SAE Aerospace
An SAE International Group

AEROSPACE STANDARD

SAE AS5710

Issued 2008-12

JAUS Core Service Set

RATIONALE

This document contains a set of service specifications that define interfaces common to most other services. The specifications may aid in simplifying the development of other service specifications that require these interfaces as a part of their own, by virtue of reuse.

INTRODUCTION

This document, the JAUS Core Service Set (AS5710), defines a message-passing interface for capabilities and services commonly found in unmanned systems. These services represent the foundation on which more complex and domain-specific JAUS Service Sets can be developed.

The primary goal of the JAUS Core Service Set is logical interoperability between communicating elements in an unmanned system. 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 Transport Specification [AS5669], is used in conjunction with this specification.

Each service in the JAUS Core Service Set can be described using the JAUS Service Interface Definition Language [JSIDL]. JSIDL creates a formal schema based on Relax NG Compact [rng] 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 Core Service Set contains both a text based and XML based representation for each service.

The AS-5710 Standard does not replace the latest JAUS Reference Architecture Version 3.3 [RA33P1, RA33P2, RA33P3], but is a direct evolution of that work. This Standard has been carefully designed to allow for the simplest possible migration of RA 3.3 implementations to a services-based framework. Even though the notion of services has come to define formal interfaces between components, the message sets in those services trace back to identical messages in the Reference Architecture.

This document uses a number of conventions to simplify the text. All names are given in Camel Case. Names start with upper case, while reference names start with a lower case.

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

SAE values your input. To provide feedback on this Technical Report, please visit
http://www.sae.org/technical/standards/AS5710

TABLE OF CONTENTS

1. SCOPE

This document defines a set of standard application layer interfaces called *JAUS Core 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 Core Services represent the infrastructure commonly found across all domains and types of unmanned systems. At present, seven services are defined in this document:

- Transport Service: Abstracts the functionality of the underlying communication transport layer
- Events Service: Establishes a publish/subscribe mechanism for automatic messaging
- Access Control: Manages preemptable exclusive control for safety critical operations
- Management: Defines component life-cycle management
- Time: Allows clients to query and set the system time for the component
- Liveness: Provides a means to maintain connection liveness between communicating components
- Discovery: Governs automatic discovery of remote entities and their capabilities

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].

1.1 Purpose

The purpose of this document is to facilitate interoperation of unmanned vehicle systems, subsystems, and payloads by standardization of the message set and associated protocol.

1.2 Compliance

The JAUS Core 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.

1.3 Document Organization

The layout of this document is as follows. Section 2 lists external references, definitions of common terms and a list of acronyms used throughout the specification. Section 3 specifies aspects of message encoding that are common across all services. Many of these encoding rules are more thoroughly discussed in [JSIDL]. Section 4 provides context for interpreting the protocol portion of a Service Definition. Section 5 specifies the JAUS Service Definition for each of the core services, with particular emphasis on the description, assumptions, message set, and protocol behavior. Section 6 describes the message encoding for each message set. 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 International, 400 Commonwealth Drive, Warrendale, PA 15096-0001, Tel: 877-606-7323 (inside USA and Canada) or 724-776-4970 (outside USA), Web address: www.sae.org.

AIR5665 Architecture Framework for Unmanned Systems

AS5669 JAUS Transport Specification

AS5684 JAUS Service Interface Definition Language

2.1.2 JAUS Technical References

RA33P1 *JAUS Reference Architecture Specification, Volume II, Part 1, Architecture Framework, Version 3.3*, June 22, 2007

RA33P2 *JAUS Reference Architecture Specification, Volume II, Part 2, Message Definition, Version 3.3*, June 22, 2007

RA33P3 JAUS Reference Architecture Specification, Volume II, Part 3, Message Set, Version 3.3, June 22, 2007 Other Publications

[bnt] Barry N. Taylor. The International System of Units (SI), National Institute of Standards and Technology Special Publication 330, 1991 Edition. [http://physics.nist.gov/Document/sp330.pdf]

[rng] Relax NG: [http://www.oasis-open.org/committees/relax-ng/spec-20011203.html] Standard lightweight XML schema language.

2.2 Definitions

2.2.1 Component

A component is a software element in a JAUS system. A component that provides one or more *services* is called a server. A component that uses one or more services is called a client.

While a component may support multiple services, two instances of the same service cannot co-exist within the same component. If two services that reside within a component inherit from the same base service, it is assumed that only one instance of that base service is present. Inheritence of service interfaces is described in [AS5684].

2.2.2 Service Definition

A Service Definition is a textual and/or XML representation of a service interface. Any Service Definition shall conform to the JAUS Service Interface Definition Language Schema defined in [AS5684]. Each Service Definition contains a service identifier, version, message set, protocol, and associated information.

2.2.3 Service Identifier

A service identifier is a globally unique string that identifies a specific *Service Definition*. Since a Service Definition mandates a message set and associated protocol, the service identifier and version number are sufficient to uniquely identify the service interface. Service Identifiers are based on a unique URI, and are specified for each of the core services defined herein.

2.2.4 JAUS Identifier

The JAUS Identifier is a 4-byte unsigned integer that corresponds to a communication end point. Messages are sent to, and received from, a JAUS Identifier. A JAUS Identifier may represent a component that hosts multiple services; the Discovery Service therefore permits the run-time determination of this one-to-many mapping between JAUS Identifiers and Service Identifiers. Refer to the Transport Service for details on the JAUS identifier.

2.2.5 Transport Layer

The JAUS Transport Layer is responsible for providing resources and mechanisms for the routing and delivery of messages over a variety of available transport domains. While a Service Definition is independent of the underlying transport, it is designed for integration with [AS5669]. Other transport layers are possible, provided they meet the requirements described in the Transport Service.

2.2.6 Serialization

Serialization (sometimes called encoding or marshalling) is the process by which a message defined by a Service Definition is converted into a well-defined sequence of bytes that will be visible in a given Transport Domain, "on the wire". The reverse process is known as deserialization. Interoperability requires that both sender and receiver follow the same syntax for serialization and deserialization or data corruption may occur.

2.2.7 Message Code

The Message Code is an identifier globally unique to each message. This code, along with the associated service version, allows receiving entities to know the type, intent, and structure of an incoming message. The Message Code is serialized in the same position within each message.

2.2.8 Presence Vector

JAUS provides for variable length messages. These messages either have repeating data or have a mixture of required and optional data fields. The Presence Vector is used to indicate which of the optional data fields are included. In the Service Message tables, optional fields are mapped in order to each bit in the presence vector, starting with bit 0.

2.2.9 Little Endian

A byte-encoding scheme where the least significant byte of a multi-byte data item comes first in the byte sequence.

2.3 List of Acronyms

ANSI American National Standards Institute

Id Identifier

API Application Programming Interface

ASCII American Standard Code for Information Interchange

BLOB Binary Large Object

DOM	Document Object Model
IEEE	Institute of Electrical and Electronics Engineers
JAUS	Joint Architecture for Unmanned Systems
JSD	JAUS Service Definition
JSIDL	JAUS Service (Interface) Definition Language
NIST	National Institute of Standards and Technology
RA	(JAUS) Reference Architecture
SMC	State Machine Compiler
UML	Unified Modeling Language
URL	Uniform Resource Locator
URN	Uniform Resource Name
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier
XML	Extensible Markup Language

3. MESSAGE SERIALIZATION

Any implementation of a JAUS Service Definition must meet the requirements established in [AS5684] for on-the-wire message serialization:

- All multi-byte data shall be encoded in little endian where the low byte comes first.
- All messages shall be encoded with a 2-byte header that contains the Message Code. This header must preceed the message body.

In addition to the above requirements for all messages, JAUS Service Definitions may make use of the following additional serialization rules to improve messaging efficiency.

3.1 Scaled Integers

The use of Scaled Integers reduces the impact of transmitting floating point and double precision values that are valid only in a finite range. Scaled integers are integer types that represent real values. The real values are first bounded by upper and lower limits and are then mapped onto the integer range of the integer field type used for the scaled integer. Each scaled integer is associated with a *bias* and *scale factor*. For additional information on computing the bias and scale factor for scaled integers, consult [AS5684].

3.2 Presence Vectors

Messages are allowed to have a mixture of required and optional data fields. 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 is indicated by the “Optional” field for each element in a message definition. The bits of the presence vector map to optional data fields or groups of data fields that follow in the message data specification. This mapping is implicit, such that the least significant bit of the presence vector maps to the first optional field, with each additional optional field mapping to the next available least significant bit. If the optional data field/group is present in the message, the bit corresponding to that field/group in the presence vector is enabled. Otherwise, the corresponding bit shall be set to zero (0).

Unused bits in the presence vector shall be set to zero (0).

3.3 Variable Length Arrays and Lists

Many messages make use of variable length fields, such as arrays and lists. Such fields must be serialized with the count field preceeding the corresponding data. The data size of the count field is specified in each message definition.

4. UNDERSTANDING PROTOCOL DESCRIPTIONS

The protocol behavior for each component is described in the form of concurrent and hierarchical finite state machines[1]. These finite state machines describe the state of the component and its behavior in response to an input message (external stimuli) or internal event (internal stimuli). The state machines are described diagrammatically as shown in Figure 1 below, and consist of two main elements:

States, described as rectangles with rounded corners, represent the situations during the life of a component. At any given time, a component is said to be in a state that satisfies some condition, performs some activity, or waits for some stimulus.

Transitions, described by labeled and directed arrows, represent state changes from a single source state to a single destination state.

[1] For a more detailed description of protocol behavior specification, refer [AS5684].

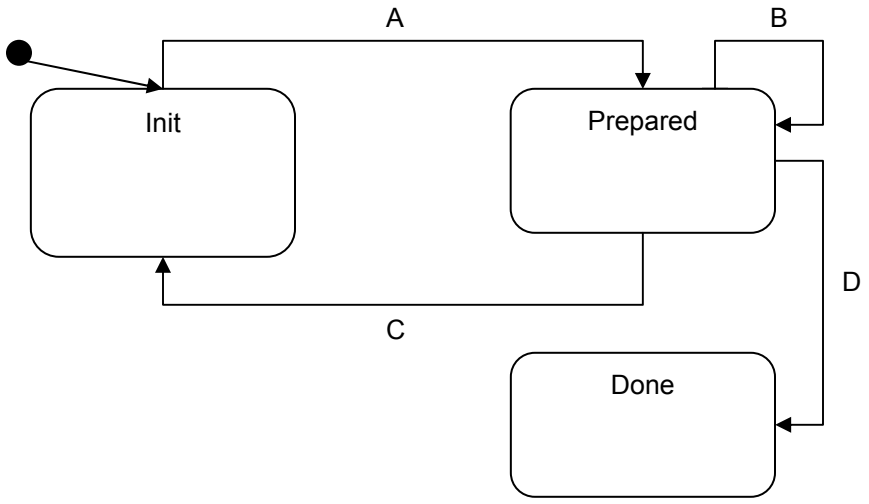


FIGURE 1 - PROTOCOL STATE MACHINE

4.1 States

Each state is given a name by which it can be uniquely identified. A special state called the pseudo-start state is described as a small filled circle. This state contains an arrow that points to the initial state, the state that the component is in when it first comes to life. Pseudo states are also used to point to a nested state that is automatically entered when their parent states are entered.

Each state may correspond to some ongoing activity that is described using a set of actions called entry and exit actions. These actions are performed when a state is entered and when it is exited, and in the order in which they are listed. Their execution is also conditional on the type of the transition that causes the state to be entered or exited. This is explained in the next sub-section on transitions. The entry action usually starts the activity on entry of the state and the exit action often terminates that activity on exit from the state. But in general, the entry and exit actions may not be related and can be used to perform protocol related functions like sending a message, resetting a timeout or performing operations on stored data. If the entry and exist actions perform operations that are local to the component, that is, they do not send out messages, then the actions are represented in italics. Entry and exit are considered to be atomic and non-interruptible. This kind of activity may also be specified in a fine grained manner with the use of nested states, each of which may have their respective entry and exit actions that are executed in the order in which the states are nested.

4.2 Transitions

Transitions cause a state machine to change its state from the source state of the transition to its destination state, and to perform a set of actions called transition actions before transitioning into the destination state. Like entry and exit actions of states, transition actions are also atomic and non-interruptible and represented in italics if they only perform local operations. The signature of a transition consists of an input message or internal event that triggers the transition, an optional guard condition that must evaluate to true for the transition to be executable, and an ordered sequence of zero or more transition actions. These signatures are listed in tabular form as shown in Table 1-3 below.

If the transition is triggered by an internal event, then the trigger for that transition is represented in italics. The transition with trigger Failure is an example of a transition that is triggered by an internal event. Since the evaluation of a guard condition is an operation that is always local to the component, all guard conditions are also represented in italics. Transitions are executed when they are found to be executable. Once a state is entered, its transitions are evaluated for executability in the order in which they are defined. If two or more transitions are executable at the same time, the transition that is defined highest in a top-down order among the ones that are executable is the transition that is chosen to be executed.

TABLE 1 - TRANSITION TABLE

Label	Trigger	Condition	Actions
A	RequestToPrepare		sendPrepared
B	SetData	*dataWritable*	*logData*, sendSuccess(LOGGED)
	SetData	NOT *dataWritable*	sendAbort(NOWRITE)
C	RollBack		*clearLog*, sendSuccess(ROLLEDBACK)
	Commit	NOT *isCommit*	sendAbort(NOCOMMIT)
	Failure		sendAbort(FAILURE)
D	Commit	*isCommit*	sendSuccess(COMMITTED)

TABLE 2 - CONDITION TABLE

Condition	Interpretation
dataWritable	True if data can be written
isCommit	True if commit was successful

TABLE 3 - ACTION TABLE

Action	Interpretation
sendPrepared	Send the prepared message
logData	Write the data to a log
sendAbort(code)	Send the abort message with indicated code
clearLog	Empty the log
sendSuccess(code)	Send the success message with indicated code

When a state machine is nested, a transition to the enclosing state (or super state) also represents a transition to a nested state only if a nested pseudo start state is used to point to the nested state. In Figure2, the transition labeled A has Prepared.Standby as its destination state due to the presence of the nested pseudo start state in the super state called Prepared. Alternately, a transition may be directed specifically to a nested state as in transition B in Figure2. A nested state may also be a source state to a transition (transition G in Figure 2, for example). A transition that has the enclosing state as the source state is equivalent to a transition that has any of the nested states as its source state.

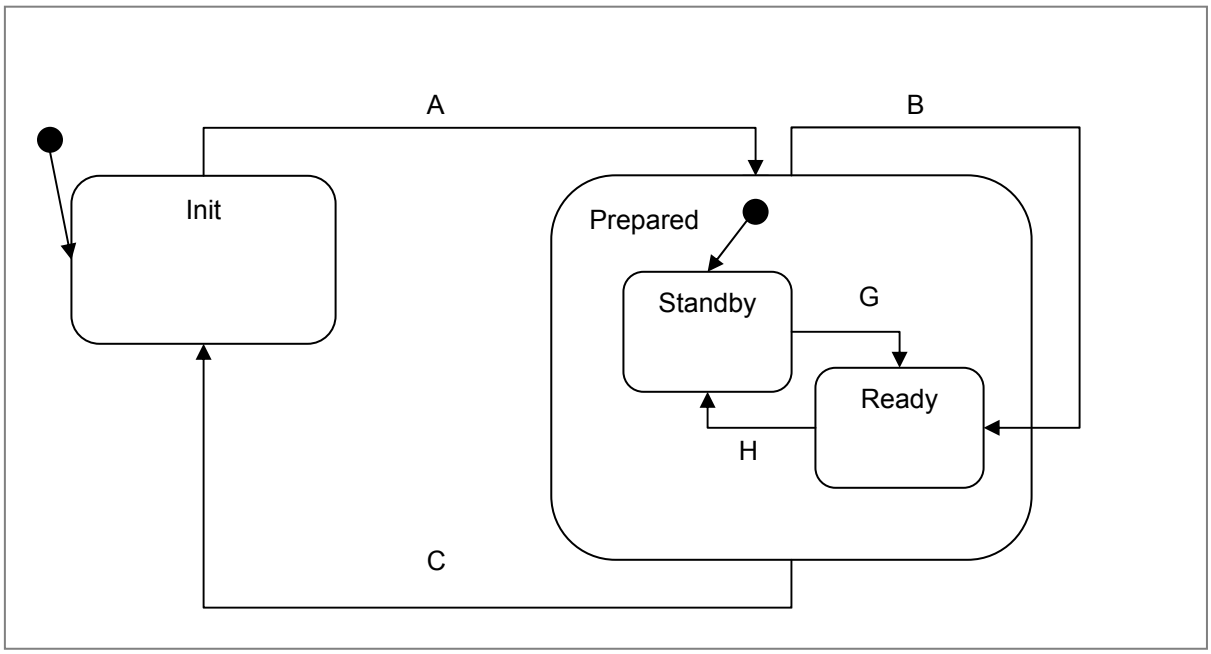


FIGURE 2 - NESTED STATES

There are four types of transitions: simple transitions, loopback transitions, push transitions and pop transitions. As the name suggests, a simple transition changes the state of the finite state machine from the transition's source state to its destination state. A loopback transition causes the finite state machine to re-enter the source state of the transition. That is, the destination state of a loopback transition is its source state. In Figure 2 above, the transition labeled A is a Simple transition from Init to Prepared, and the transition labeled B acts as a simple transition from Prepared.Standby to Prepared.Ready, but also represents a loopback transition from and to Prepared.Ready. A push transition causes the finite state machine to change its state to the destination state of the push transition in a manner that allows a subsequent pop transition to return the state machine back to the source state of the preceding push transition. This operation is similar to the last in first out (LIFO) operation of a stack. Push and Pop transitions are useful for describing subroutine type operations where control has to be returned to the caller of a subroutine that has completed its execution. Push and Pop transitions are described using unfilled circles marked either Push or Pop, as shown in Figure 3 below. In Figure 3, the transitions labeled A and B cause the finite state machine to "push" the Blocked state into a stack, while transition labeled C causes the state machine to pop the blocked state from the stack. For example, the sequence of transitions {A, B, C} results in the blocked state being stacked twice and unstacked once, leaving the state machine in a blocked state that is stacked once. The sequence {A, B, C, C} will return the state machine back to the running state.

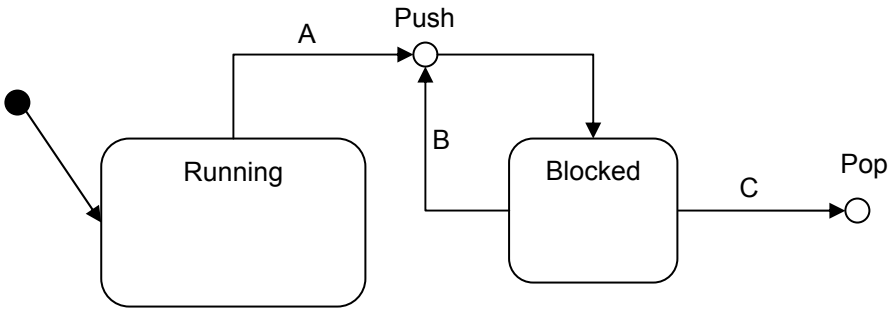


FIGURE 3 - PUSH AND POP TRANSITIONS

The execution of entry and exit actions of the source and destination states of a transition is conditional upon the type of the transition. The table below shows when these actions can and cannot be executed.

TABLE 4 - TYPES OF TRANSITIONS FOR WHICH ENTRY/EXIT ACTIONS ARE EXECUTED

Transition Type	Execute "from" state's Exit Actions?	Execute "to" state's Entry Actions?
Simple Transition	Yes	Yes
Loop back Transition	No	No
Push Transition	No	Yes
Pop Transition	Yes	No

Concurrent state machines are described graphically as separate state machines that are separated by a vertical or horizontal line as shown in Figure 4.

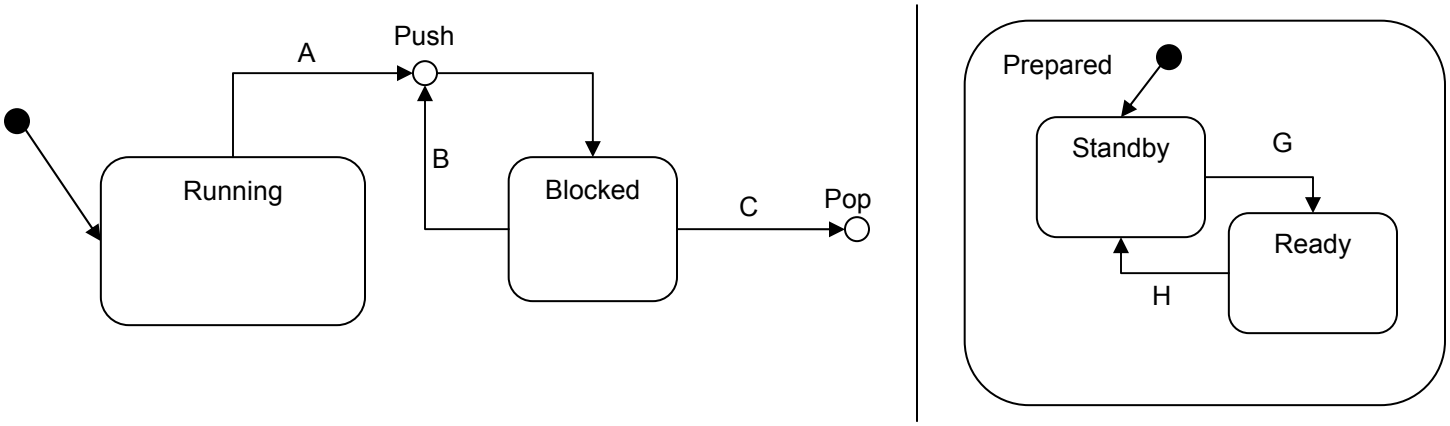


FIGURE 4 - CONCURRENT FINITE STATE MACHINES

A finite state machine can be extended by hierarchically adding to it, or its existing features using the inheritance relationship. This relationship allows only single inheritance and can be used to add new messages, nested states, new transitions and entire concurrenct state machines to the service that is being extended.The relationship also allows for the overriding of transitions defined in the service that is being extended with the constraint that the overriding transitions do not alter the behavior of the transitions that are being overridden. Refer to JSIDL for a more detailed description of the inheritance relationship [AS5684]

5. SERVICE DEFINITIONS

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

5.1 Transport

name=Transport
version=1.0
id=urn:jaus:jss:core:Transport

Description

The transport service acts as an interface to the JAUS transport layer. It models an abstract bi-directional communication channel (input queue and output queue) whose primary function is to provide the capability of sending messages to a single destination endpoint or broadcasting messages to all endpoints in the system, and to receive a message from any source endpoint. It also provides the capability to prioritize the delivery of sent messages.

