOARD SERVICE CONDITIONS TABLE

Guard	Interpretation
keyboardExists	True if a keyboard exists on the client interface

TABLE 11 - KEYBOARD SERVICE TRANSITION ACTIONS

Action	Interpretation
sendReportKeyboardAction	Send Report Keyboard Action message to the remote asset that sent the query

4.4 Digital Control Service

name=DigitalControl
id=urn:jaus:jss:HMI:DigitalControl
version=1.0

inherits-from AccessControl
  name=AccessControl
  id=urn:jaus:jss:core:AccessControl
  version=1.0

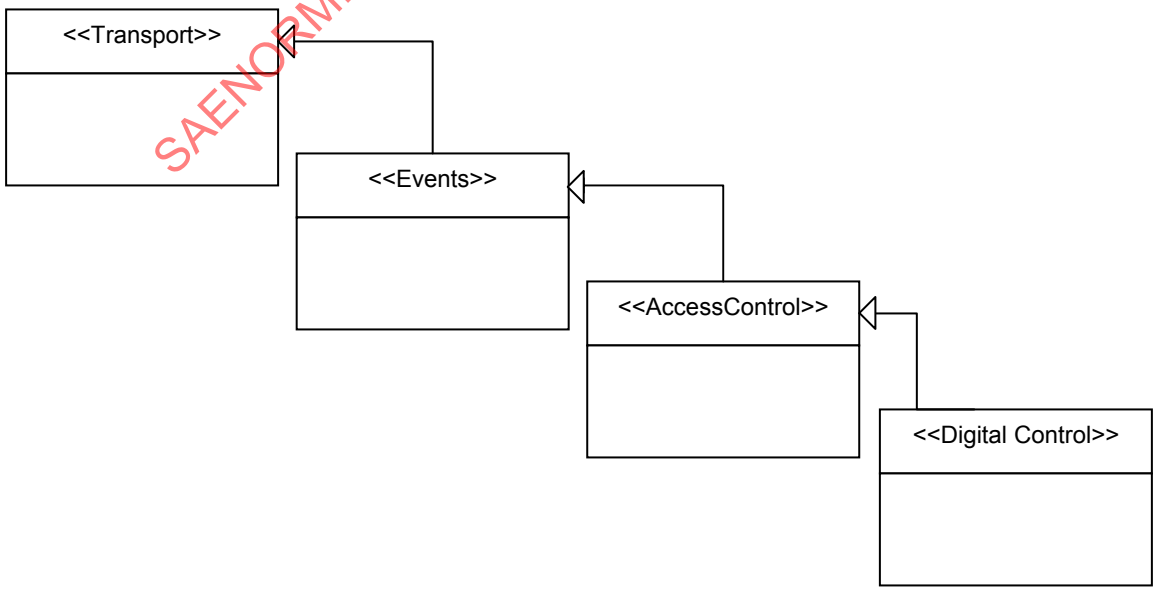


FIGURE 8 - DIGITAL CONTROL SERVICE

4.4.1 Description

An operator interface that has digital controls, such as a button, either as software or hardware may implement the Digital Control Service that would send a Report DCM message to the remote asset when a digital control push or release occurs. The Digital Control Service is used in conjunction with the Drawing Service.

The remote asset may send a Set DCM message to the operator interface to control the current state of a digital control. This allows the operator interface to alter the display of the referenced digital control to be displayed as inset or released, active or inactive and flashing or steady.

The remote asset may create an Event for the operator interface to send Digital Control Messages (DCMs). It is highly recommended that this Event is setup for "on change" and not a periodic frequency.

4.4.2 Assumptions

Messages may be delayed, lost or reordered.

4.4.3 Vocabulary

The table below lists the vocabulary of the Digital Control Service.

TABLE 12 - DIGITAL CONTROL SERVICE VOCABULARY

Message ID (hex)	**Name**	**isCommand?**
Input Set		
2706	QueryDCM	False
0706	SetDCM	True
Output Set		
4706	ReportDCM	False

4.4.4 Protocol Behavior

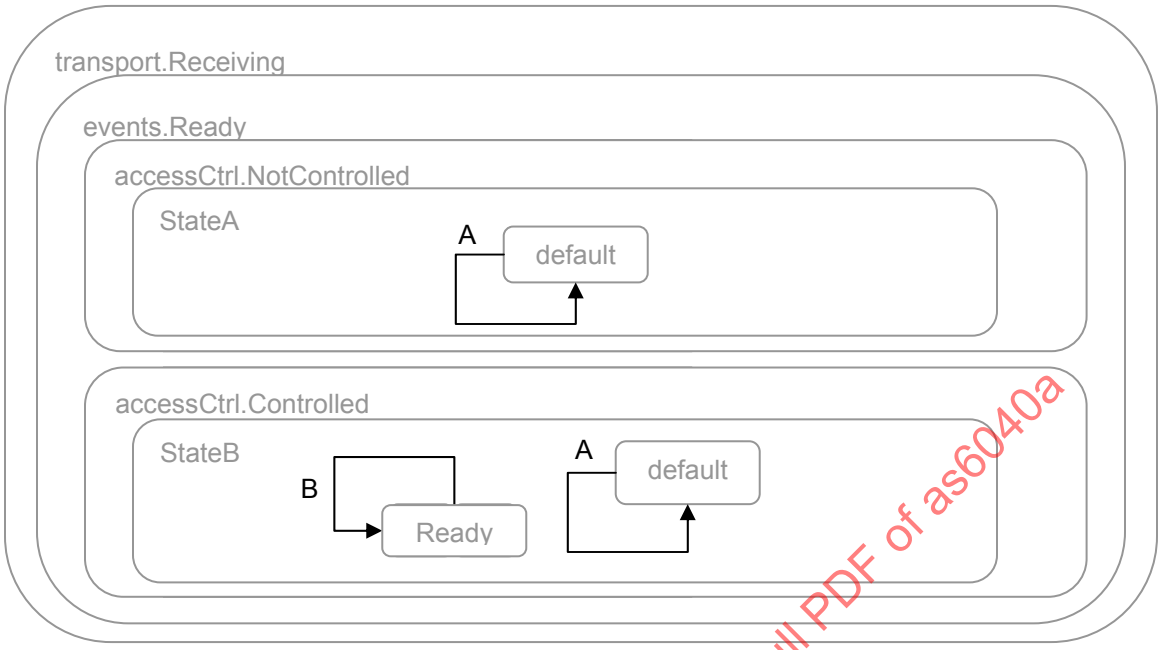


FIGURE 9 - DIGITAL CONTROL SERVICE PROTOCOL BEHAVIOR

TABLE 13 - DIGITAL CONTROL SERVICE TRANSITION TABLE

Label	Trigger	Guard	Actions
A	QueryDCM		sendReportDCM
B	SetDCM		setDCM

TABLE 14 - DIGITAL CONTROL SERVICE TRANSITION ACTIONS

Action	Interpretation
sendReportDCM	Send a Report DCM message to the remote asset that sent the query.
setDCM	Set the digital control style on the operator interface

4.5 Analog Control Service

name=AnalogControl
id=urn:jaus:jss:HMI:AnalogControl
version=1.0

inherits-from Events
  name=Events
  id=urn:jaus:jss:core:Events
  version=1.0

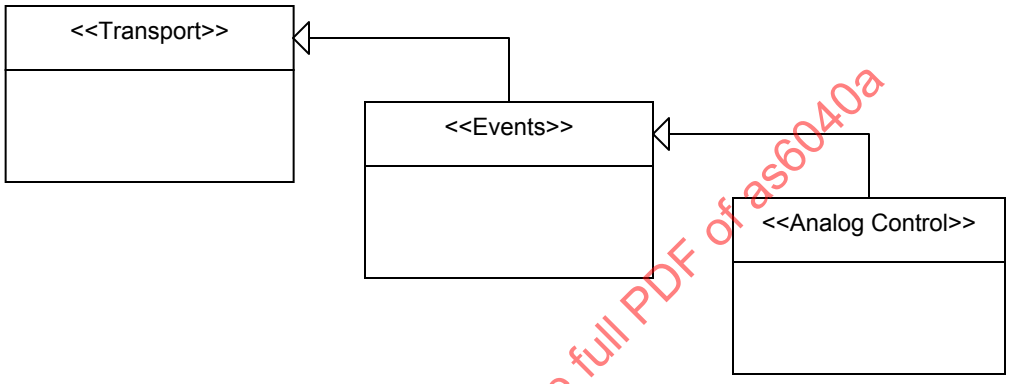


FIGURE 10 - ANALOG CONTROL SERVICE

4.5.1 Description

The Analog Control Service is used for the generation of analog operator interactions. Six fields represent the proportion of movement of an analog device in linear and rotational movement of the three cardinal axis.

The Analog Control Service does not define co-ordinate systems. However, the interpretation of operator inputs shall assume the following orientation of movement as shown in Figure 11. The longitudinal axis is positive in the forward direction, lateral movement is positive to the right and vertical movement is positive downwards. Rotational axial values increase in a clockwise direction when looking along the axis towards the greater positive value. Roll is rotation around the longitudinal axis, pitch around the lateral axis and yaw around the vertical axis.

