

---

**Contribution Number: NPF2002.429.05**

---

**Working Group: Hardware**

---

**Task Group: Message Layer Task Group**

---

**TITLE: NPF Message Layer Implementation Agreement<sup>1</sup>**

---

**SOURCE: Erik Johnson**  
Technical EditorIntel Research & Development  
2111 NE 25<sup>th</sup> Avenue, M/S JF3-206  
Hillsboro, OR 97124-5961  
USA  
Phone: +1 503 264 9950  
Email: [erik.j.johnson@intel.com](mailto:erik.j.johnson@intel.com)**Mike Lerer**  
Message Layer Task Group ChairXilinx  
Phone: +1 603-548-3704  
Email: [mlexer@fpga.com](mailto:mlexer@fpga.com)

---

**DATE: November 11, 2003**

---

**ABSTRACT:**

This specification defines the Network Processing Forum (NPF) message layer implementation agreement. The message layer, or “messaging”, enables a system designer to configure the information communicated between NPEs within a system without requiring proprietary extensions to the NPEs. Messaging is suited to NPE-to-NPE communications over a range of conveyances, including streaming and look aside. In-band messages are composed of a configured number of fixed-sized (1 byte) slots. These slots are grouped into one or more consecutive message types, which, in a message header or message trailer, carry the NPE-to-NPE information along with the message payload. An out-of-band configuration mechanism is used to determine the meaning associated with the in-band messages for both the sending and receiving NPEs. This document represents the final NPF message layer implementation agreement as it will be published.

For additional information contact:  
The Network Processing Forum, 39355 California Street,  
Suite 307, Fremont, CA 94538  
+1 510 608-5990 phone ☎ [info@npforum.org](mailto:info@npforum.org)

© 2002-2003 Network Processing Forum

---

<sup>1</sup> This document represents the final NPF message layer implementation agreement as it will be published

**Revision History**

Version	Date	Details
00	07/29/2002	Baseline selected proposal from July 2002 NPF plenary
01	10/4/2002	Updates based on npf2002.484 (added message trailer and message payload padding, message alignment restrictions), npf2002.523 (added message field syntax graphs), npf2002.521 (clarified description of the standard NPE operation configuration fields). Also includes several editorial updates including changing "slot configuration ID" to "format ID", "NPE Instruction" to "NPE Operation" and use of the term "conveyance granule". Refer to the associate editor's notes document for specific details (npf_msg_2002_429_editor_list.doc).
02	2/21/03	Version 01 was incorrectly uploaded to the NPF contribution website as version 02. Thus, version 01 and 02 are identical
03		Edited for readability.
04	6/30/03	Added resolutions for comments from straw ballot (npf2003.285.00). Also modified the settings on the Visio diagrams so that the PDF converter would stop losing text, these figure replacements were not change tracked.
05	11/10/03	Removed references to DRAFT in the header and title

**Table of Contents**

1. Overview .....	4
1.1 Conventions Used .....	4
1.2 Scope.....	4
1.3 Design Objectives .....	5
1.4 Message Layer Reference Model .....	5
1.5 Design Overview.....	10
2. NPE Requirements .....	15
2.1 Programmable and Configurable NPEs .....	15
3. SAE Operation .....	15
3.1 Alignment of Fields and Payloads.....	16
3.2 Multiple Message Header Formats for a Single Conveyance Channel .....	17
4. Formal Message Format Definition .....	18
4.1 Basic Message Parts.....	18
4.2 Containers .....	23
4.3 Data Types .....	24
4.4 Message Fields .....	27
Annexes .....	39
A Glossary.....	39
B Syntax Graph Notation Guide (Informative) .....	41
C In-band Configuration (informative) .....	42
D NPE Capabilities Grammar (normative).....	45

## 1. Overview

The mission of the messaging task group is:

*To enable multi-vendor interoperability between NPEs beyond the physical (i.e., streaming or look aside) interconnect level.*

Unlike physical or link-layer interfaces, which simply enable NPEs to physically exchange 'bits' of data, the goal of messaging is to define the meaning (e.g., configuration, status, flow identification, etc.) of these bits of data so that two NPEs in a system can share work.

### 1.1 Conventions Used

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMEND", "MAY", and "OPTIONAL" in this document are to be interpreted as described by the IETF's RFC 2119.

Footnotes used in this specification are informative only.

A reserved field or bit MUST be transmitted as zero and ignored on reception.

If there is a conflict between a state machine diagram or syntax graph and the text of this document (including a table entry), the state machine takes precedence over the text.

Figures (except state diagrams or syntax graphs) are informative only.

### 1.2 Scope

This document is a definition for NPE-to-NPE messaging. Messaging standardizes the format and types of information exchanged between NPEs, enabling a system integrator to construct a single network element (i.e., a box) using NPEs from various vendors without requiring proprietary communication mechanisms between different vendors' NPEs. Messaging is applicable to both data and control plane data exchanges within a network element, with the primary design focus being on data plane message exchanges.

#### 1.2.1 Scope of Underlying Conveyances

Messaging covers NPE-to-NPE communications within a single network element, over a range of common NPE-to-NPE conveyances including streaming, look-aside, and switch fabrics. Messaging can be layered on top of any conveyance that meets a minimal set of requirements, including, but not limited to, those being developed within the NPF such as CSIX<sup>2</sup> and SI. Message may be layered on top of non-NPF conveyances such as Ethernet, for example. Wherever possible, messaging exposes and does not duplicate the capabilities of the conveyance in which it is encapsulated.

Any conveyance used to carry NPF messages MUST:

- *Support variable sized packets.* This requirement can be accomplished in any manner but MUST minimally include conveyance-level packet length as well as packet delineation (e.g., start and end of packet markers). The conveyance determines the granularity of packet lengths (e.g., all packets are a multiple of 4-bytes long). The conveyance also determines the maximum packet length with the following additional assumption:
  - No fragmentation is performed at the messaging layer.

The system designer can deal with this assumption in any manner, including, but not limited to, selecting a conveyance that supports segmentation and reassembly or by always configuring the messages such that the total size of the message (i.e., message header, message payload and message trailer) is never larger than the MTU of the selected conveyance.

---

<sup>2</sup> CSIX support of messaging requires a currently non-ratified specification for an L2A header.

- *Provide NPE-endpoint addressability.* The conveyance provides the appropriate addressing necessary to deliver packets to its NPE endpoints. For some conveyances (e.g., point-to-point), this requirement may not physically result in the transmission of addressing information. Messaging exposes and exploits the addressing of the underlying conveyance.
- *Provide at least unidirectional operation.* A interconnect that is physically bidirectional is an acceptable conveyance, but is modeled as two unidirectional conveyances for the purposes of messaging.
- *Protect its resources.* This requirement can be accomplished in any manner including, but not limited to, providing a flow-control mechanism.
- *Provide error detection.* Packets that the conveyance detects are erroneous may either be dropped, with or without notification to the messaging layer, or may be delivered to the messaging layer with an indication of the error. Conveyances are not required to be reliable, but any error correction in the conveyance MUST be transparent to the messaging layer.

### 1.3 Design Objectives

The design objectives are designated as:

1. To expose, and not to duplicate, the functionality of the underlying conveyance where possible.
2. Support simplex unidirectional traffic (i.e., “fire-and-forget”, connectionless) over any conveyance, including, but not limited to, streaming, look aside and switch fabric style conveyances between NPEs.
3. Support the pairing of two unidirectional connections to enable bidirectional communication between two NPEs.
4. Define standard message types for out-of-the-box multi-vendor interoperability
5. Define a message header and trailer format that:
  - a. Can represent a wide range of message types
  - b. Is efficient to access, process, and transmit

### 1.4 Message Layer Reference Model

In order to describe the interface provided by the message layer to the application layer, as well as the assumptions made by the messaging layer on the conveyance layer, this section presents a message layer reference model. This reference model specifies the architectural services assumed by, and provided by, the message layer, but does not imply any particular implementation of these services.

This reference model is only applicable in a system using messaging. Messaging is OPTIONAL for a system, application, or conveyance.

Architecturally, one or more applications on an NPE use services provided by a message layer. A message layer, in turn, relies on services provided by a conveyance layer. This architectural layering is depicted in Figure 1, which shows the application, message, and conveyance layers for two NPEs using messaging for unidirectional communications. Applications requiring bi-directional communications are accomplished with two unidirectional models, as shown in Figure 2. It is acceptable (and reasonable) for a bi-directional conveyance layer to be modeled as two unidirectional conveyance layers.