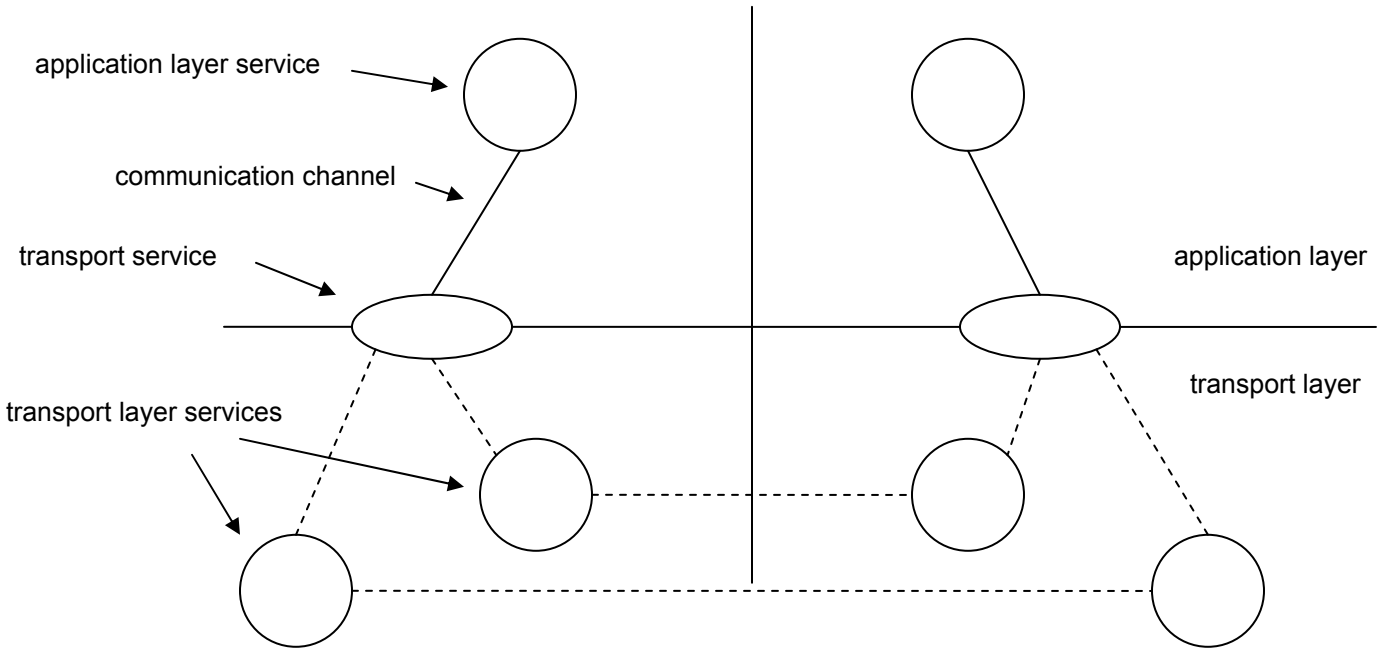


FIGURE 5 - TRANSPORT SERVICE ACTING AS AN INTERFACE TO THE TRANSPORT LAYER

This service establishes a communication endpoint whose address is defined by a triple {SubsystemID, NodeID, ComponentID} as specified by the Send and Receive internal events. Other services that need to utilize the communication channel provided by the transport service must inherit from the transport service. The SubsystemID uniquely identifies a subsystem, which is an independent and distinct unit within a system and is made up of a logical grouping of nodes. Similarly, the Node ID uniquely identifies a node, which is an independent and distinct unit within a subsystem and is made up of a logical grouping of components. And finally, the ComponentID uniquely identifies a component, which is an independent and distinct unit within a node and is made up of a logical grouping of services. Each component may contain multiple services, but the services within a single component must have disjoint message sets. Figure 6 depicts a graphical representation of this topology.

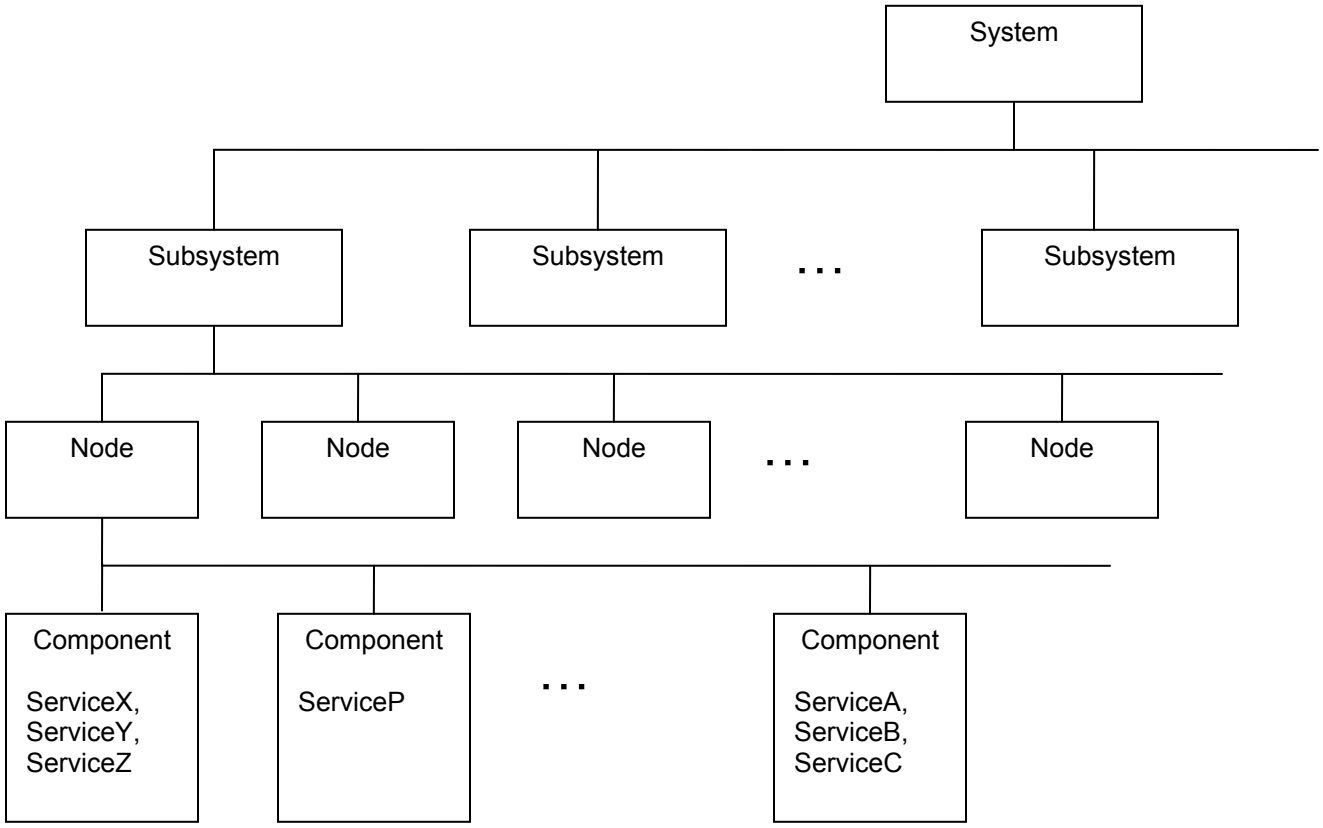


FIGURE 6 - THE SYSTEM TOPOLOGY

Assumptions

Messages may be delayed, lost or reordered.

Vocabulary

The table below lists the vocabulary of the Transport Service. This service has no input and output messages. It contains only internal events that are triggered when a message is sent or received.

TABLE 5 - TRANSPORT SERVICE INTERNAL EVENTS SET

Name	Interpretation
Receive	The Receive event is used by derived services to hand over the data that was sent by another entity to an endpoint established by this service.
Send	The Send event is used by a derived service to hand over the payload data that it needs to send to a specified destination endpoint via the transport layer. Upon receipt, this service prepares the message for delivery (marshals the message) as specified by the transport layer specification and attempts to deliver the encapsulated payload data to the destination endpoint.
BroadcastLocal	The Broadcast Local event is the same as the Send event except that this service sends the payload to all endpoints within the subsystem.
BroadcastGlobal	The Broadcast Global event is the same as the Send event except that this service sends the payload to all endpoints on all subsystems.

Encoding

TABLE 6 - RECEIVE EVENT ENCODING

body
└ **record** name=ReceiveRec

record name=ReceiveRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** SrcSubsystemID	unsigned short integer	one	false	Source Subsystem ID
2	**<fixed_field>** SrcNodeID	unsigned byte	one	false	Source Node ID
3	**<fixed_field>** SrcComponentID	unsigned byte	one	false	Source Component ID
4	**<variable_length_field>** Message Payload	count_field = unsigned long integer	format= JAUS_MESSAGE	false	Enclosed message payload

TABLE 7 - SEND EVENT ENCODING

body
└ **record** name=SendRec

record name=SendRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<presence_vector>**	unsigned_byte	one	false	
2	**<fixed_field>** DestSubsystemID	unsigned short integer	one	false	Destination Subsystem ID, where a value of 0xFFFF represents all subsystems
3	**<fixed_field>** DestNodeID	unsigned byte	one	false	Destination Node ID where a value of 0xFF represents all nodes.
4	**<fixed_field>** DestComponentID	unsigned byte	one	false	Destination Component ID where a value of 0xFF represents all components.
5	**<fixed_field>** SrcSubsystemID	unsigned short integer	one	true	Source Subsystem ID
6	**<fixed_field>** SrcNodeID	unsigned byte	one	true	Source Node ID
7	**<fixed_field>** SrcComponentID	unsigned byte	one	true	Source Component ID
8	**<fixed_field>** Priority	unsigned byte	one	true	Default priority is NORMAL value_enum 0 = LOW 1 = NORMAL 2 = HIGH 3 = SAFETY
9	**<variable_length_field>** Message Payload	count_field = unsigned long integer	format= JAUS_MESSAGE	false	Enclosed message payload

TABLE 8 - BROADCAST LOCAL EVENT ENCODING

body └ **record** name=SendRec
Encoding for the *BroadcastLocal* event shall be the same as the *Send* event.

TABLE 9 - BROADCAST GLOBAL EVENT ENCODING

body └ **record** name=SendRec
Encoding for the *BroadcastGlobal* event shall be the same as the *Send* event.

Protocol Behavior

The protocol behavior of the transport service consists of two concurrent state machines. One is used for receiving messages and the other is used for sending messages out. State machines of derived services must inherit from the Receive state machine and override the Receive transition in order to specify the receipt of messages belonging to their vocabularies (see AccessControl service). Similarly, derived services may overload the transitions of the Send state machine if more specific behavior needs to be specified for those transitions.

Receive state machine Send state machine

FIGURE 7 - TRANSPORT SERVICE PROTOCOL BEHAVIOR

TABLE 10 - TRANSPORT SERVICE STATE TRANSITION TABLE

Label	Trigger	Conditions	Actions
A	*Receive*	*false*	
B	*Send*		*Enqueue*
	BroadcastLocal		*BroadcastLocalEnqueue*
	BroadcastGlobal		*BroadcastGlobalEnqueue*

TABLE 11 - TRANSPORT SERVICE CONDITIONS

Condition	Interpretation
false	The strongest pre-condition that can only be weakened in derived services for transitions containing this guard to be executed.

TABLE 12 - TRANSPORT SERVICE TRANSITION ACTIONS

Action	Interpretation
Enqueue	Convert the destination address into an unsigned integer such that the ComponentID maps to the least significant byte, NodeID to the next least significant byte and SubsystemID maps onto the remaining two bytes of the integer. Package the message as specified by the transport layer specification and send it to its destination as per the specified priority.
BroadcastLocalEnqueue	Package the message as specified by the transport layer specification and send it to all endpoints in the local subsystem.
BroadcastGloballEnqueue	Package the message as specified by the transport layer specification and send it to all endpoints on all subsystems.

5.2 Events

name=Events
version=1.0
id=urn:jaus:jss:core:Events

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

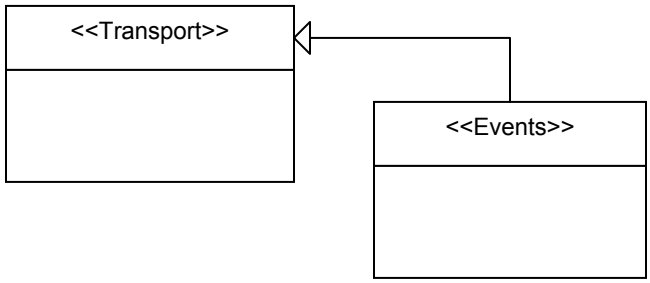


FIGURE 8 - EVENTS SERVICE

Description

This service allows clients to subscribe to changes in the information provided by report messages supported by services that derive from this service. When a client successfully creates an event, an InternalEvent occurs when the event creation criteria are met, and as a result the matching Event is sent to the subscribing client.

Assumptions

Messages may be delayed, lost or reordered.

Vocabulary

The table below lists the vocabulary of the Events Service.

TABLE 13 - EVENTS SERVICE VOCABULARY

Message Id (hex)	Name	Command
Input Set		
01F0	CreateEvent	True
01F1	UpdateEvent	True
01F2	CancelEvent	True
21F0	Query Events	False
Output Set		
01F3	ConfirmEventRequest	False
01F4	RejectEventRequest	False
41F0	ReportEvents	False
41F1	Event	False

TABLE 14 - EVENTS SERVICE INTERNAL EVENTS SET

Name	Interpretation
EventOccured	Received when an event occurs.
EventError	Received when an event error occurs.

Protocol Behavior

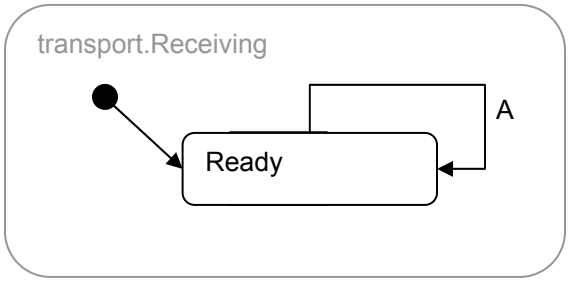


FIGURE 9 - EVENTS SERVICE PROTOCOL BEHAVIOR

TABLE 15 - EVENTS SERVICE STATE TRANSITION TABLE

Label	Trigger	Conditions	Actions
A	Query Events		sendReportEvents
	Create Event	*isSupported* AND NOT *eventExists*	*createEvent,* sendConfirmEventRequest
	Create Event	*isSupported* AND *eventExists*	*updateEvent,* sendConfirmEventRequest
	Create Event	NOT *isSupported*	sendRejectEventRequest
	Update Event	*isSupported* AND *eventExists*	*updateEvent,* sendConfirmEventRequest
	Update Event	NOT *isSupported* OR NOT *eventExists*	sendRejectEventRequest
	Cancel Event	*eventExists*	*cancelEvent,* sendConfirmEventRequest
	Cancel Event	NOT *eventExists*	sendRejectEventRequest
	Event Occurred	*eventExists*	sendEvent
	Event Error	*eventExists*	sendRejectEventRequest

TABLE 16 - EVENTS SERVICE CONDITIONS

Condition	Interpretation
isSupported	True if parameters are supported.
eventExists	True if the specified event exists.

TABLE 17 - EVENTS SERVICE TRANSITION ACTIONS

Action	Interpretation
sendReportEvents	Send Report Events message to the component that sent the query
createEvent	Create the event
sendConfirmEventRequest	Send Confirm Event Request message
sendRejectEventRequest	Send Reject Event Request message
updateEvent	Update an event
cancelEvent	Cancel an event
sendEvent	Send an Event notification

5.3 AccessControl

name=AccessControl
version=1.0
id= urn:jaus:jss:core:AccessControl

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

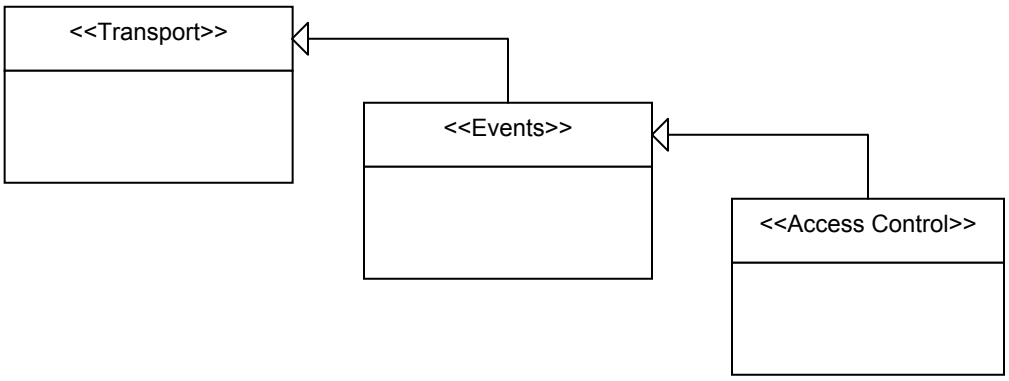


FIGURE 10 - ACCESS CONTROL SERVICE

Description

The Access Control service offers a basic interface for acquiring preemptable exclusive control to one or more related services that utilize this function. Once the exclusive control is established, the related services shall only execute commands originating from the controlling component. The authority code parameter of this service is used for preemption and is to be set equal to that of its controlling client. This service always grants control to the highest authority client that is requesting exclusive control. Commands from all other clients are ignored unless from a client with higher authority.

This service maintains two values, a default value and a current value of a field called authority code. The default value is the value that the service is pre-configured with. Access is provided to clients based on the value of their authority code in comparison to the current value of this service.

Assumptions

Messages may be delayed, lost or reordered.

Vocabulary

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

TABLE 18 - ACCESS CONTROL SERVICE VOCABULARY

Message Id (hex)	Name	Command
Input Set		
000D	RequestControl	True
000E	ReleaseControl	True
200D	QueryControl	False
2001	QueryAuthority	False
0001	SetAuthority	True
2003	QueryTimeout	False
Output Set		
400D	ReportControl	False
0010	RejectControl	False
000F	ConfirmControl	False
4001	ReportAuthority	False
4003	ReportTimeout	False

TABLE 19 - ACCESS CONTROL SERVICE INTERNAL EVENTS SET

Name	Interpretation
Timeout	Occurs when access is not re-acquired periodically

Protocol Behavior

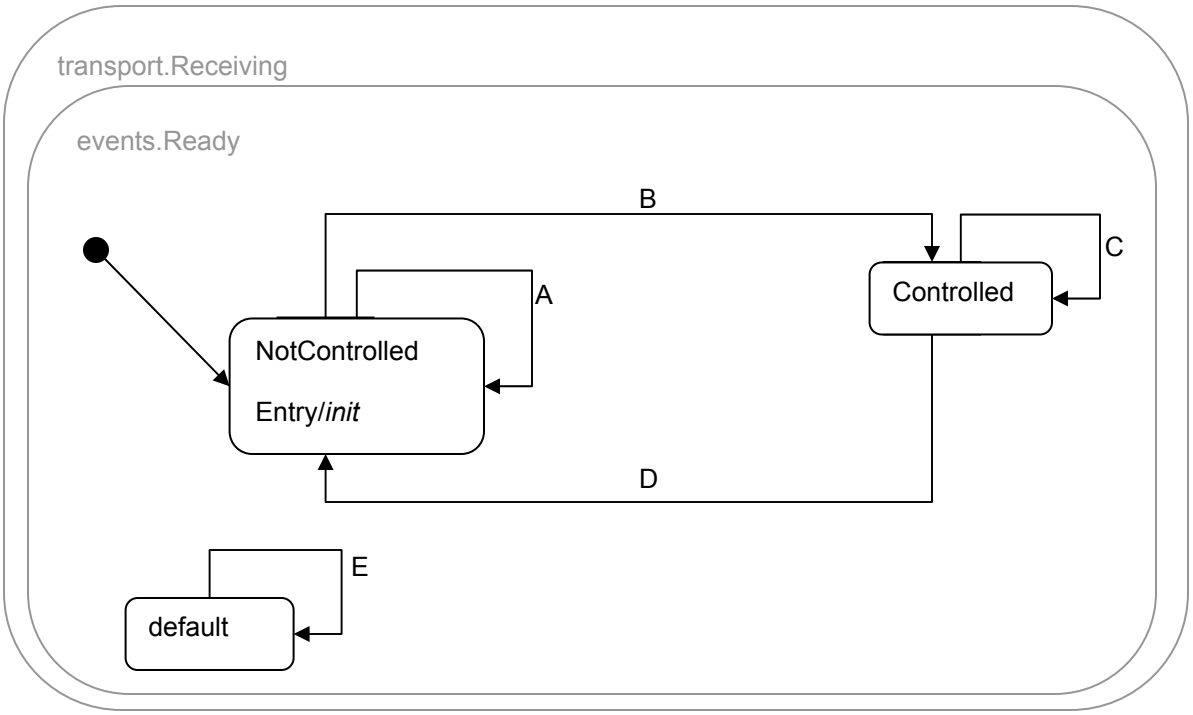


FIGURE 11 - ACCESS CONTROL SERVICE PROTOCOL BEHAVIOR

TABLE 20 - ACCCESS CONTROL SERVICE STATE TRANSITIONS

Label	Trigger	Conditions	Actions
A	RequestControl	NOT *isControlAvailable*	sendConfirmControl(NOT_AVAILABLE)
	RequestControl	*isDefaultAuthorityGreater*	sendConfirmControl(INSUFFICIENT_AUTHORITY)
	ReleaseControl		sendRejectControl (CONTROL_RELEASED)
	QueryControl		sendReportControl()
B	RequestControl	NOT *isDefaultAuthorityGreater* AND *isControlAvailable*	*storeAddress*, *setAuthority*, *resetTimer*, sendConfirmControl(CONTROL_ACCEPTED)
C	RequestControl	NOT *isControlAvailable*	sendConfirmControl(NOT_AVAILABLE)
	RequestControl	*isCurrentAuthorityLess* AND NOT *isControllingClient*	sendRejectControlToController(CONTROL_RELEASED) , *storeAddress*, *setAuthority*, *resetTimer*, sendConfirmControl(CONTROL_ACCEPTED)
	RequestControl	NOT *isCurrentAuthorityLess* AND NOT *isControllingClient*	sendConfirmControl(INSUFFICIENT_AUTHORITY)
	RequestControl	NOT *isDefaultAuthorityGreater* AND *isControllingClient*	*resetTimer*, *setAuthority*, sendConfirmControl(CONTROL_ACCEPTED)
	QueryControl		sendReportControl()
	SetAuthority	*isControllingClient* AND *isAuthorityValid* AND *isControlAvailable*	*setAuthority*
	ReleaseControl	NOT *isControlAvailable*	sendRejectControl(NOT_AVAILABLE)
D	ReleaseControl	*isControllingClient* AND *isControlAvailable*	sendRejectControl(CONTROL_RELEASED)
	RequestControl	*isDefaultAuthorityGreater* AND *isControllingClient*	sendRejectControlToController(CONTROL_RELEASED)
	Timeout	*isControlAvailable*	sendRejectControlToController(CONTROL_RELEASED)
E	QueryAuthority		sendReportAuthority
	QueryTimeout		sendReportTimeout