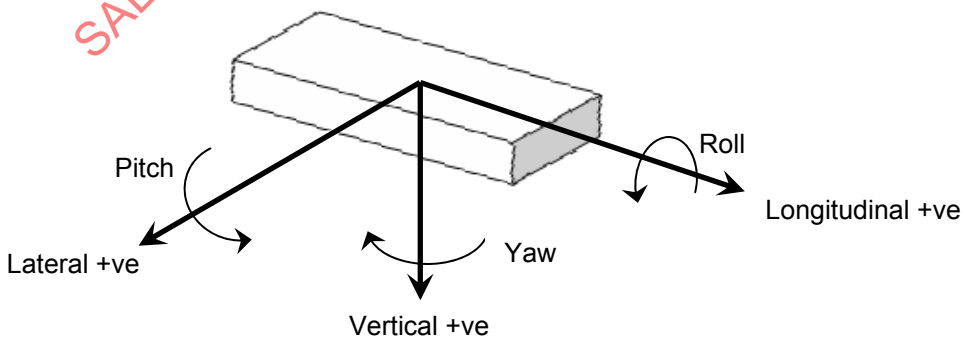


FIGURE 11 - MOVEMENT ORIENTATION

The remote asset may create an Event for the operator interface to send Report Analog Action messages. This Event can be configured to send either "on change" or at a periodic frequency. Upon receipt of this event creation, the operator interface shall return Report Analog Action messages at the requested interval.

4.5.2 Assumptions

Messages may be delayed, lost or reordered.

4.5.3 Vocabulary

The table below lists the vocabulary of the Analog Control Service.

TABLE 15 - ANALOG CONTROL SERVICE VOCABULARY

Message ID (hex)	Name	isCommand?
Input Set		
2707	QueryAnalogDevices	False
2708	QueryAnalogAction	False
Output Set		
4707	ReportAnalogDevices	False
4708	ReportAnalogAction	False

4.5.4 Protocol Behavior

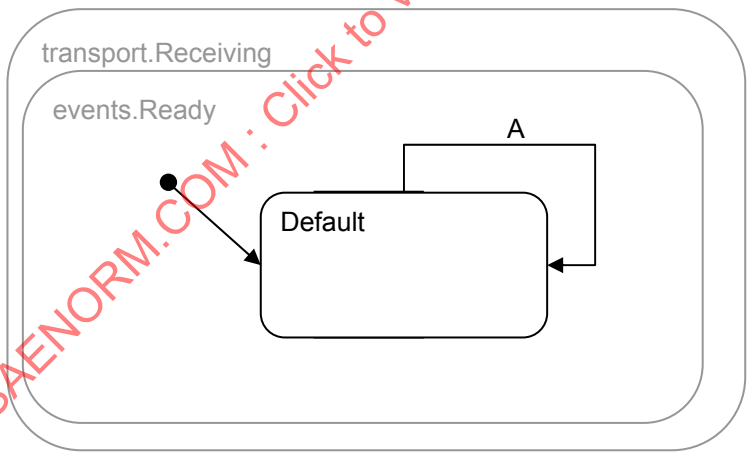


FIGURE 12 - ANALOG CONTROL SERVICE PROTOCOL BEHAVIOR

TABLE 16 - ANALOG CONTROL SERVICE STATE TRANSITION TABLE

Label	Trigger	Guard	Actions
A	QueryAnalogDevices		sendReportAnalogDevices
	QueryAnalogAction		sendReportAnalogAction

TABLE 17 - ANALOG CONTROL SERVICE TRANSITION ACTIONS

Action	Interpretation
sendReportAnalogDevices	Send Report Analog Devices message to the remote asset that sent the query
sendReportAnalogAction	Send Report Analog Action message to the remote asset that sent the query

5. DECLARED TYPES

5.1 CommandClass

5.1.1 ID 0706h: SetDCM

The remote asset uses this message to define the drawing state of a digital control on the screen with Set DCM.

TABLE 18 - SET DCM MESSAGE ENCODING

body └ **record** name= DCMRec					
record name= DCMRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** DCMReference	Unsigned Short Integer	One	False	Unique reference to the digital control
2	**<bit_field>** DCMStateCode	Unsigned Byte	One	False	The state of the digital control (see Table 19)

The Reference field is unique within the definition of the HMI Services. A remote asset may use all bits to define the look of a digital control displayed on an operator interface screen.

TABLE 19 - DIGITAL CONTROL STATES

State	Bit	Interpretation
UPDOWN	0	Bit set for down or pressed, clear for up or released
ACTIVE	1	Bit set for active, bit clear for inactive
FLASH	2	Bit set to flash digital control, clear for steady state
Flash Rate	3 & 4	Rate at which to flash the digital control in Hz if bit 2 (FLASH) is set. If a value of 00 is set then the control will appear as a steady state. Other values map directly to Hz values

5.2 QueryClass

5.2.1 ID 2700h: QueryDrawingDefinition

This message is used by an operator interface to obtain the Drawing Definition from the remote asset.

TABLE 20 - QUERY DRAWING DEFINITION MESSAGE ENCODING

body └ (empty)

5.2.2 ID 2701h: QueryDataDefinition

This message is used by an operator interface to obtain all Data Definitions from the remote asset.

TABLE 21 - QUERY DATA DEFINITION MESSAGE ENCODING

body └ (empty)

5.2.3 ID 2702h: QueryData

This message is used by an operator interface to obtain the Data relating to the Data Definition for the specified data structure.

TABLE 22 - QUERY DATA MESSAGE ENCODING

body └ **record** name= QueryDataRec

record name= QueryDataRec

Field #	Name	Type	Units	Optional?	Interpretation
1	**<fixed_field>** DataStructureReference	Unsigned Short Integer	One	False	Unique reference to the required data structure used in the system

5.2.4 ID 2703h: QueryPointingDeviceMovement

This message causes the receiving operator interface to reply with a Report Pointing Device Movement message.

TABLE 23 - QUERY POINTING DEVICE MOVEMENT MESSAGE ENCODING

body └ (empty)

5.2.5 ID 2704h: QueryPointingDeviceAction

This message causes the receiving operator interface to reply with a Report Pointing Device Action message.

TABLE 24 - QUERY POINTING DEVICE ACTION MESSAGE ENCODING

body └ (empty)

5.2.6 ID 2705h: QueryKeyboardAction

This message causes the receiving operator interface to reply with a Report Keyboard Action message.

TABLE 25 - QUERY KEYBOARD ACTION MESSAGE ENCODING

body └ (empty)

5.2.7 ID 2706h: QueryDCM

This message causes the receiving remote asset to reply with Report DCM messages for all DCMs known by the receiving remote asset.

TABLE 26 - QUERY DCM MESSAGE ENCODING

body └ (empty)

5.2.8 ID 2707h: QueryAnalogDevices

This message causes the receiving operator interface to reply with Report Analog Devices messages for all available analog devices.

TABLE 27 - QUERY ANALOG DEVICES MESSAGE ENCODING

body └ (empty)

5.2.9 ID 2708h: QueryAnalogAction

This message shall cause the receiving component to reply with a Report Analog Action message.

TABLE 28 - QUERY ANALOG ACTION MESSAGE ENCODING

body └ **record** name= QueryAnalogRec					
record name= QueryAnalogActionRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** PresenceVector	Byte	One	False	See Report Analog Action Message
2	**<fixed_field>** AnalogReference	Unsigned Short Integer	One	False	Unique reference to the analog device used in the system

5.3 InformClass

5.3.1 ID 4700h: ReportDrawingDefinition

This message is used to provide a Drawing Definition containing one or more Graphic Commands and associated data.

