THE JOINT ARCHITECTURE FOR UNMANNED SYSTEMS

Reference Architecture Specification

Volume II, Part 2

Message Definition

Version 3.3

June 27, 2007

TABLE OF CONTENTS

Title **Page**

TABLE OF FIGURES

TABLE OF TABLES

1 INTRODUCTION

This document, Part 2 of the Joint Architecture for Unmanned Systems (JAUS) Reference Architecture Specification specifies the JAUS specific protocol for transmission of JAUS messages. In support of this discussion, the JAUS Standard data types and data models are presented first. Following the Standards, this document specifies the JAUS message header and the types of JAUS messages. The specification focuses on the rules for messaging as opposed to the domain specific semantics contained in the message set itself.

2 JAUS STANDARDS

2.1 Textual Data Representation

Textual data shall conform to the Latin-1 ISO/IEC 8859 Latin-1 standard character set. Fixed length character strings (Strings) shall not require a terminator. The length of a fixed length string shall be its declared length. Variable length strings shall use the NUL (0 decimal, 00 hex) character for termination. The use of the NUL character for string termination is consistent with current best practice.

2.2 Numerical Data Representation

The reference architecture defines basic numeric data types and their representations. These data types are used in message definitions to define the data format of message parameters. **Table 2-1** lists the data types and their representation.

Table 2-1 - JAUS Data Types

Data Type	Size (in Bytes)	Representation
Byte	1	8 bit unsigned integer
Short Integer	2	16 bit signed integer
Integer	4	32 bit signed integer
Long Integer	8	64 bit signed integer
Unsigned Short Integer	2	16 bit unsigned integer
Unsigned Integer	4	32 bit unsigned integer
Unsigned Long Integer	8	64 bit unsigned integer
Float	4	IEEE 32 bit floating point number
Long Float	8	IEEE 64 bit floating point number

2.2.1 Scaled Integers

Real numbers can be represented as integer values (signed - deprecate in v4.0, or unsigned) by specifying a ***scale factor*** and ***bias***. This type of representation is referred to as a ***Scaled Integer***. A scaled integer is based upon one of the seven integer types as listed above.

2.2.1.1 Unsigned Integer Bit Ranges

The integer bit range for any unsigned integer type is $2^n - 1$, where n is the number of bits in the integer. For an Unsigned Short Integer, the upper range is 65,535 or $2^{16} - 1$.

2.2.1.2 Signed Integer Bit Ranges (Deprecate in v4.0)

The integer bit range for any signed integer type is $2^{(number\_of\_bits - 1)} - 1$, where n is the number of bits in the integer. For a Short integer, the upper range is 32767 or $2^{15} - 1$.

2.2.1.3 Computing the Bias and Scale Factor - Real to Unsigned Integer

The scale factor for a real represented by an unsigned integer type is calculated as follows:

```
Integer_Range = 2^(number_of_bits) – 1
Scale_Factor = (Real_Upper_Limit – Real_Lower_Limit)/Integer_Range
Bias = Real_Lower_Limit
```

2.2.1.4 Computing the Bias and Scale Factor - Real to Signed Integer (Deprecate in v4.0)

The scale factor and bias for a real represented by a signed integer are calculated as follows:

```
Integer_Range = 2 * (2^(number_of_bits - 1) – 1)
Scale_Factor = (Real_Upper_Limit – Real_Lower_Limit)/Integer_Range
Bias = (Real_Lower_Limit + Real_Upper_Limit)/2
```

2.2.1.5 Converting Between Integers and Real Values

The formulas for converting from an integer value to a real value and visa versa are:

```
Integer_Value = Round((Real_Value – Bias)/Scale_Factor)
Real_Value = Integer_Value*Scale_Factor + Bias
```

2.2.1.6 Example Conversion

An example of using a scaled integer value follows. Given a lower limit value of –100, and an upper limit value of 100, the scale factor and bias for a short integer type would be.

```
Scale_Factor = (100 – (-100))/(2*(2^15 – 1)) = 0.003051851
Bias = (-100 + 100) / 2 = 0.0
```

To convert the real value of 30 to its scaled integer representation, the following formula is used.

```
Integer_Value = (30.0 - Bias)/Scale_Factor = 9830
```

Note: Rounding is used to obtain the integer value, not truncation.

The node that receives the data value (9830) can determine the real number value with the following calculation.

```
Real_Value = 9830*Scale_Factor + Bias = 29.9997
```

2.2.1.7 Conversion Tables and Formulas

Table 2-2, **Table 2-3**, and **Table 2-4** show the specific calculations used for the integer types defined in **Table 2-1**. A Byte (unsigned 8 bit integer) can also be used as a scaled integer, but the formula is not explicitly given in a table. Shorthand used in the following tables includes:

Min = Real_Lower_Limit,
Max = Real_Upper_Limit,
R = Real_Value, and
I = Integer_Value

Table 2-2 - Scaled Integer Formulas for 16 Bit Integers

Operation	**Unsigned Short (16 Bits)**	**Signed Short (16 Bits) deprecate in v4.0**
Real Value to Integer Value	$I = (R - Min) * \left(\frac{(2^{16} - 1)}{(Max - Min)}\right)$	$I = \left(R - \frac{Max + Min}{2}\right) * 2\left(\frac{(2^{15} - 1)}{Max - Min}\right)$
Integer Value to Real Value	$R = I\left(\frac{(Max - Min)}{(2^{16} - 1)}\right) + Min$	$R = \frac{I}{2}\left(\frac{(Max - Min)}{(2^{15} - 1)}\right) + \frac{(Max + Min)}{2}$

Table 2-3 - Scaled Integer Formulas for 32 Bit Integers

Operation	Unsigned Integer (32 Bits)	Signed Integer (32 Bits) deprecate in v4.0
Real Value to Integer Value	$I = (R - Min) * \left(\frac{(2^{32} - 1)}{(Max - Min)} \right)$	$I = \left(R - \frac{(Max + Min)}{2} \right) * 2 \left(\frac{(2^{31} - 1)}{(Max - Min)} \right)$
Integer Value to Real Value	$R = I \left(\frac{(Max - Min)}{(2^{32} - 1)} \right) + Min$	$R = \frac{I}{2} \left(\frac{(Max - Min)}{(2^{31} - 1)} \right) + \frac{(Max + Min)}{2}$