TABLE 21 - ACCESS CONTROL SERVICE CONDITIONS

Condition	Interpretation
isControlAvailable	True if this service or services related to this service are not engaged in other operations that would prevent this service from performing its actions.
isDefaultAuthorityGreater	True if the default authority code of this service is greater than the authority code of the client service that triggered the corresponding transition
isCurrentAuthorityLess	True if the current authority value of this service is less than the authority code of the client service that triggered the corresponding transition
isAuthorityValid	True if the value of the authority code received from the client is less than or equal to the current authority value of this service, but greater than or equal to the receiving component's default authority
isControllingClient	True if the message that triggered the transition is received from the client that is in control of this service

TABLE 22 - ACCESS CONTROL SERVICE TRANSITION ACTIONS

Action	Interpretation
Entry*/init*	Set the service's current authority value to the default authority value
storeID	Store the JAUS ID of the client that sent the message that caused this action to be executed
setAuthority	Set the current authority value of this service to the specified authority
resetTimer	Reset the timer
sendConfirmControl(int ResponseCode)	Send a confirm control message with the specified response code to requesting client
sendRejectControl(int ResponseCode)	Send a Reject Control message with the specified response code to the client requesting release
sendRejectControlToController(int ResponseCode)	Send a Reject Control message with the specified response code to the client that currently has control
sendReportControl(int value)	Send a Report Control message with the specified control value
sendReportAuthority	Send a Report Authority message to querying client reporting the current authority value of this service
sendReportTimeout	Send a Report Timeout message specifying the timeout period of this service

5.4 Management

name=Management
version=1.0
id= urn:jaus:jss:core:Management

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

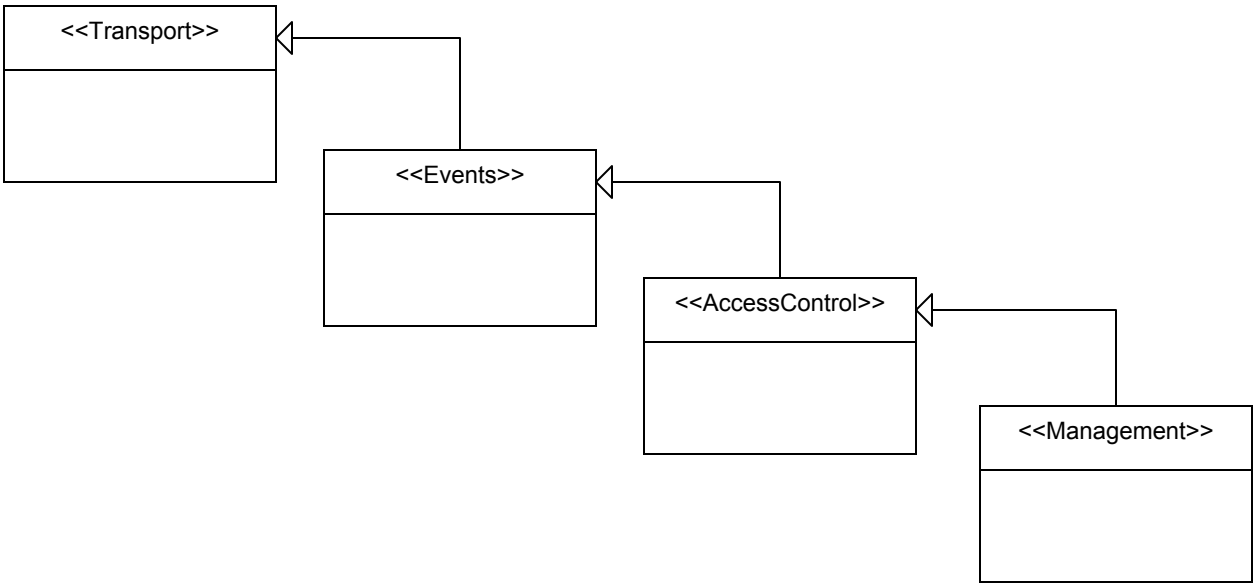


FIGURE 12 - MANAGEMENT SERVICE

Description

The Management Service provides a state machine for component life-cycle management to help clients understand how the component will react to commands and queries.

Note that the transitions for "RequestControl" and "ReleaseControl" from the Access Control service have been overridden to disallow these messages in emergency, shutdown, failure and Init states.

Assumptions

Messages may be delayed, lost or reordered.

Vocabulary

The table below lists the vocabulary of the Management Service.

TABLE 23 - MANAGEMENT SERVICE VOCABULARY

Message Id (hex)	Name	Command
Input Set		
0002	Shutdown	True
0003	Standby	True
0004	Resume	True
0005	Reset	True
0006	SetEmergency	True
0007	ClearEmergency	True
2002	QueryStatus	False
Output Set		
4002	ReportStatus	False

TABLE 24 - MANAGEMENT SERVICE INTERNAL EVENTS SET

Name	Interpretation
Initialized	An internally generated event that marks the completion of the initialization cycle.
Failure	An internally generated event that marks forces the component to a failure state.

Protocol Behavior

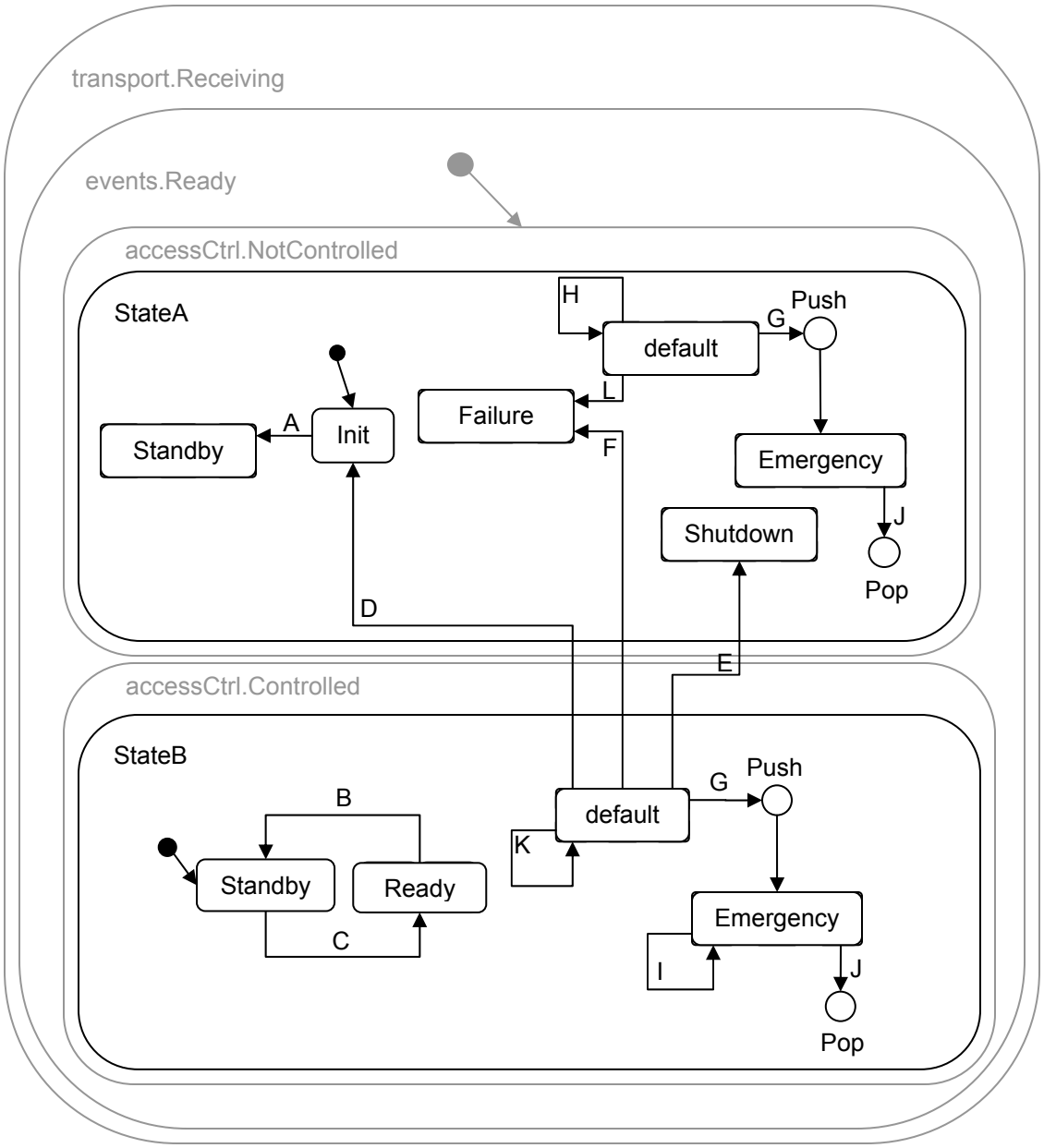


FIGURE 13 - MANAGEMENT SERVICE PROTOCOL BEHAVIOR

TABLE 25 - MANAGEMENT SERVICE TRANSITION TABLE

Label	Trigger	Conditions	Actions
A	*Initialized*		
B	Standby	*isControllingClient*	
C	Resume	*isControllingClient*	
D	Reset	*isControllingClient*	sendRejectControl(CONTROL_RELEASED)
E	Shutdown	*isControllingClient*	sendRejectControl(CONTROL_RELEASED), *emptyStateStack*
F	*Failure*		sendRejectControl(CONTROL_RELEASED), *emptyStateStack*
G	SetEmergency		*storeID*
H	QueryStatus		sendReportStatus
	RequestControl	NOT *isStateStandby*	sendConfirmControl(NOT_AVAILABLE)
I	RequestControl		sendConfirmControl(NOT_AVAILABLE)
	ReleaseControl		sendRejectControl(NOT_AVAILABLE)
	Timeout		*resetTimer*
	Reset		
J	ClearEmergency	*isIDStored*	*deleteID*
K	QueryStatus		sendReportStatus
L	*Failure*		*emptyStateStack*

TABLE 26 - MANAGEMENT SERVICE CONDITIONS

Condition	Interpretation
isControllingClient	True if the sender of the message that triggered the transition is in control of this service
isIDStored	True if the sender of the message that triggered the transition is in the list of clients that have set an emergency condition.
isStateStandby	True if the component that received the message that triggered the transition is in the NotControlled.StateA.Standby state.

TABLE 27 - ACCESS CONTROL SERVICE TRANSITION ACTIONS

Action	Interpretation
storeID	Add the sender of the message that caused this action to the list of clients that have an emergency condition.
deleteID	Remove the sender of the message that caused this action from the list of clients that have an emergency condition.
sendRejectControl	Send a Reject Control message to client that is currently controlling this service.
sendReportStatus	Send a Report Status message to the requesting client.
Init	Re-initialize the component's state machines
emptyStateStack	Empty all stacked emergency states and delete all associated stored ids

5.5 Time

name=Time
version=1.0
id= urn:jaus:jss:core:Time

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

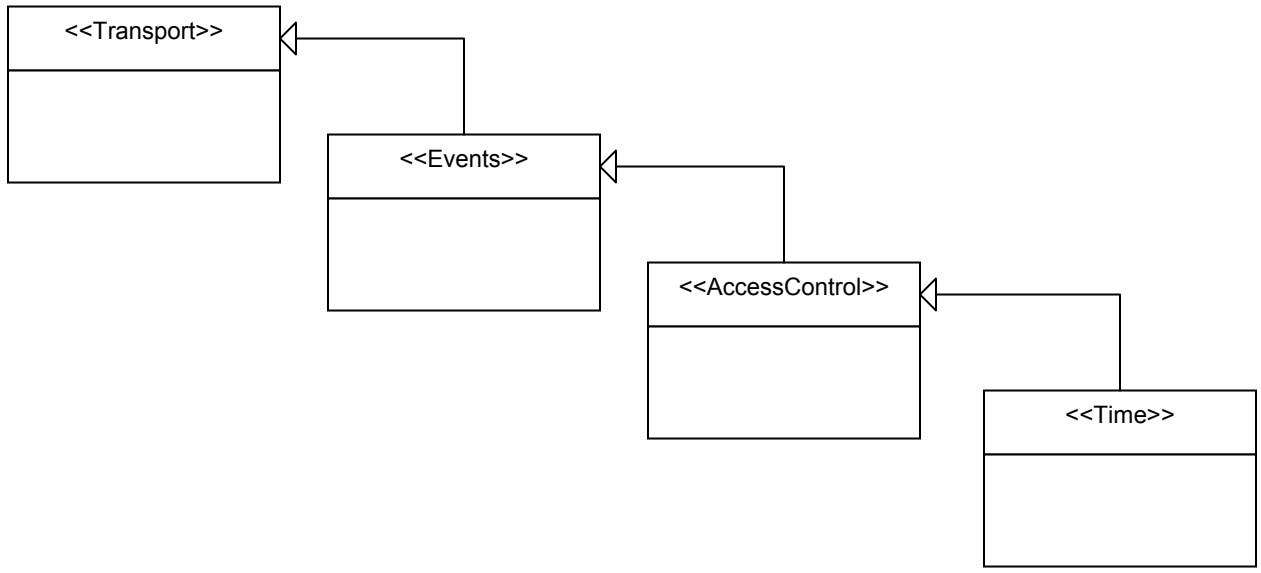


FIGURE 14 - TIME SERVICE

Description

The Time Service allows clients to query and set the system time for the component. Note that exclusive control is required to set the time, but is not required to query it.

Assumptions

Messages may be delayed, lost or reordered.

Vocabulary

The table below lists the vocabulary of the Time Service.

TABLE 28 - TIME SERVICE VOCABULARY

Message Id (hex)	Name	Command
Input Set		
0011	Set Time	True
2011	QueryTime	False
Output Set		
4011	ReportTime	False

Protocol Behavior

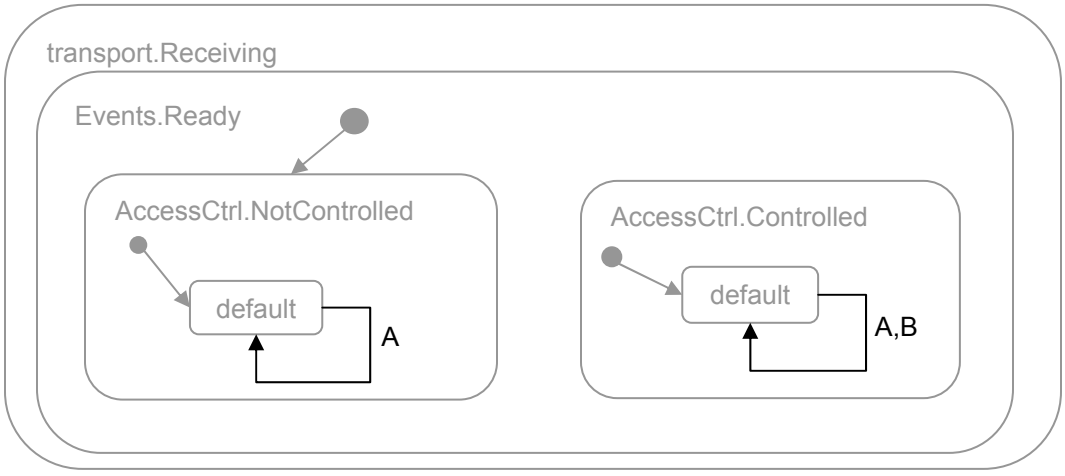


FIGURE 15 - TIME SERVICE PROTOCOL BEHAVIOR

Label	Trigger	Conditions	Actions
A	Query Time		sendReportTime
B	SetTime	*isControllingClient*	*setTime*

FIGURE 16 - TIME SERVICE TRANSITION TABLE

TABLE 29 - TIME SERVICE CONDITIONS TABLE

Condition	Interpretation
isControllingClient	True if the message that triggered the transition is received from the client that is in control of this service.

TABLE 30 - TIME SERVICE TRANSITION ACTIONS

Action	Interpretation
setTime	Set the time to the specified time.
sendReportTime	Send a Report Time message.

5.6 Liveness

name=Liveness
version=1.0
id= urn:jaus:jss:core:Liveness

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

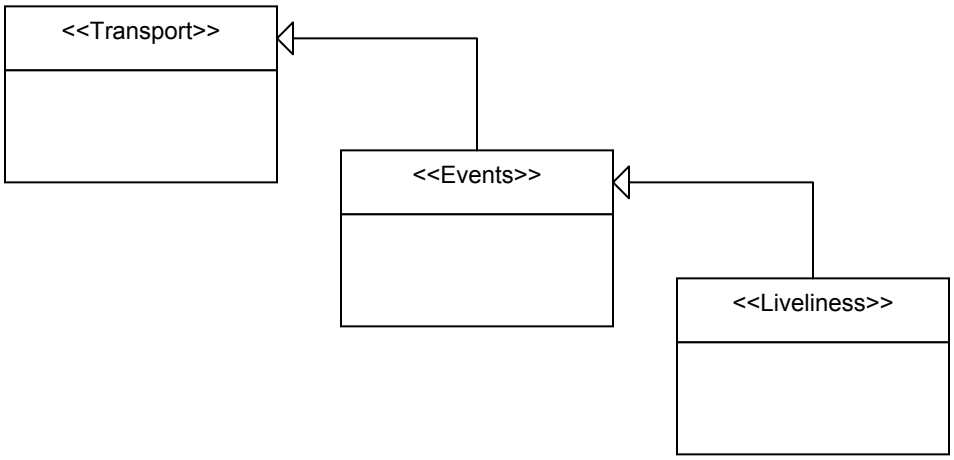


FIGURE 17 - LIVENESS SERVICE

Description

This service provides a means to maintain connection liveness between communicating components.

Assumptions

Messages may be lost of reordered.

Vocabulary

TABLE 31 - LIVENESS SERVICE VOCABULARY

Message Id (hex)	Name	Command
Input Set		
2202	QueryHeartbeatPulse	False
Output Set		
4202	ReportHeartBeatPulse	False

Protocol Behavior

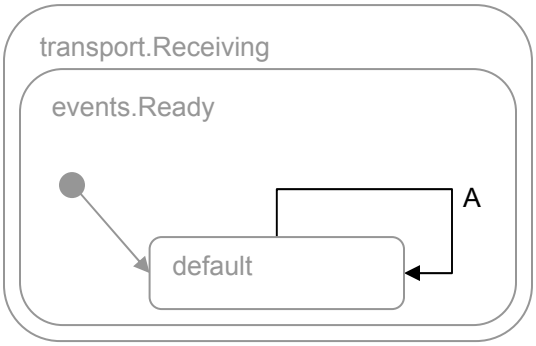


FIGURE 18 - LIVENESS SERVICE PROTOCOL BEHAVIOR

TABLE 32 - LIVENESS SERVICE STATE TRANSITIONS

Label	Trigger	Conditions	Actions
A	QueryHeartBeatPulse		sendReportHeartBeatPulse

TABLE 33 - LIVENESS SERVICE TRANSITION ACTIONS

Action	Interpretation
sendReportHeartbeatPulse	Send a Report Heartbeat Pulse message to the component that sent the query

5.7 Discovery

name=Discovery
version=1.0
id= urn:jaus:jss:core:Discovery

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

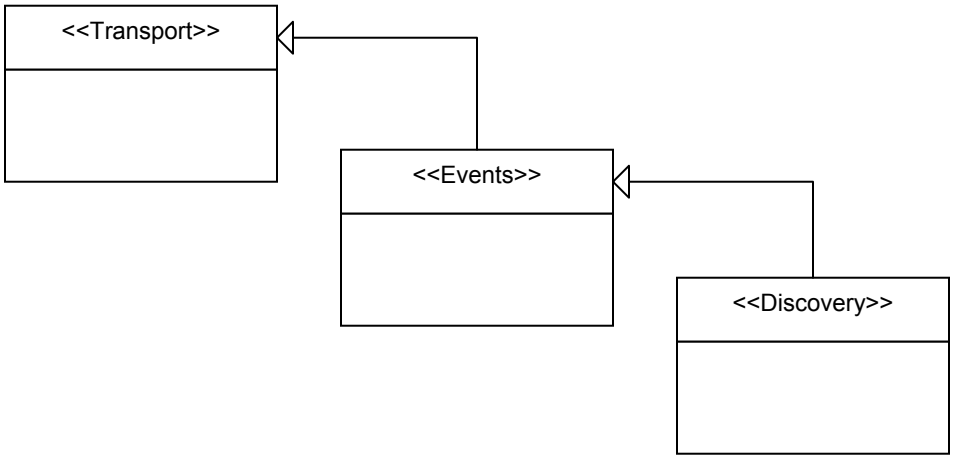


FIGURE 19 - DISCOVERY SERVICE

Description

The process of discovery is conducted at both the node level and the subsystem level. This service supports the discovery of both legacy components defined in the JAUS Reference Architecture versions 3.2+, and new components. The Component IDs of legacy components were fixed at specification time (Primitive Driver = 33 for example) and could contain only one service beyond the core service support. New components may use any component ID that is not used in the Reference Architecture for legacy component definitions. New components can also contain two or more services beyond the core service support.

The typical messaging sequence used during the discovery process is as shown in the interfaction diagram in the protocol behavior section below.