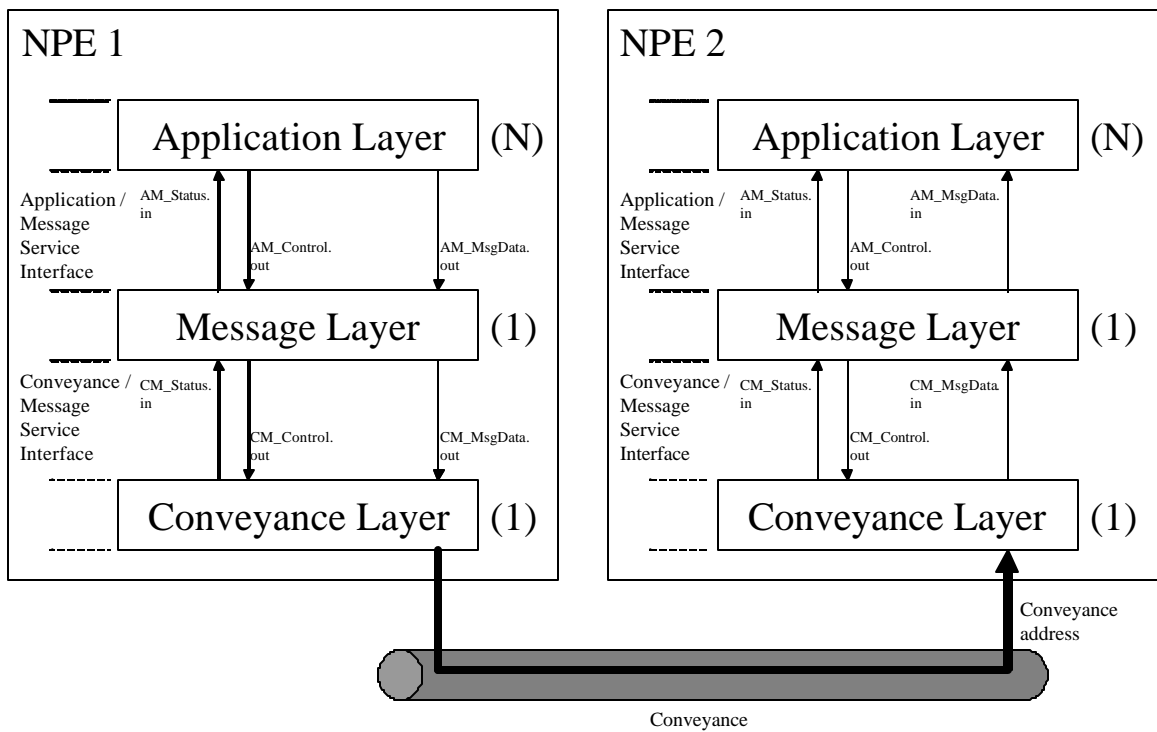


Figure 1 Unidirectional Message Layer Reference Model

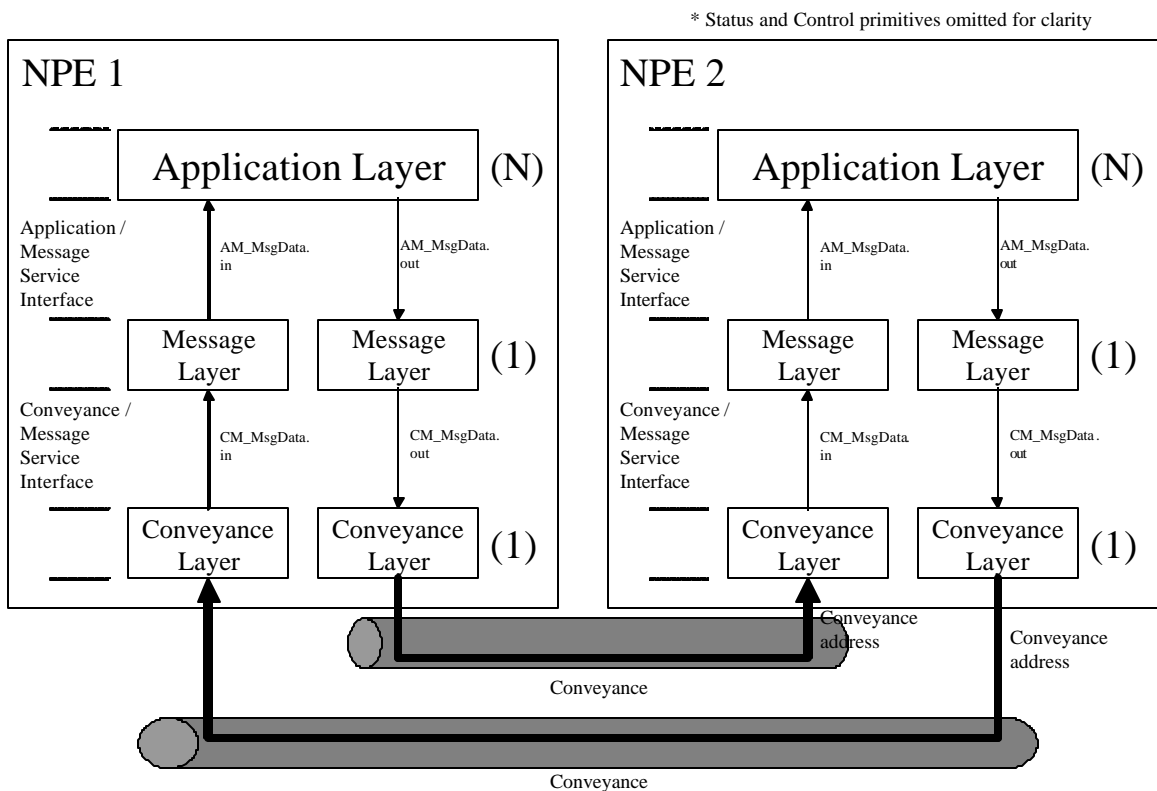


Figure 2 Bi-directional Message Layer Reference Model Using Two Unidirectional Models

As illustrated, the message layer reference model consists of three layers:

**Conveyance layer** This layer corresponds to specifications defined for the movement of data. Examples within the NPF would include the streaming and look

	aside interfaces.
Message layer	This layer corresponds to this implementation agreement.
Application layer	This layer corresponds to specifications that describe a function provided by an NPE. <i>The use of the term "application" is not meant to imply either a software or hardware implementation; The intent is to imply a function that is a source or destination of messages that provides some functionality as the result of its receipt of messages.</i>

The message layer reference model specifies a one-to-one relationship between the conveyance layer and the message layer. A one-to-many relationship is specified between the message layer and the application layer.

For convenience, the layers are described as having an order. The conveyance layer is described having the lowest order and the application layer described as the highest. The message layer is higher than the conveyance layer but is lower than the application layer.

Between these three layers, two interfaces are defined:

- The application/message service interface (AM)
- The conveyance/message service interface (CM)

Each of these interfaces contains service primitives used for passing messages between different NPEs (MsgData) as well as service primitives that are internal to the NPE (Control and Status). The internal service primitives do not result in messages carried across the conveyance.

Several important details related to these interfaces are:

- Applications use the *AM\_MsgData* service primitives to send and receive messages. For the applications on NPE1 in Figure 1, the *AM\_MsgData.out* primitive (provided by the message layer) allows messages to be sent to NPE2. Similarly, for NPE2, the *AM\_MsgData.in* primitive delivers messages to the applications on NPE2.
- The message layer uses the *CM\_MsgData* service primitives to send and receive messages on the conveyance. For example, in response to an *AM\_MsgData.out* primitive, the message layer would use the *CM\_MsgData.out* primitive to send the message on the appropriate conveyance.
- Both the *AM\_MsgData* and *CM\_MsgData* service primitives use conveyance addresses to identify the incoming or outgoing message destination. A path between two conveyance end points identified by a conveyance address is a conveyance channel. The message layer does not define its own addressing scheme but instead uses the conveyance addresses directly (see Section 1.2.1). Therefore, each conveyance address corresponds to a particular instance of the message layer in a one-to-one or many-to-one relationship. When a conveyance receives a packet, if the conveyance address indicates delivery to a message layer, the *CM\_MsgData.in* primitive MUST be used to deliver the message to the message layer. If a conveyance supports multiple addresses, it is acceptable to have more than one conveyance address on the conveyance mapped to the messaging layer. For example, two conveyance addresses could send conveyance packets to the messaging layer. The conveyance layer addresses that are used for directing packets to the message layer are defined by configuration. The intent is to allow demultiplexing at both the messaging and application layer. In addition to using the conveyance layer addressing, fields within the message are defined to allow for additional demultiplexing in the application layer.

The following sections describe each of the service layer primitives, starting with a general syntax for these descriptions.

### 1.4.1 Service Primitive Syntax

The syntax of a service primitive is:

```

xx_unit.type  [
                operand 1
                operand 2
                ]

```

where

*xx*            The service interface. Valid values are

    AM            Application/message service interface

    CM            Conveyance/message service interface

*unit*        Classification of the information being passed. Valid values are

    MsgData      Message data (NPE-to-NPE messages)

    Control      Flow control information (internal to an NPE)

    Status        Interface status, error conditions (internal to an NPE)

*type*        Primitive type. Valid values are

    out            Service request to a lower reference model layer.

    in            Service request to a higher reference model layer.

Operand1      Conveyance address (CAddr)

Operand2      Service message

## 1.4.2 Conveyance/Message Service Primitives

### 1.4.2.1 CM\_MsgData.in[

    CAddr        The destination conveyance layer address. This address is used as the primary method for directing data to the message layer.

    Msg         Message (message header, message data, and message trailer).

    ]

**Description:** This primitive is used to move messages from the conveyance layer to the messaging layer. Based on the value of the conveyance address operand (CAddr), the messaging layer passes the message to the proper application layer using the *AM\_MsgData.in* service primitive. The conveyance layer **MUST** use this primitive and the message layer **MUST** support this primitive.

### 1.4.2.2 CM\_Status.in[

    CAddr        Destination conveyance layer address. This address is used as the primary method for directing data to the message layer.

    Error        Error details .

    ]

**Description:** This primitive is used to indicate that an error occurred in the conveyance layer during reception of a message. Based on the value of the conveyance address operand (CAddr), the error is reported to the proper application layer using the *AM\_Status.in* service primitive. The conveyance layer **MAY** use this primitive and the message layer **MUST** support this primitive.

This primitive is internal to a single NPE.

### 1.4.2.3 CM\_MsgData.out[

    CAddr        Destination conveyance layer address. This is the address the conveyance layer uses when sending the data, found in the Msg operand, on the conveyance layer interface.

    Msg         Message (message header, message data, and message trailer).

]

**Description:** This primitive is used to pass messages from the message layer to the conveyance layer. The message layer uses this primitive in response to an *AM\_MsgData.out* service primitive received from the application layer. The conveyance layer **MUST** support this primitive and the message layer **MUST** use this primitive.

#### 1.4.2.4 CM\_Control.out[

CAddr      Conveyance layer address corresponding to the application layer.  
 Status      Flow control status for the conveyance layer address provided.

]

**Description:** This primitive is used to pass flow control information about a particular conveyance layer address from the message layer to the conveyance layer. If the conveyance layer supports a flow control status path, then the conveyance address is used to map into the conveyance layer's flow control structure, and the status is mapped into the report types supported by the conveyance layer's flow control status path. The message layer uses this service primitive to protect its own resources, or as a response to an *AM\_Control.out* service primitive received from the application layer. The conveyance layer **MAY** use this primitive and the message layer **MUST** support this primitive.

This primitive is internal to a single NPE.

### 1.4.3 Application/Message Service Primitives

#### 1.4.3.1 AM\_MsgData.in[

CAddr      Conveyance layer address. This address **MAY** be used by the application layer to direct the message to the proper function within the application layer.  
 Msg      Message (message header, message data, and message trailer).

]

**Description:** This primitive is used to pass message data from the message layer to the application layer. This primitive is used in response to a *CM\_MsgData.in* primitive received from the conveyance layer. The application layer **MUST** support this primitive and the message layer **MUST** use this primitive.

#### 1.4.3.2 AM\_Status.in[

CAddr      Conveyance layer address. This address **MAY** be used by the application layer to direct the message to the proper function within the application layer.  
 Error      Error details .

]

**Description:** This primitive is used to indicate that an error occurred in the conveyance layer during reception of a message. Based on the value of the conveyance address operand, the error is reported to the proper function within the application layer. This primitive is used in response to a *CM\_Status.in* primitive. The application layer **MUST** support this primitive and the message layer **MUST** use this primitive.

This primitive is internal to a single NPE.

#### 1.4.3.3 AM\_MsgData.out[

CAddr      Destination conveyance layer address. This is the address the conveyance layer uses when sending the data, found in the Msg operand, on the conveyance layer interface.  
 Msg      Message (message header, message data, and message trailer).

]

**Description:** This primitive is used to pass message data from the application layer to the message layer. The message layer uses the *CM\_MsgData.out* service primitive in response to this service

primitive. The application layer MUST use this primitive and the message layer MUST support this primitive.

#### 1.4.3.4 AM\_Control.out[

CAAddr Conveyance layer address corresponding to the function within the application layer.

Status Flow control status for the conveyance layer address provided.