TABLE 29 - REPORT DRAWING DEFINITION MESSAGE ENCODING

body
- **sequence** name= ReportDrawingDefinitionSeq
  - **record** name= ReferenceFieldRec
  - **list** name= GraphicsCommandList
    - (count_field= unsigned short integer)
    - **variant** name= GraphicsCommandVar

(vtag_field= unsigned byte)
sequence name= PageSeq
record name= BackgroundRec
record name= PenRec
record name= PenWidthRec
record name= BrushRec
record name= DigitalControlRec
record name= DigitalControlGroupRec
sequence name= LabelSeq
sequence name= ValueSeq
sequence name= RotateSeq
sequence name= AntiRotateSeq
sequence name= TranslateSeq
sequence name= AntiTranslateSeq
sequence name= RectangleSeq
sequence name= FilledRectangleSeq
sequence name= CircleSeq
sequence name= FilledCircleSeq
sequence name= EllipseSeq
sequence name= FilledEllipseSeq
sequence name= LineSeq

list name= PolylineList
- (count_field= unsigned byte)
- **sequence** name= PolylineSeq

list name=PolygonList
- (count_field= unsigned byte)
- **sequence** name= PolygonSeq

list name=FilledPolygonList
- (count_field= unsigned byte)
- **sequence** name= FilledPolygonSeq

sequence name= ArcSeq
sequence name= FilledArcSeq
variant name= PushVar
(vtag_field= unsigned byte)
variant name= PopVar
(vtag_field= unsigned byte)

variant name= IfVar
(declared_type_ref= Composite)
variant name= ElseIfVar
(declared_type_ref= Composite)
variant name= ElseVar
(vtag_field= unsigned byte)
variant name= EndIfVar
(vtag_field= unsigned byte)
variant name= SelectVar
(declared_type_ref= Composite)
variant name= CaseVar
(declared_type_ref= Composite)
variant name= DefaultVar
(vtag_field= unsigned byte)
variant name= EndSelectVar
(vtag_field= unsigned byte)
sequence name= ImageSeq

record name= ReferenceFieldRec

Field #	Name	Type	Units	Optional?	Interpretation
1	**<fixed_field>** DrawingDefinitionReference	Unsigned Short Integer	One	False	Unique reference to the Drawing Definition used in the system

5.3.1.1 Graphics Command Variant Definition

5.3.1.1.1 Page (vtag_field= 0)

The Page command defines the limits of the coordinate system used by the remote asset for the Graphics Viewport. Operator interfaces shall interpret this definition and adjust the limits such that the actual limits retain a 1:1 ratio with the physical dimensions of the operator interface Graphics Viewport. Any adjustments required to the page command coordinates will retain the center position on the operator interface Graphics Viewport. The page command shall appear first in a Drawing Definition.

TABLE 30 – PAGE MESSAGE ENCODING

sequence name= PageSeq
variant (Composite) name= MinimumX // User defined minimum X coordinate of the display frame
variant (Composite) name= MinimumY // User defined minimum Y coordinate of the display frame
variant (Composite) name= MaximumX // User defined maximum X coordinate of the display frame
variant (Composite) name= MaximumY // User defined maximum Y coordinate of the display frame

5.3.1.1.2 Background (vtag_field= 1)

The Background command allows the background color of the drawing area to be set. Limited color displays shall suitably convert the RGB value. By default, the background color shall be white RGB (255, 255, 255).

TABLE 31 – BACKGROUND MESSAGE ENCODING

record name= BackgroundRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** RedColor	Byte	One	False	Red value of the background color of drawing area
2	**<fixed_field>** GreenColor	Byte	One	False	Green value of the background color of drawing area
3	**<fixed_field>** BlueColor	Byte	One	False	Blue value of the background color of drawing area

5.3.1.1.3 Pen (vtag_field= 2)

The Pen command allows the definition of the active pen color for drawing. Limited color displays shall suitably convert the RGB value. By default, the pen color shall be black RGB (0, 0, 0).

TABLE 32 – PEN MESSAGE ENCODING

record name= PenRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** RedColor	Byte	One	False	Red value of the new pen color
2	**<fixed_field>** GreenColor	Byte	One	False	Green value of the new pen color
3	**<fixed_field>** BlueColor	Byte	One	False	Blue value of the new pen color

5.3.1.1.4 Pen Width (vtag_field= 3)

Defines the active drawing pen line width. By default, the pen width shall be 1 pixel.

TABLE 33 – PEN WIDTH MESSAGE ENCODING

record name= PenWidthRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** Width	Byte	Pixel	False	Width of drawing pen in pixels

5.3.1.1.5 Brush (vtag_field= 4)

The Brush command defines the fill color for drawing filled shapes. Limited color displays shall suitably convert the RGB value. By default, the brush color shall be black RGB (0, 0, 0).

TABLE 34 – BRUSH MESSAGE ENCODING

record name= BrushRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** RedColor	Byte	One	False	Red value of the new brush color
2	**<fixed_field>** GreenColor	Byte	One	False	Green value of the new brush color
3	**<fixed_field>** BlueColor	Byte	One	False	Blue value of the new brush color

5.3.1.1.6 Digital Control (vtag_field= 5)

The Digital Control command allows the remote asset to define a digital control at a position on the screen. The operator interface defines the size of the digital control determined by guidelines or standards for the application domain. The Drawing Service defines the digital controls outside the Graphics Viewport in four Digital Control Groups. Each Digital Control Group can contain five digital controls.

Operation of a digital control by the operator will generate a Report DCM message (see Digital Control Service section 4.4) to be sent to the remote asset informing it of the action taken by the operator. The remote asset can send a Set DCM message to the operator interface to state the desired display of the digital control. The remote asset has complete control over the behaviour of the digital control allowing it to create toggle or momentary types (see Table 19 for digital control states).

TABLE 35 – DIGITAL CONTROL MESSAGE ENCODING

record name= DigitalControlRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** Reference	Unsigned Short Integer	One	False	Reference to the digital control used to define the state of the digital control in DCM messages
2	**<fixed_field>** Group	Byte	One	False	0 = Top 1 = Left 2 = Right 3 = Bottom
3	**<fixed_field>** DigitalControl	Byte	One	False	Digital Control position within Group 0 = Top/Left … 4 = Lower/Right
4	**<fixed_length_string>** Label	String	12 bytes	False	The text to label the digital control in a null terminated ASCII String

5.3.1.1.7 Digital Control Group (vtag_field= 6)

The Digital Control Group command assigns a label to one of the four Digital Control Groups. The Digital Control Group text may be used on displays that cannot display 20 digital controls simultaneously.

TABLE 36 – DIGITAL CONTROL GROUP MESSAGE ENCODING

record name= DigitalControlGroupRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** Group	Byte	One	False	0 = Top 1 = Left 2 = Right 3 = Bottom
2	**<fixed_length_string>** Label	String	12 bytes	False	The text to label the Digital Control Group in a null terminated ASCII String

5.3.1.1.8 Label (vtag_field= 7)

The Label command allows the remote asset to create output only text onto the display. The text is displayed at the current transform rotation (section 5.3.1.1.10).

TABLE 37 – LABEL MESSAGE ENCODING

sequence name= LabelSeq └ **variant** (Composite) name= X // X coordinate in User Space **variant** (Composite) name= Y // Y coordinate in User Space **record** name= AlignmentRec **variant** (Composite) name= size // Font point size (72 points per inch) **record** name= TextRec					
record name= AlignmentRec					
1	**<fixed_field>** Alignment	Byte	One	False	See note below* 0 = Top Left 1 = Middle Left 2 = Bottom Left 3 = Middle Center 4 = Middle Right 5 = Bottom Center 6 = Bottom Right 7 = Top Center 8 = Top Right
record name= TextRec					
1 & 2	**<variable_length_string>** Text	count_field= unsigned byte	One	False	Text of label in a null terminated ASCII String. The count field value includes the null terminator.

*The alignment field defines where the text is to be displayed in relation to the X, Y point e.g. a value of 3 would align the text middle center, which is centered both horizontally and vertically at the point (X, Y) specified.

5.3.1.1.9 Value (vtag_field= 8)

The Value command allows the remote asset to display a variable value on the operator interface Graphics Viewport. The variable Reference defines the text displayed.

TABLE 38 – VALUE MESSAGE ENCODING

sequence name= ValueSeq
└ **variant** (Composite) name= X
variant (Composite) name= Y
record name= AlignmentRec
variant (Composite) name= size
variant (Composite) name= DataReference
	// X coordinate in User Space
	// Y coordinate in User Space
	
	// Font point size (72 points per inch)
	// Reference to the data to display. Characterize flag will always be set to 1 (reference value)

record name= AlignmentRec

1	**<fixed_field>** Alignment	Byte	One	False	See note below* 0 = Top Left 1 = Middle Left 2 = Bottom Left 3 = Middle Center 4 = Middle Right 5 = Bottom Center 6 = Bottom Right 7 = Top Center 8 = Top Right

* The alignment field defines where the text is to be displayed in relation to the X, Y point e.g. a value of 3 would align the text middle center, which is centered both horizontally and vertically at the point (X, Y) specified.

5.3.1.1.10 Rotate (vtag_field= 9)

The Rotate command defines an angular rotation of the graphics transformation matrix. Positive angles are a clockwise rotation. By default, the rotation angle shall be 0 micro-radians from the horizontal.

TABLE 39 – ROTATE MESSAGE ENCODING

sequence name= RotateSeq └ **variant** (Composite) name= Angle	// The angular rotation in micro-radians

5.3.1.1.11 Anti-Rotate (vtag_field= 10)

The Anti-rotate command defines a rotation in an anti-clockwise direction, to complement the ‘Rotate’ command above.

TABLE 40 – ANTI-ROTATE MESSAGE ENCODING

sequence name= AntiRotateSeq	
└ **variant** (Composite) name= Angle	// The negative angular rotation in micro-radians

5.3.1.1.12 Translate (vtag_field= 11)

The Translate command defines a positional shift of the graphics transformation matrix. By default, the translation shall be (0, 0) from the origin.

TABLE 41 – TRANSLATE MESSAGE ENCODING

sequence name= TranslateSeq	
└ **variant** (Composite) name= TX	// The X transformation in User Space
variant (Composite) name= TY	// The Y transformation in User Space

5.3.1.1.13 Anti-Translate (vtag_field= 12)

The Anti-translate command defines a positional shift in the opposite direction to the Translate command above. This allows the same variable references to be used to obtain an opposite action to a translate command.