Assumptions

Messages may be lost or reordered

Vocabulary

TABLE 34 - DISCOVERY SERVICE VOCABULARY

Message Id (hex)	Name	Command
Input Set		
0B00	RegisterServices	True
2B00	QueryIdentification	False
2B01	QueryConfiguration	False
2B02	QuerySubsystemList	False
2B03	QueryServices	False
Output Set		
4B00	ReportIdentification	False
4B01	ReportConfiguration	False
4B02	ReportSubsystemList	False
4B03	ReportServices	False

Protocol Behavior

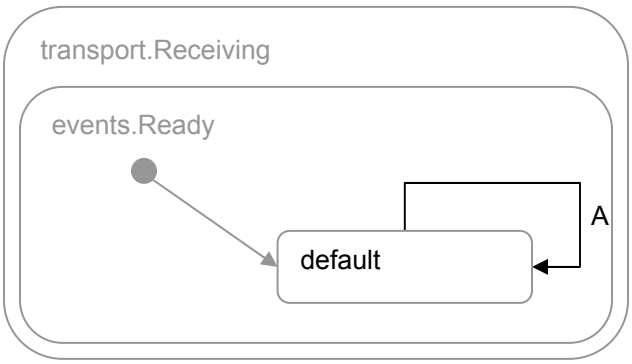


FIGURE 20 - DISCOVERY SERVICE PROTOCOL BEHAVIOR

TABLE 35 - DISCOVERY SERVICE STATE TRANSITIONS

Label	Trigger	Conditions	Actions
A	RegisterServices		*publishServices*
	QueryIdentification		sendReportIdentification
	QueryConfiguration		sendReportConfiguration
	QuerySubsystemList		sendReportSubsystemList
	QueryServices		sendReportServices

TABLE 36 - DISCOVERY SERVICE TRANSITION ACTIONS

Action	**Interpretation**
sendReportIdentification	Send a Report Identification message to the component that sent the query.
sendReportConfiguration	Send a Report Configuration message to the component that sent the query
sendReportSubsystemList	Send a Report Subsystem List message to the component that sent the query
sendReportServices	Send a Report Services message to the component that sent the query
publishServices	Add the component that sent the message to the list of reported services

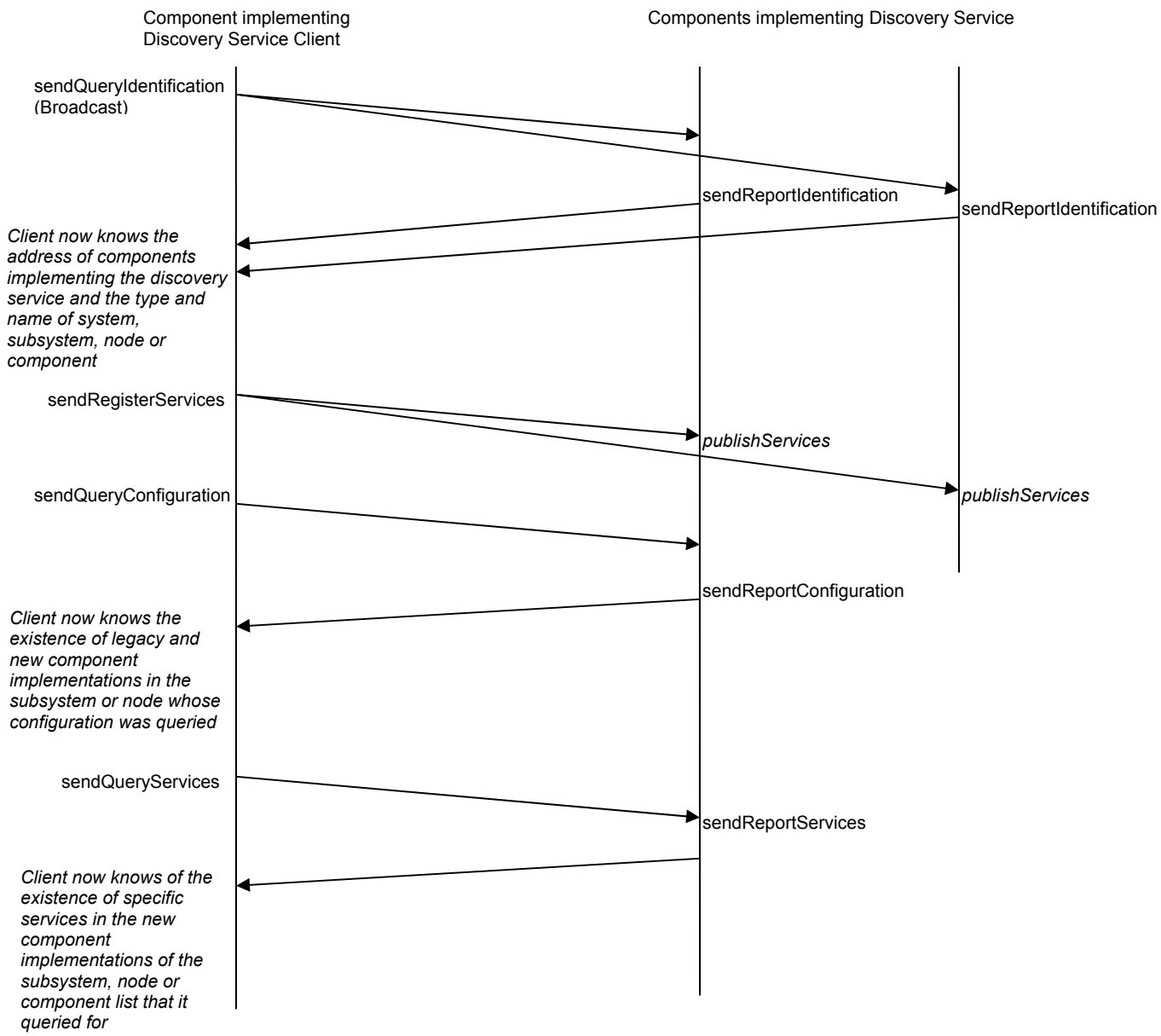


FIGURE 21 - SEQUENCE DIAGRAM OF SERVICE DISCOVERY

6. DECLARED TYPES

6.1 CommandClass

Command class messages are used to precipitate actions within a component.

6.1.1 ID 0001h: SetAuthority

This message shall set the command authority of the receiving component. The authority bits range in value from 0 to 255 with 255 being the highest.

TABLE 37 - SET AUTHORITY MESSAGE ENCODING

body
└ **record** name=SetAuthorityRec

record name=SetAuthorityRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** `AuthorityCode`	unsigned byte	one	false	range: 0-255

6.1.2 ID 0002h: Shutdown

This message is used to cause the receiving component to free all of the resources allocated to its process by the system and then to shutdown.

TABLE 38 - SHUTDOWN MESSAGE ENCODING

body
└ (empty)

6.1.3 ID 0003h: Standby

This message is used to transition the receiving component to the standby state.

TABLE 39 - STANDBY MESSAGE ENCODING

body
└ (empty)

6.1.4 ID 0004h: Resume

This message is used to transition the receiving component to the ready state.

TABLE 40 - RESUME MESSAGE ENCODING

body
└ (empty)

6.1.5 ID 0005h: Reset

This message is used to cause the receiving component to reinitialize.

TABLE 41 - RESET MESSAGE ENCODING

body
└ (empty)

6.1.6 ID 0006h: SetEmergency

This message is used to alert the component to a safety critical situation. The component that sends the emergency command shall set the message priority to the safety critical priority range as defined by the Transport.Receive of the emergency command shall result in the component transitioning into the emergency state.

TABLE 42 - SET EMERGENCY MESSAGE ENCODING

body └ **record** name=SetEmergencyRec

record name=SetEmergencyRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** EmergencyCode	unsigned short integer	one[2]	False	value_enum 1 = Stop

6.1.7 ID 0007h: ClearEmergency

This message shall notify the receiving component that the current emergency condition is to be reset and that the component shall transition back to the Ready or Standby state, provided that all emergency conditions have been cleared.

JAUS currently defines only one emergency condition, the “Stop” condition. Future versions of this document could define other emergency conditions.

[2] ‘one’ represents a dimensionless quantity in SI units [bnt]

TABLE 43 - CLEAR EMERGENCY MESSAGE ENCODING

body
└ **record** name=ClearEmergencyRec

record name=ClearEmergencyRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** EmergencyCode	unsigned short integer	one[2]	False	value_enum 1 = Stop

6.1.8 ID 000Dh: RequestControl

This message is used to request interruptible control of the receiving component. Once control is established, the receiving component shall only execute commands from the sending component. The authority code parameter is to be set equal to that of the sending component. The receiving component must always accept the control of the highest authority component that is requesting uninterruptible control. Commands from all other components are ignored unless from a component with higher authority.

TABLE 44 - REQUEST CONTROL MESSAGE ENCODING

body
└ **record** name=RequestControlRec

record name=RequestControlRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** `AuthorityCode`	unsigned byte	one[2]	false	range: 0-255

6.1.9 ID 000Eh: ReleaseControl

This message is used to relinquish uninterruptible control of the receiving component.

TABLE 45 - RELEASE CONTROL MESSAGE ENCODING

body
└ (empty)

6.1.10 ID 000Fh: ConfirmControl

The Confirm Control message is used to notify a component that it accepts control from that component (or not). When control has been granted, response code of 0, the component under control will only execute messages from the controlling component until control is released or interrupted. When the requesting component has lower authority than the current controlling entity, the response will be 2. For components not supporting interruptible control, or if the component is engaged in other operations that would prevent this service from performing its actions to grant control, the response code value of 1 can be used.

TABLE 46 - CONFIRM CONTROL MESSAGE ENCODING

body
└ **record** name=ConfirmControlRec

record name=ConfirmControlRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** ResponseCode	unsigned byte	one[2]	false	value_enum 0 CONTROL ACCEPTED value_enum 1 NOT AVAILABLE value_enum 2 INSUFFICIENT AUTHORITY

6.1.11 ID 0010h: RejectControl

The Reject Coontrol message is used to notify a component that control has been released (response code = 0), or a request to release control could not be processed (response code = 1).

TABLE 47 - REJECT CONTROL MESSAGE ENCODING

body
└ **record** name=RejectControlRec

record name=RejectControlRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** ResponseCode	unsigned byte	one[2]	false	value_enum 0 CONTROL RELEASED value_enum 1 NOT AVAILABLE

6.1.12 ID 0011h: SetTime

Time is configured for a JAUS component using the following message. Accuracy of the time may be dependent on latencies in the transmission of the message. Proper systems engineering procedures should be used to insure the accuracy of the time messages are within the system tolerance. All times are in Coordinated Universal Time (UTC).

TABLE 48 - SET TIME MESSAGE ENCODING

body
└ **record** name=TimeRec

record name=TimeRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<presence_vector>**	unsigned byte	one[2]	false	
2	**<bit_field>** Time	unsigned integer	one[2]	true	Bits 0-9: milliseconds, range 0...999 Bits 10-15: Seconds, range 0...59 Bits 16 – 21: Minutes, range 0...59 Bits 22-26: Hour (24 hour clock), range 0..23 Bits 27-31: Day, range 1...31
3	**<bit_field>** Date	unsigned short integer	one[2]	true	Bits 0-4: Day, range 1...31 Bits 5-8: Month, range 1...12 Bits 9 – 15: Year, range 2000...2127 Where 0 is 2000, 1 is 2001, etc.

6.1.13 ID 01F0h: CreateEvent

This message is used to set up an event. Field 1 is a local request ID that the event provider returns in the Confirm or Reject message. Field 2 is the Event Type, which allows the requester to specify the type of event – Periodic specifies that the event is triggered at regular intervals in time, in which case field 3 (Requested periodic rate) must be set to a non-zero value. Event type of Every Change specifies that the corresponding Report message should be sent every time the data associated with that message changes.

Field 4 contains the size of the Query message that is to specify the contents of the Report. Field 5 contains the Query message (including its two byte header).

TABLE 49 - CREATE EVENT MESSAGE ENCODING

body └ **record** name=CreateEventRec					
record name=CreateEventRec					
Field #	**Name**	**Type**	**Units**	**Optional**	**Interpretation**
1	<fixed_field> RequestID	unsigned byte	one[2]	false	Local Request ID for use in confirm event
2	<fixed_field> EventType	unsigned byte	one[2]	false	value_enum 0 Periodic 1 Every change
3	<fixed_field> Requested PeriodicRate	unsigned short integer	hertz	false	Must be specified for periodic event, and set to 0 for every change scale_range real_lower_limit=0 real_upper_limit=1092
4&5	<variable_length_field> QueryMessage	count_field = unsigned integer	one[2]	false	The JAUS Query message to be used by the receiving component to generate the report message(s)

6.1.14 ID 01F1h: UpdateEvent

The Update Event message allows the requester to request a rate or change. The format is the same as the Create Event, only with the addition of Event ID field to specify the event that needs updating.

TABLE 50 - UPDATE EVENT MESSAGE ENCODING

body
└ **record** name=UpdateEventRec

record name=UpdateEventRec

Field #	Name	Type	Units	Optional	Interpretation
1	<fixed_field> RequestID	unsigned byte	one[2]	false	Local Request ID for use in confirm event
2	<fixed_field> EventType	unsigned byte	one[2]	false	value_enum 0 Periodic 1 Every change
3	<fixed_field> RequestedPeriodic Rate	unsigned short integer	hertz	false	Must be specified for periodic event, and set to 0 for every change scale_range real_lower_limit=0 real_upper_limit=1092
4	Event ID	unsigned byte	one[2]	false	Unique identifier of existing event to update
5&6	<variable_length_field> QueryMessage	count_field = unsigned integer	one[2]	false	The JAUS Query message to be used by the receiving component to generate the report message(s)

6.1.15 ID 01F2h: CancelEvent

The Cancel Event message is used by the requester to cancel and/or request deletion of the specified event.

TABLE 51 - CANCEL EVENT MESSAGE ENCODING

body
└ **record** name=CancelEventRec

record name=CancelEventRec

Field #	Name	Type	Units	Optional	Interpretation
2	<fixed_field> RequestID	unsigned byte	one[2]	false	Local Request ID for use in confirm/reject message
4	<fixed_field> EventID	unsigned byte	one[2]	false	Unique identifier of existing event to be removed

6.1.16 ID 01F3h: ConfirmEventRequest

The Confirm Event message is used to confirm an Event has been created/updated/or cancelled. Field 1 represents the Request ID from the Create, Update, or Cancel message that initiated this message. The Request ID's scope is local to the requesting client only. Field 2, Event ID, is a globally unique ID that is established by the service when the event is created. Field 3 is used to specify the closest rate that the service can provide if it cannot match the requested rate.

TABLE 52 - CONFIRM EVENT REQUEST MESSAGE ENCODING

body
└ **record** name=ConfirmEventRequestRec

record name=ConfirmEventRequestRec

Field #	Name	Type	Units	Optional	Interpretation
1	<fixed_field> RequestID	unsigned byte	one[2]	false	ID of the event maintenance request (Create, Update, or Cancel)
2	<fixed_field> EventID	unsigned byte	one[2]	false	The identifier of the specific event
3	<fixed_field> ConfirmedPeriodic Rate	unsigned short integer	hertz	false	Requested rate or closest to requested rate scale_range lower_limit=0 upper_limit=1092

6.1.17 ID 01F4h: RejectEventRequest

The Reject Event Request message is used to reject an Event creation, update or cancellation. Field 2 represents the Request ID from the Create, Update, or Cancel message that initiated this message. The Request ID's scope is local to the requesting client only.

TABLE 53 - REJECT EVENT REQUEST MESSAGE ENCODING

body
└ **record** name=RejectEventRequestRec

record name=RejectEventRequestRec

Field #	Name	Type	Units	Optional	Interpretation
1	<presence_vector>	unsigned byte	one[2]	false	
2	<fixed_field> RequestID	unsigned byte	one[2]	false	ID of the event maintenance request (create, update, or cancel)
3	<fixed_field> ResponseCode	unsigned byte	one[2]	false	value_enum 1 periodic events not supported 2 change based events not supported 3 connection refused 4 invalid event setup 5 message not supported 6 error, invalid event ID for update event request
4	<fixed_length_string> ErrorMessage	String of size 80 bytes	one[2]	true	String for additional information

6.1.18 ID 0B00h: RegisterServices

This message allows a component to register its capabilities with a Discovery service. If a component ID is specified in the JAUS Reference Architecture version 3.2+, it may report only one service beyond the core message support, and this service must be equal to the component ID. If a component ID is not listed in the RA, it may report any number of services. For example, a component with ID 33 must provide only service 33. The exception to this rule is component ID 1 (the Node Manager) which may provide any number of services in addition to core message support. Note that this restriction may be removed in future versions of this Standard.

TABLE 54 - REGISTER SERVICES MESSAGE ENCODING

body
└ **list** name=ServiceList
(count_field = unsigned byte)
└ **Record** name=ServiceRec

record name=ServiceRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<variable_length_ string>** URI	count_field= unsigned byte	one[2]	false	Service URI
2	**<fixed_field>** MajorVersionNumber	unsigned byte	one[2]	false	Major version number
3	**<fixed_field>** MinorVersionNumber	unsigned byte	one[2]	false	Minor version number

6.2 QueryClass

Query class messages are used to request information from a service. The response to a Query Class message is an Inform class message.

6.2.1 ID 2001h: QueryAuthority

This message is used by clients to query the current value of the authority code of this service.

TABLE 55 - QUERY AUTHORITY MESSAGE ENCODING

body
└ (empty)

6.2.2 ID 2002h: QueryStatus

This message is used by clients of this service to query the state of this service.

TABLE 56 - QUERY STATUS MESSAGE ENCODING

body
└ (empty)

6.2.3 ID 2003h: QueryTimeout

This message is used by clients of this service to query the timeout period of the service.

TABLE 57 - QUERY TIMEOUT MESSAGE ENCODING

body
└ (empty)

6.2.4 ID 2011h: QueryTime

This message is used by cleints of this service to query the current time of this service.

TABLE 58 - QUERY TIME MESSAGE ENCODING

body
└ **record** name=QueryTimeRec

record name=QueryTimeRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** PresenceVector	unsigned byte	one[2]	false	See corresponding ReportTime message for presence vector mapping

6.2.5 ID 200Dh: QueryControl

This message is used by clients to query the current control state of this service.

TABLE 59 - QUERY CONTROL MESSAGE ENCODING

body
└ (empty)

6.2.6 ID 21F0h: QueryEvents

The Query Events message is used to request detail on events. Queries can be made by message ID, event type or Event ID. If no filter is specified, all events should be reported.

TABLE 60 - QUERY EVENTS MESSAGE ENCODING

body
└ **variant** name=QueryEventsVar
(vtag_field = unsigned byte)
record name=MessageIDRec
record name=EventTypeRec
record name=EventIDRec
record name=AllEventsRec

record name=MessageIDRec

Field #	Name	Type	Units	Optional	Interpretation
1	<fixed_field> MessageID	unsigned short integer	one[2]	false	Query Message ID of the Event message that the receiving component is inquiring about.

record name=EventTypeRec

Field #	Name	Type	Units	Optional	Interpretation
1	<fixed_field> EventType	unsigned byte	one[2]	false	value_enum 0 Periodic 1 Every change

record name=EventIDRec

Field #	Name	Type	Units	Optional	Interpretation
1	<fixed_field> EventID	unsigned byte	one[2]	false	Event ID returned by Confirm Event for details on specific event.

record name=AllEventsRec

Field #	Name	Type	Units	Optional	Interpretation
1	<fixed_field> AllEvents	unsigned byte	one[2]	false	All events should be reported. value_range(0,0)

6.2.7 ID 2202h: QueryHeartbeatPulse

This message shall be used to query for a heartbeat pulse.

TABLE 61 - QUERY HEARTBEAT PULSE MESSAGE ENCODING

body
└ **empty**

6.2.8 ID 2B00h: QueryIdentification

This message shall request the identification summary of a Subsystem, Node, or Component.

TABLE 62 - QUERY IDENTIFICATION MESSAGE ENCODING

body
└ **record** name=QueryIdentificationRec

record name=QueryIdentificationRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** QueryType	unsigned byte	one[2]	false	0: Reserved 1: System Identification 2: Subsystem Identification 3: Node Identification 4: Component Identification 5 – 255: Reserved

6.2.9 ID 2B01h: QueryConfiguration

This message shall request the configuration summary of a subsystem or node. For example, to get the complete configuration of a subsystem, field 1 shall be set to 2.

TABLE 63 - QUERY CONFIGURATION MESSAGE ENCODING

body
└ **record** name=QueryConfigurationRec

record name=QueryConfigurationRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** QueryType	unsigned byte	one[2]	false	0: Reserved 1: Reserved 2: Subsystem Configuration 3: Node Configuration 4– 255: Reserved

6.2.10 ID 2B02h: QuerySubsystemList

This message shall request the Report Subsystem List message. There are no data fields associated with this message.

TABLE 64 - QUERY SUBSYSTEM LIST MESSAGE ENCODING

body
└ (empty)

6.2.11 ID 2B03h: QueryServices

This message allows a component to request the service information of an entire subsystem or node, or a single component. The corresponding Report Services message will respond with service information only for new component implementations. It will not report service information for legacy component implementations.

TABLE 65 - QUERY SERVICES MESSAGE ENCODING

body
└ **list** name=NodeList
(count_field = unsigned byte)
sequence name=NodeSeq
record name=NodeRec
list name = ComponentList
(count_field = unsigned byte)
record name = ComponentRec

record name=NodeRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** Node ID	unsigned byte	one[2]	false	Use 255 if service information from all nodes in the subsystem is required value_range 0: Reserved 1-254: valid Node IDs 255: All nodes in the subsystem

record name=ComponentRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** Component ID	unsigned byte	one[2]	false	Use 255 if service information from components in the node are required value_range 0: Reserved 1-254: valid Component IDs 255: All components in the node

6.3 InformClass

Inform class messages are used to report information from a service. The Inform Class message is in response to a Query Class message.

6.3.1 ID 4001h: ReportAuthority

This message is used to report the current command authority and associated time-out value.

TABLE 66 - REPORT AUTHORITY MESSAGE ENCODING

body
└ **record** name=AuthorityRec

record name=AuthorityRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<declared_field>** `AuthorityCode`	Unsigned byte	one[2]	false	

6.3.2 ID 4002h: ReportStatus

This message is used to report the current status of the sender of the message.

TABLE 67 - REPORT STATUS MESSAGE ENCODING

body └ **record** name=ReportStatusRec					
record name=ReportStatusRec					
Field #	**Name**	**Type**	**Units**	**Optional**	**Interpretation**
1	**<fixed_field>** Status	unsigned byte	one[2]	false	value_enum 0 INIT value_enum 1 READY value_enum 2 STANDBY value_enum 3 SHUTDOWN value_enum 4 FAILURE value_enum 5 EMERGENCY
2	**<fixed_field>** reserved	unsigned integer	one[2]	false	This field is reserved for compatibility with previous versions of the Standard.

6.3.3 ID 4003h: ReportTimeout

This message is used to report the timeout period of the service.

TABLE 68 - REPORT TIMOUT MESSAGE ENCODING

body └ **record** name=ReportTimeoutRec					
record name=ReportStatusRec					
Field #	**Name**	**Type**	**Units**	**Optional**	**Interpretation**
2	**<fixed_field>** Timeout	Unsigned byte	Seconds	False	Clients must re-request control to prevent being rejected when the timeout expires. A value of zero indicates this feature is disabled.

6.3.4 ID 4011h: ReportTime

This message is used to report the current time of this service to querying clients. All times are in Coordinated Universal Time (UTC).

TABLE 69 - REPORT TIME MESSAGE ENCODING

body
└ **record** name=TimeRec

record name=TimeRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<presence_vector>**	unsigned byte	one[2]	false	
2	**<bit_field>** Time	unsigned integer	one[2]	true	Bits 0-9: milliseconds, range 0...999 Bits 10-15: Seconds, range 0...59 Bits 16 – 21: Minutes, range 0...59 Bits 22-26: Hour (24 hour clock), range 0..23 Bits 27-31: Day, range 1...31
3	**<bit_field>** Date	unsigned short integer	one[2]	true	Bits 0-4: Day, range 1...31 Bits 5-8: Month, range 1...12 Bits 9 – 15: Year, range 2000...2127 Where 0 is 2000, 1 is 2001, etc.