]

**Description:** This primitive is used to pass flow control information from the application layer to the message layer. This flow control information describes the status of the function within the application layer indicated by the conveyance layer address. The application layer uses this primitive to protect its own resources. The message layer MUST respond to this primitive by either using the *CM\_Control.out* primitive, or by using the status of its own resources to determine when the *CM\_Control.out* primitive is sent. The message layer could honor the status or it MAY rely on the conveyance layer to honor the status. For the later case, the message layer MUST immediately respond by using the *CM\_Control.out* primitive. The application layer MAY use this primitive and the message layer MUST support this primitive.

This primitive is internal to a single NPE.

## 1.5 Design Overview

### 1.5.1 Basic NPE-to-NPE Messaging Topologies

Figure 3 shows a basic system composed of multiple NPEs interconnected with streaming, look aside, and switch fabric style conveyances, as examples. Other conveyance types are possible with messaging. Messaging is appropriate for NPE-to-NPE communications across all of these conveyances, with examples as follows:

- **NPE1 to NPE2:** Messaging is utilized across a unidirectional streaming conveyance channel to pass packets with packet attributes to a specific destination NPE, enabling NPE2 to benefit from the processing done in NPE1. A typical usage of messaging in this instance would be to communicate the resulting tag from a classification coprocessor to an NPU, or a queue and class of service from an NPU to a traffic management coprocessor.
- **NPE2 to NPE3:** Messaging is utilized across a look aside conveyance (represented with two unidirectional conveyance channels) to pass some combination of commands, packet data and headers, and attributes, enabling NPE3 to service a request from NPE2 and for NPE2 to accept a response from NPE3. A typical usage of messaging in this instance would be to communicate the lookup key and table identifier from an NPU to a TCAM, or a cryptographic key, or index, and packet payload to an encryption coprocessor.
- **NPE2 to NPE4:** Messaging is utilized across a switch fabric conveyance (either unidirectional or bi-directional) to pass packets with packet attributes to a specific destination NPE, enabling NPE4 to benefit from the processing done in NPE2. Much like the streaming conveyance, this usage of messaging might typically occur between an ingress and egress NPU to communicate output port and quality-of-service metrics.

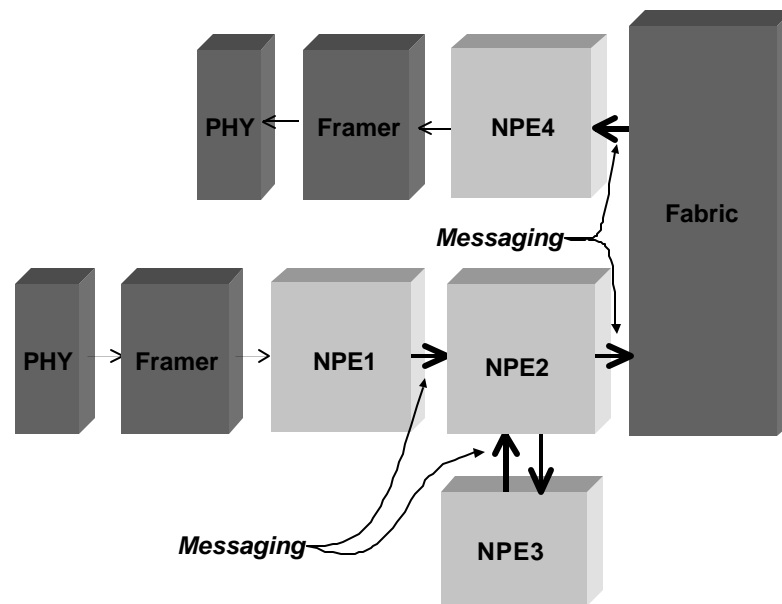


Figure 3 NPE-to-NPE topologies appropriate for messaging

### 1.5.2 Messaging Design Elements

The messaging solution consists of the following key items:

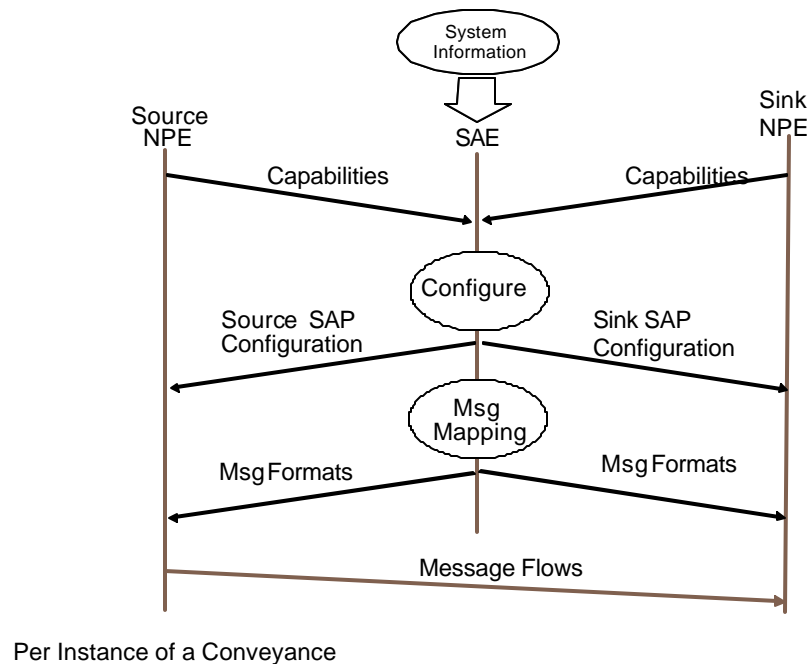
- A format for in-band message headers and trailers (using slots) along with message payloads.
- A definition of common message fields and sizes.
- An out-of-band configuration mechanism, called the system architectural entity (SAE), which gathers capabilities and requirements for the system and NPEs, and then, per-conveyance, configures the conveyance and application (i.e., service access point, or SAP) information in each NPE. Finally, the SAE determines and communicates message formats for each conveyance by mapping message fields into slots.

As shown in Figure 4, NPF messaging begins with the SAE gathering both system information and, per-conveyance, source and sink NPE capabilities. The system information, which may include details such as physical topologies in the network element and required application support, is obtained in some manner outside the scope of this implementation agreement. Each NPE's capabilities, which include supported message field types and sizes, are communicated to the SAE in a standard format as specified in normative Annex D.

Given this system and NPE information, the SAE determines and configures the service-access point (SAP) information for both the source and sink NPE. This SAP configuration defines the mapping from applications to the conveyance addressing scheme underneath messaging. The configuration information, which is for both the source and sink NPEs, and which is likely different for the two NPEs, is communicated to these NPEs in a manner specified conceptually in informative Annex C.

Following configuration, the SAE performs message mapping. Message mapping includes identification of, and message format definition for, communicating SAPs. Message format definition entails the assignment of standard message field types to slots in the message header and trailer and a description of the message payload (e.g., fixed or variable size, padded or unpadded, etc.). The slot assignment MUST honor alignment constraints of both the conveyance and NPE's processing subsystem. Once message mapping is complete, both source and sink NPEs are provided with the resulting message formats in a manner specified conceptually in informative Annex C.

Finally, the NPEs can begin passing messages according to the configuration and message formats established by the SAE.



**Figure 4 The basic messaging design**

This design enables message specifications to be optimized for processing and bandwidth efficiency, while honoring any sending or receiving alignment constraints. This design also enables a wide range of message formats. This design allows the message specification to be specific to an application, while providing a lowest-common-denominator structure that allows simple messages to be passed with minimal overhead.

The message header and trailer formats are specified as fixed-size (1 byte) slots that carry the NPE-to-NPE message fields. This format supports messaging across all of the conveyances described in the previous section. The SAE, an out-of-band configuration mechanism, is used to map the message fields into specific slots.

The system illustrated by Figure 4 has the following advantages:

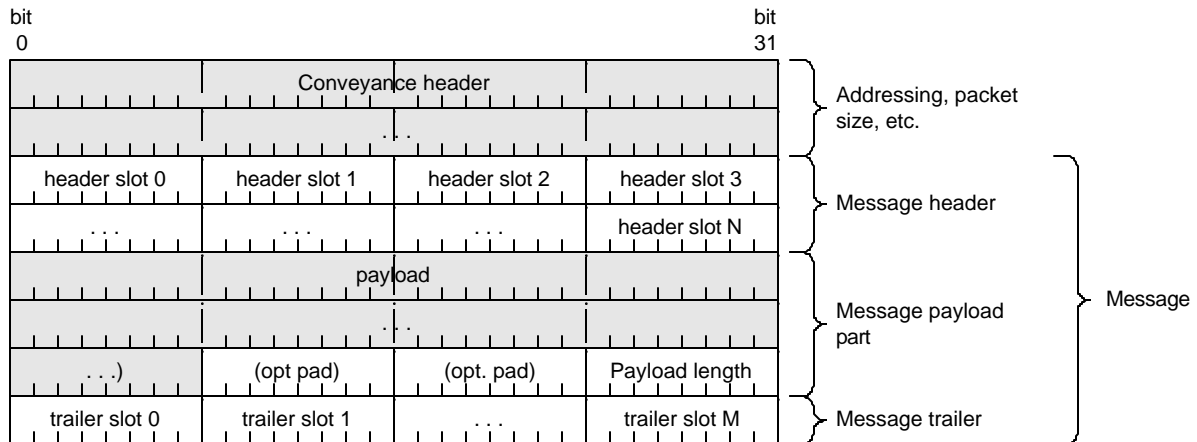
- No message bandwidth is consumed representing the length or positions of the message fields. This information is part of the configuration performed by the SAE, and is thus under the control of the system designer. This allows the system designer to control conveyance-layer over-speed requirements, if present, by restricting the maximum size message header and trailer added to any payload.
- Slot accesses are processing efficient (i.e., direct access to any slot).
- The message header and trailer (i.e., slot usage) MAY be defined differently for each conveyance channel. This enables each conveyance channel to be optimized for the particular communication performed across that conveyance channel. Furthermore, new message specifications can be setup by the SAE at run-time, if both the sending and receiving NPE support such run-time configuration.
- Each conveyance channel MAY support multiple, different, messages specifications, where the first message field (the `format-id`) MUST be used to identify the message specification.
- The SAE MUST make slot assignments that maintain processing granule alignment of certain fields for NPEs that incur costs for unaligned data access.

### 1.5.3 Message Format and Terminology Overview

This section provides a brief overview of the format of a message with the intention of quickly bringing readers up to speed on the message format and terminology. Several examples are provided to

illustrate the parts, and basic physical encoding, of a message within a conveyance layer. The formal definition of a message, including all of the various standard field types, their orderings, and semantic values, is covered in Section 4.

