



AEROSPACE RECOMMENDED PRACTICE

ARP6227™

Issued 2013-02
Reaffirmed 2024-03

J AUS Messaging over the OMG Data Distribution Service (DDS)

RATIONALE

The AS5684A standard defines the JAUS Service Interface Definition Language (JSIDL) that, in part, specifies the serialization of JAUS messages that can be exchanged over multiple transports like JAUS-over-UDP (JUDP). This ARP6227 document defines a mapping of the JSIDL messageset and typeset schema elements to OMG Interface Definition Language (IDL) used by the OMG Data Distribution Service (DDS) standard.

DDS provides a standard transport layer with interoperable support by multiple vendors. JAUS users have been moving toward implementing JAUS services over alternate transports, and significant new systems have already started integrating JAUS messaging over DDS.

This ARP allows the JAUS community to begin hosting JAUS on interoperable DDS transports, and opens the discussion for continuing work on transport considerations that might produce a JAUS / DDS standard. This document alone is NOT sufficient to create a JAUS compliant or interoperable JAUS implementation.

ARP6227 has been reaffirmed to comply with the SAE Five-Year Review policy.

SAE Executive Standards Committee Rules provide that: "This report is published by SAE to advance the state of technical and engineering sciences. The use of this report is entirely voluntary, and its applicability and suitability for any particular use, including any patent infringement arising therefrom, is the sole responsibility of the user."

SAE reviews each technical report at least every five years at which time it may be revised, reaffirmed, stabilized, or cancelled. SAE invites your written comments and suggestions.

Copyright © 2024 SAE International

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SAE.

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

SAE WEB ADDRESS:

Full document details: SAE ARP6227

For more information on this standard, visit
<https://www.sae.org/standards/content/ARP6227/>

Documents
Document Sharing Hub

TABLE OF CONTENTS

1.	SCOPE	3
2.	REFERENCES	3
2.1	Applicable Documents	3
2.1.1	SAE Publications	3
2.1.2	Object Management Group (OMG) Publications	3
3.	OMG DDS INTERFACE DEFINITION LANGUAGE (IDL).....	4
4.	JSIDL to DDS IDL Mapping	4
4.1	Using @JAUS() Annotations in DDS IDL	6
4.2	JAUS DDS IDL Support for JAUS 'presence vectors' for JSIDL Records and Sequences.....	6
4.3	JSIDL Declared Type Set	7
4.4	Simple Fields.....	7
4.4.1	Fixed Field.....	8
4.4.2	Variable Field	9
4.4.3	Bit Field	9
4.4.4	Fixed-Length String Field	10
4.4.5	Variable-Length String Field	10
4.4.6	Variable-Length Field	11
4.4.7	Variable-Format Field.....	11
4.5	Composite Fields	12
4.5.1	Array.....	13
4.5.2	Record.....	13
4.5.3	List.....	14
4.5.4	Variant.....	15
4.5.5	Sequence	16
5.	NOTES	18
FIGURE 1	COMPARISON OF A SIMPLE JAUS MESSAGE IN JSIDL AND DDS IDL	5
TABLE 1	JSIDL - DDS IDL PRIMITIVE DATA TYPE MAPPING	8
TABLE 2	JAUS COMPOSITE FIELDS MATRIX	12

TYPOGRAPHICAL CONVENTIONS

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

- Arial - 10 pt.: Standard body text
- **Courier - 10 pt. Bold:** Programming language elements.

1. SCOPE

This document defines a standard representation of JAUS AS5684A message data in DDS IDL defined by the Object Management Group (OMG) CORBA 3.2 specification. This document does NOT address how JAUS transport considerations or JAUS service protocols are implemented on OMG DDS platforms.

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), www.sae.org.

AS5684A JAUS Service Interface Definition Language, July 2010

2.1.2 Object Management Group (OMG) Publications

Available from the Object Management Group at the URL provided.

CORBA3_2 Common Object Request Broker Architecture (CORBA), Specification, Version 3.2, Part 1: CORBA Interfaces, November 2011.

OMG Document Number: formal/2011-11-01

Standard document URL: <http://www.omg.org/spec/CORBA/3.2/Interfaces/PDF>

DDS1_2 Data Distribution Service for Real-time Systems Version 1.2, January 2007.