6.3.5 ID 400Dh: ReportControl

This message is used to report the current state of control of this service. If the serivce is in the Controlled state, this message reports the ID of the controlling component. The ID fields shall be set to zero (0) if this service is in the NotControlled state..

TABLE 70 - REPORT CONTROL MESSAGE ENCODING

body
└ **record** name=ReportControlRec

record name=ReportControlRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** SubsystemID	unsigned short integer	one[2]	false	value_range (0,65535)
2	**<fixed_field>** NodeID	unsigned byte	one[2]	false	value_range (0,254)
3	**<fixed_field>** ComponentID	unsigned byte	one[2]	false	value_range (0,254)
4	**<fixed_field>** AuthorityCode	unsigned byte	one[2]	false	value_range (0,255)

6.3.6 ID 41F0h: ReportEvents

This message is used to report the active event requests that match the requirements provided in the QueryEvents message.

TABLE 71 - REPORT EVENTS MESSAGE ENCODING

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

record name=ReportEventsRec

Field #	Name	Type	Units	Optional	Interpretation
1	<fixed_field> EventType	unsigned byte	one[2]	false	value_enum 0 Periodic 1 Every change
2	<fixed_field> EventID	unsigned byte	one[2]	false	Unique Identifier of existing event to update
3&4	<variable_length_field> QueryMessage	count_field = unsigned integer	one[2]	false	The JAUS Query message to be used by the receiving component to generate the report message(s)

6.3.7 ID 41F1h: Event

The Event message is sent when an event is triggered. It includes the Event ID and a sequence number to allow the client to keep track of event processing.

TABLE 72 - EVENT MESSAGE ENCODING

body
└ **record** name=EventRec

record name=EventRec

Field #	Name	Type	Units	Optional	Interpretation
1	<fixed_field> Event ID	unsigned byte	N/A	False	Unique identifier of the enclosed event
2	<fixed_field> Sequence Number	unsigned byte	N/A	False	Sequential count of the number of times this event has been issued
3&4	<variable_length_field> ReportMessage	count_field = unsigned integer	N/A	False	The JAUS Report message including the message header

6.3.8 ID 4202h: ReportHeartbeat Pulse

This message notifies the receiver that the sender is alive.

TABLE 73 - REPORT HEARTBEAT PULSE MESSAGE ENCODING

body └ **empty**

6.3.9 ID 4B00h: ReportIdentification

This message shall provide the requesting component an identification summary of the Subsystem, Node, or Component.

TABLE 74 - REPORT IDENTIFICATION MESSAGE ENCODING

body └ **record** name=ReportIdentificationRec					
record name=ReportIdentificationRec					
Field #	**Name**	**Type**	**Units**	**Optional**	**Interpretation**
1	**<fixed_field>** QueryType	unsigned byte	one[2]	false	value_enum: 0: Reserved 1: System Identification 2: Subsystem Identification 3: Node Identification 4: Component Identification 5 – 255: Reserved
2	**<fixed_field>** Type	unsigned short	one[2]	false	This field identifies the particular unmanned vehicle (subsystem) type, node type or component type based on the following enumeration value_enum: 00000: Reserved 10001: VEHICLE 10002-20000: Reserved 20001: OCU 20002-30000: Reserved 30001: OTHER_SUBSYSTEM 30002-40000: Reserved 40001: NODE 40002-50000: Reserved 50001: PAYLOAD 50002-60000: Reserved 60001: COMPONENT 60002-65535: Reserved
3	**<variable_length_ string>** Identification	count_field = unsigned byte	one[2]	false	Human-recognizable name of the Subsystem, Node or Component.

6.3.10 ID 4B01h: ReportConfiguration

This message is used to provide the receiver a table of all existing components located on the source's subsystem or node depending on the value of field 1 of the ID 2B01h: Query Configuration message.

TABLE 75 - REPORT CONFIGURATION MESSAGE ENCODING

body
└ **list** name=NodeList
  (count_field = unsigned byte)
  └ **sequence** name=NodeSeq
    └ **record** name=NodeRec
      list name=ComponentList
        (count_field = unsigned byte)
        └ **record**
      name=ComponentRec

record name=NodeRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** NodeID	unsigned byte	one[2]	false	Node ID. For single Node or Component reporting this field shall contain the Node ID as specified in the Destination Address of the Query Configuration message

record name=ComponentRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** ComponentID	unsigned byte	one[2]	false	Comp ID. For Single Component reporting this field shall contain the Component ID as specified in the Destination Address of the Query Configuration message and shall be the only Component reported
2	**<fixed_field>** InstanceID	unsigned byte	one[2]	false	Inst ID when legacy Components are reported; a value of zero (0) shall be used for non-legacy components.

6.3.11 ID 4B02h: ReportSubsystemList

This message shall provide the receiving component a table of all subsystems located in the source's system. It also provides the ID of the component to send a Query Configuration message within the subsystem.

TABLE 76 - REPORT SUBSYSTEM LIST MESSAGE ENCODING

body
└ **list** name=SubsystemList
  (count_field = unsigned byte)
  └ **record** name=SubsystemRec

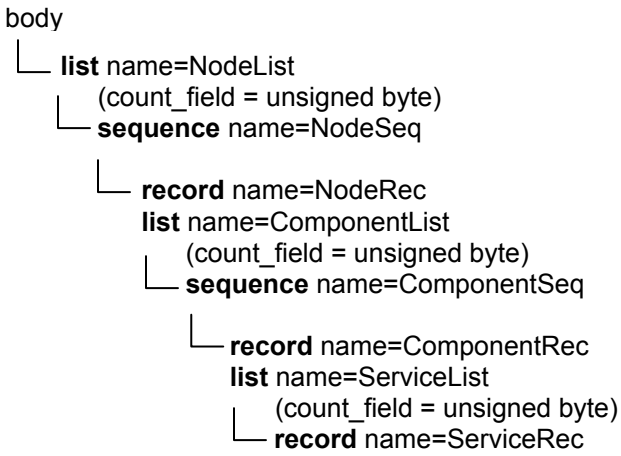
record name=SubsystemRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** SubsystemID	unsigned short integer	one[2]	false	Subsystem ID.
2	**<fixed_field>** NodeID	unsigned byte	one[2]	false	Node ID used for Query Configuration.
3	**<fixed_field>** ComponentID	unsigned byte	one[2]	false	Component ID used for Query Configuration

6.3.12 ID 4B03h: ReportServices

This message allows a component to publish its capabilities, according to the Service Dictionary presented below. If a component ID is specified in the RA, it may report only one service beyond the core message support, and this service must be equal to the component ID. If a component ID is not listed in the RA, it may report any number of services. For example, a component with ID 33 must provide only service 33. The exception to this rule is component ID 1 (the Node Manager) which may provide any number of services in addition to core message support.

TABLE 77 - REPORT SERVICES MESSAGE ENCODING



record name=NodeRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** NodeID	unsigned byte	one[2]	false	Node ID. For single Node or Component reporting this field shall contain the Node ID as specified in the Destination Address of the Query Configuration message

record name=ComponentRec

Field #	Name	Type	Units	Optional	Interpretation
1	**<fixed_field>** ComponentID	unsigned byte	one[2]	false	Comp ID. For Single Component reporting this field shall contain the Component ID as specified in the Destination Address of the Query Configuration message and shall be the only Component reported
2	**<fixed_field>** InstanceID	unsigned byte	one[2]	false	Inst ID when legacy Components are reported; a value of zero (0) shall be used for non-legacy components.

record name=ServiceRec

Field #	Name	Type	Units	Optional	Interpretation
1&2	**<variable_length_ string>** URI	count_field = unsigned byte	one[2]	false	Service URI
3	**<fixed_field>** MajorVersionNumber	unsigned byte	one[2]	false	Major version number
4	**<fixed_field>** MinorVersionNumber	unsigned byte	one[2]	false	Minor version number

7. NOTES

7.1 A change bar (l) located in the left margin is for the convenience of the user in locating areas where technical revisions, not editorial changes, have been made to the previous issue of this document. An (R) symbol to the left of the document title indicates a complete revision of the document, including technical revisions. Change bars and (R) are not used in original publications, nor in documents that contain editorial changes only.

APPENDIX A - XML FOR SERVICE DEFINITIONS

A.1 TRANSPORT

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def xmlns="urn:jaus:jsidl:1.0" id="urn:jaus:jss:core:Transport"
  name="Transport" version="1.0">
  <description xml:space="preserve">
    The transport service acts as an interface to the JAUS transport layer. It models an
abstract bi-directional communication channel (input queue and output queue) whose primary
function is to provide the capability of sending messages to a single destination endpoint
or broadcasting messages to all endpoints in the system, and to receive a message from any
source endpoint. It also provides the capability to prioritize the delivery of sent
messages.
    This service establishes a communication endpoint whose address is defined by a triple
{SubsystemID, NodeID, ComponentID} as specified by the Send and Receive internal events.
Other services that need to utilize the communication channel provided by the transport
service must inherit from the transport service.
  </description>
  <assumptions> Messages may be delayed, lost or reordered. </assumptions>
  <declared_type_set name="TransportTypeSet">
    <record name="SendRec" optional="false">
      <presence_vector field_type_unsigned="unsigned byte"/>
      <fixed_field name="DestSubsystemID" field_type="unsigned short integer"
        field_units="one" interpretation="Destination Subsystem ID, where a value of
0xFFFF represents all subsystems" optional="false"/>
      <fixed_field name="DestNodeID" field_type="unsigned byte" field_units="one"
        interpretation="Destination Node ID where a value of 0xFF represents all nodes."
optional="false"/>
      <fixed_field name="DestComponentID" field_type="unsigned byte" field_units="one"
        interpretation="Destination Component ID where a value of 0xFF represents all
components." optional="false"/>
      <fixed_field name="SrcSubsystemID" field_type="unsigned short integer"
field_units="one"
        interpretation="Source Subsystem ID" optional="true"/>
      <fixed_field name="SrcNodeID" field_type="unsigned byte" field_units="one"
        interpretation="Source Node ID" optional="true"/>
      <fixed_field name="SrcComponentID" field_type="unsigned byte" field_units="one"
        interpretation="Destination Component ID" optional="false"/>
      <fixed_field name="Priority" field_type="unsigned byte" field_units="one"
optional="true"
        interpretation="Priority level of this message. If not specified, the normal
priority
        level is used.">
        <value_set offset_to_lower_limit="false">
          <value_range lower_limit="0" lower_limit_type="inclusive" upper_limit="3"
            upper_limit_type="inclusive" interpretation="priority range"/>
          <value_enum enum_index="0" enum_const="LOW"/>
          <value_enum enum_index="1" enum_const="NORMAL"/>
          <value_enum enum_index="2" enum_const="HIGH"/>
          <value_enum enum_index="3" enum_const="SAFETY"/>
        </value_set>
      </fixed_field>
      <variable_length_field name="MessagePayload" field_format="JAUS MESSAGE"
        optional="false" interpretation="Message payload.">
        <count_field field_type_unsigned="unsigned long integer" min_count="2"/>
      </variable_length_field>
    </record>
  </declared_type_set>
  <message_set>
    <input_set/>
    <output_set/>
```

```
  </message_set>
  <internal_events_set>
    <event_def name="Send">
      <description xml:space="preserve">
        The Send event is used by a derived service to hand over the payload data that it
needs to send to a specified destination endpoint via the transport layer. Upon receipt,
this service prepares the message for delivery (marshals the message) as specified by the
transport layer specification and attempts to deliver the encapsulated payload data to the
destination endpoint.
      </description>
      <header name="Header"/>
      <body name="Body">
        <declared_record name="SendRec" declared_type_ref="TransportRecord"
optional="false"/>
      </body>
      <footer name="Footer"/>
    </event_def>
    <event_def name="BroadcastLocal">
      <description xml:space="preserve">
        The Broadcast Local event is the same as the Send event except that this service
sends the payload to all endpoints within the subsystem.
      </description>
      <header name="Header"/>
      <body name="Body">
        <declared_record name="SendRec" declared_type_ref="TransportRecord"
optional="false"/>
      </body>
      <footer name="Footer"/>
    </event_def>
    <event_def name="BroadcastGlobal">
      <description xml:space="preserve">
        The Broadcast Global event is the same as the Send event except that this service
sends the payload to all endpoints on all subsystems.
      </description>
      <header name="Header"/>
      <body name="Body">
        <declared_record name="SendRec" declared_type_ref="TransportRecord"
optional="false"/>
      </body>
      <footer name="Footer"/>
    </event_def>
    <event_def name="Receive">
      <description xml:space="preserve">
        The Receive event is used by derived services to hand over the data that was sent
by another service to an endpoint established by this service.
      </description>
      <header name="Header"/>
      <body name="Body">
        <record name="ReceiveRec" optional="false" interpretation="transport data">
          <fixed_field name="SrcSubsystemID" field_type="unsigned short integer"
field_units="one"
            interpretation="Source Subsystem ID" optional="false"/>
          <fixed_field name="SrcNodeID" field_type="unsigned byte" field_units="one"
            interpretation="Source Node ID" optional="false"/>
          <fixed_field name="SrcComponentID" field_type="unsigned byte" field_units="one"
            interpretation="Destination Component ID" optional="false"/>
          <variable_length_field name="MessagePayload" field_format="JAUS MESSAGE"
            optional="false" interpretation="message payload">
            <count_field field_type_unsigned="unsigned long integer" min_count="2"/>
          </variable_length_field>
        </record>
      </body>
```

```
      <footer name="Footer"/>
    </event_def>
  </internal_events_set>
  <protocol_behavior>
    <start state_machine_name="ReceiveFSM" state_name="Receiving"/>
    <start state_machine_name="SendFSM" state_name="Sending"/>
    <state_machine name="ReceiveFSM">
      <state name="Receiving">
        <transition name="Receive">
          <simple/>
        </transition>
      </state>
    </state_machine>
    <state_machine name="SendFSM">
      <state name="Sending">
        <transition name="Send">
          <parameter type="Send" value="msg"/>
          <simple/>
          <action name="Enqueue" interpretation="Convert the destination address into an
unsigned
            integer such that the ComponentID maps to the least significant byte, NodeID
to the next
            least significant byte and SubsystemID maps onto the remaining two bytes of
the integer.
            Package the message as specified by the transport layer specification and send
it to its
            destination as per the specified priority.">
            <argument value="msg"/>
          </action>
        </transition>
        <transition name="BroadcastLocal">
          <parameter type="BroadcastLocal" value="msg"/>
          <simple/>
          <action name="BroadcastLocalEnqueue" interpretation="Package the message as
specified by the transport layer specification and send it to all endpoints in the local
subsystem.">
            <argument value="msg"/>
          </action>
        </transition>
        <transition name="BroadcastGlobal">
          <parameter type="BroadcastGlobal" value="msg"/>
          <simple/>
          <action name="BroadcastGlobalEnqueue" interpretation="Package the message as
specified by the transport layer specification and send it to all endpoints in on all
subsystems.">
            <argument value="msg"/>
          </action>
        </transition>
      </state>
    </state_machine>
  </protocol_behavior>
</service_def>
```

A.2 EVENTS

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def xmlns="urn:jaus:jsidl:1.0" name="Events" id="urn:jaus:jss:core:Events"
  version="1.0">
  <description>This service is used to set up event notifications. Since this service does
not
    contain any messages and data on which events can be setup, it is useful only when
derived by
    other services that contain messages and data on which events can be
defined.</description>
  <assumptions>Messages may be delayed, lost or reordered.</assumptions>
  <references>
    <inherits_from name="transport" id="urn:jaus:jss:core:Transport" version="1.0"/>
  </references>
  <declared_type_set name="Types">
    <declared_type_set_ref name="core" id="urn:jaus:jss:core:MessageSet" version="1.0"/>
  </declared_type_set>
  <message_set>
    <input_set>
      <declared_message_def name="CreateEvent"
declared_type_ref="core.CommandClass.CreateEvent"/>
      <declared_message_def name="UpdateEvent"
declared_type_ref="core.CommandClass.UpdateEvent"/>
      <declared_message_def name="CancelEvent"
declared_type_ref="core.CommandClass.CancelEvent"/>
      <declared_message_def name="QueryEvents"
declared_type_ref="core.QueryClass.QueryEvents"/>
    </input_set>
    <output_set>
      <declared_message_def name="ConfirmEventRequest"
declared_type_ref="core.CommandClass.ConfirmEventRequest"/>
      <declared_message_def name="RejectEventRequest"
declared_type_ref="core.CommandClass.RejectEventRequest"/>
      <declared_message_def name="ReportEvents"
declared_type_ref="core.InformClass.ReportEvents"/>
      <declared_message_def name="Event" declared_type_ref="core.InformClass.Event"/>
    </output_set>
  </message_set>
  <internal_events_set>
    <event_def name="EventOccurred">
      <description> Received when an event occurs. </description>
      <header name="Header"/>
      <body name="Body"/>
      <footer name="Footer"/>
    </event_def>
    <event_def name="EventError">
      <description> Received when an event error occurs. </description>
      <header name="Header"/>
      <body name="Body"/>
      <footer name="Footer"/>
    </event_def>
  </internal_events_set>
  <protocol_behavior is_stateless="false">
    <start state_machine_name="transport.ReceiveFSM" state_name="Receiving.Ready"/>
    <state_machine name="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">
          <transition name="transport.Receive">
```

```
            <parameter type="QueryEvents" value="msg"/>
            <parameter type="ReceiveRec" value="transportData"/>
            <simple/>
            <action name="transport.Send" interpretation="Send Report Events message to
the
              component that sent the query">
              <argument value=" 'ReportEvents' "/>
              <argument value="msg"/>
              <argument value="transportData"/>
            </action>
          </transition>
          <transition name="transport.Receive">
            <parameter type="CreateEvent" value="msg"/>
            <parameter type="ReceiveRec" value="transportData"/>
            <guard condition="isSupported(msg) &amp;amp;&amp;amp; ! eventExists(msg)"
interpretation="True if parameters are supported and the event does not already exist."/>
            <simple/>
            <action name="createEvent" interpretation="create the event">
              <argument value="msg"/>
            </action>
            <action name="transport.Send" interpretation="Send Confirm Event Request
              message">
              <argument value=" 'ConfirmEventRequest' "/>
              <argument value="msg"/>
              <argument value="transportData"/>
            </action>
          </transition>
          <transition name="transport.Receive">
            <parameter type="CreateEvent" value="msg"/>
            <parameter type="ReceiveRec" value="transportData"/>
            <guard condition="isSupported(msg) &amp;amp;&amp;amp; eventExists(msg)"
interpretation="True if parameters are supported and the event already exists."/>
            <simple/>
            <action name="updateEvent" interpretation="update the event">
              <argument value="msg"/>
            </action>
            <action name="transport.Send" interpretation="Send Confirm Event Request
              message">
              <argument value=" 'ConfirmEventRequest' "/>
              <argument value="msg"/>
              <argument value="transportData"/>
            </action>
          </transition>
          <transition name="transport.Receive">
            <parameter type="CreateEvent" value="msg"/>
            <parameter type="ReceiveRec" value="transportData"/>
            <guard condition="! isSupported(msg)" interpretation="True if parameters are
not
              supported."/>
            <simple/>
            <action name="transport.Send" interpretation="Send Reject Event Request
message">
              <argument value=" 'RejectEventRequest' "/>
              <argument value="msg"/>
              <argument value="transportData"/>
            </action>
          </transition>
          <transition name="transport.Receive">
            <parameter type="UpdateEvent" value="msg"/>
            <parameter type="ReceiveRec" value="transportData"/>
            <guard condition="isSupported(msg) &amp;amp;&amp;amp; eventExists(msg)"
```

```
              interpretation="True if parameters are supported and if the specified event
exists"/>
            <simple/>
            <action name="updateEvent" interpretation="update the event">
              <argument value="msg"/>
            </action>
            <action name="transport.Send" interpretation="Send Confirm Event Request
              message">
              <argument value=" 'ConfirmEventRequest' "/>
              <argument value="msg"/>
              <argument value="transportData"/>
            </action>
          </transition>
          <transition name="transport.Receive">
            <parameter type="UpdateEvent" value="msg"/>
            <parameter type="ReceiveRec" value="transportData"/>
            <guard condition="! isSupported(msg) || ! eventExists(msg)"/>
            <simple/>
            <action name="transport.Send" interpretation="Send Reject Event Request
message">
              <argument value=" 'RejectEventRequest' "/>
              <argument value="msg"/>
              <argument value="transportData"/>
            </action>
          </transition>
          <transition name="transport.Receive">
            <parameter type="CancelEvent" value="msg"/>
            <parameter type="ReceiveRec" value="transportData"/>
            <guard condition="! eventExists(msg)"/>
            <simple/>
            <action name="transport.Send" interpretation="Send Reject Event Request
message">
              <argument value=" 'RejectEventRequest' "/>
              <argument value="msg"/>
              <argument value="transportData"/>
            </action>
          </transition>
          <transition name="transport.Receive">
            <parameter type="CancelEvent" value="msg"/>
            <parameter type="ReceiveRec" value="transportData"/>
            <guard condition="eventExists(msg)"/>
            <simple/>
            <action name="cancelEvent" interpretation="cancel the event">
              <argument value="msg"/>
            </action>
            <action name="transport.Send" interpretation="Send Confirm Event Request
              message">
              <argument value=" 'ConfirmEventRequest' "/>
              <argument value="msg"/>
              <argument value="transportData"/>
            </action>
          </transition>
          <transition name="EventOccurred">
            <guard condition="eventExists()"/>
            <simple/>
            <action name="sendEvent" interpretation="send an event notification"/>
          </transition>
          <transition name="EventError">
            <guard condition="eventExists()"/>
            <simple/>
            <action name="sendRejectEventRequest" interpretation="Send Reject Event
Request message"/>
```

```
          </transition>
        </state>
      </state>
    </state_machine>
  </protocol_behavior>
</service_def>
```

A.3 ACCESS CONTROL

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def xmlns="urn:jaus:jsidl:1.0" name="AccessControl"
  id="urn:jaus:jss:core:AccessControl" version="1.0">
  <description xml:space="preserve">
    The Access Control service offers a basic interface for acquiring preemptable
exclusive control to one or more related services that utilize this function. Once the
exclusive control is established, the related services shall only execute commands
originating from the controlling component. The authority code parameter of this service
is used for preemption and is to be set equal to that of its controlling client. This
service always grants control to the highest authority client that is requesting exclusive
control. Commands from all other clients are ignored unless from a client with higher
authority.
    This service maintains two values, a default value and a current value of a field
called authority code. The default value is the value that the service is pre-configured
with. Access is provided to clients based on the value of their authority code in
comparison to the current value of this service.
  </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="core" id="urn:jaus:jss:core:MessageSet" version="1.0"/>
  </declared_type_set>
  <message_set>
    <input_set>
      <declared_message_def name="RequestControl"
        declared_type_ref="core.commandClass.RequestControl"/>
      <declared_message_def name="ReleaseControl"
        declared_type_ref="core.commandClass.ReleaseControl"/>
      <declared_message_def name="QueryControl"
declared_type_ref="core.queryClass.QueryControl"/>
      <declared_message_def name="QueryAuthority"
declared_type_ref="core.queryClass.QueryAuthority"/>
      <declared_message_def name="SetAuthority"
declared_type_ref="core.commandClass.SetAuthority"/>
      <declared_message_def name="QueryTimeout"
declared_type_ref="core.queryClass.QueryTimeout"/>
    </input_set>
    <output_set>
      <declared_message_def name="ReportControl"
declared_type_ref="core.informClass.ReportEvent"/>
      <declared_message_def name="RejectControl"
declared_type_ref="core.commandClass.RejectEvent"/>
      <declared_message_def name="ConfirmControl"
        declared_type_ref="core.commandClass.ConfirmControl"/>
      <declared_message_def name="ReportAuthority"
        declared_type_ref="core.informClass.ReportAuthority"/>
      <declared_message_def name="ReportTimeout"
declared_type_ref="core.informClass.ReportTimeout"
      />
    </output_set>
  </message_set>
```