Table 2-4 - Scaled Integer Formulas for 64 Bit Integers

Operation	Unsigned Long (64 Bits)	Signed Long (64 Bits) deprecate in v4.0
Real Value to Integer Value	$I = (R - Min) * \left(\frac{(2^{64} - 1)}{(Max - Min)} \right)$	$I = \left(R - \frac{(Max + Min)}{2} \right) * 2 \left(\frac{(2^{63} - 1)}{(Max - Min)} \right)$
Integer Value to Real Value	$R = I \left(\frac{(Max - Min)}{(2^{64} - 1)} \right) + Min$	$R = \frac{I}{2} \left(\frac{(Max - Min)}{(2^{63} - 1)} \right) + \frac{(Max + Min)}{2}$

2.3 Byte Ordering

Machine architectures interpret data in different ways and JAUS specifies *Little Endian* byte ordering as the default method that shall be used for all message transactions between nodes. Data communicated between nodes shall be formatted into byte streams for transmission. Data streams that span more than one byte shall be formatted to transmit the least significant byte first.

2.4 Platform Orientation

The parameters Ψ, θ, and ϕ define the orientation of the vehicle. This definition is illustrated in **Figure 2.1** where an XYZ coordinate system has been attached to the vehicle with its X axis pointed forward and its Z axis downward.

Global platform orientation is defined by initially aligning the X axis in a northerly direction and the Z axis along the gravity vector as shown in (a). The vehicle is then rotated by an angle Ψ in a right-handed sense about the Z axis as shown in (b). Subsequently the vehicle is rotated by an angle θ about the modified Y axis as shown in (c) followed by a rotation of ϕ about the modified X axis as shown in (d). Local platform orientation is defined in a similar manner by initially aligning the platform coordinate system with the local coordinate system and then rotating it in the same sequence as defined for the global platform orientation.

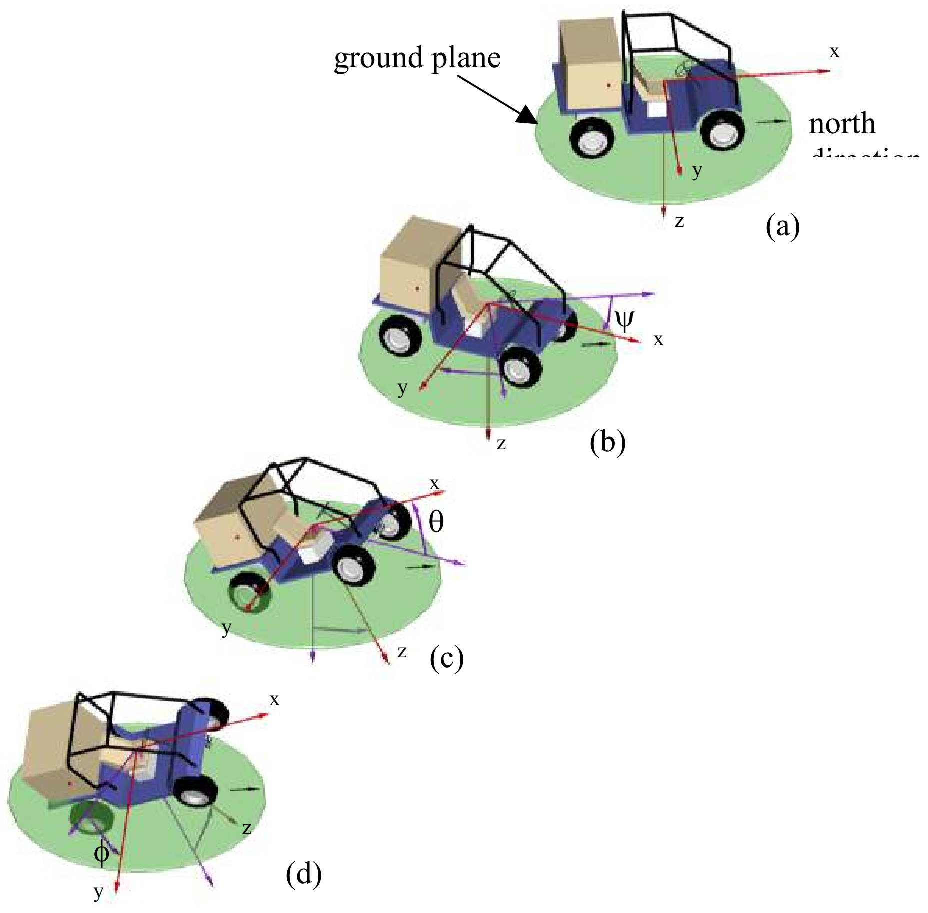
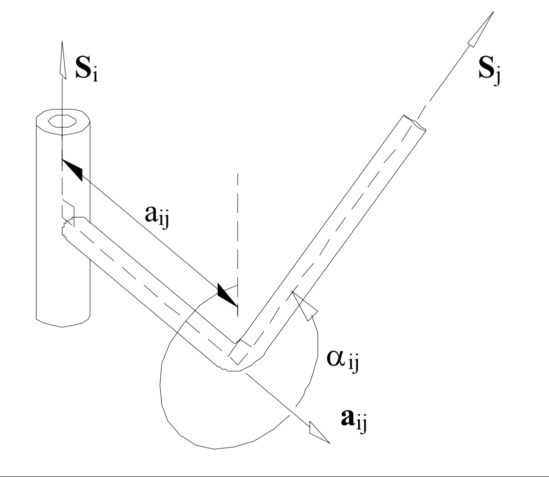


Figure 2.1 - Definition of Platform Orientation

2.5 Manipulator Linkage Notation

Manipulators are often used on Unmanned Systems to alter the environment. To standardize the method of referencing manipulator links (or joints), the following figures illustrate the notation that will be used later in the document. **Figure 2.2** illustrates the parameters used for link ij. **Figure 2.3** and **Figure 2.4** illustrates additional parameters especially used for rotational joints and prismatic joints respectively.