TABLE 42 – ANTI-TRANSLATE MESSAGE ENCODING

sequence name= AntiTranslateSeq	
└ **variant** (Composite) name= TX	// The negative X transformation in User Space
variant (Composite) name= TY	// The negative Y transformation in User Space

5.3.1.1.14 Rectangle (vtag_field= 13)

The Rectangle command specifies a box drawing. X, Y defines the lower left corner (assuming 0, 0) of the rectangle. Width and height may be positive or negative. The border of the rectangle is drawn using the active Pen color.

TABLE 43 – RECTANGLE MESSAGE ENCODING

sequence name= RectangleSeq	
└ **variant** (Composite) name= X	// The X coordinate in User Space
variant (Composite) name= Y	// The Y coordinate in User Space
variant (Composite) name= Width	// The width value in User Space
variant (Composite) name= Height	// The height value in User Space

5.3.1.1.15 Filled Rectangle (vtag_field= 14)

The Filled Rectangle command specifies a box drawing. X, Y defines the lower left corner (assuming 0, 0) of the rectangle. Width and height may be positive or negative. The border of the rectangle is drawn using the active Pen color and filled with the active Brush color.

TABLE 44 – FILLED RECTANGLE MESSAGE ENCODING

sequence name= FilledRectangleSeq	
└ **variant** (Composite) name= X	// The X coordinate in User Space
variant (Composite) name= Y	// The Y coordinate in User Space
variant (Composite) name= Width	// The width value in User Space
variant (Composite) name= Height	// The height value in User Space

5.3.1.1.16 Circle (vtag_field= 15)

The Circle command defines a circle drawing. X and Y define the center of the circle. All values are in user coordinate space. The border of the circle is drawn using the active Pen color.

TABLE 45 – CIRCLE MESSAGE ENCODING

sequence name= CircleSeq	
└ **variant** (Composite) name= X	// The X coordinate in User Space
variant (Composite) name= Y	// The Y coordinate in User Space
variant (Composite) name= Radius	// The radius of the circle in User Space

5.3.1.1.17 Filled Circle (vtag_field= 16)

The Filled Circle command defines a circle drawing. The X and Y fields define the center of the circle. The border of the circle is drawn using the active Pen color and filled with the active Brush color.

TABLE 46 – FILLED CIRCLE MESSAGE ENCODING

sequence name= FilledCircleSeq	
└ **variant** (Composite) name= X	// The X coordinate in User Space
variant (Composite) name= Y	// The Y coordinate in User Space
variant (Composite) name= Radius	// The radius of the circle in User Space

5.3.1.1.18 Ellipse (vtag_field= 17)

The Ellipse command defines an ellipse drawing. X and Y define the bottom left corner of the ellipse and the width and height parameters define the size of the ellipse. The border of the ellipse is drawn using the active Pen color.

TABLE 47 – ELLIPSE MESSAGE ENCODING

sequence name= EllipseSeq	
└ **variant** (Composite) name= X	// The X coordinate in User Space
variant (Composite) name= Y	// The Y coordinate in User Space
variant (Composite) name= Width	// The width of the ellipse in User Space
variant (Composite) name= Height	// The height of the ellipse in User Space

5.3.1.1.19 Filled Ellipse (vtag_field= 18)

The Filled Ellipse command defines an ellipse drawing. X and Y define the center of the circle. The border of the ellipse is drawn using the active Pen color and filled with the active Brush color.

TABLE 48 – FILLED ELLIPSE MESSAGE ENCODING

sequence name= FilledEllipseSeq	
└ **variant** (Composite) name= X	// The X coordinate in User Space
variant (Composite) name= Y	// The Y coordinate in User Space
variant (Composite) name= Width	// The width of the ellipse in User Space
variant (Composite) name= Height	// The height of the ellipse in User Space

5.3.1.1.20 Line (vtag_field= 19)

The Line command allows the user to define two points joined by a line to be drawn in the current Pen color and width.

TABLE 49 – LINE MESSAGE ENCODING

sequence name= LineSeq	
└ **variant** (Composite) name= X1	// The start X coordinate in User Space
variant (Composite) name= Y1	// The start Y coordinate in User Space
variant (Composite) name= X2	// The end X coordinate in User Space
variant (Composite) name= Y2	// The end Y coordinate in User Space

5.3.1.1.21 Polyline (vtag_field= 20)

Draws a sequence of connected lines defined by arrays of X and Y coordinates. Each pair of (X, Y) coordinates defines a point. The figure shall not close if the first point differs from the last point. The first X, Y pair in the list is the start point. The polyline is drawn in the current Pen color and width.

TABLE 50 – POLYLINE MESSAGE ENCODING

sequence name= PolylineSeq	
└ **variant** (Composite) name= X	// The X coordinate in User Space
variant (Composite) name= Y	// The Y coordinate in User Space

5.3.1.1.22 Polygon (vtag_field= 21)

Draws a sequence of connected lines defined by arrays of X and Y coordinates. Each pair of (X, Y) coordinates defines a point. The figure shall not close if the first point differs from the last point. The first X, Y pair in the list is the start point. The polygon is drawn in the current Pen color and width.

TABLE 51 – POLYGON MESSAGE ENCODING

sequence name= PolygonSeq	
└ **variant** (Composite) name= X	// The X coordinate in User Space
variant (Composite) name= Y	// The Y coordinate in User Space

5.3.1.1.23 Filled Polygon (vtag_field= 22)

Draws a closed polygon defined by arrays of X and Y coordinates. Each pair of (X, Y) coordinates defines a point. The figure closes automatically by drawing a line connecting the final point to the first point, if those points are different. The first X, Y pair in the list is the start point. The border of the polygon is drawn using the active Pen color and filled with the active Brush color.

TABLE 52 – FILLED POLYGON MESSAGE ENCODING

sequence name= FilledPolygonSeq	
└ **variant** (Composite) name= X	// The X coordinate in User Space
variant (Composite) name= Y	// The Y coordinate in User Space

5.3.1.1.24 Arc (vtag_field= 23)

Draws the outline of a circular or elliptical arc covering the specified rectangle. The resulting arc begins at StartAngle and extends for ArcAngle degrees. Angles are interpreted such that 0 degrees is at the 3 o'clock position. A positive value indicates a counter-clockwise rotation while a negative value indicates a clockwise rotation. The center of the arc is the center of the rectangle whose origin is (X, Y). The arc is drawn in the current Pen color and width.

TABLE 53 – ARC MESSAGE ENCODING

sequence name= ArcSeq	
└ **variant** (Composite) name= X	// The X coordinate in User Space
variant (Composite) name= Y	// The Y coordinate in User Space
variant (Composite) name= Width	// The width of the arc to be drawn in User Space
variant (Composite) name= Height	// The height of the arc to be drawn in User Space
variant (Composite) name= StartAngle	// The beginning angle
variant (Composite) name= ArcAngle	// The angular extent of the arc, relative to the start angle

5.3.1.1.25 Filled Arc (vtag_field= 24)

Fills a circular or elliptical arc covering the specified rectangle. The resulting arc begins at StartAngle and extends for ArcAngle degrees. Angles are interpreted such that 0 degrees is at the 3 o'clock position. A positive value indicates a counter-clockwise rotation while a negative value indicates a clockwise rotation. The center of the arc is the center of the rectangle whose origin is (X, Y). The border of the arc is drawn using the active Pen color and filled with the active Brush color.

TABLE 54 – FILLED ARC MESSAGE ENCODING

sequence name= FilledArcSeq	
└ **variant** (Composite) name= X	// The X coordinate in User Space
variant (Composite) name= Y	// The Y coordinate in User Space
variant (Composite) name= Width	// The width of the arc to be drawn in User Space
variant (Composite) name= Height	// The height of the arc to be drawn in User Space
variant (Composite) name= StartAngle	// The beginning angle
variant (Composite) name= ArcAngle	// The angular extent of the arc, relative to the start angle

5.3.1.1.26 Push (vtag_field= 25)

The Push command increases the stack level of the graphics transformation matrix stack, copying the current top-level matrix. There are no parameters for this definition.

5.3.1.1.27 Pop (vtag_field= 26)

The Pop command decreases the stack level of the graphics transformation matrix stack, removing the current top-level matrix and returning to the transformation defined prior to the previous Push. There are no parameters for this definition.

5.3.1.1.28 If (vtag_field= 27)

The If command starts a conditional block. The parameter for this defintion is the variable to test for true/false. Non-zero = false.

5.3.1.1.29 Else If (vtag_field= 28)

The Else If command starts an alternative conditional block. The parameter for this defintion is the variable to test for true/false. Non-zero = false.

5.3.1.1.30 Else (vtag_field= 29)

The Else command starts a block of definitions for execution if preceding If and Else If statements were false. There are no parameters for this definition.