```
  <internal_events_set>
    <event_def name="Timeout">
      <description> Occurs when access is not re-acquired periodically </description>
      <header name="Header"/>
      <body name="Body"/>
      <footer name="Footer"/>
    </event_def>
  </internal_events_set>
  <protocol_behavior is_stateless="false">
    <start state_machine_name="events.transport.ReceiveFSM"
      state_name="Receiving.Ready.NotControlled"/>
    <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="NotControlled" interpretation="redefine state
in order to
          extend">
          <state name="NotControlled">
            <entry interpretation="Set the service's current authority value to the
default
              authority value">
              <action name="init"/>
            </entry>
            <transition name="events.transport.Receive">
              <parameter type="RequestControl" value="msg" interpretation="enveloped
request
                control message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <guard condition="! isControlAvailable(msg)" interpretation="True if this
service or
                services related to this service are engaged in other operations that
would
                prevent this service from performing its actions."/>
              <simple/>
              <action name="events.transport.Send" interpretation="Send a confirm
component control
                message with the specified response code to requesting client">
                <argument value=" 'ConfirmControl' "/>
                <argument value=" 'NOT_AVAILABLE' "/>
              </action>
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="RequestControl" value="msg" interpretation="enveloped
request
                control message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <guard condition="! isDefaultAuthorityGreater(msg) &amp;amp;amp;
                isControlAvailable(msg)" interpretation="True if the base
authority
                code of this service is greater than the authority code of the client
                service that triggered the corresponding transition and if this service or
services
                related to this service are not engaged in other operations that would
prevent this
                service from performing its actions"/>
              <simple>
                <end_state state="Receiving.Ready.Controlled"/>
```

```
              </simple>
              <action name="StoreAddress" interpretation="Store the address of the client
that
                sent the message that caused this action to be executed">
                <argument value="transportData"/>
              </action>
              <action name="SetAuthority" interpretation="Set the current authority value
of this
                service to the specified authority">
                <argument value="msg"/>
              </action>
              <action name="ResetTimer" interpretation="Reset the timer"/>
              <action name="events.transport.Send" interpretation="Send a confirm control
                message with the specified response code to requesting client">
                <argument value=" 'ConfirmControl' "/>
                <argument value=" 'CONTROL_ACCEPTED' "/>
              </action>
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="RequestControl" value="msg" interpretation="enveloped
request
                control message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <guard condition="isDefaultAuthorityGreater(msg)" interpretation="True if
the
                default authority code of this service is greater than the authority code
of the
                client service that triggered the corresponding transition"/>
              <simple/>
              <action name="events.transport.Send" interpretation="Send a confirm control
                message with the specified response code to requesting client">
                <argument value=" 'ConfirmControl' "/>
                <argument value=" 'INSUFFICIENT_AUTHORITY' "/>
              </action>
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="ReleaseControl" value="msg" interpretation="enveloped
release
                control message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <simple/>
              <action name="events.transport.Send" interpretation="Send a Reject Control
message to the client requesting release">
                <argument value=" 'RejectControl' "/>
                <argument value=" 'transportData' "/>
                <argument value=" 'CONTROL_RELEASED' "/>
              </action>
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="QueryControl" value="msg" interpretation="enveloped query
control
                message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <simple/>
              <action name="events.transport.Send" interpretation="Send a Report Control
message
                with the                  specified control value">
                <argument value=" 'ReportControl' "/>
                <argument value=" 'transportData' "/>
```

```
              </action>
            </transition>
          </state>
          <state name="Controlled">
            <transition name="events.transport.Receive">
              <parameter type="RequestControl" value="msg" interpretation="enveloped
request
                control message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <guard condition="! isControlAvailable(msg)" interpretation="True if this
service or
                services related to this service are engaged in other operations that
would
                prevent this service from performing its actions."/>
              <simple/>
              <action name="events.transport.Send" interpretation="Send a confirm control
                message with the specified response code to requesting client">
                <argument value=" 'ConfirmControl' "/>
                <argument value=" 'NOT_AVAILABLE' "/>
              </action>
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="SetAuthority" value="msg" interpretation="enveloped set
authority
                message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <guard condition="isAuthorityValid(msg) &amp;amp;&amp;amp; isControllingClient(msg)
                &amp;amp;&amp;amp; isControlAvailable()" interpretation="True if the value of the
                authority                  code received from the client is less than or
equal to
                the current authority value                  of this service, , but
greater than or
                equal to the receiving components default authority,                and
if the
                message that triggered the transition is received from the
  client
                that is in control of this service"/>
              <simple/>
              <action name="SetAuthority" interpretation="Set the current authority value
of this
                service to the specified authority">
                <argument value="msg"/>
              </action>
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="RequestControl" value="msg" interpretation="enveloped
request
                control message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <guard condition="isCurrentAuthorityLess(msg) &amp;amp;&amp;amp; !
                isControllingClient(msg)" interpretation="True if the current authority
value of
                this service is less than the authority code of the client service that
triggered
                the corresponding transition, and if the message that triggered the
transition is
                not received from the client that is in control of this service"/>
              <simple/>
```

```
              <action name="events.transport.Send" interpretation="Send a Reject Control
message to
                current                  controlling client">
                <argument value=" 'RejectControlToController' " interpretation="Send a
Reject Control message with the specified response code to the client that currently has
control."/>
                <argument value=" 'CONTROL_RELEASED' "/>
              </action>
              <action name="StoreAddress" interpretation="Store the address of the client
that
                sent                    the message that caused this action to be
executed">
                <argument value="msg"/>
              </action>
              <action name="SetAuthority" interpretation="Set the current authority value
of this
                service to the specified authority">
                <argument value="msg"/>
              </action>
              <action name="ResetTimer" interpretation="Reset the timer"/>
              <action name="events.transport.Send" interpretation="Send a confirm control
                message with the specified response code to requesting client">
                <argument value=" 'CONTROL_ACCEPTED' "/>
                <argument value="transportData"/>
              </action>
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="RequestControl" value="msg" interpretation="enveloped
request
                control message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <guard condition="! isCurrentAuthorityLess(msg) &amp;&amp; !
                isControllingClient(msg)" interpretation="True if the current authority
value of
                this service is not less than the authority code of the client service
that
                triggered the corresponding transition, and if the message that triggered
                the transition is received from the client that is in control of this
service"/>
              <simple/>
              <action name="events.transport.Send" interpretation="Send a confirm control
                message with the specified response code to requesting client">
                <argument value=" 'ConfirmControl' "/>
                <argument value=" 'INSUFFICIENT_AUTHORITY' "/>
                <argument value="transportData"/>
              </action>
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="RequestControl" value="msg" interpretation="enveloped
request
                control message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <guard condition="! isDefaultAuthorityGreater(msg) &amp;&amp;
                isControllingClient(msg)" interpretation="True if the default authority
code of
                this service is not greater than the authority code of the client service
that
                triggered the corresponding transition, and if the message that triggered
the
                transition is received from the client that is in control of this
service"/>
```

```
              <simple/>
              <action name="ResetTimer" interpretation="Reset the timer"/>
              <action name="SetAuthority" interpretation="Set the current authority value
of this
                service to the specified authority"/>
              <action name="events.transport.Send" interpretation="Send a confirm control
                message with the specified response code to requesting client">
                <argument value=" 'ConfirmControl' "/>
                <argument value=" 'CONTROL_ACCEPTED' "/>
                <argument value="transportData"/>
              </action>
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="QueryControl" value="msg" interpretation="enveloped query
control
                message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <simple/>
              <action name="events.transport.Send" interpretation="Send a Report Control
message
                with the                  specified control value">
                <argument value=" 'ReportControl' "/>
                <argument value="transportData"/>
              </action>
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="ReleaseControl" value="msg" interpretation="enveloped
request
                control message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <guard condition="! isControlAvailable(msg)" interpretation="True if this
service or
                services related to this service are engaged in other operations that
would
                prevent this service from performing its actions."/>
              <simple/>
              <action name="events.transport.Send" interpretation="Send a reject control
                message                 with the specified response code to the client
requesting release">
                <argument value=" 'RejectControl' "/>
                <argument value=" 'NOT_AVAILABLE' "/>
              </action>
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="ReleaseControl" value="msg" interpretation="enveloped
release
                control message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <guard condition="isControllingClient(msg) &amp;amp;&amp;amp;
isControlAvailable(msg)"
                interpretation="True if the message that                triggered the
transition is
                received from the client that is in control of this
service, and if
                this service or services related to this service are not engaged in other
                operations that would prevent this service from performing its actions"/>
              <simple>
                <end_state state="Receiving.Ready.NotControlled"/>
              </simple>
```

```
              <action name="events.transport.Send" interpretation="Send a Reject Control
message with the specified response code to the client requesting release">
                <argument value=" 'RejectControl' "/>
                <argument value="transportData"/>
                <argument value=" 'CONTROL_RELEASED' "/>
              </action>
            </transition>
            <transition name="transport.Receive">
              <parameter type="RequestControl" value="msg"/>
              <parameter type="ReceiveRec" value="transportData"/>
              <guard condition="isDefaultAuthorityGreater(msg) &amp;amp;&amp;amp;
                isControllingClient(msg)" interpretation="True if the default authority
code of this
                service is greater than the authority code of the client service that
triggered the
                corresponding transition, and if the message that triggered the transition
is
                received from the client that is in control of this service"/>
              <simple>
                <end_state state="Receiving.Ready.NotControlled"/>
              </simple>
              <action name="transport.Send" interpretation="Send a Reject Control
message">
                <argument value=" 'RejectControlToController' " interpretation="Send a
Reject Control message with the specified response code to the client that currently has
control"/>
                <argument value="CONTROL_RELEASED"/>
                <argument value="transportData"/>
              </action>
            </transition>
            <transition name="Timeout">
              <guard condition="isControlAvailable(msg)" interpretation="True if this
service or
                services related to this service are not engaged in other operations that
would
                prevent this service from performing its actions."/>
              <simple>
                <end_state state="Receiving.Ready.NotControlled"/>
              </simple>
              <action name="events.transport.Send" interpretation="Send a Reject Control
message to
                controlling client">
                <argument value=" 'RejectControlToController' " interpretation="Send a
Reject Control message with the specified response code to the client that currently has
control"/>
                <argument value=" 'CONTROL_RELEASED' "/>
              </action>
            </transition>
          </state>
          <default_state>
            <transition name="events.transport.Receive">
              <parameter type="QueryAuthority" value="msg" interpretation="enveloped query
                authority message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <simple/>
              <action name="events.transport.Send" interpretation="Send a Report Authority
message
                to                querying client">
                <argument value=" 'ReportAuthority' "/>
                <argument value="transportData"/>
              </action>
```

```
            </transition>
            <transition name="events.transport.Receive">
              <parameter type="QueryTimeout" value="msg" interpretation="enveloped query
                timeout message"/>
              <parameter type="ReceiveRec" value="transportData" interpretation="transport
data"/>
              <simple/>
              <action name="events.transport.Send" interpretation="Send a Report Timeout
message to
                querying client">
                <argument value=" 'ReportTimeout' "/>
                <argument value="transportData"/>
              </action>
            </transition>
          </default_state>
        </state>
      </state>
    </state_machine>
  </protocol_behavior>
</service_def>
```

A.4 MANAGEMENT

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def name="Management" id="urn:jaus:jss:core:Management" version="1.0"
  xmlns="urn:jaus:jsidl:1.0">
  <description>The Management Service provides a state machine for component life-cycle
management
    to help clients understand how the component will react to commands and
queries.</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="core" id="urn:jaus:jss:core:MessageSet" version="1.0"/>
  </declared_type_set>
  <message_set>
    <input_set>
      <declared_message_def name="Shutdown"
declared_type_ref="core.commandClass.Shutdown"/>
      <declared_message_def name="Standby" declared_type_ref="core.commandClass.Standby"/>
      <declared_message_def name="Resume" declared_type_ref="core.commandClass.Resume"/>
      <declared_message_def name="Reset" declared_type_ref="core.commandClass.Reset"/>
      <declared_message_def name="SetEmergency"
declared_type_ref="core.commandClass.SetEmergency"/>
      <declared_message_def name="ClearEmergency"
        declared_type_ref="core.commandClass.ClearEmergency"/>
      <declared_message_def name="QueryStatus"
declared_type_ref="core.queryClass.QueryStatus"/>
    </input_set>
    <output_set>
      <declared_message_def name="ReportStatus"
declared_type_ref="core.queryClass.ReportStatus"/>
    </output_set>
  </message_set>
  <internal_events_set>
    <event_def name="Initialized">
      <description>An internally generated event that marks the completion of the
initialization
        cycle.</description>
```

```
      <header name="Header"/>
      <body name="Body"/>
      <footer name="Footer"/>
    </event_def>
    <event_def name="Failure">
      <description>An internally generated event that marks forces the component to a
failure state.</description>
      <header name="Header"/>
      <body name="Body"/>
      <footer name="Footer"/>
    </event_def>
  </internal_events_set>
  <protocol_behavior>
    <start state_machine_name="accessControl.events.transport.ReceiveFSM"
      state_name="Receiving.Ready.NotControlled.StateA.Init"/>
    <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">
            <state name="StateA" initial_state="Init">
              <state name="Standby"/>
              <state name="Init">
                <entry>
                  <action name="initialize" interpretation="Re-initialize the component's
state
                    machines."/>
                </entry>
                <transition name="Initialized">
                  <simple>
                    <end_state state="Receiving.Ready.NotControlled.StateA.Standby"/>
                  </simple>
                </transition>
              </state>
              <state name="Failure"/>
              <state name="Shutdown"/>
              <state name="Emergency">
                <transition name="accessControl.events.transport.Receive"
interpretation="If
                  multiple emergency conditions exist, then all conditions that have been
set must
                  be specifically cleared before the component can transition out of the
emergency
                  state.">
                  <parameter type="ClearEmergency" value="msg" interpretation="enveloped
clear
                    emergency                      message"/>
                  <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                  <guard condition="isIDStored( transportData )" interpretation="True if
the sender of the message that triggered the transition is in the list of clients that
have set an emergency condition."/>
                  <pop/>
                  <action name="DeleteID" interpretation="Remove the sender of the message
that
                    caused                      this action from the list of clients that
have an
                    emergency condition.">
```

```
                    <argument value="transportData"/>
                  </action>
                </transition>
              </state>
              <default_state>
                <transition name="accessControl.events.transport.Receive">
                  <parameter type="QueryStatus" value="msg" interpretation="enveloped
query status
                    message"/>
                  <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                  <simple/>
                  <action name="accessControl.events.transport.Send" interpretation="Send
a Report
                    Status message with the                    current status">
                    <argument value=" 'ReportStatus' "/>
                    <argument value="transportData"/>
                  </action>
                </transition>
                <transition name="accessControl.events.transport.Receive">
                  <parameter type="SetEmergency" value="msg" interpretation="enveloped set
emergency
                    message"/>
                  <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                  <push>
                    <end_state state="Receiving.Ready.NotControlled.StateA.Emergency"/>
                  </push>
                  <action name="StoreID" interpretation="Add the sender of the message
that caused
                    this action to the list of clients that have an emergency
                    condition.">
                    <argument value="transportData"/>
                  </action>
                </transition>
                <transition name="Failure">
                  <simple>
                    <end_state state="Receiving.Ready.NotControlled.StateA.Failure"/>
                  </simple>
                  <action name="emptyStateStack" interpretation="Empty all stacked
emergency states
                    and delete all associated stored ids"/>
                </transition>
                <transition name="accessControl.events.transport.Receive">
                  <parameter type="accessControl.RequestControl" value="msg"
                    interpretation="enveloped request control message belonging
                    to the inherited access control service's vocabulary"/>
                  <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                  <guard condition="! isStateStandby()" interpretation="True if the
component that
                    received the message that triggered the transition is not in the
                    NotControlled.StateA.Standby state."/>
                  <simple/>
                  <action name="transport.Send" interpretation="Send a confirm component
control
                    message with the specified response code">
                    <argument value=" 'ConfirmControl' "/>
                    <argument value=" 'NOT_AVAILABLE' "/>
                  </action>
                </transition>
              </default_state>
```

```
            </state>
          </state>
          <state name="Controlled" initial_state="StateB">
            <state name="StateB" initial_state="Standby">
              <state name="Standby">
                <transition name="accessControl.events.transport.Receive">
                  <parameter type="Resume" value="msg" interpretation="enveloped resume
message"/>
                  <parameter type="ReceiveRec" value="transportData"
interpretation="enveloped
                    resume message"/>
                  <guard condition="isControllingClient( transportData )"
interpretation="True if
                    the message that triggered the transition is received from
                    the client that is in control of this service"/>
                  <simple>
                    <end_state state="Receiving.Ready.Controlled.StateB.Ready"/>
                  </simple>
                </transition>
              </state>
              <state name="Ready">
                <transition name="accessControl.events.transport.Receive">
                  <parameter type="Standby" value="msg" interpretation="enveloped standby
message"/>
                  <parameter type="ReceiveRec" value="transportData"
interpretation="enveloped
                    resume message"/>
                  <guard condition="isControllingClient( transportData )"
interpretation="True if
                    the message that triggered the transition is received from
                    the client that is in control of this service"/>
                  <simple>
                    <end_state state="Receiving.Ready.Controlled.StateB.Standby"/>
                  </simple>
                </transition>
              </state>
              <state name="Emergency">
                <transition name="accessControl.events.transport.Receive"
interpretation="If
                  multiple emergency conditions exist, then all conditions that have been
set must
                  be specifically cleared before the component can transition out of the
emergency
                  state.">
                  <parameter type="ClearEmergency" value="msg" interpretation="enveloped
clear
                    emergency message"/>
                  <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                  <guard condition="isIDStored( transportData )" interpretation="True if
the sender
                    of the message that triggered the transition is in the list
                    of clients that have set an emergency condition."/>
                  <pop/>
                  <action name="DeleteID" interpretation="Remove the sender of the message
that
                    caused this action from the list of clients that have an
                    emergency condition.">
                    <argument value="transportData"/>
                  </action>
                </transition>
                <transition name="accessControl.events.transport.Receive">
```

```
                    <parameter type="accessControl.RequestControl" value="msg"
                      interpretation="enveloped request control message"/>
                    <simple/>
                    <action name="transport.Send" interpretation="Send a confirm component
control
                      message with the specified response code">
                      <argument value=" 'ConfirmControl' "/>
                      <argument value=" 'NOT_AVAILABLE' "/>
                    </action>
                  </transition>
                  <transition name="accessControl.events.transport.Receive">
                    <parameter type="accessControl.ReleaseControl" value="msg"
                      interpretation="enveloped request control message"/>
                    <simple/>
                    <action name="transport.Send" interpretation="Send a reject control
                      message with the specified response code">
                      <argument value=" 'RejectControl' "/>
                      <argument value=" 'NOT_AVAILABLE' "/>
                    </action>
                  </transition>
                  <transition name="accessControl.Timeout">
                    <simple/>
                    <action name="accessControl.resetTimer" interpretation="reset the timer
beloning
                      to the inherited access control service"/>
                  </transition>
                  <transition name="accessControl.events.transport.Receive">
                    <parameter type="Reset" value="msg" interpretation="enveloped reset
message"/>
                    <simple/>
                  </transition>
                </state>
                <default_state>
                  <transition name="accessControl.events.transport.Receive">
                    <parameter type="QueryStatus" value="msg" interpretation="enveloped
query status
                      message"/>
                    <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                    <simple/>
                    <action name="transport.Send" interpretation="Send a Report Status
message with
                      the current status">
                      <argument value=" 'ReportStatus' "/>
                      <argument value="transportData"/>
                    </action>
                  </transition>
                  <transition name="accessControl.events.transport.Receive">
                    <parameter type="SetEmergency" value="msg" interpretation="enveloped set
emergency
                      message"/>
                    <parameter type="ReceiveRec" value="transportData"
interpretation="transport data"/>
                    <push>
                      <end_state state="Receiving.Ready.Controlled.StateB.Emergency"/>
                    </push>
                    <action name="StoreID" interpretation="Add the sender of the message
that caused
                      this action to the list of clients that have an emergency
                      condition.">
                      <argument value="transportData"/>
                    </action>
```

```
                </transition>
                <transition name="Failure">
                  <simple>
                    <end_state state="Receiving.Ready.NotControlled.StateA.Failure"/>
                  </simple>
                  <action name="transport.Send" interpretation="Send a Reject Control
message to the
                    client that is currently controlling this service.">
                    <argument value=" 'RejectControl' "/>
                    <argument value="CONTROL_RELEASED"/>
                  </action>
                  <action name="emptyStateStack" interpretation="Empty all stacked
emergency states
                    and delete all associated stored ids"/>
                </transition>
                <transition name="accessControl.events.transport.Receive">
                  <parameter type="Reset" value="msg" interpretation="enveloped reset
message"/>
                  <parameter type="ReceiveRec" value="transportData"/>
                  <guard condition="isControllingClient(  transportData )"
interpretation="True if
                    the message that triggered the transition is received from
                    the client that is in control of this service"/>
                  <simple>
                    <end_state state="Receiving.Ready.NotControlled.StateA.Init"/>
                  </simple>
                  <action name="transport.Send" interpretation="Send a Reject Control
message to the
                    client that is currently controlling this service.">
                    <argument value=" 'RejectControl' "/>
                    <argument value="transportData"/>
                    <argument value="CONTROL_RELEASED"/>
                  </action>
                </transition>
                <transition name="accessControl.events.transport.Receive">
                  <parameter type="Shutdown" value="msg" interpretation="enveloped
shutdown message"/>
                  <parameter type="ReceiveRec" value="transportData"/>
                  <guard condition="isControllingClient(  transportData )"
interpretation="True if
                    the message that triggered the transition is received from
                    the client that is in control of this service"/>
                  <simple>
                    <end_state state="Receiving.Ready.NotControlled.StateA.Shutdown"/>
                  </simple>
                  <action name="transport.Send" interpretation="Send a Reject Control
message to the
                    client that is currently controlling this service.">
                    <argument value=" 'RejectControl' "/>
                    <argument value="transportData"/>
                    <argument value="CONTROL_RELEASED"/>
                  </action>
                  <action name="emptyStateStack" interpretation="Empty all stacked
emergency states
                    and delete all associated stored ids"/>
                </transition>
              </default_state>
            </state>
          </state>
        </state>
      </state>
    </state_machine>
```

```
  </protocol_behavior>
</service_def>
```