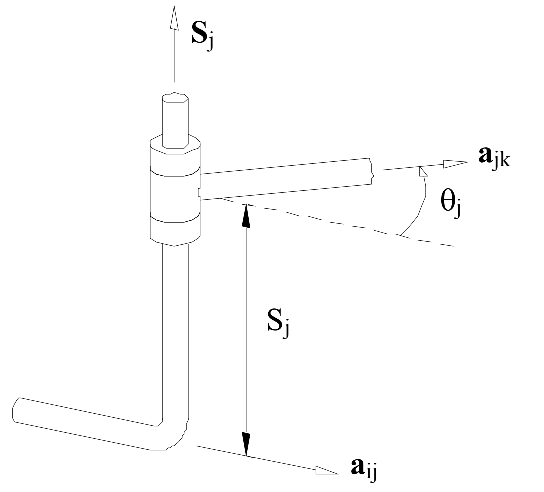
a_{ij} – link length
α_{ij} – twist angle

$\mathbf{a}_{ij}$ – unit vector along link ij
$\mathbf{S}_i$ – unit vector along joint axis i
$\mathbf{S}_j$ – unit vector along joint axis j

Link length a_{ij} is measured as the perpendicular distance between joint axis i and joint axis j along the vector $\mathbf{a}_{ij}$. Note that it can have a negative value.

Twist angle α_{ij} is the angle between $\mathbf{S}_i$ and $\mathbf{S}_j$ measured in a right hand sense about $\mathbf{a}_{ij}$.

Figure 2.2 – Manipulator Linkage Parameters for link ij



S_j – constant joint offset distance
θ_j – variable joint angle

$\mathbf{a}_{ij}$ – unit vector along link ij
$\mathbf{a}_{jk}$ – unit vector along link jk
$\mathbf{S}_j$ – unit vector along joint axis j

Joint offset S_j is measured as the perpendicular distance between link a_{ij} and link a_{jk} along the vector $\mathbf{S}_j$. Note that it can have a negative value.

Joint angle θ_j is the angle between $\mathbf{a}_{ij}$ and $\mathbf{a}_{jk}$ measured in a right hand sense about $\mathbf{S}_j$.

Figure 2.3 – Manipulator Linkage Parameters for Revolute Joint j

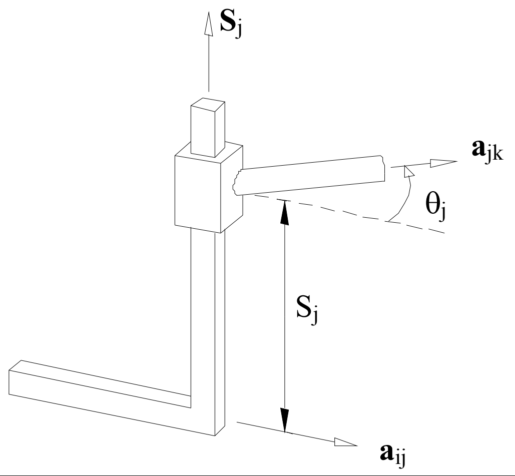
 $\mathbf{S}_j$ $\mathbf{a}_{jk}$ θ_j S_j $\mathbf{a}_{ij}$	S_j – variable joint offset distance θ_j – constant joint angle $\mathbf{a}_{ij}$ – unit vector along link ij $\mathbf{a}_{jk}$ – unit vector along link jk $\mathbf{S}_j$ – unit vector along joint axis j Joint offset S_j is measured as the perpendicular distance between link a_{ij} and link a_{jk} along the vector $\mathbf{S}_j$. Note that it can have a negative value. Joint angle θ_j is the angle between $\mathbf{a}_{ij}$ and $\mathbf{a}_{jk}$ measured in a right hand sense about $\mathbf{S}_j$.

Figure 2.4 – Manipulator Linkage Parameters for Prismatic Joint j

3 MESSAGE SPECIFICATION

3.1 Overview

The interoperability inherent in a JAUS compliant subsystem is due in large part to the use of a well-defined set of messages. Messages cause actions to commence, information to be exchanged and events to occur. Everything that occurs in a JAUS system is precipitated by messages. This strategy makes JAUS a component based, message-passing architecture.

All references to the term ***"message"*** that appear in this document refer explicitly to JAUS defined messages. All of the messages that JAUS currently recognizes are defined in Part 3 of this specification.

The following definitions are stated for proper interpretation of references made hereafter.

Reserved – Indicates that the range of values or bits cannot be used by developers and is reserved for future use. Reserved bits shall be set to zero unless otherwise noted.

Available – Indicates that the range of values or bits is not specified within JAUS. Developers may utilize the available range of values or bits for the sole purpose of debug and testing. Under normal operating conditions, these bits are set to zero.

Specified – Indicates that the range of values or bits are defined within JAUS. Developers must observe the defined range of values or bits.

3.2 Message Classes

JAUS defines six classes of messages at the component level. **Table 3.1** lists the message classes, segmented by command code, that are currently defined by JAUS.

Segmentation is used by JAUS as a method of organizing messages. Segmentation also allows message command codes to be masked prior to evaluation as an aid to determining the type of message transaction in process.

Table 3.1 - Segmentation of Command Codes by Class

Message Class	Offset Range (0000h to FFFFh)
Command	0000h – 1FFFh
Query	2000h – 3FFFh
Inform	4000h – 5FFFh
Event Setup	6000h – 7FFFh (Deprecate v4.0)
Event Notification	8000h – 9FFFh (Deprecate v4.0)
Node Management	A000h – BFFFh
Reserved	C000h – CFFFh
Experimental Message	D000h – FFFFh

3.2.1 Command Class Message

Command class messages are used to effect system mode changes, actuation control, alter the state of a component or subsystem or to otherwise initiate some type of action.

3.2.2 Query Class Messages

Query class messages are used to solicit information from another component. An inform class message is generated in response to a query class message.

3.2.3 Inform Class Messages

Inform class messages allow components to transmit information to each other. Status reports, geographic position, and state information are some examples of inform messages.

3.2.4 Event Setup Class Messages (Deprecate in v4.0)

Event setup class messages are used to setup the parameters for an Event Notification message and to have a component start monitoring for the trigger event. An Event Notification Message is generated whenever parameters are set within the Event Setup message.

3.2.5 Event Notification Class Messages (Deprecated in v4.0)

Event notification class messages communicate the occurrence of an event. Engine over-temperature, oil overpressure, etc. are examples of unsolicited events that can occur.

3.2.6 Node Management Class Message

Node Management class messages are only used by the Node Management task (see **Section 3.4.2.2**). These messages are used for node specific communications including the conveyance of configuration information and component registration.