5.3.1.1.31 End If (vtag_field= 30)

Terminates a conditional block. There are no parameters for this definition.

5.3.1.1.32 Select (vtag_field= 31)

Starts a case statement. The parameter for this defintion is the reference to value to be evaluated for each Case statement.

5.3.1.1.33 Case (vtag_field= 32)

Starts a block of conditional commands executed if the specific value of the Case statement matches the current value of the Select statement. The parameter for this defintion is the value to be compared to the reference data in the Select statement.

5.3.1.1.34 Default (vtag_field= 33)

Starts a block for execution if no other Case statement command blocks were executed within a Select block. There are no parameters for this definition.

5.3.1.1.35 End Select (vtag_field= 34)

The End Select statement terminates a Select block. There are no parameters for this definition.

5.3.1.1.36 Image (vtag_field= 35)

The Image statement defines an area where a static image can be displayed. The image shall be scaled to fit within the defined area. X, Y defines the lower left corner (assuming 0, 0) of the image.

TABLE 55 – IMAGE MESSAGE ENCODING

sequence name= ImageSeq

- **variant** (Composite) name= X // The X coordinate in User Space
- **variant** (Composite) name= Y // The Y coordinate in User Space
- **variant** (Composite) name= Width // The width of the image to be drawn in User Space
- **variant** (Composite) name= Height // The height of the image to be drawn in User Space
- **variant** (Composite) name= Reference // Reference to the data of image to display. Characterize flag will always be set to 1 (reference value)
- **record** name= ImageTypeRec

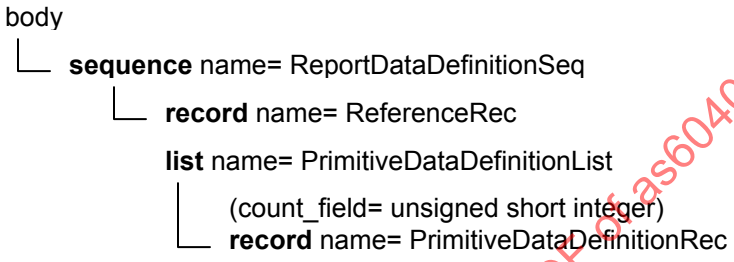
record name= ImageTypeRec

1	**<fixed_field>** ImageType	Byte	One	False	0 = Portable PixMap (PPM) 1 = Joint Photographic Experts Group (JPEG) 2 = Tagged Image File Format (TIFF) 3 = National Imagery Transmission Format (NITFS) 4 = Bitmap (BMP)

5.3.2 ID 4701h: ReportDataDefinition

The Report Data Definition message allows a remote asset to define the data content of data structures transmitted using Report Data messages for use within the Drawing Definition (section 5.3.1). The reference for the Report Data Definition message should correspond to the reference for the Report Data message for a given data structure. The array type will define a one dimensional array of element types as identified. Character arrays shall use the ASCII null character for termination.

TABLE 56 - REPORT DATA DEFINITION MESSAGE ENCODING



record name= ReferenceRec

Field #	Name	Type	Units	Optional?	Interpretation
1	**<fixed_field>** DataStructureReference	Unsigned Short Integer	One	False	Unique reference to the data structure
record name= PrimitiveDataDefinitionRec					
1	**<fixed_field>** PrimitiveDataDefinition	Unsigned Byte	One	False	The elements/fields of the data structure (see TABLE 57). The first data element to be defined is referenced as field 0; the second is 1 etc…

TABLE 57 - DATA STRUCTURE DEFINITION MESSAGE CODES

Name	Representation	Interpretation
Byte	8 bit unsigned integer	Set to 0
Short Integer	16 bit signed integer	Set to 1
Integer	32 bit signed integer	Set to 2
Long Integer	64 bit signed integer	Set to 3
Unsigned Short Integer	16 bit unsigned integer	Set to 4
Unsigned Integer	32 bit unsigned integer	Set to 5
Unsigned Long Integer	64 bit unsigned integer	Set to 6
Float	IEEE 32 bit floating point number	Set to 7
Long Float	IEEE 64 bit floating point number	Set to 8
Character	Defines a single ASCII character	Set to 9
List of Bytes	List of 8 bit unsigned integers	Set to 10
List of Short Integers	List of 16 bit signed integers	Set to 11
List of Integers	List of 32 bit signed integers	Set to 12
List of Long Integers	List of 64 bit signed integers	Set to 13
List of Unsigned Short Integers	List of 16 bit unsigned integers	Set to 14
List of Unsigned Integers	List of 32 bit unsigned integers	Set to 15
List of Unsigned Long Integers	List of 64 bit unsigned integers	Set to 16
List of Floats	List of IEEE 32 bit floating point numbers	Set to 17
List of Long Floats	List of IEEE 64 bit floating point numbers	Set to 18
List of Characters	List of ASCII characters	Set to 19

5.3.3 ID 4702h: ReportData

A remote asset outputs Report Data messages to pass blocks of data in a pre-determined format (section 5.3.2) to the operator interface for use within the Drawing Definition (section 5.3.1).

The Data field contains the data in a format described by a previously transmitted Report Data Definition message with the same reference number.

TABLE 58 - REPORT DATA MESSAGE ENCODING

```
body
└─ sequence name= ReportDataSeq
     └─ record name= ReportDataRec
        list name= DataList
        │  (count_field= unsigned short integer)
        └─ variant name= StructureFieldVar
           (vtag_field= unsigned byte)
           record name= ByteRec
           record name= ShortIntRec
           record name= IntegerRec
           record name= LongIntRec
           record name= UShortIntRec
           record name= UIntergerRec
           record name= ULongIntRec
           record name= FloatRec
           record name= LongFloatRec
           record name= ByteRec
           list name= ByteList
           │  (count_field= unsigned byte)
           └─ record name= ByteRec
           list name= ShortIntList
           │  (count_field= unsigned byte)
           └─ record name= ShortIntRec
           list name= IntegerList
           │  (count_field= unsigned byte)
           └─ record name= IntegerRec
           list name= LongIntList
           │  (count_field= unsigned byte)
           └─ record name= LongIntRec
           list name= UShortIntList
           │  (count_field= unsigned byte)
           └─ record name= UShortIntRec
           list name= UIntegerList
           │  (count_field= unsigned byte)
           └─ record name= UIntegerRec
           list name= ULongIntList
           │  (count_field= unsigned byte)
           └─ record name= ULongIntRec
           list name= FloatList
           │  (count_field= unsigned byte)
           └─ record name= FloatRec
           list name= LongFloatList
           │  (count_field= unsigned byte)
           └─ record name= LongFloatRec
           list name= CharList
           │  (count_field= unsigned byte)
           └─ record name= ByteRec
```

record name= ReportDataRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** DataStructureReference	Unsigned Short Integer	One	False	Unique reference to the data structure
record name= ByteRec					
1	**<fixed_field>** Data	Byte	One	False	The data being transmitted as required by Data Definition (section 5.3.2)
record name= ShortIntRec					
1	**<fixed_field>** Data	Short integer	One	False	The data being transmitted as required by Data Definition (section 5.3.2)
record name= IntegerRec					
1	**<fixed_field>** Data	Integer	One	False	The data being transmitted as required by Data Definition (section 5.3.2)
record name= LongIntRec					
1	**<fixed_field>** Data	Long Integer	One	False	The data being transmitted as required by Data Definition (section 5.3.2)
record name= UShortIntRec					
1	**<fixed_field>** Data	Unsigned Short Integer	One	False	The data being transmitted as required by Data Definition (section 5.3.2)
record name= UIntegerRec					
1	**<fixed_field>** Data	Unsigned Integer	One	False	The data being transmitted as required by Data Definition (section 5.3.2)
record name= ULongIntRec					
1	**<fixed_field>** Data	Unsigned Long Integer	One	False	The data being transmitted as required by Data Definition (section 5.3.2)
record name= FloatRec					
1	**<fixed_field>** Data	Float	One	False	The data being transmitted as required by Data Definition (section 5.3.2)
record name= LongFloatRec					
1	**<fixed_field>** Data	Long Float	One	False	The data being transmitted as required by Data Definition (section 5.3.2)

5.3.4 ID 4703h: ReportPointingDeviceMovement

An operator interface creates the Report Pointing Device Movement message when the user moves a graphic input device over the Graphics Viewport. The message returns the coordinates of the pointing device in the Graphics Viewport defined by the remote asset. The two fields represent position of the mouse in user coordinate space.

TABLE 59 - REPORT POINTING DEVICE MOVEMENT MESSAGE ENCODING

body └ **record** name=ReportPointingDeviceMovementRec					
record name= ReportPointingDeviceMovementRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** X	Integer	One	False	The X coordinate in User Space
2	**<fixed_field>** Y	Integer	One	False	The Y coordinate in User Space
3	**<fixed_field>** Valid	Byte	One	False	If the pointing device is within the Graphics Viewport this value is set to 1, otherwise set to 0 with X and Y values set to 0

5.3.5 ID 4704h: ReportPointingDeviceAction

An operator interface creates the Report Pointing Device Action message when the user operates a pointing device button over the Graphics Viewport. The message returns the pointing device coordinates and the state of the button. All pointing device button clicks (push and release) shall generate two messages.