As shown in Figure 5 a message consists of three *parts*: a header, payload, and trailer. These parts are physically encapsulated and delivered via an underlying conveyance. The message header comes immediately after the conveyance header and **MUST** be of appropriate size to ensure the payload is aligned<sup>3</sup>. The payload starts, if present, at the first byte following the message header. The payload ends with either padding and a length field (as shown) or with opaque padding and no length field (not shown). In either case, the payload is sized to ensure the message trailer, or start of the next message, is aligned. The trailer starts, if present, at the first byte following the message payload and **MUST** be of appropriate size to ensure the message is properly sized. The conveyance layer's mechanism for delineating packets is used to determine the end of the message.



The data shown is transferred in left-to-right, top-to-bottom order at both the bit and byte level

**Figure 5** Message format encapsulated in an example conveyance layer with 4-byte granule

In Figure 5, a generic conveyance header with a 4-byte *conveyance granule* is shown. The granule of a conveyance is defined as the minimum, logical, allocation of bandwidth (in bytes) for the conveyance. For example, CSIX has a 1-byte granule and SI has a 2-byte granule. Messaging uses the conveyance granule (along with a processing granule that is explained later) to constrain the sizes of the header, payload, and trailer. Specifically, the size of each part of a message **MUST** be an integral number of conveyance granules. This requirement ensures the overall alignment of the message parts with respect to the underlying conveyance. See Section 3 for more details, including the notion of a processing granule that may impose stricter constraints on message parts than the conveyance granule. For simplicity, the remaining figures illustrating the messaging layer in this document do not include the conveyance layer content, even though the text refers to the notion of the conveyance granule.

The *format* of the message header and trailer are determined by the SAE during the design phase of the network system. For any given format, the size and layout of the message header and trailer are fixed and well-known.

Both the message header and trailer are divided into a series of fixed-sized *slots*. Each slot is 1 byte (8 bits) in length. By dividing the message header and trailer into fixed-sized slots, each slot can be accessed directly. For example, referring to Figure 5, an NPE that needs to read the value in header slot 3 **MUST** simply read the 4<sup>th</sup> byte in the message header. Similarly, an NPE that needs to read the value in trailer slot 2 **MUST** simply read the third byte from the start of the message trailer.

*Slot numbering* is always zero-based and relative to the start of the message part (i.e., header or trailer). For the header, the slot number can be easily translated into a physical offset from the start of the message (as explained above). However, due to the (potentially) variable size of the payload, the start of the trailer cannot be computed from the start of the message. Instead, for an NPE to access a trailer slot by number, the end of the message, along with the size of the trailer can be used. Again, note that

<sup>3</sup> Alignment is described in detail in subsequent sections.

for any given format, the size of the trailer is fixed and known. Thus, a particular trailer slot can be accessed directly by number using the following pseudo-code:

```
&trailer_start = message_ptr + length_of_message - size_of_trailer;
trailer_slot = trailer_start[trailer_slot_num];
```

A message header and trailer are comprised of a series of message *fields*<sup>4</sup>. The fields MAY be used to provide information about the message payload<sup>5</sup>, or MAY indicate the type of processing and related information (operands).

Certain message fields MUST be aligned on *processing granules*. A processing granule is defined as the minimum, logical, computational access (in bytes) for the message. Each NPE endpoint participating in messaging can specify its processing granule. Intuitively the processing granule is computed using the natural size of an NPE-endpoints' data paths. Messaging uses the processing granule to align fields so as to make processing on NPEs most efficient. It should be noted that both the conveyance granule and processing granule can be specified as 1-byte when conveyance overhead (i.e., bandwidth) is the most desired system design feature to conserve.

A field is one or more consecutive slots that correspond to one type of information associated with the message. For example, Figure 6 shows a streaming classification coprocessor communicating the results of packet classification to an NPU using a 2-slot field of type *flow-id*<sup>6</sup>. No field is a priori bound to any particular slot(s) except for the *format-id* (see Section 3.2), and only a few fields are REQUIRED in any particular NPE-to-NPE communications (but even these fields can be *implicit* in that they do not appear in the physical instantiation of the message). The fields do, however, have some restrictions on when they may exist and on their relative ordering as explained in Section 4. The NPEs (classification coprocessor and NPU) are both separately configured to reference the same size and position of the *flow-id* field via the SAE.

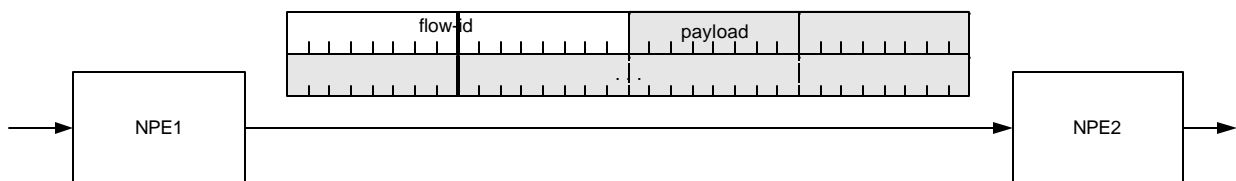


Figure 6 Single, 2-slot header field per message

Multiple fields, including fields in the message trailer, are allowed within a single message, as shown in Figure 7. In this example, an NPU communicates a 2-slot *flow-id* field and a 1-slot *destination-port* field in the message header and a *crc-32* field in the message trailer to a traffic management coprocessor. Again, the SAE determines the order and size of the fields based on the system design.

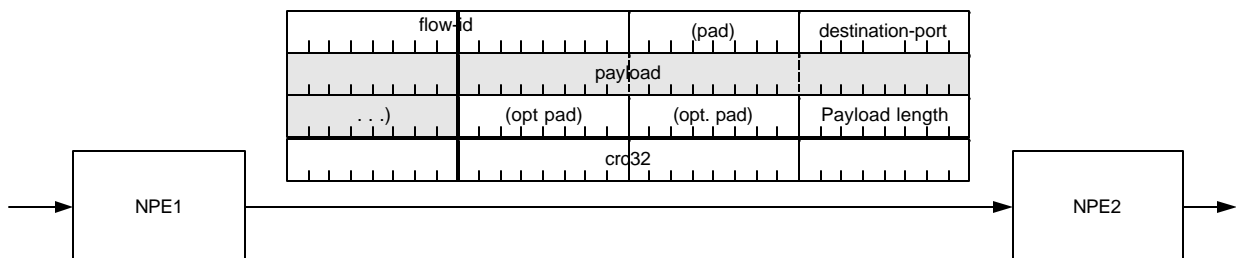


Figure 7 Multiple fields per message

<sup>4</sup> Referred to as just fields in the remainder of this document

<sup>5</sup> Referred to as just payload in the remainder of this document

<sup>6</sup> *flow-id* is the type field identifier.

## 2. NPE Requirements

The two requirements for an NPE to enable messaging are:

- The NPE **MUST** be able to express its expected and generated message fields, alignment requirements (i.e., processing granule), and any fixed slot assignments, for each conveyance channel (see Annex B). The expression of expected and generated message fields **MAY** be as simple as documentation associated with an NPE.
- The NPE **MUST** ignore any slot that it has not been configured to use.

Both of these requirements are aimed at enabling messaging within a system with minimal burden to NPE designs. The first requirement is the minimum functionality for determining off-the-shelf interoperability between NPEs. For example, it is possible for a fixed-function device to meet this requirement with little or no modifications. The second requirement allows the SAE to properly align the parts of the message (e.g., fields, header, trailer and payload). By requiring the NPE to ignore non-configured slots, the SAE can control message field and message part alignments as well as total system processing requirements.

### 2.1 Programmable and Configurable NPEs

While the first requirement enables determination of NPE interoperability, for programmable or configurable NPEs, two additional capabilities can enable more messaging options for the system designer. These capabilities are:

- An NPE **SHOULD** be able to accept a fixed-value (i.e., implicit value) for any field at configuration time. This enables the SAE to eliminate fields that do not change (i.e., are constant) from appearing in every message, and thus consuming bandwidth.
- An NPE **SHOULD** be able to read and write its properly-ordered message fields into any properly aligned set of consecutive slots. A set of message fields is properly ordered if they conform to the relative ordering set forth in Section 4. A set of consecutive slots of size N slots is properly aligned if it starts on a slot number evenly divisible by the processing granule. For maximum flexibility within the SAE, an NPE **MAY** support the ability to read and write its message fields into any slot, regardless of alignment (i.e., an NPE **MAY** support a processing granule of 1 for maximum flexibility).

These additional capabilities state that a programmable or configurable NPE **SHOULD** contain some capability (e.g., slot offset register or table) to allow the system integrator to assign and refer to constant-valued fields, and to position the explicit-value input and output fields of the NPE. With these capabilities, the system designer has the flexibility to create off-the-shelf interoperability between NPEs. Some existing NPEs might not be capable of meeting these additional requirements if their outputs have been hard coded to certain offsets within the message. This does not preclude the use of these NPEs within a system using messaging. Indeed, many systems could function quite well with such NPEs. However, these NPEs could cause irresolvable system conflicts for a system designer. (E.g., one NPE only writes information into slot 0 and the second only reads from slot 2).

## 3. SAE Operation

The SAE, a logical entity, is a human or a software-based allocation engine tool. The SAE is used by a system designer, and, if implemented as software, is not required or expected to execute on any NPE. The SAE determines the mapping of fields to one or more slots based on system-wide knowledge. This mapping is determined once during initialization or infrequently as part of re-configuration.<sup>7</sup> The SAE has no performance requirements associated with it because it is irrelevant to the functionality and performance of this design where or how the SAE is implemented.

At a high-level, the SAE performs roughly the following steps, in order:

---

<sup>7</sup> This agreement does not cover the details of reconfiguration. In particular, the issue of atomically updating all NPEs with the new configuration is an issue left open to the system designer.

1. Determine the SAP mappings using the system information.
2. Communicate the SAP mappings to each NPE.
3. Determine each NPEs input requirements and output capabilities on a per conveyance channel basis.
4. Using the NPE conveyance channel topology and the values obtained from the previous step, for each necessary input field for any NPE:
  - a. Ensure the sending NPE generates this field. If not, the system is inconsistent so report a failure.
5. Following the formal message grammar, for each of the output-to-input mappings obtained in the previous step, map the field communicated between the NPEs into the next available set of slots of appropriate number. Compaction of different data types occurs automatically as a result. For systems that include an NPE that is not configurable, the mapping of the fixed NPEs determines the final mapping. For these devices it is possible that no system-wide slot mapping exists. For configurable NPE mappings, the next available slot(s), within the constraints of the message grammar, MAY be chosen.
6. For each programmable or configurable NPE, communicate the resulting field-to-slot-group mapping for each conveyance channel of that NPE.

### 3.1 Alignment of Fields and Payloads

The SAE MUST honor any requirements from an NPE for field or payload alignment within the message. In particular, the SAE MUST ensure that the message header, message trailer, and message payload part of the message each are an integral number of *message granules*. The message granule is defined as the larger of the processing or conveyance granule, when the message contains a trailer, and simply the conveyance granule otherwise.

The SAE can ensure the message header and trailer meet this requirement through the use of ignored slots. Since the SAE cannot determine the size of the payload, it MUST ensure the existence of message payload padding if the payload is not guaranteed to be an integral number of conveyance granules.