3.2.7 Experimental Class Message

Experimental class messages are used to provide a mechanism for experimentation with new messages that are not defined within the JAUS RA. The intent is to have the experimental messages incorporated into a future version of the RA. When an Experimental class message is incorporated into the RA, it will be assigned a command code from one of the standard JAUS message classes. Due to interoperability issues, these messages should be kept to a minimum and should not be expected to exist outside the standard for a long period of time.

3.3 Message Composition

All messages are composed of a required header and the data fields. The format of the required header is common to all messages. This allows JAUS to employ an embedded protocol. Use of an embedded protocol means that certain fields within the header provide information on how to handle the message and on how data is encoded before transmission or decoded.

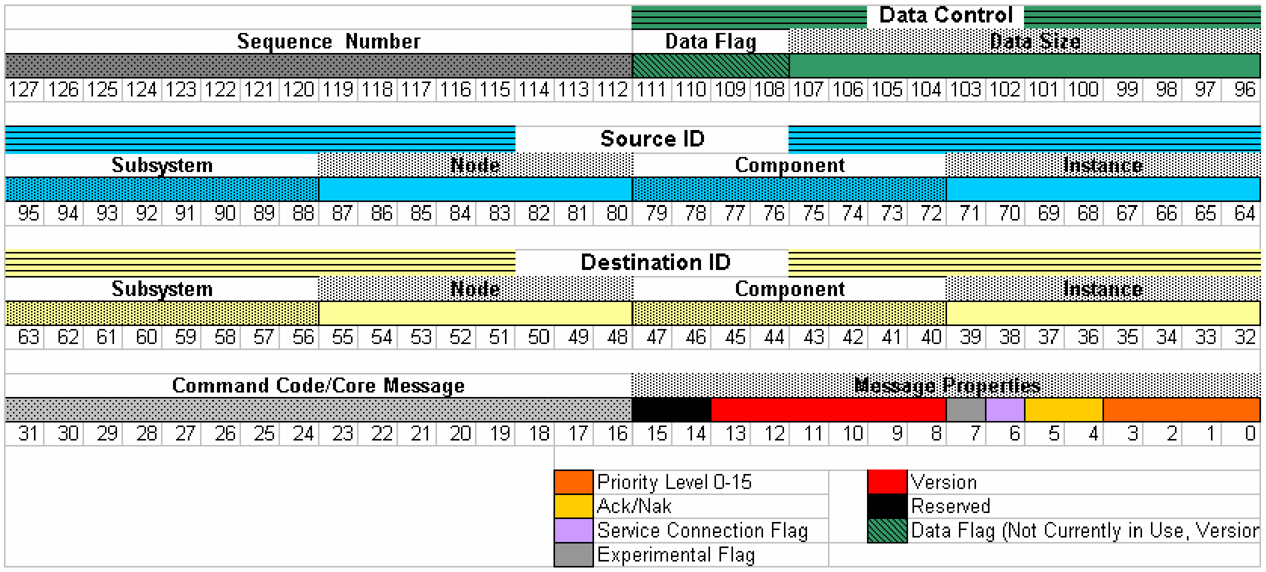


Figure 3.1 – JAUS Message Header Detail

3.3.1 Message Header

The message header format is shown in **Table 3.2**. Each field is interpreted as an unsigned integer value. The header contains information regarding its properties, data size, handling requirements, data encoding/decoding and message routing.

➢ *The message header, at a minimum, shall be included on all messages.*

Table 3.2 - Message Header Data Format

Field #	Field Description	Type	Size (Bytes)
1	Message Properties	Unsigned Short	2
2	Command Code	Unsigned Short	2
3	Destination Instance ID	Byte	1
4	Destination Component ID	Byte	1
5	Destination Node ID	Byte	1
6	Destination Subsystem ID	Byte	1
7	Source Instance ID	Byte	1
8	Source Component ID	Byte	1
9	Source Node ID	Byte	1
10	Source Subsystem ID	Byte	1
11	Data Control (bytes)	Unsigned Short	2
12	Sequence Number	Unsigned Short	2
	Total Bytes		**16**

3.3.1.1 Message Properties

The header defines the Message Properties in Field #1. The properties are segmented into six distinct bit-fields. The format of this 2-byte field is presented in Error! Reference source not found.**Figure 3.2**.

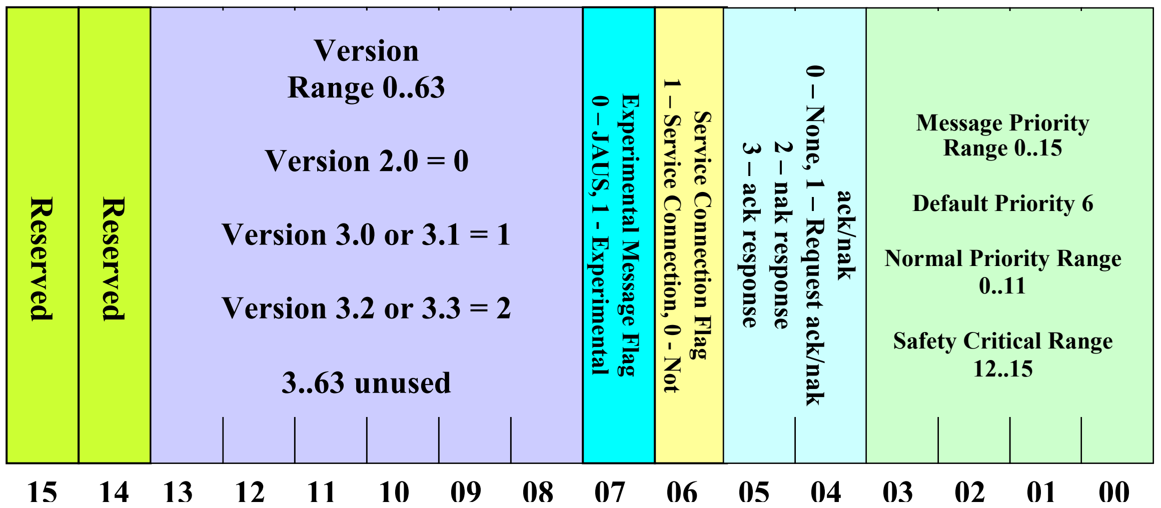


Figure 3.2 - Bit Layout of Message Properties

3.3.1.1.1 Priority

Property bits 0-3 are used to set the message priority. This field can support values ranging from 0 to 15. Priorities are defined as follows:

Normal Priority Message (Range 0-11)

Priority = 0	Low priority message
Priority = 6	Default priority message
Priority = 11	High priority message

Safety Critical Priority Message (Range 12-15)

Priority = 12	Lowest priority safety critical message
Priority = 15	Highest priority safety critical message

- *Safety critical messages always have priority over normal priority messages.*

3.3.1.1.2 ACK/NAKs

Property bits 4-5 are used to set the message Acknowledge/Negative Acknowledge (ACK/NAK) behavior. ACK/NAK bit usage is defined as follows:

Message Originator (0 or 1)

ACK/NAK = 0	No response required
ACK/NAK = 1	Response required

Message Responder (2 or 3)

ACK/NAK = 2	Message negative acknowledge
ACK/NAK = 3	Message acknowledged OK

- *Messages requiring ACK/NAK must provide a sequence number in Field #12.*

3.3.1.1.3 Special Bits

Property bit 6 indicates that the message is to be treated as a service connection. Service connection messages have different behaviors than normal messages. These differences are discussed in **Section 3.6**.

Property bit 7 is a flag used to indicate that the message is an experimental message. The command code for this message must be in the experimental command space ranging from D000h – FFFFh. Non-Experimental messages shall have a command code within one of the standard JAUS message class ranges, see **Table 3.1**. Use of this bit is discussed in **Section 3.7.2**.

3.3.1.1.4 Version

Property bits 8-13 are used to set the message version. The version refers to the JAUS RA version. This field is treated as a numeric, binary encoded field. It is to be encoded as follows:

0 = Version 2.0 and 2.1 compatible message

1 = Version 3.0 through 3.1 compatible message

2 = Version 3.2 and 3.3 compatible message

3 to 63 Reserved for Future Versions.

3.3.1.1.5 Reserved Bits

Bits 14 and 15 are reserved and each shall be set to zero.

3.3.1.2 Command Code

The header defines the Command Code in Field #2. The command code is a unique 2-byte numeric value that specifies the type of message, the size of its required data (if any) and its encoding/decoding characteristics.

3.3.1.3 Source and Destination IDs

The header defines message routing information in Fields #3-10. The source and destination of a message identify the sender (source) and receiver (destination) of the message. **Section 3.4** presents detailed information on routing.

3.3.1.4 Data Control

The header defines the Data Control in Field #11. Data control is implemented as two bit fields as follows:

• Data Size (Bits 0-11)	Size for up to 4080 bytes of data per transaction. The range for the bytes of data in the body of this message is 0 to 4080. the 12 bits provides for a range of 0 to 4095, but given the 16 byte JAUS header, 4080 is the maximum valid value.
• Data Flags (Bits 12-15)	Used to control single/multi packet transactions

The Data Size value represents the number of 8 bit bytes within the message body exclusive of the 16 header bytes. Thus, the total number of bytes within a single transaction is limited to 4096. Further rationale for this size is that support for a wide variety of processors and transport protocols is provided.

Section 3.5 presents detailed information on how large data sets shall be handled.

3.3.1.5 Sequence Number

The header defines the Sequence Number in Field #12. The sequence number is used to serialize messages. The first message in a serial message set shall use zero as the Sequence Number. Each successive message shall increment the sequence number by one. When the sequence number reaches its maximum unsigned value (65535), it shall reset back to zero.

When use of this field is mandatory, that fact is specified in the appropriate section.

3.3.2 Message Data

Any data that may be associated with a message is determined by the command code of the message. JAUS organizes all messages by command code and defines their data formats.

3.4 Message Routing

As mentioned previously, messages are routed according to the source and destination information contained within the header. The source and destination of a message equate to the sender and receiver.

The current reliance of JAUS on manual configuration of important parameters creates a significant impact on message routing. Since component IDs are the only IDs defined by JAUS, the subsystem(s) and node(s), must be assigned unique IDs by the engineer.

Component instancing is not directly effected by this limitation because multi-component instancing is most efficiently managed in software; i.e. the design of the component.

Messages shall follow the routing rules as follows:

- ***Messages to and from a Subsystem shall go through the Communicator.***
- ***Messages to and from a Node shall go through the Node Manager.***
- ***JAUS Messages internal to a Node shall go through the Node Manager.***

3.4.1 Composite IDs

The terms ***"source"*** and ***"destination"*** are used throughout this document to simplify routing terminology. They are really composites made up of the constituent Source and Destination ID elements defined in the header. For documentation purposes, the composite Destination ID and Source IDs are denoted by the mnemonic `xxx:xxx:xxx:xxx`. The first field represents the Subcomponent ID, the second field represents the Node ID, the third represents the Component ID, and the fourth field represents the Instance ID. The range of each ID is 0-255 inclusive.

3.4.2 Routing Scope

The byte-wide ID numbering scheme used by JAUS allows for a maximum of 256 different ID variations in each field. JAUS specifies the following ID field restrictions:

- ➢ *Zero is never used as a valid ID*
- ➢ *255 is always used as a broadcast ID*
- ➢ *254 IDs are valid and assignable*

Table 3.3, **Table 3.4**, and **Table 3.5** present the valid Component, Node, and Subsystem ID ranges defined within JAUS.

Table 3.3 - Component IDs

Component Type	Component ID Range	Number of IDs
Broadcast to all Components on Node	255	1
Invalid	0	1
Node Manager	1	1
User-Defined	2 … 31	30
Component Ids	32 … 254	223

Table 3.4 - Node IDs

Node Type	Node ID Range	Number of IDs
Broadcast to all Nodes on Subsystem	255	1
Invalid	0	1
Node Ids	1 … 254	254

Table 3.5 - Subsystem IDs

Subsystem Type	Subsystem ID Range	Number of IDs
Broadcast to all Subsystems in System	255	1
Invalid	0	1
Subsystem Ids	1 … 254	254

3.4.2.1 Broadcast

The Broadcast capability is provided primarily to simplify configuration and to reduce bandwidth utilization. It allows the originator of the broadcast message to send only one message and rely on the Node Manager facility to handle propagation to components.