A.5 TIME

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def name="Time" id="urn:jaus:jss:core:Time" version="1.0"
xmlns="urn:jaus:jsidl:1.0">
  <description>The Time Service allows clients to query and set the system time for the
component.
    Note that exclusive control is required to set the time, but is not required to query
it. </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="core" id="urn:jaus:jss:core:MessageSet" version="1.0"/>
  </declared_type_set>
  <message_set>
    <input_set>
      <declared_message_def name="SetTime" declared_type_ref="core.commandClass.SetTime"/>
      <declared_message_def name="QueryTime"
declared_type_ref="core.queryClass.QueryTime"/>
    </input_set>
    <output_set>
      <declared_message_def name="ReportTime"
declared_type_ref="core.informClass.ReportTime"/>
    </output_set>
  </message_set>
  <internal_events_set/>
  <protocol_behavior>
    <start state_machine_name="accessControl.events.transport.ReceiveFSM"
      state_name="Receiving.Ready.NotControlled"/>
    <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" interpretation="redefine state in order to
   extend">
            <default_state>
              <transition name="accessControl.events.transport.Receive">
                <parameter type="QueryTime" value="msg" interpretation="enveloped query
time
                  message"/>
                <parameter type="TransportRec" value="transportData"
interpretation="transport data"/>
                <simple/>
                <action name="accessControl.events.transport.Send" interpretation="Send a
Report
                  Time message with the                    current time">
                  <argument value=" 'ReportTime' "/>
                  <argument value=" 'currentTime' "/>
                  <argument value=" 'transportData' "/>
                </action>
              </transition>
```

```
            </default_state>
          </state>
          <state name="Controlled" interpretation="redefine state in order to
extend">
            <default_state>
              <transition name="accessControl.events.transport.Receive">
                <parameter type="QueryTime" value="msg" interpretation="enveloped query
time
                  message"/>
                <parameter type="TransportRec" value="transportData"
interpretation="transport data"/>
                <simple/>
                <action name="accessControl.events.transport.Send" interpretation="Send a
Report
                  Time message with the                    current time">
                  <argument value=" 'ReportTime' "/>
                  <argument value=" 'currentTime' "/>
                  <argument value=" 'transportData' "/>
                </action>
              </transition>
              <transition name="accessControl.events.transport.Receive">
                <parameter type="SetTime" value="msg" interpretation="enveloped set time
message"/>
                <parameter type="TransportRec" value="transportData"
interpretation="transport data"/>
                <guard condition="isControllingClient(transportData)" interpretation="True
if the message that
                  triggered the transition is received from the client that is in control
of this
                  service"/>
                <simple/>
                <action name="setTime" interpretation="Set the time of this component to
the
                  specified                    time">
                  <argument value="msg"/>
                </action>
              </transition>
            </default_state>
          </state>
        </state>
      </state>
    </state_machine>
  </protocol_behavior>
</service_def>
```

A.6 LIVENESS

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def xmlns="urn:jaus:jsidl:1.0" name="Liveness" id="urn:jaus:jss:core:Liveness"
  version="1.0">
  <description>This service provides a means to maintain connection liveness between
communicating
    components.</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="core" id="urn:jaus:jss:core:MessageSet" version="1.0"/>
  </declared_type_set>
  <message_set>
    <input_set>
```

```
      <declared_message_def name="QueryHeartbeatPulse"
        declared_type_ref="core.queryClass.QueryHeartbeatPulse"/>
    </input_set>
    <output_set>
      <declared_message_def name="ReportHeartbeatPulse"
        declared_type_ref="core.informClass.ReportHeartbeatPulse"/>
    </output_set>
  </message_set>
  <internal_events_set/>
  <protocol_behavior is_stateless="false">
    <start state_machine_name="events.transport.ReceiveFSM" state_name="Receiving.Ready"/>
    <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" interpretation="redefine state in order to extend">
          <default_state>
            <transition name="transport.Receive">
              <parameter type="QueryHeartbeatPulse" value="msg"/>
              <parameter type="ReceiveRec" value="transportData"/>
              <simple/>
              <action name="transport.Send" interpretation="Send a Report Heartbeat pulse
                to querying client">
                <argument value=" 'ReportHeartbeatPulse' "/>
                <argument value="transportData"/>
              </action>
            </transition>
          </default_state>
        </state>
      </state>
    </state_machine>
  </protocol_behavior>
</service_def>
```

A.7 DISCOVERY

```
<?xml version="1.0" encoding="UTF-8"?>
<service_def xmlns="urn:jaus:jsidl:1.0" name="Discovery" id="urn:jaus:jss:core:Discovery"
  version="1.0">
  <description>The process of discovery is conducted at both the node level and the
subsystem level.
    This service supports the discovery of both legacy components defined in the JAUS
Reference
    Architecture versions 3.2+, and new components. The Component IDs of legacy components
were
    fixed at specification time (Primitive Driver = 33 for example) and could contain only
one
    service beyond the core service support. New components may use any component ID that
is outside
    the range of IDs that have been allocated to legacy component definitions. New
components can
    also contain two or more services beyond the core service support. </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="core" id="urn:jaus:jss:core:MessageSet" version="1.0"/>
  </declared_type_set>
  <message_set>
```

```
    <input_set>
      <declared_message_def name="QueryIdentification"
        declared_type_ref="core.queryClass.QueryIdentification"/>
      <declared_message_def name="QueryConfiguration"
        declared_type_ref="core.queryClass.QueryConfiguration"/>
      <declared_message_def name="QuerySubsystemList"
        declared_type_ref="core.queryClass.QuerySubsystemList"/>
      <declared_message_def name="QueryServices"
declared_type_ref="core.queryClass.QueryServices"/>
      <declared_message_def name="RegisterServices"
        declared_type_ref="core.commandClass.RegisterServices"/>
    </input_set>
    <output_set>
      <declared_message_def name="ReportIdentification"
        declared_type_ref="core.informClass.ReportIdentification"/>
      <declared_message_def name="ReportConfiguration"
        declared_type_ref="core.informClass.ReportConfiguration"/>
      <declared_message_def name="ReportSubsystemList"
        declared_type_ref="core.informClass.ReportSubsystemList"/>
      <declared_message_def name="ReportServices"
        declared_type_ref="core.informClass.ReportServices"/>
    </output_set>
  </message_set>
  <internal_events_set/>
  <protocol_behavior is_stateless="false">
    <start state_machine_name="events.transport.ReceiveFSM" state_name="Receiving.Ready"/>
    <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">
          <default_state>
            <transition name="transport.Receive">
              <parameter type="QueryIdentification" value="msg"/>
              <parameter type="ReceiveRec" value="transportData"/>
              <simple/>
              <action name="transport.Send" interpretation="Send a Report Identification
                message               to querying client">
                <argument value=" 'ReportIdentification' "/>
                <argument value="transportData"/>
              </action>
            </transition>
            <transition name="transport.Receive">
              <parameter type="QueryConfiguration" value="msg"/>
              <parameter type="ReceiveRec" value="transportData"/>
              <simple/>
              <action name="transport.Send" interpretation="Send a Report Configuration
                message               to querying client">
                <argument value=" 'ReportConfiguration' "/>
                <argument value="transportData"/>
              </action>
            </transition>
            <transition name="transport.Receive">
              <parameter type="QuerySubsystemList" value="msg"/>
              <parameter type="ReceiveRec" value="transportData"/>
              <simple/>
              <action name="transport.Send" interpretation="Send a Report Subsystem List
                message               to querying client">
                <argument value="ReportSubsystemList"/>
                <argument value="transportData"/>
```

```
            </action>
          </transition>
          <transition name="transport.Receive">
            <parameter type="QueryServices" value="msg"/>
            <parameter type="ReceiveRec" value="transportData"/>
            <simple/>
            <action name="transport.Send" interpretation="Send a Report Services message
              to querying client">
              <argument value=" 'ReportServices' "/>
              <argument value="transportData"/>
            </action>
          </transition>
          <transition name="transport.Receive">
            <parameter type="RegisterServices" value="msg"/>
            <simple/>
            <action name="PublishServices" interpretation="Add the component that sent
the message
              to the list of reported services"/>
          </transition>
        </default_state>
      </state>
    </state>
  </state_machine>
 </protocol_behavior>
</service_def>
```

APPENDIX B - XML FOR DECLARED TYPE SETS

B.1 CORE SUBGROUP MESSAGE DEFINITIONS

```
<?xml version="1.0" encoding="UTF-8"?>
<declared_type_set xmlns="urn:jaus:jsidl:0.11" name="MessageSet"
  id="urn:jaus:jss:core:MessageSet" version="1.0">
  <declared_type_set_ref name="commandClass"
id="urn:jaus:jss:core:MessageSet:CommandClass"
    version="1.0"/>
  <declared_type_set_ref name="informClass" id="urn:jaus:jss:core:MessageSet:InformClass"
    version="1.0"/>
  <declared_type_set_ref name="queryClass" id="urn:jaus:jss:core: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:0.11" name="CommandClass"
  id="urn:jaus:jss:core:MessageSet:CommandClass" version="1.0">
  <declared_type_set_ref name="basicTypes" id="urn:jaus:jss:core:MessageSet:BasicTypes"
    version="1.0"/>
  <message_def name="SetAuthority" message_id="0001" is_command="true">
    <description xml:space="preserve">
      This message shall set the command authority of the receiving component.  The
authority bits range in value from 0 to 255 with 255 being the highest.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="authorityRec" optional="false">
        <declared_fixed_field name="AuthorityCode"
declared_type_ref="basicTypes.AuthorityCode"
          optional="false"/>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="Shutdown" message_id="0002" is_command="true">
    <description xml:space="preserve">
      This message is used to cause the receiving component to free all of the resources
allocated to its process by the system and then to shutdown.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="Standby" message_id="0003" is_command="true">
    <description xml:space="preserve">
      This message is used to transition the receiving component to the standby state.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="Resume" message_id="0004" is_command="true">
    <description xml:space="preserve">
      This message is used to transition the receiving component to the ready state.
    </description>
```

```
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="Reset" message_id="0005" is_command="true">
    <description xml:space="preserve">
      This message is used to cause the receiving component to reinitialize.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="SetEmergency" message_id="0006" is_command="true">
    <description xml:space="preserve">
      This message is used to alert the component to a safety critical situation. The
component that sends the emergency command shall set the message priority to the safety
critical priority range as defined by the Transport. Receive of the emergency command
shall result in the component transitioning into the emergency state.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="SetEmergencyRec" optional="false">
        <declared_fixed_field name="EmergencyCode"
declared_type_ref="basicTypes.EmergencyCode"
          optional="false"/>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ClearEmergency" message_id="0007" is_command="true">
    <description xml:space="preserve">
      This message shall notify the receiving component that the current emergency
condition is to be reset and that the component shall transition back to the Ready or
Standby state, provided that all emergency conditions have been cleared.
      JAUS currently defines only one emergency condition, the “Stop” condition.  Future
versions of this document could define other emergency conditions.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="ClearEmergencyRec" optional="false">
        <declared_fixed_field name="EmergencyCode"
declared_type_ref="basicTypes.EmergencyCode"
          optional="false"/>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="RequestControl" message_id="000D" is_command="true">
    <description xml:space="preserve">
      This message is used to request interruptible control of the receiving component.
Once control is established, the receiving component shall only execute commands from the
sending component.  The authority code parameter is to be set equal to that of the sending
component.  The receiving component must always accept the control of the highest
authority component that is requesting uninterruptible control.  Commands from all other
components are ignored unless from a component with higher authority.
```

```
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="RequestControlRec" optional="false">
        <declared_fixed_field name="AuthorityCode"
declared_type_ref="basicTypes.AuthorityCode"
          optional="false"/>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ReleaseControl" message_id="000E" is_command="true">
    <description xml:space="preserve">
      This message is used to relinquish uninterruptible control of the receiving
component.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ConfirmControl" message_id="000F" is_command="false">
    <description xml:space="preserve">
      The Confirm Control message is used to notify a component that it accepts control
from that component (or not).  When control has been granted, response code of 0, the
component under control will only execute messages from the controlling component until
control is released or interrupted.  When the requesting component has lower authority
than the current controlling entity, the response will be 2.  For components not
supporting interruptible control, or if the component is engaged in other operations that
would prevent this service from performing its actions to grant control, the response code
value of 1 can be used.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="ConfirmControlRec" optional="false">
        <fixed_field name="ResponseCode" field_type="unsigned byte" field_units="one"
          optional="false">
          <value_set offset_to_lower_limit="false">
            <value_enum enum_index="0" enum_const="CONTROL_ACCEPTED"/>
            <value_enum enum_index="1" enum_const="NOT_AVAILABLE"/>
            <value_enum enum_index="2" enum_const="INSUFFICIENT_AUTHORITY"/>
          </value_set>
        </fixed_field>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="RejectControl" message_id="0010" is_command="false">
    <description xml:space="preserve">
      The Reject Control message is used to notify a component that control has been
released (response code = 0), or a request to release control could not be processed
(response code = 1).
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="RejectControlRec" optional="false">
        <fixed_field name="ResponseCode" field_type="unsigned byte" field_units="one"
          optional="false">
          <value_set offset_to_lower_limit="false">
```

```
            <value_enum enum_index="0" enum_const="CONTROL_RELEASED"/>
            <value_enum enum_index="1" enum_const="NOT_AVAILABLE"/>
          </value_set>
        </fixed_field>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="SetTime" message_id="0011" is_command="true">
    <description xml:space="preserve">
      Time is configured within a JAUS system using the following message.  Accuracy of
the time may be dependent on latencies in the transmission of the message.  Proper systems
engineering procedures should be used to insure the accuracy of the time messages are
within the system tolerance.  All times are in Coordinated Universal Time (UTC).
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <declared_record name="TimeRec" declared_type_ref="basicTypes.TimeRec"
optional="false"/>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="CreateEvent" message_id="01F0" is_command="true">
    <description xml:space="preserve">
      This message is used to set up an event.   Field 1 is a local request ID that the
event provider returns in the Confirm or Reject message.  Field 2 is the Event Type, which
allows the requester  to specify the type of event – Periodic specifies that the event is
a service connection request and should not be queued, in which case field 3 (Requested
periodic rate) must be set to a non-zero value.  Event type of Every Change specifies that
the corresponding Report message should be sent every time the data associated with that
message changes.
      Field 4 contains the size of the Query message that is to specify the contents of
the Report.  Field 5 contains the Query message (including its two byte header).
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="CreateEventRec" optional="false">
        <fixed_field name="RequestID" field_type="unsigned byte" field_units="one"
optional="false"
          interpretation="Local request ID for use in confirm event"/>
        <fixed_field name="EventType" field_type="unsigned byte" field_units="one"
optional="false"
          interpretation="Type of event">
          <value_set offset_to_lower_limit="false">
            <value_enum enum_index="0" enum_const="Periodic (SC)"/>
            <value_enum enum_index="1" enum_const="Every change"/>
          </value_set>
        </fixed_field>
        <fixed_field name="RequestedPeriodicRate" field_type="unsigned short integer"
          field_units="hertz" optional="false" interpretation="Must be specified for
periodic event,
          and set to 0 for every change ">
          <scale_range integer_function="round" real_lower_limit="0"
real_upper_limit="1092"/>
        </fixed_field>
        <variable_length_field name="QueryMessage" field_format="JAUS MESSAGE"
optional="false"
          interpretation="The JAUS Query message to be used by the receiving
component to
          generate the report message(s)">
```

```
          <count_field min_count="2" field_type_unsigned="unsigned integer"/>
        </variable_length_field>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="UpdateEvent" is_command="true" message_id="01F1">
    <description xml:space="preserve">
      The Update Event message allows the requester to request a rate or change.  The
format is the same as the Create Event, only with the addition of Event ID field to
specify the event that needs updating.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="UpdateEventRec" optional="false">
        <fixed_field name="RequestID" field_type="unsigned byte" field_units="one"
optional="false"
          interpretation="Local request ID for use in confirm event"/>
        <fixed_field name="EventType" field_type="unsigned byte" field_units="one"
optional="false"
          interpretation="Type of event">
          <value_set offset_to_lower_limit="false">
            <value_enum enum_index="0" enum_const="Periodic (SC)"/>
            <value_enum enum_index="1" enum_const="Every change"/>
          </value_set>
        </fixed_field>
        <fixed_field name="RequestedPeriodicRate" field_type="unsigned short integer"
          field_units="hertz" optional="false" interpretation="Must be specified for
periodic event,
          and set to 0 for every change ">
          <scale_range integer_function="round" real_lower_limit="0"
real_upper_limit="1092"/>
        </fixed_field>
        <fixed_field name="EventID" field_type="unsigned byte" field_units="one"
optional="false"
          interpretation="Unique identifier of existing event to update"/>
        <variable_length_field name="QueryMessage" field_format="JAUS MESSAGE"
optional="false"
          interpretation="The JAUS Query message to be used by the receiving
component to
          generate the report message(s)">
          <count_field min_count="2" field_type_unsigned="unsigned integer"/>
        </variable_length_field>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="CancelEvent" is_command="true" message_id="01F2">
    <description xml:space="preserve">
      The Cancel Event message is used by the requester to cancel and/or request deletion
of the specified event. </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="CancelEventRec" optional="false">
        <fixed_field name="RequestID" field_type="unsigned byte" field_units="one"
          interpretation="Local           request id for use in confirm/reject message"
          optional="false"/>
        <fixed_field name="EventID" field_type="unsigned byte" field_units="one"
          interpretation="unique id           of the event to be removed"
optional="false"/>
```

```
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ConfirmEventRequest" is_command="false" message_id="01F3">
    <description xml:space="preserve">
      The Confirm Event message is used to confirm an Event has been created/updated/or
cancelled.  Field 1 represents the Request ID from the Create, Update, or Cancel message
that initiated this message.  The Request ID's scope is local to the requesting client
only.  Field 2, Event ID, is a globally unique ID that is established for the event. Field
3 is used to specify the closest rate that the service can provide if it cannot match the
requested rate.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="ConfirmEventRequestRec" optional="false">
        <fixed_field name="RequestID" field_type="unsigned byte" field_units="one"
optional="false"
          interpretation="ID of the event maintenance request (Create, Update, or
Cancel)"/>
        <fixed_field name="EventID" field_type="unsigned byte" field_units="one"
interpretation="The
          identifier of the specific event " optional="false"/>
        <fixed_field name="ConfirmedPeriodicRate" field_type="unsigned short integer"
          field_units="hertz" optional="false" interpretation="Requested rate or closest
to
          requested rate">
          <scale_range integer_function="round" real_lower_limit="0"
real_upper_limit="1092"/>
        </fixed_field>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="RejectEventRequest" is_command="false" message_id="01F4">
    <description xml:space="preserve">
      The Reject Event Request message is used to reject an Event creation, update or
cancellation. Field 2 represents the Request ID from the Create, Update, or Cancel message
that initiated this message. The Request ID's scope is local to the requesting client
only. Field 4, Event ID, is a globally unique ID that is established for the event.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="RejectEventRequestRec" optional="false">
        <presence_vector field_type_unsigned="unsigned byte"/>
        <fixed_field name="RequestID" field_type="unsigned byte" field_units="one"
optional="false"
          interpretation="ID of the event maintenance request (create, update, or
cancel)"/>
        <fixed_field name="ResponseCode" field_type="unsigned byte" field_units="one"
          optional="true">
          <value_set offset_to_lower_limit="false">
            <value_enum enum_index="1" enum_const="periodic events not supported"/>
            <value_enum enum_index="2" enum_const="change based events not supported"/>
            <value_enum enum_index="3" enum_const="connection refused"/>
            <value_enum enum_index="4" enum_const="invalid event setup"/>
            <value_enum enum_index="5" enum_const="message not supported"/>
            <value_enum enum_index="6" enum_const="error, invalid event ID for update
event
              request"/>
```

```
          </value_set>
        </fixed_field>
        <fixed_length_string name="ErrorMessage" string_length="80" optional="true"
          interpretation="String for additional information"/>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="RegisterServices" is_command="true" message_id="0B00">
    <description xml:space="preserve">
      This message allows a component to register its capabilities with a Discovery
service.  If a component ID is specified in the JAUS Reference Architecture version 3.2+,
it may report only one service beyond the core message support, and this service must be
equal to the component ID.  If a component ID is not listed in the RA, it may report any
number of services.  For example, a component with ID 33 must provide only service 33.
The exception to this rule is component ID 1 (the Node Manager) which may provide any
number of services in addition to core message support. Note that this restriction may be
removed in future versions of this Standard.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="RegisterServicesBody">
      <list name="ServiceList" optional="false">
        <count_field field_type_unsigned="unsigned byte" interpretation="Number of
services
          registered for this component"/>
        <record name="ServiceRec" optional="false">
          <variable_length_string name="URI" optional="false" interpretation="Service
URI">
            <count_field field_type_unsigned="unsigned byte"/>
          </variable_length_string>
          <fixed_field name="MajorVersionNumber" field_type="unsigned byte"
field_units="one"
            optional="false" interpretation="Major version number of the service"/>
          <fixed_field name="MinorVersionNumber" field_type="unsigned byte"
field_units="one"
            optional="false" interpretation="Minor version number of the service"/>
        </record>
      </list>
    </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:0.11" name="QueryClass"
  id="urn:jaus:jss:core:MessageSet:QueryClass" version="1.0">
  <declared_type_set_ref name="basicTypes" id="urn:jaus:jss:core:MessageSet:BasicTypes"
    version="1.0"/>
  <message_def name="QueryAuthority" message_id="2001" is_command="false">
    <description xml:space="preserve">
      This message is used by clients to query the current value of the authority code of
this service.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryStatus" message_id="2002" is_command="false">
```

```
    <description xml:space="preserve">
      This message is used by clients of this service to query the state of this service.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryTimeout" message_id="2003" is_command="false">
    <description xml:space="preserve">
      This message is used by clients of this service to query the timeout period of this
service.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryTime" message_id="2011" is_command="false">
    <description xml:space="preserve">
      This message is used by cleints of this service to query the current time of this
service.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="QueryTimeRec" optional="false">
        <fixed_field name="PresenceVector" field_type="unsigned byte" field_units="one"
          optional="false" interpretation="See corresponding report time message for
presence vector
          mapping"/>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryControl" message_id="200D" is_command="false">
    <description xml:space="preserve">
      This message is used by clients to query the current control state of this service.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryEvents" message_id="21F0" is_command="false">
    <description xml:space="preserve">
      The Query Events message is used to request detail on events. Queries can be made by
message ID, event type or Event ID.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <variant name="QueryEventsVar" optional="false">
        <vtag_field field_type_unsigned="unsigned byte" min_count="0" max_count="3"/>
        <record name="MessageIDRec" optional="false">
          <fixed_field name="MessageCode" field_type="unsigned short integer"
field_units="one"
            optional="false" interpretation="Query Message ID of the Event message that
the
            receiving component is inquiring about."/>
        </record>
        <record name="EventTypeRec" optional="false">
```

```
          <fixed_field name="EventType" field_type="unsigned byte" field_units="one"
            optional="false" interpretation="Type of event">
            <value_set offset_to_lower_limit="false">
              <value_enum enum_index="0" enum_const="Periodic (SC)"/>
              <value_enum enum_index="1" enum_const="Every change"/>
            </value_set>
          </fixed_field>
        </record>
        <record name="EventIDRec" optional="false">
          <fixed_field name="EventID" field_type="unsigned byte" field_units="one"
optional="false"
            interpretation="Event ID returned by Confirm Event for details on specific
event."/>
        </record>
        <record name="AllEventsRec" optional="false">
          <fixed_field name="AllEvents" field_type="unsigned byte" field_units="one"
            optional="false" interpretation="All events should be reported.">
            <value_set offset_to_lower_limit="false">
              <value_range lower_limit="0" lower_limit_type="inclusive" upper_limit="0"
                upper_limit_type="inclusive"/>
            </value_set>
          </fixed_field>
        </record>
      </variant>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryHeartbeatPulse" message_id="2202" is_command="false">
    <description xml:space="preserve">
      This message shall be used to query for a heartbeat pulse.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryIdentification" message_id="2B00" is_command="false">
    <description xml:space="preserve">
      This message shall request the identification summary of a Subsystem, Node, or
Component.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="QueryIdentificationRec" optional="false">
        <fixed_field name="QueryType" field_type="unsigned byte" field_units="one"
optional="false">
          <value_set offset_to_lower_limit="false">
            <value_enum enum_index="0" enum_const="Reserved"/>
            <value_enum enum_index="1" enum_const="System Identification"/>
            <value_enum enum_index="2" enum_const="Subsystem Identification"/>
            <value_enum enum_index="3" enum_const="Node Identification"/>
            <value_enum enum_index="4" enum_const="Component Identification"/>
            <value_range lower_limit="5" upper_limit="255" lower_limit_type="inclusive"
              upper_limit_type="inclusive" interpretation="Reserved"/>
          </value_set>
        </fixed_field>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryConfiguration" message_id="2B01" is_command="false">
```

```
    <description xml:space="preserve">
      This message shall request the configuration summary of a subsystem or node. For
example, to get the complete configuration of a subsystem, field 1 shall be set to 2.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="QueryConfigurationRec" optional="false">
        <fixed_field name="QueryType" field_type="unsigned byte" field_units="one"
optional="false">
          <value_set offset_to_lower_limit="false">
            <value_enum enum_index="0" enum_const="Reserved"/>
            <value_enum enum_index="1" enum_const="Reserved"/>
            <value_enum enum_index="2" enum_const="Subsystem Identification"/>
            <value_enum enum_index="3" enum_const="Node Identification"/>
            <value_range lower_limit="4" upper_limit="255" lower_limit_type="inclusive"
              upper_limit_type="inclusive" interpretation="Reserved"/>
          </value_set>
        </fixed_field>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QuerySubsystemList" message_id="2B02" is_command="false">
    <description xml:space="preserve">
      This message shall request the Report Subsystem List message.  There are no data
fields associated with this message.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="QueryServices" message_id="2B03" is_command="false">
    <description xml:space="preserve">
      This message allows a component to request the service information of an entire
subsystem or node, or a single component. The corresponding Report Services message will
respond with service information only for new component implementations. It will not
report service information for legacy component implementations.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <list name="NodeList" optional="false" interpretation="# of Nodes reported. For a
single Node
        Report this field shall be 1">
        <count_field field_type_unsigned="unsigned byte" min_count="1" max_count="255"/>
        <sequence name="NodeSeq" optional="false">
          <record name="NodeRec" optional="false">
            <fixed_field name="NodeID" field_type="unsigned byte" field_units="one"
optional="false"
              interpretation="Use 255 if service information from all nodes in the
subsystem is
              required">
              <value_set offset_to_lower_limit="false">
                <value_enum enum_index="0" enum_const="Reserved"/>
                <value_range lower_limit="1" upper_limit="254"
lower_limit_type="inclusive"
                  upper_limit_type="inclusive" interpretation="Valid Node IDs"/>
                <value_enum enum_index="255" enum_const="All nodes in the subsystem"/>
              </value_set>
            </fixed_field>
```

```
          </record>
          <list name="ComponentList" optional="false">
            <count_field field_type_unsigned="unsigned byte" min_count="1"
max_count="255"/>
            <record name="ComponentRec" optional="false">
              <fixed_field name="ComponentID" field_type="unsigned byte" field_units="one"
                optional="false" interpretation="Use 255 if service information from
components in
                the node are                 required">
                <value_set offset_to_lower_limit="false">
                  <value_enum enum_index="0" enum_const="Reserved"/>
                  <value_range lower_limit="1" upper_limit="254"
lower_limit_type="inclusive"
                    upper_limit_type="inclusive" interpretation="Valid Component IDs"/>
                  <value_enum enum_index="255" enum_const="All components in the
subsystem"/>
                </value_set>
              </fixed_field>
            </record>
          </list>
        </sequence>
      </list>
    </body>
    <footer name="Footer"/>
  </message_def>
</declared_type_set>
```