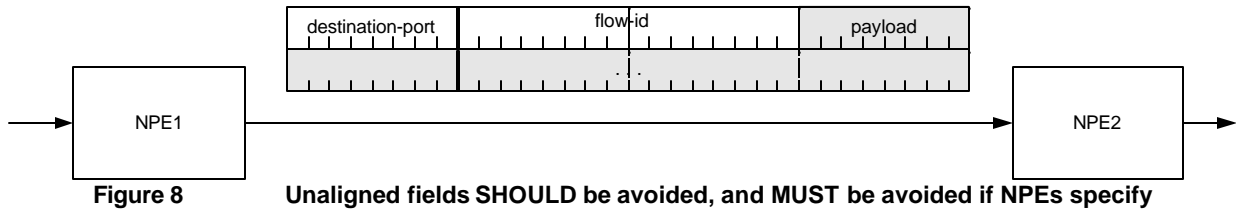
It is beyond the scope of the SAE to determine whether the message itself starts at a properly aligned location in the conveyance channel. Instead, the SAE only ensures that alignment is enforced with respect to the start of the message. Any use of the term 'aligned' in this specification is assumed to carry this qualification unless explicitly noted otherwise. It is RECOMMENDED that the mapping of the messaging layer into a conveyance do so in a manner that aligns the start of the message to the conveyance granule.

Certain message fields<sup>8</sup> MUST also be aligned--according to the processing granule--if requested by either the sending or receiving NPE. For example, when needed by an NPE, any 2-slot padded field MUST be aligned on a 2byte boundary with respect to the start of the message (i.e., start at an even slot number), any 4-slot padded field would be aligned on a 4-byte boundary, and so on. For example, Figure 7 contains aligned fields, where as the alternate arrangement of fields shown in Figure 8 makes the `flow-id` field unaligned<sup>9</sup>. This alternate arrangement SHOULD be avoided and MUST NOT be used if either NPE requires a processing granule larger than 1.

---

<sup>8</sup> Those defined as containing 'padding'

<sup>9</sup> This configuration also violates the field ordering constraints specified in Section 4. The example here is intended to only illustrate the field alignment issues and thus this point is ignored.



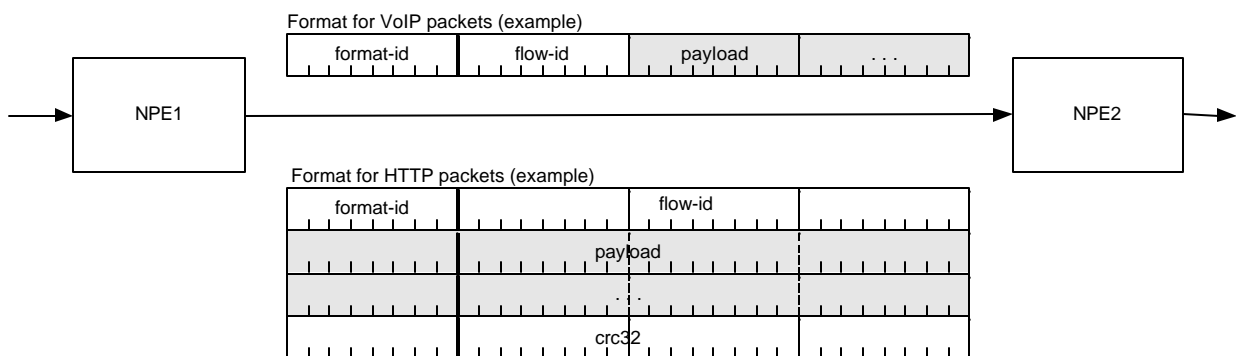
In certain situations the SAE can insert ignored slots to provide either field or payload alignment. For example, as shown in Figure 7, with a processing granule of 4-bytes, the SAE would be REQUIRED to insert an empty slot, as shown. The empty slot is shown in slot 2 to align the payload on a 4byte boundary and to place the destination-port into the least-significant byte of a 2-byte field.

### 3.2 Multiple Message Header Formats for a Single Conveyance Channel

At this point, only a single message header format per conveyance channel has been discussed. However, some applications require different message header formats for the same conveyance channel. For example, the system might need one set of fields for traffic of type A and a second set of fields for traffic of type B. If both types of traffic were present on the same conveyance channel, then, with the above solution, the system would need to be able to create a message header with fields for both sets of traffic types. The receiving NPE would then pick and choose which attributes to use for any given packet. Such a solution is both wasteful of bandwidth, and increases the processing necessary on both of the NPEs involved.

Multiple message header formats per-conveyance channel is achieved by a field that represents the format of both the message header and OPTIONAL trailer (called the format-id). For each packet transmitted across a conveyance channel, a unique format-id MUST be placed in the first slot within the message header. Note that for NPEs that support the ability to configure certain fields to a constant value, the implicit format-id can be removed from the message for conveyances requiring only a single message format.

For example, Figure 9 shows two NPEs communicating. Across the single conveyance channel between NPE1 and NPE2 the majority of messages are small VoIP packets and only carry an 8-bit flow-id. However, occasionally, larger HTTP packets are also sent across this same conveyance channel. These HTTP packets require a larger 3 byte flow-id and a crc-32 field in the trailer.



In order for NPE2 to discern the different types of message header and trailer formats arriving across this single conveyance channel, the format-id field is inserted into all packets. Format-id is a 1-byte quantity and MUST be placed into slot 0 when the format-id is not implicit in the message. Each format-id value indicates the format of the remaining slots in the message header and trailer for these messages.

So for example, for all VoIP packets, the first slot in the message header contains a configured value (as determined by the SAE) that specifies that the remaining slots are formatted as follows: Slot 1 represents the `flow-id`. Likewise, a different configured `format-id` is given to all HTTP traffic and is formatted as follows: Header slots 1 through 3 represent the `flow-id` and trailer slots 0 through 3 represent the `crc-32`.

The basic operation of NPE2 is to first read slot 0 (the `format-id`), and use this information to reference configuration information provided by the SAE to correctly parse the message header and trailer and determine the content of the remaining slots.

How certain types of traffic are mapped to different message header formats is beyond the scope of this implementation agreement. The SAE is responsible for deciding when and how to use multiple message header formats.

## 4. Formal Message Format Definition

This section contains the formal definition of message format, including standard message fields, their relative ordering restrictions, sizes (number of slots), and semantic meanings. These standard message fields have been grouped into categories for convenience of presentation. By restricting the possible orderings of message fields, simpler NPE devices can be built. With a few limitations, the SAE can mix any of these standard message fields for an individual message header or trailer. These limitations are illustrated in the syntax graphs and also listed within the description section of the affected standard message fields.

Each vendor implementing this messaging specification MUST supply a *score card* (in the format of the SAE grammar) indicating which of, and in what sizes, these standard message fields are supported by their NPE. System designers can then use this information to determine compatibility between different NPEs and design systems using NPEs from various vendors.

### 4.1 Basic Message Parts

This section presents the format of the message header, trailer, and payload using syntax graphs<sup>10</sup> to graphically illustrate both field, as well as message part, ordering. This section does not describe the semantic meaning of each field, rather each standard field type is described in Section 4.4.

Figure 10 shows the syntax graph for a message. A message is composed of either a message header only, or a message header and payload with an OPTIONAL trailer. In all instances, a message header SHOULD have at least one explicit field, the payload MUST have at least one explicit octet, and the trailer, when present, MUST have at least one explicit field. Each of these message *parts* (header, payload and trailer) MUST be an integral number of message granules (see Section 3.1). Each of these parts is further refined in subsequent sections.

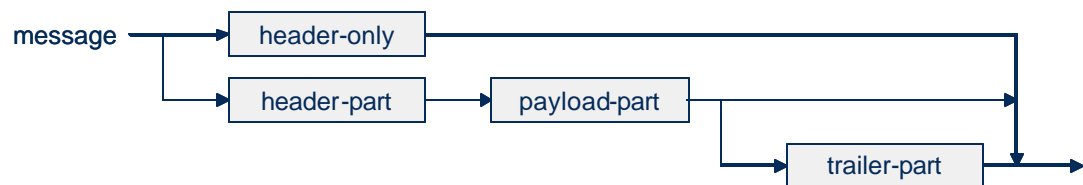
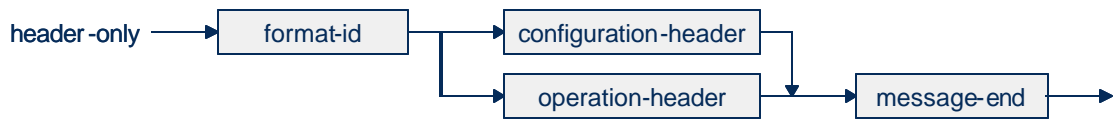


Figure 10 The syntax graph for a message

Figure 11 shows the syntax graph for the *header-only* portion of a message. All message headers MUST contain a `format-id`, which, as shown later, can be implicit and thus MAY not physically be part of the message. Following the `format-id`, messages with only headers contain either configuration or

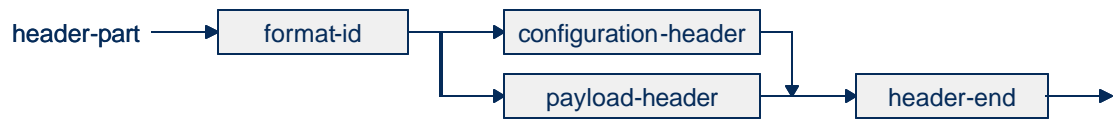
<sup>10</sup> A syntax graph notation guide is provided in Annex B.

operational fields, followed by a terminator. For the case of *header-only*, the terminator is for the entire message.



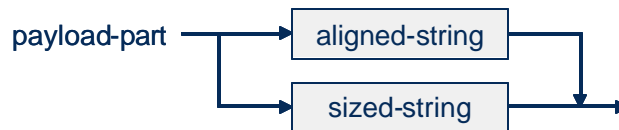
**Figure 11** The syntax graph for header-only (i.e., header without a payload or trailer)

Figure 12 shows the syntax graph for the header when included with a payload. Like the *header-only* syntax graph, headers with payloads MUST contain a `format-id`, which can be implicit. Following the `format-id`, headers with payloads contain either configuration or payload fields, followed by a terminator. The *header-part* terminator is for the header itself. Notice the difference between the two headers are in the terminators (one is for the message, the other for the header) and in the types of fields that belong in the non-configuration header. Messages without payloads have a restricted number of legal non-configuration header fields. Specifically, those fields not appropriate without a payload are not part of the *operation-header* but are part of the *payload-header*.



**Figure 12** The syntax graph for header-part (i.e., header with a payload and optional trailer)

Figure 13 shows the syntax graph for the *payload-part* of the message. A payload is a string (i.e., a series of bytes whose values are unknown to the SAE) of length that MUST always be an integral number of message granules (*aligned-string*), or is a string that contains padding and a length field in which the total size MUST be an integral number of message granules (*sized-string*).