OMG Document Number: formal/07-01-01

Standard document URL: <http://www.omg.org/spec/DDS/2.1>

DDS_XTYPES Extensible and Dynamic Topic Types for DDS (DDS-XTypes), v1.0

OMG Document Number: formal/2012-11-10

Standard document URL: <http://www.omg.org/spec/DDS-XTypes/1.0/PDF>

3. OMG DDS INTERFACE DEFINITION LANGUAGE (IDL)

As of [DDS1_2], data structures defined for the OMG Data Distribution Service (DDS) comply with interface definition language (IDL) defined by [CORBA3_2]. DDS vendors often extend this IDL to provide features requested by customers. In response, the OMG DDS group has recently published [DDS_XTYPES] to provide significant new functionality, including support for bitfields (BitFieldType) and value initialization.

Because it will take some time for this new standard to be fully supported by DDS providers, the JAUS/DDS IDL mappings will conform to [CORBA3_2]. The JAUS/DDS task group will track developments with DDS XTypes and address in future versions of this and other documents.

4. JSIDL TO DDS IDL MAPPING

This document describes an approach for converting Joint Architecture for Unmanned Systems (JAUS) message types defined in a JAUS Service Interface Definition Language (JSIDL) file into corresponding Data Distribution Service (DDS) data types defined in a DDS IDL file. Using this approach results in DDS data types that can then be compiled by a DDS IDL compiler and the resultant code used to create JAUS over DDS (JDDS) topics.

A sample JSIDL definition of a JAUS message compared with the corresponding DDS IDL definition is shown in Figure 1.

The DDS IDL representation of JAUS messages makes heavy use of module declarations to segregate the namespace and avoid name collisions, since JAUS fields can be reused in multiple contexts within the same message:

1. All JDDS data types are defined within a JDDS module.
2. Each JAUS message type is defined within a separate module. The name is taken from the **name** attribute of the **message_def** element in the JSIDL. For example, note the **message_def** element in the JSIDL below named **SimpleMessage** and the corresponding **module SimpleMessage** declaration in the DDS IDL.
3. Each JAUS message contains a header, body, and footer segment. Each of these segments is defined in a separate module with a corresponding name.
4. Each JAUS composite field type is defined in a separate module. The name is taken from the **name** attribute of the corresponding composite field element in the JSIDL. For example, note the record element in the JSIDL below named **SimpleRecord** and the corresponding **module SimpleRecord** declaration in the DDS IDL.
5. JAUS data properties that cannot be represented in standard CORBA IDL can optionally be provided as annotations enclosed in '@JAUS(...)'. This is an optional feature of JSIDL / IDL translators that allows development tooling to use the additional JAUS properties from JSIDL as represented in the IDL.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<service_def name="SimpleServer"
  id="urn:jbus:exp:dds:SimpleServer" version="1.0"
  xmlns="urn:jbus:jsidl:1.0">
  <description xml:space="preserve">Simple Server used for unit
  testing during JDDS development and testing
  </description>
  <message_set>
  <input_set>
  <message_def name="SimpleMessage"
    message_id="B001"
    is_command="false">
    <header name="Jbus_Header">
      <record name="HeaderRec" optional="false">
        <fixed_field name="MessageID"
          optional="false"
          field_units="one"
          field_type="unsigned short integer"/>
      </record>
    </header>
    <body name="SimpleBody">
      <record name="SimpleRecord"
        optional="false">
        <fixed_field name="SimpleFixedField"
          optional="false"
          field_units="one"
          field_type="integer"/>
      </record>
    </body>
    <footer name="EmptyFooter"/>
  </message_def>
  </input_set>
  </message_set>
  ...
</service_def>

```

```

module SimpleServer {
  // @JAUS(id="urn:jbus:exp:dds:SimpleServer",
  version="1.0")
  // Simple Server used for unit testing during JDDS
  development and testing
  module JDDS {