Without the broadcast capability, the originator would have to send the message to each destination individually. This would require the originator to know of every subsystem, node and component in the overall system in order to be wholly broadcast capable.

A message that contains the broadcast value (255) in any ID field automatically requires that the message be treated as a Node Management class message. Broadcast handling shall comply with the examples presented in **Table 3.6**.

Table 3.6 - Required Broadcast Message Handling

Destination ID	Required Handling Of Broadcast Message
001:002:003:255	Broadcast to all instances of component 3 on node 2, subsystem 1
001:002:255:255	Broadcast to all instances of all components on node 2, subsystem 1
001:255:255:255	Broadcast to all instances of all components on all nodes, subsystem 1
255:255:255:255	Broadcast to all instances of all components on all nodes and all subsystems
001:255:003:255	Broadcast to all instances of component 3 on all nodes of subsystem 1
255:255:003:255	Broadcast to all instances of component 3 on all nodes and subsystems
001:255:003:001	Broadcast to component 3, instance 1 on all nodes of subsystem 1
255:255:003:001	Broadcast to component 3, instance 1 on all nodes and subsystems

3.4.2.2 Node Manager ID

The Node Manager has a special component ID. All nodes shall provide one, and only one, node manager task. The node manager task is responsible for five important operations. The scope of each of these operations is defined within the indicated section.

The Node Manager shall:

1. Handle all broadcast message propagation. See **Section3.4.2.1**
2. Handle all Node Management command class messages.
3. Provide all routing services of messages between components. See **Section 3.4**
4. Provide adequate handling of all service connection messages. See **Section 3.6**
5. Comply with all of the requirements of a standard JAUS component.

3.4.2.3 User Defined IDs

User-Defined component IDs are provided by JAUS so that message capable processes that are not JAUS component processes can also send and receive messages. These tasks might be client/server processes, monitoring, or any other facility that is designed into the system by the engineer and/or required buy the system functional specification. These component ID's are reserved for support tasks not already available through the existing component set.

3.5 Large Data Sets

As defined in **Section 3.3.1.4**, the header's Data Control field is segmented into two bit fields – the data size and data flags. Messages larger than 4095 bytes, including the 16-byte header, are considered large data sets. Transmission of dynamic configuration file data or streaming video data are examples of large data sets that will be supported in a future JAUS RA revision.

The data flag bit field is used to control data handling by the receiver. The bits are mutually exclusive and any occurrence of more than one bit set at a time, shall cause the message to be discarded (or NAK'ed if the message's ACK/NAK = 1).

Bit field usage is defined in **Table 3.7**.

Table 3.7 - Data Flag Bit field

Bit Value	Description
0000	Only data packet in single-packet stream
0001	First data packet in multi-packet stream
0010	Normal data packet
0100	Retransmitted data packet
1000	Last data packet in stream

The sender of a multi-packet data stream may require acknowledgement or not as best suits the implementation. ACK/NAK rules will be applied in either case.

The following rules shall be enforced when sending multi-packet messages:

- *The first message in the stream shall set the Data Flag Bit field = 1 decimal.*
- *The last message in the stream shall set the Data Flag Bit field = 8 decimal.*

- ➢ *The Sequence Number field shall identify each message. The first message shall use a sequence number of zero. Each successive message will be incremented by one.*
- ➢ *Each message between the first and last message shall set the flags = 2 initially. The packet shall be retransmitted when ACK was required, but NAK was returned. Retransmission will contain the same sequence number, but will set the flags = 4.*

3.6 Service Connections (Deprecate in V4.0)

Service Connections (SCs) are a feature of the JAUS RA designed to facilitate message transfer that takes place on a regular, periodic basis. The service connection is meant to establish a virtual data circuit between two components. A service connection shall be used to send an Inform or Command class message only.

The service connection bit is used to indicate to all message handlers that the command code is either a service connection management code or that the message is a service connection transaction. Transactions are any normal message traffic. Service connection commands are concerned with managing the connection. They are defined as follows:

• Create Service Connection	Create a new service connection
• Confirm Service Connection	Confirm creation of a service connection
• Activate Service Connection	Activate a suspended service connection
• Suspend Service Connection	Suspend an active service connection
• Terminate Service Connection	Kill a service connection regardless of state

Some fundamental rules apply to service connection messaging that differs from normal messaging. They are as follows:

- ➢ *Service connection messages are never queued. They are to be used immediately upon arrival or in the event that the message is stored in some intermediate data storage, then the data storage shall be overwritten each time another message is sent.*
- ➢ *Service connection messages are required to use the sequence number in the header.*
- ➢ *Service connection messages shall never set the ACK/NAK field to any value other than zero. All components are forbidden from acknowledging any service connection message.*
- ➢ *A service connection is unidirectional; messages always flow in the same direction.*
- ➢ *A service connection supports only the one command code for which it was created.*

Senders of service connection messages update the sequence number as an aid to message receivers. It helps the receiver to determine if the data is being under or over sampled. Because the message is always overwritten, the receiver only has access to the most recent data.

3.6.1 Establishing a Service Connection

The Node Manager shall provide facilities for handling service connections between JAUS components. A connection can be established between components on the same node or on different nodes. The supporting facilities of the node, which handle SC management, shall make sufficient provisions so as to handle all routing issues internally.

In the examples that follow, the distinction between handling SC messages and being SC capable will be made evident.

Two types of service connections are permitted; Inform or Command. The type of SC established is determined by the class of message that shall be sent through the SC. The SC sender shall send the required message once each period, as specified by the periodic rate established when the connection was confirmed.

Hereafter, all service connection handling mechanisms *provided by the Node Manager* shall be referred to simply as "**SC Management**". All subsequent discussions referring to duties provided by SC Management imply an SC capable node.

The following rules apply to establishing, managing and terminating all SCs:

- ➢ *All service connection command messages shall be sent from the source component to the destination component via all node SC Management services along the route.*
- ➢ *The SC Management service shall not be required to handle normal SC message traffic.*

3.6.1.1 The Inform Service Connection

Inform type service connections shall have only one message source and shall have at least one, and possibly many, destinations. A new destination is added each time another instance of the service connection is created by a different destination component. This is referred as a one-to-many type of service connection.

- The component receiving Inform messages via the service connection is referred to as the **“requester”** of the service.
- The component sending Inform messages via the service connection is referred to as the **“provider”** of the service.