**Figure 13** The syntax graph for payload-part

Finally, to complete the highest level of syntax graphs, Figure 14 shows the format of the *trailer-part* of a message. A trailer has, OPTIONALLY, payload fields, much like the payload fields of a *header-part*, and MUST contain the message terminator fields.

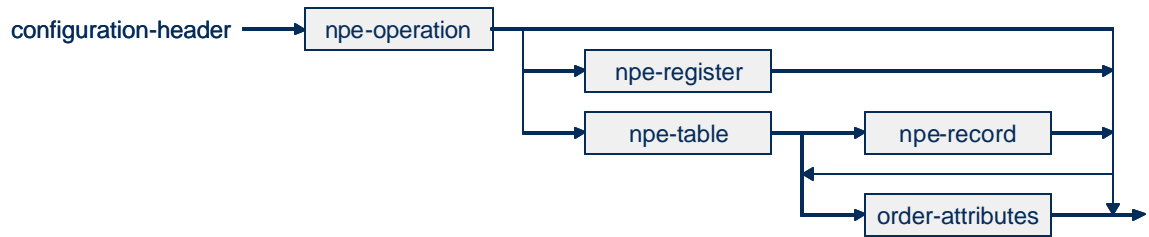


**Figure 14** The syntax graph for trailer-part

### 4.1.1 Header and Trailer Formats

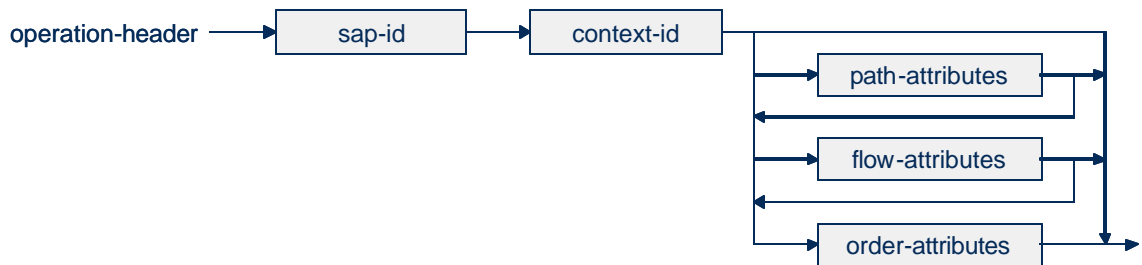
This section covers the legal constructions for the header and trailer parts of messages. Figure 15 shows the *configuration-header* syntax graph. As illustrated, a configuration header MUST contain an `npe-operation` field, OPTIONALLY followed by either an `npe-register` or `npe-table`, and OPTIONALLY one of the ordering attribute fields (e.g., timestamps or sequencing fields). In the case of the configuration header specifying an `npe-table` access, the `npe-record` field MUST be specified to indicate a particular entry within the table. Thus, when specifying a configuration operation, one can

simply specify the operation to perform (e.g., reset), the operation to perform with respect to a particular register (e.g., read register), or the operation to perform with respect to a particular table entry (e.g., modify the 5<sup>th</sup> element of a table).



**Figure 15** The syntax graph for configuration-header

Figure 16 shows the syntax graph for the *operation-header*. Recall, a message containing only a header consists of either a *configuration-header* or an *operation-header*. *Operation-headers* are used to carry fields associated with network traffic (i.e., non-configuration traffic), but not necessarily associated with the payload. All *operation-headers* MUST contain a service access point ID (*sap-id*), which specifies an application on top of the messaging layer, and a context ID (*context-id*) which specifies a SAP-specific state, or context. Like the *format-id*, these fields can be missing from the physical message. For example, if only one implicit SAP endpoint is using the associated conveyance channel, the *sap-id* MAY be implicit. Similarly, if the SAP has only one *context-id*, or the *context-id* is available from the conveyance, then the *context-id* MAY be implicit. Following the *sap-id*, the *operational-header* contains (individually optional, but in order) path, flow, and ordering attributes. For example, path attributes include ingress and egress endpoint information, flow attributes include flow, QoS, and state identifiers, and ordering attributes include timestamps and sequence numbers. Each of these syntax graphs are covered in detail in subsequent sections.



**Figure 16** The syntax graph for operation-header

Figure 17 shows the syntax graph for the *payload-header*. *Payload-headers* are a super-set of *operational-headers*. *Payload-headers* are an *operational-header* followed, optionally, by payload attributes. Payload attributes contain fields that are only appropriate when the message contains a payload.



**Figure 17** The syntax graph for payload-header

Figure 18 shows the syntax graph for the *payload-trailer*. Legal trailer fields are restricted in comparison to headers. In particular, configuration fields and operational fields MUST NOT be present in trailers. Instead, *payload-trailers* contain, OPTIONALLY, payload attributes followed OPTIONALLY by order attributes. Notice a trailer MUST NOT be empty. Instead the top-most message syntax graph allows the entire trailer to be skipped. In this case, the trailer is referred to as not present as opposed to empty.



Figure 18 The syntax graph for payload-trailer

4.1.1.1 Header Components

This section covers syntax graphs for attributes contained only in headers. The header syntax graphs that correspond to standard message fields are covered in a subsequent section.

Figure 19 shows the *path-attributes* and related syntax graphs. If included in the operational header, egress point fields MUST come before ingress point fields. Both egress and ingress points contain a port ID and, optionally, a channel ID, in that order. As detailed later, path attributes correspond to the path of the message through the system, not (necessarily) between two directly connected NPEs.

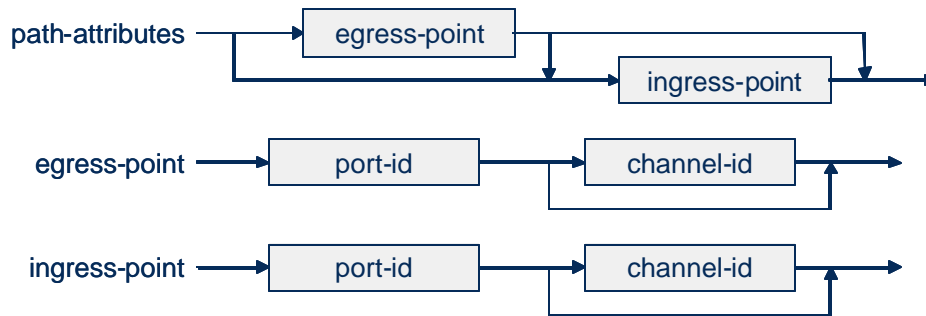


Figure 19 The syntax graphs for path-attributes, egress-point, and ingress-point

Figure 20 shows the *flow-attributes* syntax graph. Flow attributes MUST contain a *flow-id*, followed OPTIONALLY by a *flow-qos* field and then OPTIONALLY by a *flow-state* field, in that order. The meaning of each of these standard fields is provided in a subsequent section.

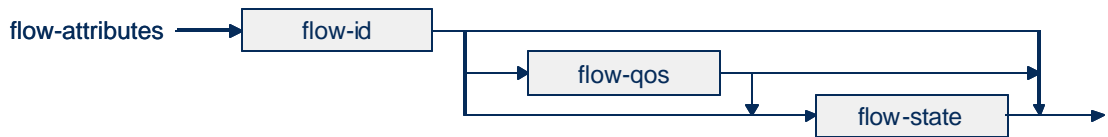


Figure 20 The syntax graph for flow-attributes

Figure 21 shows the syntax graph for header-end. The end of a header MUST contain OPTIONAL padding as well as, OPTIONALLY, exactly one of two integrity checks: a 1's compliment 16-bit checksum (*checksum-16*), or a *crc-32*. In this figure the value *z* represents the message granule, in bits.

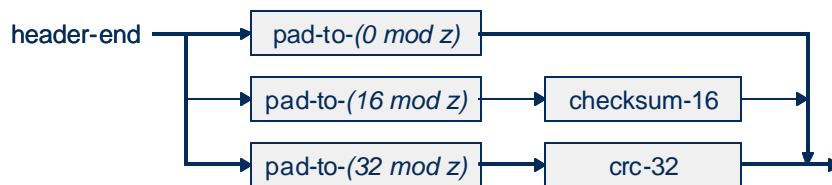
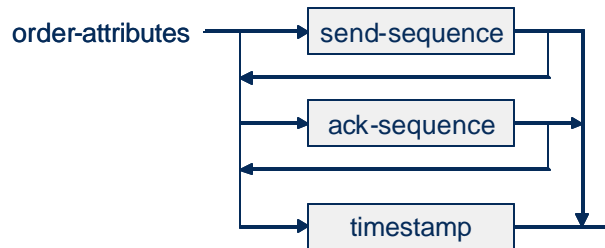


Figure 21 The syntax graph for header-end (z is the message granule in bits)

4.1.1.2 Header and Trailer Components

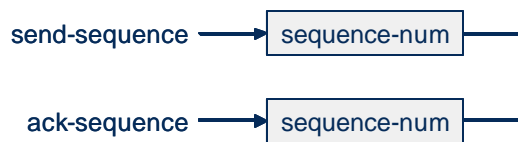
This section covers syntax graphs for attributes contained in both headers and trailers.

Figure 22 shows the *order-attributes* syntax graph. Order attributes can contain sequencing fields (*send-sequence* and *ack-sequence*) as well as a *timestamp*. Each of these fields can be **OPTIONALLY** present in either a header or trailer. When order attributes are present, they are relative to the *flow-id*, if present. Otherwise, the order attributes are relative to the context ID, SAP ID, and **OPTIONALLY** the payload, as defined by the application. For configuration-headers, order attributes implicitly apply to the single 'flow' of configuration. If the *flow-id* field is not present in a message with order attributes, the exact relationship of the order attributes to other parts of the message is beyond the scope of this agreement.



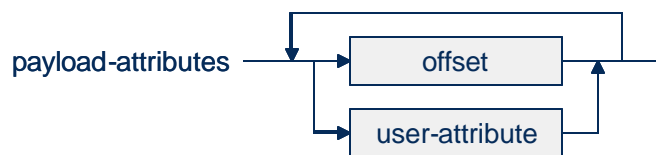
**Figure 22** The syntax graph for *order-attributes*

As shown in Figure 23, both the *send-sequence* and *ack-sequence* non-terminal nodes contain sequence numbers (with different semantic meanings, as defined in Section 4.4).



**Figure 23** The syntax graphs for *send-sequence* and *ack-sequence*

Figure 24 shows the syntax graph for *payload-attributes*. Payload attributes consist of one or more *offsets* and *user-attributes*. While it is likely that applications will associate semantic meanings with *offset/user-attribute* pairs, it is beyond the scope of this specification to define when an *offset* is associated with a preceding or following *user-attribute* field. Together these fields represent some application-specific processing that should occur on the payload of the message.