TABLE 60 - REPORT POINTING DEVICE ACTION MESSAGE ENCODING

body └ **record** name= ReportPointingDeviceActionRec					
record name= ReportPointingDeviceActionRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** Button	Byte	One	False	Button code. 0 = Left 1 = Middle 2 = Right
2	**<fixed_field>** State	Byte	One	False	The state of the button position. 0 = Up 1 = Down
3	**<fixed_field>** X	Integer	One	False	The X coordinate in User Space
4	**<fixed_field>** Y	Integer	One	False	The Y coordinate in User Space
5	**<fixed_field>** Valid	Byte	One	False	If the pointing device is within the Graphics Viewport this value is set to 1, otherwise set to 0 with X and Y values set to 0

5.3.6 ID 4705h: ReportKeyboardAction

An operator interface sends a Report Keyboard Action message to a remote asset when the operator uses a keyboard. All keyboard button clicks (push and release) shall generate two messages.

TABLE 61 - REPORT KEYBOARD ACTION MESSAGE ENCODING

body └ **record** name= ReportKeyboardActionRec					
record name= ReportKeyboardActionRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** Key	Short Integer	One	False	The 16-Bit Unicode value of the key
2	**<fixed_field>** KeyState	Byte	One	False	If the key is pressed this value is set to 1. If the key is subsequently released this value is set to 0

5.3.7 ID 4706h: ReportDCM

An operator interface may output Report DCM (Digital Control Message) messages when the operator interacts with one of the referenced digital controls. All Digital Control clicks (push and release) shall generate two messages.

TABLE 62 - REPORT DCM MESSAGE ENCODING

body └ **record** name= ReportDCMRec					
record name= ReportDCMRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<fixed_field>** DCMReference	Unsigned Short Integer	One	False	Unique reference to the digital control
2	**<fixed_field>** DCMStateCode	Byte	One	False	Bit 0 set for down or pressed, bit clear for up or released

5.3.8 ID 4707h: ReportAnalogDevices

An operator interface sends a Report Analog Devices message to a remote asset when requested by the remote asset.

TABLE 63 - REPORT ANALOG DEVICES MESSAGE ENCODING

body
└ **list** name= ReportAnalogDevicesList
└ (count_field= unsigned byte)
record name= ReportAnalogDevicesRec

record name= ReportAnalogDevicesRec

Field #	Name	Type	Units	Optional?	Interpretation
1	**<fixed_field>** Reference	Unsigned Short Integer	One	False	Unique reference to the analog device
2 & 3	**<variable_length_string>** Name	count_field= unsigned byte	One	False	Textural identifier for the analog device in a null terminated ASCII String. The count field value includes the null terminator.

5.3.9 ID 4708h: ReportAnalogAction

An operator interface sends a Report Analog Action message to a remote asset when the operator uses an analog device.

The Report Analog message is an isochronous output by the operator interface reporting the current deflection values of the analog control of the operator interface. The operator interface determines the proportional movement of the control (proportional to the min/max values supported by the data type) and outputs a normalized value indicating this movement.

For example, if an analog device had a range of -5v to +5v this would be mapped to -127 to +127 for an 8 bit data type. An analog device with a range of -2.5v to +2.5v would also map to -127 to +127 for an 8 bit data type

The fields may be sent as 8, 16 or 32 bit signed data. All fields shall be the same format and therefore it is not permitted to mix 8, 16 and 32 bit fields within the same message.

Where a presence vector is used, or the analog device does not support all 6 axis, a zero demand (no movement) should be assumed for the unreported field(s).

TABLE 64 - REPORT ANALOG MESSAGE ENCODING

body └ **record** name= ReportAnalogActionRec					
record name= ReportAnalogActionRec					
Field #	**Name**	**Type**	**Units**	**Optional?**	**Interpretation**
1	**<presence_vector>**	Byte	One	False	
2	**<fixed_field>** Reference	Unsigned Short Integer	One	False	Unique reference to the analog device
3	**<variable_field>** Longitudinal	Byte, Short Integer or Integer	One	True	The forward demand. Positive forward
4	**<variable_field>** Lateral	Byte, Short Integer or Integer	One	True	The sideways demand. Positive right
5	**<variable_field>** Vertical	Byte, Short Integer or Integer	One	True	The vertical demand. Positive down
6	**<variable_field>** Roll	Byte, Short Integer or Integer	One	True	The roll demand. Positive clockwise
7	**<variable_field>** Pitch	Byte, Short Integer or Integer	One	True	The pitch demand. Positive nose up
8	**<variable_field>** Yaw	Byte, Short Integer or Integer	One	True	The yaw demand. Positive nose right

6. NOTES

APPENDIX A – XML FOR SERVICE DEFINITIONS

A.1 DRAWING SERVICE

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def xmlns="urn:jaus:jsidl:1.0" name="Drawing" id=" urn:jaus:jss:HMI:Drawing"
version="1.0">
  <description>The Drawing Service allows remote assets to define a graphical output to an
operator interface.  A remote asset creates a Drawing Definition that it shall send to the
operator interface.  The Drawing Definition message consists of a series of commands that
allow the remote asset to create a graphical representation and offer mechanisms for its
control.
Variable data can be represented in a Drawing Definition to allow the remote asset to
update aspects of the display on the operator interface using a mechanism of Data
Definition messages.  Data Definition messages allow a remote asset to define a data
structure that can be updated independent of the Drawing Definition.  This may be used to
update information such as joint angles reflecting the correct position and orientation
displayed on the operator interface.  Variable data is indicated within a Drawing
Definition by the use of the Composite data type (section 3.2).  A reference to a data
structure element is substituted for the literal value if the Composite type has been set
as variable data.  A remote asset can then send Data messages that correspond with the
Data Definition reference, allowing the operator interface to read the data and update the
display accordingly.
It is highly recommended that the operator interface creates an Event for the remote asset
to send Report Drawing Definition and Report Data messages upon a change to receive any
updates that may occur.</description>
  <assumptions>Messages may be delayed, lost or reordered.</assumptions>
  <references>
    <inherits_from name="Events" id="urn:jaus:jss:core:Events" version="1.0"/>
  </references>
  <declared_type_set name="Types">
    <declared_type_set_ref name="HMI" id="urn:jaus:jss:HMI:MessageSet" version="1.0"/>
  </declared_type_set>
  <message_set>
    <input_set>
      <declared_message_def name="QueryDrawingDefinition"
declared_type_ref="HMI.queryClass.QueryDrawingDefinition"/>
      <declared_message_def name="QueryDataDefinition"
declared_type_ref="HMI.queryClass.QueryDataDefinition"/>
      <declared_message_def name="QueryData"
declared_type_ref="HMI.queryClass.QueryData"/>
    </input_set>
    <output_set>
      <declared_message_def name="ReportDrawingDefinition"
declared_type_ref="HMI.informClass.ReportDrawingDefinition"/>
      <declared_message_def name="ReportDataDefinition"
declared_type_ref="HMI.informClass.ReportDataDefinition"/>
      <declared_message_def name="ReportData"
declared_type_ref="HMI.informClass.ReportData"/>
    </output_set>
  </message_set>
  <internal_events_set/>
  <protocol_behavior>
    <start state_machine_name="events.transport.ReceiveFSM"
state_name="Receiving.Ready.StateA"/>
    <state_machine name="events.transport.ReceiveFSM" interpretation="extending ReceiveFSM
of base service (transport)">
      <state name="Receiving" initial_state="Ready" interpretation="redefine state in
order to extend">
        <state name="Ready" initial_state="StateA">
          <state name="StateA" interpretation="redefine state in order to extend">
            <default_state>
              <transition name="events.transport.Receive">
```