When another SC instance is created, the provider shall determine if a periodic rate change is required. **Section 3.6.2** specifies how the service provider manages periodic rate.

All inform type service connections, are automatically connected by SC Management. It shall be the responsibility of SC Management to appropriately suspend and/or resume connections as commanded by the requester.

The creator of the inform SC is the only component legally able to send Activate, Suspend or Terminate Service Connection messages. They shall only be sent via SC Management. When all connections have been terminated, the SC shall be closed.

Inform type service connections are normally instantiated by components requiring access to information that is continually changing, such as position or velocity. The initial state of an inform SC is the active state.

Figure 3.3 presents a simple implementation of the one-to-many virtual switching needs of an inform service connection. All active virtual circuits are connected continuously.

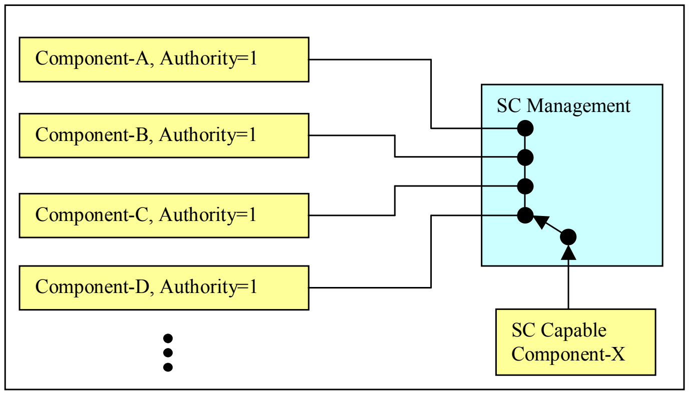


Figure 3.3 - Inform SC Switching

The following sequence shall be followed to establish an inform SC:

1. Component-A sends Create SC to Component-X.
2. The Node or Component-X replies with Confirm SC.
    1. Response Code = 0, Connection successful
    2. Response Code = 1, Node not SC capable
    3. Response Code = 2, Component not SC capable
3. The SC is immediately active (Component-A is now the SC destination).

Reminder: All of the above message transactions are sent via SC Management.

3.6.1.2 The Command Service Connection

Command type service connections shall have at least one, and possibly many, message sources and shall have only one destination. A new source is added each time another instance of the service connection is created by a different source component. This is referred as a many-to-one type of service connection.

- The component receiving Command messages via the service connection is referred to as the **"provider"** of the service.
- The component sending Command messages via the service connection is referred to as the **"commander"** of the service.

When another SC instance is created, the provider shall determine if a periodic rate change is required. **Section 3.6.2** specifies how the service provider manages periodic rate.

Command service connections are automatically connected or disconnected by SC Management. As connection states change, it shall be the responsibility of SC Management to appropriately suspend and/or resume all source components that must incur a change in state.

The following rules apply to establishing, managing and terminating all command SCs:

➢ *The creator of the SC must have an equal or higher authority than the component with which it connects or the connection shall be refused.*

➢ *The SC Management service shall only permit one active connection. The active connection shall be based on commander authority.*

- *The SC Management service shall keep track of the commander order of precedence, based on commander authority. This facility shall be responsible for automatically reactivating the next highest commander (when multiple commanders are known) that is in a suspended state, after the currently active commander terminates.*
- *The message queue of the SC provider shall ignore all command class messages that are sent to its message queue, other than core command class message, while any SC is active.*

The creator of the command SC is the only component, other than SC Management, legally able to send Activate, Suspend or Terminate Service Connection messages. They shall only be sent via SC Management. When all connections have been terminated, the SC shall be closed. The initial state of a command SC is the inactive state.

Figure 3.4 presents a simple implementation of the many-to-one virtual switching needs of a command service connection. Only the highest authority, active virtual circuit is connected at any given time.

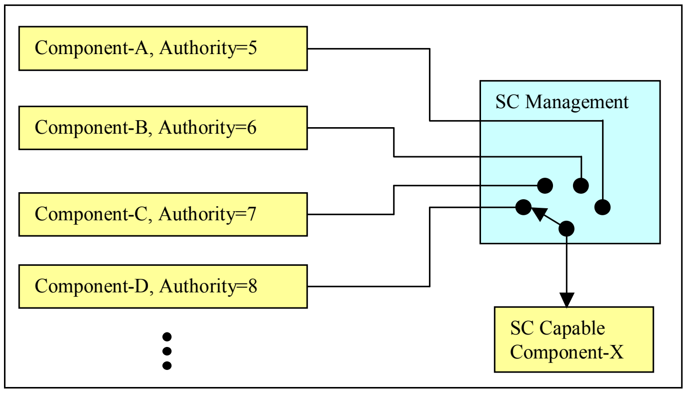


Figure 3.4 - Command SC Switching

The following sequence shall be followed to establish a command SC:

1. Component-A sends Create SC to Component-X.
2. The Node or Component-X replies with Confirm SC.
    a. Response Code = 0, Connection successful
    b. Response Code = 1, Node not SC capable
    c. Response Code = 2, Component not SC capable
3. Component-A is the proposed SC source commander (SC is inactive).

4. Component-A sends Activate SC to Component-X.
5. SC Management makes component-A the active SC source commander.

The following sequence shall be followed to establish a second command SC:

1. Component-B sends Create SC to Component-X.
2. Component-X replies with Confirm SC, Response Code = 0
3. Component-B is the proposed SC source commander (SC is inactive).
4. SC Management sends Suspend SC to Component-A.
5. Component-B sends Activate SC to Component-X.
6. SC Management makes component-B the active SC source commander.

The following sequence shall be followed to terminate the second command SC:

1. Component-B sends Terminate SC to Component-X.
2. SC Management sends Activate SC to Component-A.
3. SC Management reactivates component-A as the SC source commander.

Reminder: All of the above message transactions are sent via SC Management.

3.6.2 Service Connection Periodic Rate

Every SC is created with a specified periodic rate, which controls the interval at which messages are generated. Even when SC creators specify a rate higher than supportable by the provider, they shall still receive a positive confirmation from the SC provider (when SC capable). The confirmed rate shall indicate the rate was either accepted as specified or indicate the highest rate that the provider can support.