**Figure 24** The syntax graph for *payload-attributes*

Figure 25 shows the *message-end* syntax graph. *Message-end* is defined in a manner similar to *header-end*, except the end of the message is padded to the conveyance granule in bits (*x*) instead of the message granule. This padding scheme is used because no message processing occurs beyond the end of the message so padding the end of the message to the message granule, which accounts for the processing granule, is unnecessary.



Figure 25 The syntax graph for message-end (x is the conveyance granule in bits)

## 4.2 Containers

This section defines the three fundamental containers that compose all message parts. Containers have the only non-null terminal entries in the syntax of a message and are used to construct the data types (Section 4.3) and message fields (Section 4.4) of messages.

The three types of containers are slot, string and pad.

Slot containers, as shown in Figure 26, represent any values in the message header or trailer that are specified by NPF messaging (i.e., the SAE determines the fields mapped into slot containers). The maximum sized slot container is defined by the SAE and MUST be at least as large as the largest specified field size. Currently this is 16.

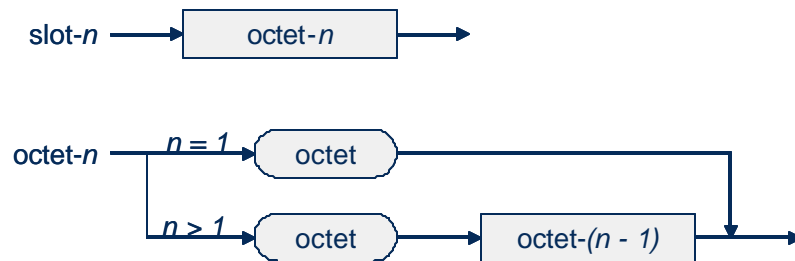
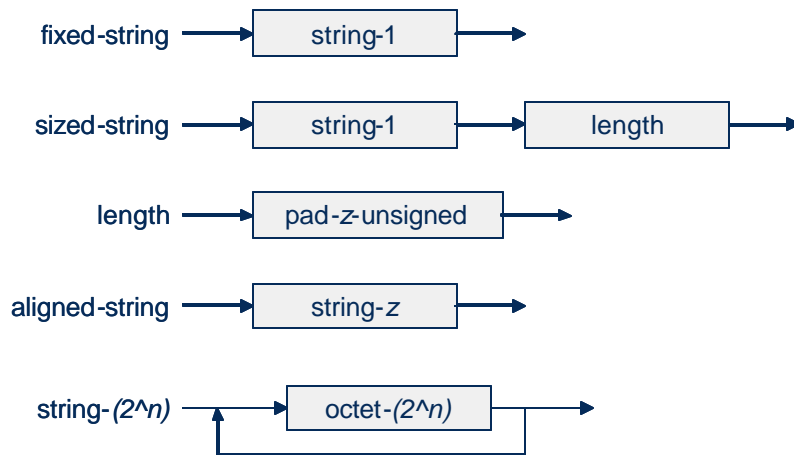


Figure 26 The syntax graph for slot containers

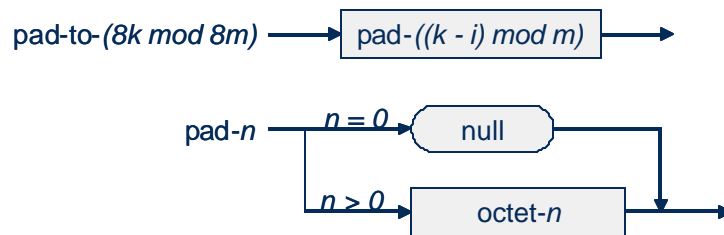
String containers, as shown in Figure 27, are transparent to NPF messaging, but contain application-specified values. There are three types of strings: fixed, sized and aligned. *Fixed-strings* can be of any length, but the length must be known at configuration time by the SAE. Both *sized-string* and *aligned-string* MUST have a total length that is an integral multiple of the message granule (z). In the case of *sized-strings*, this is ensured through the use of padding and a total length field. For *aligned-string*, the enforcement of this requirement is outside the scope of this implementation agreement. However, it is acceptable (and reasonable) for aligned strings to contain padding that is opaque to the messaging layer but is known to the application implicitly, or explicitly through fields in the conveyance or messaging headers.

The length field is the number of bytes in the payload of the message. The payload, in this definition, does not include the message header or trailer, or any padding in the payload part. The sending NPE must send the appropriate number of optional padding bytes after the message payload and before this field to ensure the payload part (payload, padding, and this length field) are an integral number of message granules.



**Figure 27** The syntax graph for string containers

Pad containers, as shown in Figure 28, hold no data and their contents MUST be ignored by the receiving NPE and SHOULD be set to zero by the sending NPE. The largest sized pad is determined by the largest supported message granule. Padding itself is simply nothing (i.e., null) or a series of content-unspecified octets. Pad-to containers are defined in terms of the input variables  $k$  and  $m$  (both in bits), and the current number of octets in the message,  $i$ . Here  $k$  and  $m$  are typically specified in terms of the system ( $z$ ), processing ( $y$ ), or conveyance ( $x$ ) granules and  $i$  is added to ensure that the padding goes up to the next given granule.



**Figure 28** The syntax graph for pad containers

### 4.3 Data Types

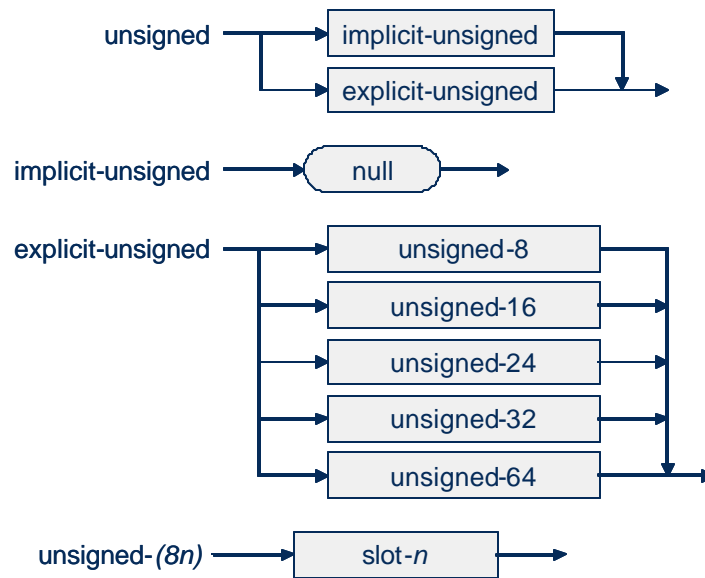
This section defines messaging data types. These data types are constructed out of containers and are used in the construction of the message fields (Section 4.4). Data types are either numeric or bit vectors.

#### 4.3.1.1 Numeric Data Types

The numeric data types represent integers (both signed and unsigned) of various sizes, with and without leading padding.

Figure 29 shows the unsigned, unpadded data types. These data types are used to support fields requiring unsigned arithmetic, ordered comparisons, and fields which will typically be used as indices into tables. Unsigned data types are either implicit or explicit. An implicit unsigned data type is represented with a null terminator and thus does not physically appear in the message, but its value is implicitly defined by the SAE or available from the conveyance (or another lower) layer. For example, the `format-id` is always present in a message header, but may be implicitly defined if only one message format is being used on the conveyance endpoint.

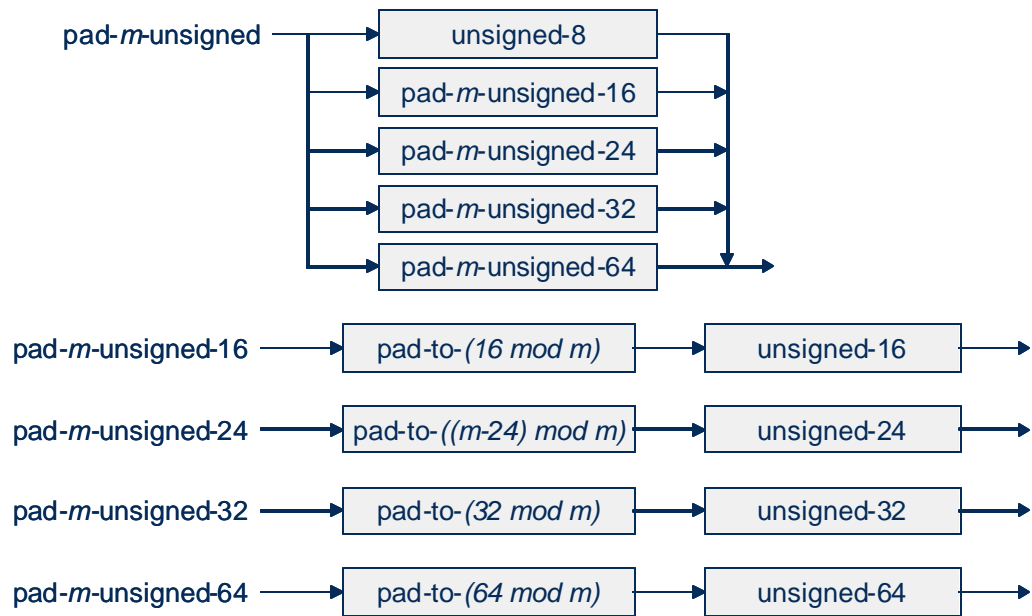
An explicit unsigned data type can be one of five different lengths, specifically 8, 16, 24, 32, and 64 bits. Each of these explicit unsigned data types are held in slot containers of appropriate size.



**Figure 29 The syntax graph for unsigned, unpadding data types**

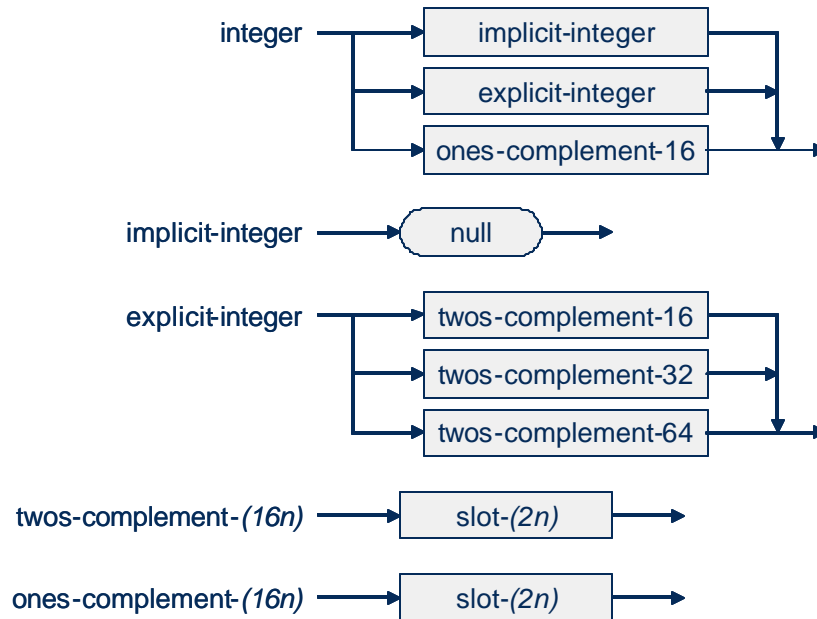
Figure 30 shows the unsigned, padded data types. These data types are used to support aligned unsigned data fields. Typically the padding is based on the processing granule to ensure natural alignment for the processing of an unsigned, padded data type. For example, padding a 16-bit data type to a 32-bit granule results in (potentially) an additional 16-bits of padding placed in front of the data type, depending on where the start of the data type was with respect to the start of the message.