```
                <parameter type="QueryDrawingDefinition" value="msg" interpretation="query
drawing definition message"/>
                <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                <simple/>
                <action name="events.transport.Send" interpretation="send a report drawing
definition message">
                  <argument value=" 'sendReportDrawingDefinition'"/>
                  <argument value="transportData"/>
                </action>
              </transition>
              <transition name="events.transport.Receive">
                <parameter type="QueryDataDefinition" value="msg" interpretation="query
data definition message"/>
                <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                <simple/>
                <action name="events.transport.Send" interpretation="send a report data
definition message">
                  <argument value="'sendReportDataDefinition'"/>
                  <argument value="transportData"/>
                </action>
              </transition>
              <transition name="events.transport.Receive">
                <parameter type="QueryData" value="msg" interpretation="query data
message"/>
                <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                <simple/>
                <action name="events.transport.Send" interpretation="send a report data
message">
                  <argument value="'sendReportData'"/>
                  <argument value="transportData"/>
                </action>
              </transition>
            </default_state>
          </state>
        </state>
      </state>
    </state_machine>
  </protocol_behavior>
</service_def>
```

A.2 POINTING DEVICE

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def xmlns="urn:jaus:jsidl:1.0" name="PointingDevice" id="
urn:jaus:jss:HMI:PointingDevice" version="1.0">
  <description>The Pointing Device Movement Service is used to detect mouse, or other
similar pointing device coordinate position on the Graphics Viewport.  The Pointing Device
Movement Service is used in conjunction with the Drawing Service.
The remote asset may create an Event for the operator interface to send Report Pointing
Device Movement messages.  This Event can either be upon a change or at a periodic
frequency.  Upon receipt of this Event creation, the operator interface shall return
Report Pointing Device Movement messages at the requested interval.
The remote asset may create an Event for the operator interface to send Report Pointing
Device Action messages.  This Event can either be upon a change or at a periodic
frequency.  Upon receipt of this Event creation, the operator interface shall return
Report Pointing Device Action messages at the requested interval.</description>
  <assumptions>Messages may be delayed, lost or reordered.</assumptions>
  <references>
    <inherits_from name="Events" id="urn:jaus:jss:core:Events" version="1.0"/>
  </references>
```

```
  <declared_type_set name="Types">
    <declared_type_set_ref name="HMI" id="urn:jaus:jss:HMI:MessageSet" version="1.0"/>
  </declared_type_set>
  <message_set>
    <input_set>
      <declared_message_def name="QueryPointingDeviceMovement"
declared_type_ref="HMI.queryClass.QueryPointingDeviceMovement"/>
      <declared_message_def name="QueryPointingDeviceAction"
declared_type_ref="HMI.queryClass.QueryPointingDeviceAction"/>
    </input_set>
    <output_set>
      <declared_message_def name="ReportPointingDeviceMovement"
declared_type_ref="HMI.informClass.ReportPointingDeviceMovement"/>
      <declared_message_def name="ReportPointingDeviceAction"
declared_type_ref="HMI.informClass.ReportPointingDeviceAction"/>
    </output_set>
  </message_set>
  <internal_events_set/>
  <protocol_behavior>
    <start state_machine_name="events.transport.ReceiveFSM"
state_name="Receiving.Ready.StateA"/>
    <state_machine name="events.transport.ReceiveFSM" interpretation="extending ReceiveFSM
of base service (transport)">
      <state name="Receiving" initial_state="Ready" interpretation="redefine state in
order to extend">
        <state name="Ready" initial_state="StateA">
          <state name="StateA" interpretation="redefine state in order to extend">
            <default_state>
              <transition name="events.transport.Receive">
                <parameter type="QueryPointingDeviceMovement" value="msg"
interpretation="query pointing device movement message"/>
                <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                <guard condition="pointingDeviceExists( msg )" interpretation="True if a
pointing device exists on the client interface"/>
                <simple/>
                <action name="events.transport.Send" interpretation="send a report
pointing device message">
                  <argument value="'sendReportPointingDeviceMovement'"/>
                  <argument value="transportData"/>
                </action>
              </transition>
              <transition name="events.transport.Receive">
                <parameter type="QueryPointingDeviceAction" value="msg"
interpretation="query pointing device action message"/>
                <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                <guard condition="pointingDeviceExists( msg )" interpretation="True if a
pointing device exists on the client interface"/>
                <simple/>
                <action name="events.transport.Send" interpretation="send a report
pointing device action message">
                  <argument value="'sendReportPointingDeviceAction'"/>
                  <argument value="transportData"/>
                </action>
              </transition>
            </default_state>
          </state>
        </state>
      </state>
    </state_machine>
  </protocol_behavior>
</service_def>
```

A.3 KEYBOARD

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def xmlns="urn:jaus:jsidl:1.0" name="Keyboard" id="urn:jaus:jss:HMI:Keyboard"
version="1.0">
  <description>An operator interface that has a keyboard device attached may implement the
Keyboard Service to return key strokes to the remote asset.
The remote asset may create an Event for the operator interface to send Report Keyboard
Action messages.  It is highly recommended that this Event is set up for on change and not
periodic frequency.</description>
  <assumptions>Messages may be delayed, lost or reordered.</assumptions>
  <references>
    <inherits_from name="Events" id="urn:jaus:jss:core:Events" version="1.0"/>
  </references>
  <declared_type_set name="Types">
    <declared_type_set_ref name="HMI" id="urn:jaus:jss:HMI:MessageSet" version="1.0"/>
  </declared_type_set>
  <message_set>
    <input_set>
      <declared_message_def name="QueryKeyboardAction"
declared_type_ref="HMI.queryClass.QueryKeyboardAction"/>
    </input_set>
    <output_set>
      <declared_message_def name="ReportKeyboardAction"
declared_type_ref="HMI.informClass.ReportKeyboardAction"/>
    </output_set>
  </message_set>
  <internal_events_set/>
  <protocol_behavior>
    <start state_machine_name="events.transport.ReceiveFSM"
state_name="Receiving.Ready.StateA"/>
    <state_machine name="events.transport.ReceiveFSM" interpretation="extending ReceiveFSM
of base service (transport)">
      <state name="Receiving" initial_state="Ready" interpretation="redefine state in
order to extend">
        <state name="Ready" initial_state="StateA">
          <state name="StateA" interpretation="redefine state in order to extend">
            <default_state>
              <transition name="events.transport.Receive">
                <parameter type="QueryKeyboardAction" value="msg" interpretation="query
keyboard action message"/>
                <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                <guard condition="keyboardExists( msg )" interpretation="True if a
keyboard exists on the client interface"/>
                <simple/>
                <action name="events.transport.Send" interpretation="send a report
keyboard action message">
                  <argument value="'sendReportKeyboardAction'"/>
                  <argument value="transportData"/>
                </action>
              </transition>
            </default_state>
          </state>
        </state>
      </state>
    </state_machine>
  </protocol_behavior>
</service_def>
```


A.4 DIGITAL CONTROL

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def xmlns="urn:jaus:jsidl:1.0" name="DigitalControl"
id="urn:jaus:jss:HMI:DigitalControl" version="1.0">
  <description>An operator interface that has digital controls, such as a button, either
as software or hardware may implement the Digital Control Service that would send a Report
DCM message to the remote asset when a digital control push or release occurs.  The
Digital Control Service is used in conjunction with the Drawing Service.
The remote asset may send a Set DCM message to the operator interface to control the
current state of a digital control.  This allows the operator interface to alter the
display of the referenced digital control to be displayed as inset or released, active or
inactive and flashing or steady.
The remote asset may create an Event for the operator interface to send Digital Control
Messages (DCMs).  It is highly recommended that this Event is setup for on change and not
periodic frequency.</description>
  <assumptions>Messages may be delayed, lost or reordered.</assumptions>
  <references>
    <inherits_from name="accessControl" id="urn:jaus:jss:core:AccessControl"
version="1.0"/>
  </references>
  <declared_type_set name="Types">
    <declared_type_set_ref name="HMI" id="urn:jaus:jss:HMI:MessageSet" version="1.0"/>
  </declared_type_set>
  <message_set>
    <input_set>
      <declared_message_def name="QueryDCM" declared_type_ref="HMI.queryClass.QueryDCM"/>
      <declared_message_def name="SetDCM" declared_type_ref="HMI.commandClass.SetDCM"/>
    </input_set>
    <output_set>
      <declared_message_def name="ReportDCM"
declared_type_ref="HMI.informClass.ReportDCM"/>
    </output_set>
  </message_set>
  <internal_events_set/>
  <protocol_behavior>
    <start state_machine_name="accessControl.events.transport.ReceiveFSM"
state_name="Receiving.Ready.NotControlled.StateA"/>
    <state_machine name="accessControl.events.transport.ReceiveFSM"
interpretation="extending ReceiveFSM of base service (transport)">
      <state name="Receiving" initial_state="Ready" interpretation="redefine state in
order to extend">
        <state name="Ready" initial_state="NotControlled" interpretation="redefine state
in order to extend">
          <state name="NotControlled" initial_state="StateA" interpretation="redefine
state in order to extend">
            <state name="StateA" interpretation="redefine state in order to extend">
              <default_state>
                <transition name="accessControl.events.transport.Receive">
                  <parameter type="QueryDCM" value="msg" interpretation="query DCM
message"/>
                  <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                  <simple/>
                  <action name="accessControl.events.transport.Send" interpretation="send
a report DCM message">
                    <argument value="'ReportDCM'"/>
                    <argument value="transportData"/>
                  </action>
                </transition>
              </default_state>
            </state>
          </state>
          <state name="Controlled" initial_state="StateB" interpretation="redefine state
in order to extend">
```

```
            <state name="StateB" interpretation="redefine state in order to extend">
              <state name="Ready">
                <transition name="accessControl.events.transport.Receive">
                  <parameter type="SetDCM" value="msg" interpretation="set DCM message"/>
                  <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                  <simple/>
                  <action name="setDCM" interpretation="set the digital control style on
the operator interface."/>
                </transition>
              </state>
              <default_state>
                <transition name="accessControl.events.transport.Receive">
                  <parameter type="QueryDCM" value="msg" interpretation="query DCM
message"/>
                  <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                  <simple/>
                  <action name="accessControl.events.transport.Send" interpretation="send
a report DCM message">
                    <argument value="'ReportDCM'"/>
                    <argument value="transportData"/>
                  </action>
                </transition>
              </default_state>
            </state>
          </state>
        </state>
      </state>
    </state_machine>
  </protocol_behavior>
</service_def>
```