Upon receipt of the confirmed periodic rate, the SC creator shall accept the confirmed rate or if unacceptably low, send the Terminate SC command to terminate the SC.

When another SC instance is created, the provider must determine if a change to the periodic rate is justified. The rate shall be increased to successively higher rates as new instances are created. The service provider always controls adjustment of the update rate.

As instances on the SC terminate, the provider shall remain at the last confirmed rate.

3.7 Additional Messaging Rules

This section presents additional rules, not yet presented, that shall be followed when sending or receiving messages. Proper message handling is crucial to seamless interoperability between nodes and/or systems. Therefore, **ALL** of the rules that follow must be supported by each component in order for the component to be considered JAUS compliant.

In the discussions that follow, the terms **"sender"** and **"receiver"** are used to indicate the originator and consumer of a message, respectively. All messages are unidirectional. In other words, a message flows from the sender to the receiver and is there consumed. Any response by the receiver forces a change in the sender/receiver roles and a new message shall be created to handle the reply.

In addition, the term **"router"** is used to indicate a message handler that is neither the source nor the destination of the message. Any number of complex message handling mechanisms may be incorporated into the system as is determined by the systems engineer. JAUS places no limitation on how many "hops" a message may take between the source and destination. A router assists message trafficking.

3.7.1 Bit Precedence

Each defined bit or bit field in the message property field shall comply with the following order of evaluation. The order of evaluation matches that of the presentation.

1. Bits 8-13, Version - Must be compatible with supporting control software.
2. Bit-6, Service Connection - When enabled, the message shall be treated as a SC message. If ACK/NAK is anything other than zero, the message shall be discarded.
3. Bits 4-5, ACK/NAK - Subject to conditions set forth in **Section 3.7.3**.
4. Bits 0-3, Priority - Subject to internal message handling capabilities.
5. Bits 14-15, Reserved - Ignored (assumed zero).

3.7.2 Use of Experimental

A developer may create and test messages that are not currently defined in the standard JAUS message set. Experimental messages may be used when no JAUS message exists to meet with a requirement under development, but shall not be used to bypass standard JAUS messages. Experimental messages shall utilize the rest of JAUS message header as defined.

3.7.3 Use of ACK/NAKs

The sender uses the ACK/NAK field to control the receiver's response behavior. The following rules apply:

Concerning the original unsolicited message:

- *The sender shall set ACK/NAK=0 to disable an acknowledgement reply*
- *The sender shall set ACK/NAK=1 to enable an acknowledgement reply*
- *The sender shall always set ACK/NAK=0 on service connection messages*
- *The sender shall always set ACK/NAK=1 on node management messages*

Concerning the solicited response (original message's ACK/NAK=1):

- *The sender shall set ACK/NAK=2 to signal that the command is not supported*

  OR
- *The sender shall set ACK/NAK=2 to signal that the destination is unknown to the Node Manager*

  OR
- *The sender shall set ACK/NAK=3 to signal that the command was received OK*

The response message shall consist of the original message header, with destination and source IDs swapped. The message data shall not be returned and the data size shall be reset to zero. The returned sequence number shall match that of the original message.

If a message is received with an ACK bit set, and if the receiver of that message does not know where to find that message's destination address, then Node Manager at the receiver will reply with a NAK message using the destination address. This approach allows the node manager to NAK on behalf of the component. However, this approach does not resolve the ambiguity of whether the target component generated the NAK or the Node Manager generated it in its stead.

3.7.4 Use of the Service Connection Bit (Deprecate in V4.0)

The service connection bit is used to indicate to all message handlers that the command code is either a service connection management command or that the message is simply a transaction via a service connection.

As stated previously, all SC management messages shall be relayed from the source to the destination via SC Management. In the event that the message contains the Create Service Connection command, then SC Management shall determine the class of the command code proposed for use on the SC, prior to delivery to the destination. One of the following two scenarios shall occur:

The command code evaluation passes:

- ➢ *The message shall be forwarded to its destination.*

The command code evaluation fails (not a command or inform class):

- ➢ *SC Management shall return the Confirm SC message to the sender, with the Response Code set to five (5=invalid parameters).*
- ➢ *The Create SC message shall not be delivered to the intended destination.*

3.7.5 Transmission Retries

A three-retry rule shall be in effect for all response solicitations. After 3 successive attempts by the sender to transmit a message that requires a response and none is forthcoming, the sender shall stop sending the message. A valid response includes both an Inform Type message appropriate with the request or a NAK response. The time interval between attempts for a valid response shall be set to a value representative of the priority, authority, urgency and expected latency within the system performance requirements.

3.7.6 Messages with Optional Data Fields

Messages are allowed to have a mixture of required and optional data fields. Required data fields must precede any/all optional data fields in the message data specification. A presence vector bit is mapped to each optional data field, or data group. Presence vectors can be specified as byte, unsigned short integer, or unsigned integer. The maximum number of bits for a presence vector is 32, which corresponds to an unsigned integer type.

The use of optional data fields in the message is indicated by a flag bit field. The flag bits are contained in the presence vector field, located at the beginning of the message data section. The flag bits of the presence vector map to optional data fields or groups of data fields that follow in the message data specification. If the optional data field/group is present in the message, the flag bit corresponding to that field/group is enabled.

The mapping of presence vector bits to optional data fields is defined for each message that uses this feature. Reserved bits in the Presence Vector shall be set to zero (0).

3.7.7 Cross-Platform Requirements

Messages exchanged between subsystems may require special handling. The message may require that extra data be included or that a fundamental change in format be made to the message being sent. These issues are case specific and shall be handled as is best determined by the systems engineer. Handling is still subject to JAUS messaging rules.

The rules governing cross-platform messaging appear below:

- ➢ *Any message receiver on a Big Endian host shall convert all numeric data fields in every message received from Little Endian format. The offsets for all numeric fields that must be converted shall be determined by the message's command code. The command code must also be byte-swapped. This can easily be accomplished because its size and offset from the beginning of the message are already known.*
- ➢ *Any message sender on a Big Endian host shall convert all numeric data fields in every message to Little Endian format prior to transmission.*
- ➢ *Messages requiring encryption or that a Cyclic Redundancy Check, Checksum or other data stream validation field be included, must travel between non-JAUS (user-defined) components that provide this extra functionality.*
- ➢ *JAUS compliance shall be maintained between all JAUS components.*