B.4 INFORM CLASS

```
<?xml version="1.0" encoding="UTF-8"?>
<declared_type_set xmlns="urn:jaus:jsidl:0.11" name="InformClass"
  id="urn:jaus:jss:core:MessageSet:InformClass" version="1.0">
  <declared_type_set_ref name="basicTypes" id="urn:jaus:jss:core:MessageSet:BasicTypes"
    version="1.0"/>
  <message_def name="ReportAuthority" message_id="4001" is_command="false">
    <description xml:space="preserve">
      This message is used to report the current command authority.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="ReportAuthorityRec" optional="false">
        <declared_fixed_field name="AuthorityCode"
declared_type_ref="basicTypes.AuthorityCode"
          optional="false"/>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ReportStatus" message_id="4002" is_command="false">
    <description xml:space="preserve">
      This message is used to report the current status of the sender of the message.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="ReportStatusRec" optional="false">
        <fixed_field name="Status" field_type="unsigned byte" field_units="one"
optional="false">
          <value_set offset_to_lower_limit="false">
            <value_enum enum_index="0" enum_const="INITIALIZE"/>
            <value_enum enum_index="1" enum_const="READY"/>
```

```
            <value_enum enum_index="2" enum_const="STANDBY"/>
            <value_enum enum_index="3" enum_const="SHUTDOWN"/>
            <value_enum enum_index="4" enum_const="FAILURE"/>
            <value_enum enum_index="5" enum_const="EMERGENCY"/>
          </value_set>
        </fixed_field>
        <fixed_field name="Reserved" field_type="unsigned integer" field_units="one"
          optional="false" interpretation="This field is reserved for compatibility with
previous
          versions of the Standard."/>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ReportTimeout" message_id="4003" is_command="false">
    <description xml:space="preserve">
      This message is used to report the timeout period of this message.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="ReportTimoutRec" optional="false">
        <fixed_field name="Timeout" field_type="unsigned byte" field_units="second"
optional="false">
          <value_set offset_to_lower_limit="false">
            <value_enum enum_index="0" enum_const="TIMEOUT_FEATURE _DISABLED"/>
          </value_set>
        </fixed_field>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ReportTime" message_id="4011" is_command="false">
    <description xml:space="preserve">
      This message is used to report the current time of this service to querying clients.
All times are in Coordinated Universal Time (UTC).
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <declared_record name="TimeRec" declared_type_ref="basicTypes.TimeRec"
optional="false"/>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ReportControl" message_id="400D" is_command="false">
    <description xml:space="preserve">
      This message is used to report the current state of control of this service.  If the
serivce is in the Controlled state, this message reports the ID of the controlling
component. The ID fields shall be set to zero (0) if this service is in the NotControlled
state..
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="ReportControlRec" optional="false">
        <fixed_field name="SubsystemID" field_type="unsigned short integer"
          field_units="one" interpretation="Subsystem ID" optional="false"/>
        <fixed_field name="NodeID" field_type="unsigned byte" field_units="one"
          interpretation="Node ID" optional="false"/>
        <fixed_field name="ComponentID" field_type="unsigned byte" field_units="one"
          interpretation="Component ID" optional="false"/>
```

```
        <declared_fixed_field name="AuthorityCode"
declared_type_ref="basicTypes.AuthorityCode"
          optional="false"/>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ReportEvents" message_id="41F0" is_command="false">
    <description xml:space="preserve">
      This message is used to report the active event requests that match the requirements
provided in the QueryEvents message.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <list name="EventList" optional="false" interpretation="List of reported events">
        <count_field min_count="0" max_count="255" field_type_unsigned="unsigned byte"/>
        <record name="ReportEventRec" optional="false">
          <fixed_field name="EventType" field_type="unsigned byte" field_units="one"
            optional="false" interpretation="Type of event">
            <value_set offset_to_lower_limit="false">
              <value_enum enum_index="0" enum_const="Periodic (SC)"/>
              <value_enum enum_index="1" enum_const="Every change"/>
            </value_set>
          </fixed_field>
          <fixed_field name="EventID" field_type="unsigned byte" field_units="one"
optional="false"
            interpretation="Unique identifier of event"/>
          <variable_length_field name="QueryMessage" field_format="JAUS MESSAGE"
optional="false"
            interpretation="The JAUS Query message to be used by the receiving component
to
            generate the Report message(s)">
            <count_field min_count="2" field_type_unsigned="unsigned integer"/>
          </variable_length_field>
        </record>
      </list>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="Event" message_id="41F1" is_command="false">
    <description xml:space="preserve">
      The Event message is sent when event is triggered. It includes the Event ID and a
      sequence number to allow the client to keep track of event processing.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="EventRec" optional="false">
        <fixed_field name="EventID" field_type="unsigned byte" field_units="one"
optional="false"
          interpretation="Unique identifier of the enclosed event"/>
        <fixed_field name="SequenceNumber" field_type="unsigned byte" field_units="one"
          optional="false" interpretation="Sequential count of the number of times this
event has
          been issued"/>
        <variable_length_field name="ReportMessage" field_format="JAUS MESSAGE"
optional="false"
          interpretation="The JAUS Report message including the message header">
          <count_field min_count="2" field_type_unsigned="unsigned integer"/>
        </variable_length_field>
```

```
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ReportHeartbeatPulse" message_id="4202" is_command="false">
    <description xml:space="preserve">
      This message notifies the receiver that the sender is alive.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body"/>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ReportIdentification" message_id="4B00" is_command="false">
    <description xml:space="preserve">
      This message shall provide the requesting component an identification summary of the
Subsystem, Node, or Component.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <record name="ReportIdentificationRec" optional="false">
        <fixed_field name="QueryType" field_type="unsigned byte" field_units="one"
optional="false">
          <value_set offset_to_lower_limit="false">
            <value_enum enum_index="0" enum_const="Reserved"/>
            <value_enum enum_index="1" enum_const="System Identification"/>
            <value_enum enum_index="2" enum_const="Subsystem Identification"/>
            <value_enum enum_index="3" enum_const="Node Identification"/>
            <value_enum enum_index="4" enum_const="Component Identification"/>
            <value_range lower_limit="5" upper_limit="255" lower_limit_type="inclusive"
              upper_limit_type="inclusive" interpretation="Reserved"/>
          </value_set>
        </fixed_field>
        <fixed_field name="Type" field_type="unsigned short integer" field_units="one"
          optional="false" interpretation="This field identifies the particular unmanned
vehicle
          (subsystem) type, node type or component type based on the following
enumeration">
          <value_set offset_to_lower_limit="false">
            <value_enum enum_index="0" enum_const="RESERVED"/>
            <value_enum enum_index="10001" enum_const="VEHICLE"/>
            <value_range lower_limit="10002" upper_limit="20000"
lower_limit_type="inclusive"
              upper_limit_type="inclusive" interpretation="Reserved"/>
            <value_enum enum_index="20001" enum_const="OCU"/>
            <value_range lower_limit="20002" upper_limit="30000"
lower_limit_type="inclusive"
              upper_limit_type="inclusive" interpretation="Reserved"/>
            <value_enum enum_index="30001" enum_const="OTHER_SUBSYSTEM"/>
            <value_range lower_limit="30002" upper_limit="40000"
lower_limit_type="inclusive"
              upper_limit_type="inclusive" interpretation="Reserved"/>
            <value_enum enum_index="40001" enum_const="NODE"/>
            <value_range lower_limit="40002" upper_limit="50000"
lower_limit_type="inclusive"
              upper_limit_type="inclusive" interpretation="Reserved"/>
            <value_enum enum_index="50001" enum_const="PAYLOAD"/>
            <value_range lower_limit="50002" upper_limit="60000"
lower_limit_type="inclusive"
              upper_limit_type="inclusive" interpretation="Reserved"/>
            <value_enum enum_index="60001" enum_const="COMPONENT"/>
```

```
            <value_range lower_limit="60002" upper_limit="65535"
lower_limit_type="inclusive"
              upper_limit_type="inclusive" interpretation="Reserved"/>
          </value_set>
        </fixed_field>
        <variable_length_string name="Identification" optional="false"
          interpretation="Human-recognizable name of the Subsystem, Node or Component. ">
          <count_field field_type_unsigned="unsigned byte"/>
        </variable_length_string>
      </record>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ReportConfiguration" message_id="4B01" is_command="false">
    <description xml:space="preserve">
      This message is used to provide the receiver a table of all existing components
located on the source’s subsystem or node depending on the value of field 1 of the Code
2B01h: Query Configuration message.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <list name="NodeList" optional="false" interpretation="# of Nodes reported. For a
single Node
        Report this field shall be 1">
        <count_field field_type_unsigned="unsigned byte" min_count="0" max_count="255"/>
        <sequence name="NodeSeq" optional="false">
          <record name="NodeRec" optional="false">
            <fixed_field name="NodeID" field_type="unsigned byte" field_units="one"
optional="false"
              interpretation="Node ID.  For single Node or Component reporting this field
shall
              contain the Node ID as specified in the Destination Address of the Query
Configuration
              message"/>
            </record>
            <list name="ComponentList" optional="false">
              <count_field field_type_unsigned="unsigned byte" min_count="0"
max_count="255"/>
              <record name="ComponentRec" optional="false">
                <fixed_field name="ComponentID" field_type="unsigned byte" field_units="one"
                  optional="false" interpretation="Component ID.  For Single Component
reporting this
                  field shall                contain the Component ID as specified in the
Destination
                  Address of the Query                Configuration message and shall be
the only
                  Component reported"/>
                <fixed_field name="InstanceID" field_type="unsigned byte" field_units="one"
                  optional="false" interpretation="Inst ID when legacy Components are
reported; a
                  value of zero (0) shall be used for non-legacy components."/>
              </record>
            </list>
          </sequence>
        </list>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ReportSubsystemList" message_id="4B02" is_command="false">
    <description xml:space="preserve">
```

```
      This message shall provide the receiving component a table of all subsystems located
in the source's system.  It also provides the ID of the component to send a Query
Configuration message within the subsystem.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <list name="SubsystemList" optional="false">
        <count_field field_type_unsigned="unsigned byte"/>
        <record name="SubsystemRec" optional="false">
          <fixed_field name="SubsystemID" field_type="unsigned short integer"
field_units="one"
            optional="false" interpretation="Subsystem ID"/>
          <fixed_field name="NodeID" field_type="unsigned byte" field_units="one"
optional="false"
            interpretation="Node ID used for Query Configuration."/>
          <fixed_field name="ComponentID" field_type="unsigned byte" field_units="one"
            optional="false" interpretation="Component ID used for Query Configuration."/>
        </record>
      </list>
    </body>
    <footer name="Footer"/>
  </message_def>
  <message_def name="ReportServices" message_id="4B03" is_command="false">
    <description xml:space="preserve">
      This message allows a component to publish its capabilities, according to the
Service Dictionary presented below.  If a component ID is specified in the RA, it may
report only one service in beyond the core message support, and this service must be equal
to the component ID.  If a component ID is not listed in the RA, it may report any number
of services.  For example, a component with ID 33 must provide only service 33.   The
exception to this rule is component ID 1 (the Node Manager) which may provide any number
of services in addition to core message support.
    </description>
    <declared_header name="AppHeader"
declared_type_ref="basicTypes.JAUSApplicationLayerHeader"/>
    <body name="Body">
      <list name="NodeList" optional="false" interpretation="# of Nodes reported. For a
single Node
        Report this field shall be 1">
        <count_field field_type_unsigned="unsigned byte" min_count="0" max_count="255"
          interpretation="Number of node sequences in the message"/>
        <sequence name="NodeSeq" optional="false">
          <record name="NodeRec" optional="false">
            <fixed_field name="NodeID" field_type="unsigned byte" field_units="one"
optional="false"
              interpretation="Node ID.  For single Node or Component reporting this field
shall
              contain the Node ID as specified in the Destination Address of the Query
Configuration
              message"/>
          </record>
          <list name="ComponentList" optional="false">
            <count_field field_type_unsigned="unsigned byte" min_count="0"
max_count="255"/>
            <sequence name="ComponentSeq" optional="false">
              <record name="ComponentRec" optional="false">
                <fixed_field name="ComponentID" field_type="unsigned byte"
field_units="one"
```

```
                  optional="false" interpretation="Component ID.  For Single Component
reporting
                  this field shall                    contain the Component ID as specified
in the
                  Destination Address of the Query                    Configuration message
and shall
                  be the only Component reported"/>
                <fixed_field name="InstanceID" field_type="unsigned byte"
field_units="one"
                  optional="false" interpretation="Inst ID when legacy Components are
reported; a
                  value of zero (0) shall be used for non-legacy components."/>
              </record>
              <list name="ServiceList" optional="false">
                <count_field field_type_unsigned="unsigned byte" min_count="0"
max_count="255"/>
                <record name="ServiceRec" optional="false">
                  <variable_length_string name="URI" optional="false">
                    <count_field field_type_unsigned="unsigned byte"/>
                  </variable_length_string>
                  <fixed_field name="MajorVersionNumber" field_type="unsigned byte"
                    field_units="one" optional="false"/>
                  <fixed_field name="MinorVersionNumber" field_type="unsigned byte"
                    field_units="one" optional="false"/>
                </record>
              </list>
            </sequence>
          </list>
        </sequence>
      </list>
    </body>
    <footer name="Footer"/>
  </message_def>
</declared_type_set>
```

B.5 BASIC TYPES

```
<?xml version="1.0" encoding="UTF-8"?>
<declared_type_set name="BasicTypes" id="urn:jaus:jss:core:MessageSet:BasicTypes"
version="1.0"
  xmlns="urn:jaus:jsidl:0.11">
  <header name="JAUSApplicationLayerHeader">
    <record name="HeaderRec" optional="false">
      <fixed_field name="MessageID" field_type="unsigned short integer" field_units="one"
        optional="false" interpretation="A two byte field to hold the message ID of a
message"/>
    </record>
  </header>
  <fixed_field name="AuthorityCode" field_type="unsigned byte" field_units="one"
optional="false">
    <value_set offset_to_lower_limit="false">
      <value_range lower_limit_type="inclusive" lower_limit="0" upper_limit="255"
        upper_limit_type="inclusive"/>
    </value_set>
  </fixed_field>
  <fixed_field name="EmergencyCode" field_type="unsigned short integer" field_units="one"
    optional="false">
    <value_set offset_to_lower_limit="false">
      <value_enum enum_index="1" enum_const="STOP"/>
    </value_set>
```

```
  </fixed_field>
  <!-- TimeStamp -->
  <bit_field name="TimeStamp" field_type_unsigned="unsigned integer" optional="false">
    <sub_field name="Milliseconds">
      <bit_range from_index="0" to_index="9"/>
      <value_set offset_to_lower_limit="false">
        <value_range lower_limit_type="inclusive" lower_limit="0" upper_limit="999"
          upper_limit_type="inclusive"/>
      </value_set>
    </sub_field>
    <sub_field name="Seconds">
      <bit_range from_index="10" to_index="15"/>
      <value_set offset_to_lower_limit="false">
        <value_range lower_limit_type="inclusive" lower_limit="0" upper_limit="59"
          upper_limit_type="inclusive"/>
      </value_set>
    </sub_field>
    <sub_field name="Minutes">
      <bit_range from_index="16" to_index="21"/>
      <value_set offset_to_lower_limit="false">
        <value_range lower_limit_type="inclusive" lower_limit="0" upper_limit="59"
          upper_limit_type="inclusive"/>
      </value_set>
    </sub_field>
    <sub_field name="Hour">
      <bit_range from_index="22" to_index="26"/>
      <value_set offset_to_lower_limit="false">
        <value_range lower_limit_type="inclusive" lower_limit="0" upper_limit="23"
          upper_limit_type="inclusive"/>
      </value_set>
    </sub_field>
    <sub_field name="Day">
      <bit_range from_index="27" to_index="31"/>
      <value_set offset_to_lower_limit="false">
        <value_range lower_limit_type="inclusive" lower_limit="1" upper_limit="31"
          upper_limit_type="inclusive"/>
      </value_set>
    </sub_field>
  </bit_field>
  <!-- DateStamp -->
  <bit_field name="DateStamp" field_type_unsigned="unsigned short integer"
optional="false">
    <sub_field name="Day">
      <bit_range from_index="0" to_index="4"/>
      <value_set offset_to_lower_limit="false">
        <value_range lower_limit_type="inclusive" lower_limit="1" upper_limit="31"
          upper_limit_type="inclusive"/>
      </value_set>
    </sub_field>
    <sub_field name="Month">
      <bit_range from_index="5" to_index="8"/>
      <value_set offset_to_lower_limit="false">
        <value_range lower_limit_type="inclusive" lower_limit="1" upper_limit="12"
          upper_limit_type="inclusive"/>
      </value_set>
    </sub_field>
    <sub_field name="Year">
      <bit_range from_index="9" to_index="15"/>
      <value_set offset_to_lower_limit="false">
        <value_range lower_limit_type="inclusive" lower_limit="0" upper_limit="127"
          upper_limit_type="inclusive" interpretation="Where 0 is 2000, 1 is 2001, etc."/>
```

```
      </value_set>
    </sub_field>
  </bit_field>
  <record name="TimeRec" optional="false">
    <presence_vector field_type_unsigned="unsigned byte"/>
    <declared_bit_field declared_type_ref="TimeStamp" optional="true" name="TimeStamp"/>
    <declared_bit_field declared_type_ref="DateStamp" optional="true" name="DateStamp"/>
  </record>
</declared_type_set>
```