A.5 ANALOG CONTROL

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def xmlns="urn:jaus:jsidl:1.0" name="AnalogControl" id="
urn:jaus:jss:HMI:AnalogControl" version="1.0">
  <description>The Analog Control Service is used to for the generation of analog operator
interactions.  Six fields represent the proportion of movement of an analog device in
linear and rotational movement of the three cardinal axis.
The Analog Control Service does not define co-ordinate systems.  However, the
interpretation of operator inputs shall assume the following orientation of movement.  The
longitudinal axis is positive in the forward direction, lateral movement is positive to
the right and vertical movement is positive downwards.  Rotational axial values increase
in a clockwise direction when looking along the axis towards the greater positive value.
Roll is rotation around the longitudinal axis, pitch around the lateral axis and yaw
around the vertical axis.
The remote asset may create an Event for the operator interface to send Report Analog
Action messages.  This event can either be upon a change or at a periodic frequency.  Upon
receipt of this event creation, the operator interface shall return Report Analog Action
messages at the requested interval.</description>
  <assumptions>Messages may be delayed, lost or reordered.</assumptions>
  <references>
    <inherits_from name="Events" id="urn:jaus:jss:core:Events" version="1.0"/>
  </references>
  <declared_type_set name="Types">
    <declared_type_set_ref name="HMI" id="urn:jaus:jss:HMI:MessageSet" version="1.0"/>
  </declared_type_set>
  <message_set>
    <input_set>
      <declared_message_def name="QueryAnalogDevices"
declared_type_ref="HMI.queryClass.QueryAnalogDevices"/>
```

```
      <declared_message_def name="QueryAnalogAction"
declared_type_ref="HMI.queryClass.QueryAnalogAction"/>
    </input_set>
    <output_set>
      <declared_message_def name="ReportAnalogDevices"
declared_type_ref="HMI.informClass.ReportAnalogDevices"/>
      <declared_message_def name="ReportAnalogAction"
declared_type_ref="HMI.informClass.ReportAnalogAction"/>
    </output_set>
  </message_set>
  <internal_events_set/>
  <protocol_behavior>
    <start state_machine_name="events.transport.ReceiveFSM"
state_name="Receiving.Ready.StateA"/>
    <state_machine name="events.transport.ReceiveFSM" interpretation="extending ReceiveFSM
of base service (transport)">
      <state name="Receiving" initial_state="Ready" interpretation="redefine state in
order to extend">
        <state name="Ready" initial_state="StateA">
          <state name="StateA" interpretation="redefine state in order to extend">
            <default_state>
              <transition name="events.transport.Receive">
                <parameter type="QueryAnalogDevices" value="msg" interpretation="query
analog devices message"/>
                <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                <simple/>
                <action name="events.transport.Send" interpretation="send a report analog
devices message">
                  <argument value="'sendReportAnalogDevices'"/>
                  <argument value="transportData"/>
                </action>
              </transition>
              <transition name="events.transport.Receive">
                <parameter type="QueryAnalogAction" value="msg" interpretation="query
analog action message"/>
                <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                <simple/>
                <action name="events.transport.Send" interpretation="send a report analog
action message">
                  <argument value="'sendReportAnalogAction'"/>
                  <argument value="transportData"/>
                </action>
              </transition>
            </default_state>
          </state>
        </state>
      </state>
    </state_machine>
  </protocol_behavior>
</service_def>
```

APPENDIX B– XML FOR DECLARED TYPE SETS

B.1 HMI SUBGROUP MESSAGE DEFINITIONS

```
<?xml version="1.0" encoding="UTF-8"?>
<declared_type_set xmlns="urn:jaus:jsidl:1.0" name="MessageSet"
id="urn:jaus:jss:HMI:MessageSet" version="1.0">
  <declared_type_set_ref name="commandClass" id="urn:jaus:jss:HMI:MessageSet:CommandClass"
version="1.0"/>
  <declared_type_set_ref name="informClass" id="urn:jaus:jss:HMI:MessageSet:InformClass"
version="1.0"/>
  <declared_type_set_ref name="queryClass" id="urn:jaus:jss:HMI:MessageSet:QueryClass"
version="1.0"/>
</declared_type_set>
```

B.2 COMMAND CLASS

```
<?xml version="1.0" encoding="UTF-8"?>
<declared_type_set xmlns="urn:jaus:jsidl:1.0" name="CommandClass"
id="urn:jaus:jss:HMI:MessageSet:CommandClass" version="1.0">
  <declared_type_set_ref name="HMITypes" id="urn:jaus:jss:HMI:MessageSet:HMITypes"
version="1.0"/>
  <message_def name="SetDCM" message_id="0706" is_command="true">
    <description xml:space="preserve">
      This message shall set the digital control style on the operator interface.
    </description>
    <declared_header name="MsgHeader" declared_type_ref="basicTypes.JAUSMessageHeader"/>
    <body name="Body">
      <record name="DCMRec" optional="false">
        <fixed_field name="DCMReference" field_type="unsigned short integer"
field_units="one" optional="false" interpretation="Unique reference to the digital
control."/>
        <bit_field name="DCMStateCode" field_type_unsigned="unsigned byte"
optional="false" interpretation="The state of the digital control.">
          <sub_field name="Updown">
            <bit_range from_index="0" to_index="0"/>
            <value_set offset_to_lower_limit="false">
              <value_enum enum_index="0" enum_const="Released"/>
              <value_enum enum_index="1" enum_const="Pressed"/>
            </value_set>
          </sub_field>
          <sub_field name="Active">
            <bit_range from_index="1" to_index="1"/>
            <value_set offset_to_lower_limit="false">
              <value_enum enum_index="0" enum_const="Inactive"/>
              <value_enum enum_index="1" enum_const="Active"/>
            </value_set>
          </sub_field>
          <sub_field name="Flash">
            <bit_range from_index="2" to_index="2"/>
            <value_set offset_to_lower_limit="false">
              <value_enum enum_index="0" enum_const="Steady"/>
              <value_enum enum_index="1" enum_const="Flashing"/>
            </value_set>
          </sub_field>
          <sub_field name="FlashRate" interpretation="Rate at which to flash the digital
control in Hz.  Only valid for Flashing state.">
            <bit_range from_index="3" to_index="4"/>
            <value_set offset_to_lower_limit="false">
              <value_range lower_limit="0" upper_limit="3" lower_limit_type="inclusive"
upper_limit_type="inclusive"/>
            </value_set>
          </sub_field>
```

```
        </bit_field>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
</declared_type_set>
```

B.3 QUERY CLASS

```
<?xml version="1.0" encoding="UTF-8"?>
<declared_type_set xmlns="urn:jaus:jsidl:1.0" name="QueryClass"
id="urn:jaus:jss:HMI:MessageSet:QueryClass" version="1.0">
  <declared_type_set_ref name="HMITypes" id="urn:jaus:jss:HMI:MessageSet:HMITypes"
version="1.0"/>
  <message_def name="QueryDrawingDefinition" message_id="2700" is_command="false">
    <description xml:space="preserve">
      This message shall cause the receiving component to reply to the requestor with a
      ID 4700h: ReportDrawingDefinition message.
    </description>
    <declared_header name="MsgHeader" declared_type_ref="basicTypes.JAUSMessageHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryDataDefinition" message_id="2701" is_command="false">
    <description xml:space="preserve">
      This message shall cause the receiving component to reply to the requestor with a
      ID 4701h: ReportDataDefinition message.
    </description>
    <declared_header name="MsgHeader" declared_type_ref="basicTypes.JAUSMessageHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryData" message_id="2702" is_command="false">
    <description xml:space="preserve">
      This message shall cause the receiving component to reply to the requestor with a
      ID 4702h: ReportData message.
    </description>
    <declared_header name="MsgHeader" declared_type_ref="basicTypes.JAUSMessageHeader"/>
    <body name="Body">
      <record name="QueryDataRec" optional="false">
        <fixed_field name="DataStructureReference" field_type="unsigned short integer"
field_units="one" optional="false" interpretation="Unique reference to the required data
structure used in the system."/>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryPointingDeviceMovement" message_id="2703" is_command="false">
    <description xml:space="preserve">
      This message shall cause the receiving component to reply to the requestor with a
      ID 4703h: ReportPointingDeviceMovement message.
    </description>
    <declared_header name="MsgHeader" declared_type_ref="basicTypes.JAUSMessageHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryPointingDeviceAction" message_id="2704" is_command="false">
    <description xml:space="preserve">
      This message shall cause the receiving component to reply to the requestor with a
      ID 4704h: ReportPointingDeviceAction message.
    </description>
    <declared_header name="MsgHeader" declared_type_ref="basicTypes.JAUSMessageHeader"/>
    <body name="Body"/>
```