    // /// message //////////////////////////////////////
    module SimpleMessage {
      // @JAUS(query,id=B001)
      typedef struct JbusAddress {
        unsigned short SubsystemID;
        octet NodeID;
        octet ComponentID;
      } JbusAddress_;
      module header {
        typedef struct HeaderRec {
          unsigned short MessageID;
        } HeaderRec_;
      };
      module body {
        module SimpleRecord {
          typedef struct SimpleRecord {
            long SimpleFixedField; //
            @JAUS(units="one",required)
          } SimpleRecord_;
        };
      };
      struct Message {
        JbusAddress_ source;
        JbusAddress_ destination;
        header::HeaderRec_ JTS_Header;
        body::SimpleRecord::SimpleRecord_
        SimpleBody;
      };
      // pragma is an OpenSplice element, RTI and
      // others identify keys differently. DDS XTypes
      // has a standard annotation for keys.
      #pragma keylist Message
        source.SubsystemID
        source.NodeID
        source.ComponentID
      }; // message: SimpleMessage
    };
  };
};

```

FIGURE 1 - COMPARISON OF A SIMPLE JAUS MESSAGE IN JSIDL AND DDS IDL

Each of the JSIDL composite types has a corresponding typedef in the DDS IDL. These typedefs are then used in the declaration of a DDS structure named **Message** toward the end of the DDS IDL file listed above. The **Message** structure forms the actual DDS data type that will be used to create JDDS topics corresponding to the associated JAUS message. The **Message** structure contains members for the header, body, and footer. In the above example, the footer is empty, so no element is included for the footer.

The **Message** structure also contains a declaration of the source JAUS address. The source address is used as the key for the DDS data type. A particular JAUS message type may be sent by multiple components in the system. Designating the source address as the key field allows JDDS messages of the same type published by two different JDDS components to form separate DDS instances and, therefore, be distinguished from one another.

The **Message** structure also contains a declaration of the destination JAUS address. The destination address allows subscribers to filter JDDS messages, ignoring those directed to another JDDS component. JAUS messages may also be broadcast locally to all JAUS components within a JAUS subsystem by setting the component ID and node ID of the destination address to 0xFF. Global broadcasts to all JAUS components in all JAUS subsystems may be made by also setting the subsystem ID to 0xFFFF.

The DDS IDL in Figure 1 shows an example of a JAUS DDS message keyed by its source JAUS identifier using the OpenSplice syntax (`#pragma`). This document defines how data in message payloads is mapped from JSIDL to DDS IDL and does not mandate specific DDS implementation decisions about DDS Topics, Topic types, and Topic type keys. Such decisions may be the subject of a future specification.

DDS allows topic data types to have key fields defined [DDS1_2]. DDS IDL does not specify a standard way to annotate IDL to specify key fields. The example in Figure 1 shows the use of `source.SubsystemID`, `source.NodeID`, and `source.ComponentID` as keys for all JAUS-DDS topics. JSIDL to DDS IDL processing software should take this into account, possibly with options to generate IDL with extensions for multiple popular DDS middleware IDL processors.

4.1 Using @JAUS() Annotations in DDS IDL

The JSIDL XML schema supports a much richer set of information than the [CORBA3_2] IDL grammar supports.

Any JAUS DDS IDL created per this Aerospace Recommended Practice (ARP), MUST have an associated, complete and valid JSIDL specification. The JSIDL specification is authoritative in all respects.

With that in mind, it is helpful to include *some* of the JSIDL information in JAUS DDS IDL to improve readability and understandability. This is done through the use of annotations embedded in comments per the following rules:

1. All annotations are optional.
2. If an annotation disagrees with the corresponding JSIDL, the JSIDL is correct.
3. Every annotation is embedded in a comment and enclosed with '@JAUS()'.
 - An annotation following a semi-colon annotates the element just before the semicolon.
 - An annotation at the beginning of a block (just after an opening '{') annotates the entire element defined by the block.
4. An annotation contains a list of tokens or name-value pairs. String values are quoted. For example:
`@JAUS(units="meter",range="(0.0,1.0]",required).`
5. Numeric value ranges in JSIDL can have open (exclusive) or closed (inclusive) boundaries. In the JAUS DDS annotations, we use the convention of parens for open boundaries and square brackets for closed boundaries. For example, in the previous case `range="(0.0,1.0]"` denotes all values x such that $x > 0.0$ and $x \leq 1.0$.
6. An annotation's position determines which syntactic DDS IDL element it is associated with.

4.2 JAUS DDS IDL Support for JAUS 'presence vectors' for JSIDL Records and Sequences

JSIDL and JAUS messages support 'optional' items whose data may or may not be valid and whose data may or may not be transmitted depending on the setting of a bit in the "presence vector". Each JAUS record or sequence that contains one or more optional items MUST have a presence vector whose size is the minimum number of bytes required to encode a bitfield marking which optional items are present (1) or absent (0).

In JSIDL, the association of optional items to bits in the `presence_vector` bitfield is implied by the order of items within the record or sequence in the XML container. For example, if, in a record, only the second and fourth items are set with `optional="true"`, then the associated presence vector assigns bit 0 to the second item and bit 1 to the fourth item.

DDS IDL does not support the notion of optional fields that can be removed from messages to reduce bandwidth. Since such optimization is more appropriately a transport-level optimization, JAUS DDS does not try to emulate optional fields. Instead, each JSIDL `presence_vector` maps to a `presence_vector` with the same size and bit interpretation, but the bits should be interpreted as 'validity' bits. An `optional="true"` JSIDL field will always be present in every DDS data item, but its `presence_vector` bit will indicate whether it contains valid data.