The padding for each of the possible unsigned data type sizes depends on both the data size and the padding granule. The only case of interest is the padding of 24-bit data types to a given size  $m$ . In this case, because 24 is not a power of two, the padding is based on the equation  $(m-24) \bmod m$ . Here mod is defined in the traditional mathematical sense (as opposed to the common programming representation of remainder), which is: the amount by which a number exceeds the largest integer multiple of the divisor that is not greater than that number. This is critical to understand when dealing with mod for negative numbers. For example, if  $(m-24) \bmod m$  was computed for  $m = 16$ , the result would be  $(16 - 24) \bmod 16$ , or  $-8 \bmod 16$  which is congruent to 8. The result would be up to 8-bits of padding added to the front of the data type.



**Figure 30 The syntax graph for unsigned, padded data types**

Figure 31 shows the syntax graphs for the signed, unpadded data types.<sup>11</sup> In a manner analogous to the unsigned, unpadded data types, signed data types are used for signed arithmetic and comparison operations. Similarly, signed data types are either implicit or explicit. However, unlike the unsigned types, explicit signed types can either be 2's complement numbers (of three different sizes) or a 16-bit, 1's complement number. The 1's complement number is added to accommodate typical Internet-style checksums.



**Figure 31 The syntax graph for signed data types**

<sup>11</sup> Signed, padded data types are not defined in the specification because no field currently requires them, but such data types could be trivially added to this specification.

### 4.3.1.2 Bit Vector Data Types

For data types that are not used for arithmetic, ordered comparisons, or indexing into tables, the bit vector data types are defined. Bit vectors are used for bitwise logic operations as well as unordered comparisons. Figure 32 shows the syntax graphs for the bit vector data types. Again, both implicit and explicit bit vectors are supported. For the explicit bit vectors, seven different sizes are defined, including a 48-bit length data type for Ethernet MAC addresses.

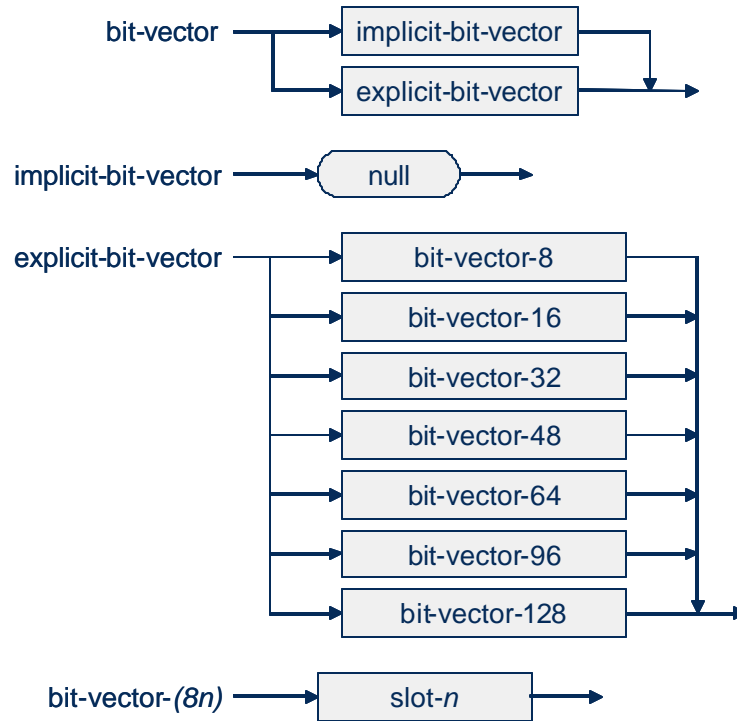
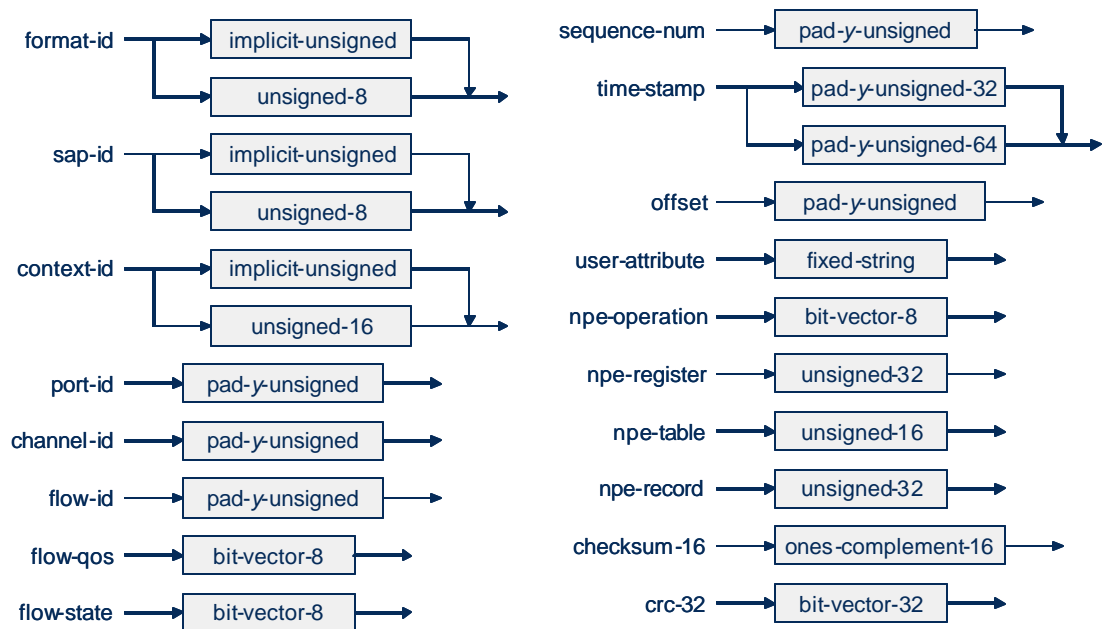


Figure 32 The syntax graph for bit vector data types

## 4.4 Message Fields

This section contains syntax graphs for the message fields as well as descriptions of the semantic meanings associated with each field. Message fields are grouped and ordered within message headers and trailers as defined in Section 4.1, and are constructed out of the data types in Section 4.3.

Figure 33 shows syntax graphs for all of the standard message fields. The following sections describe each field in detail.



**Figure 33** The syntax graph for all message fields

## 4.4.1 Identifiers

### 4.4.1.1 Format Identifier

**SAE name:** `format-id`

**Data type:** `implicit-unsigned` or `unsigned-8`

**Valid settings:** 0 – 255

**Description:** Specifies which format to use to parse the remaining slots of the message header and trailer. For any given `format-id` value, the remaining message parts are in a known order and of known fields (header and trailer).

The greater the number of formats supported, the greater the number of simultaneous message header formats that can be supported across a single conveyance channel. `Format-id` MUST be explicit when more than one message format is defined for a single conveyance channel. `Format-id` MUST be assigned to slot 0 when explicitly defined. The value of 0 is reserved for the in-band configuration message.

### 4.4.1.2 Service Access Point Identifier

**SAE name:** `sap-id`

**Data type:** `implicit-unsigned` or `unsigned-8`

**Valid settings:** 0 – 255

**Description:** Specifies the application endpoint of the message. `Sap-id` MAY be implicitly defined when only a single application endpoint is using the conveyance channel. `Sap-id` MUST be explicit when more than one application endpoint is using the conveyance channel, and `sap-id` MUST be the first field in the operational header when present.

### 4.4.1.3 Context Identifier

**SAE name:** `context-id`

**Data type:** `implicit-unsigned` or `unsigned-16`

**Valid settings:** 0 – 65535

**Description:** Specifies the application-endpoint-specific state of the message. `Context-id` SHOULD be implicitly defined when the application endpoint does not require it, or when it is available from the conveyance (or another lower) layer. `Context-id` MUST be explicit when the application endpoint requires it to be so, and `context-id` MUST be the second field in the operational header when the `sap-id` is not implicit, and MUST be the first field in the operational header otherwise..

## 4.4.2 Flow Attribute Fields

### 4.4.2.1 Flow Identification Number

**SAE name:** `flow-id`

**Data type:** padded to processing granule, unsigned

**Valid settings:** 0 – n, where n is full magnitude of data types

**Description:** Specifies a value to map the message payload to a unique transfer of packets, a flow. The determination of what constitutes a flow is outside the scope of this implementation agreement.

### 4.4.2.2 Flow Quality of Service Level

**SAE name:** `flow-qos`

**Data type:** bit-vector-8

**Valid settings:** Defined by the application

**Description:** Specifies the quality of service associated with the flow, as defined by the application.

### 4.4.2.3 Flow State / Status

**SAE name:** `flow-state`

**Data type:** bit-vector-8

**Valid settings:** Determined by interoperating vendors to meet the requirements of the system architecture

**Description:** Specifies the known state or status of a flow. Provides a method to pass non-NPF defined information about a flow within an NPF defined type. For example, two NPEs could use this field to communicate the state (SYN received, established, FIN received, etc.) of a TCP stream, thus enabling flow-based resources to be properly managed and reused between NPEs.

## 4.4.3 Order Attributes

### 4.4.3.1 Send Sequence Number

**SAE name:** `send-sequence`

**Data type:** padded to processing granule, unsigned

**Valid settings:** 0 – n, where n is full magnitude of the data type. Once the full magnitude of the field is reached, the value rolls over to zero and continues counting for normal operation.

**Description:** Specifies an incrementing number of messages (starting from 0), relative to the flow, that has been generated by the sending NPE. The receiving NPE can use the sequence number to look for loss of packets (or duplication). When two messaging links are paired to build a bi-directional link, this field allows for a windowing form of “flow control”.

### 4.4.3.2 Sequence Acknowledge Number

**SAE name:** `ack-sequence`

**Data type:** padded to processing granule, unsigned

**Valid settings:** 0 – n, where n is full magnitude of data type.

**Description:** Specifies the last Sequence Number processed by the receiving NPE relative to the flow. All messages up to this sequence number were correctly received and processed. If the sending

NPE receives a `ack-sequence` with a value less than the last value sent, the sender **MUST** assume loss of messages. Any other messages beyond this sequence number were dropped by the receiving NPE due to an error, buffer overflow or other problem. The sending NPE **MAY** continue to transfer additional messages with the sequence number one past the sequence number returned in this field. The sequence acknowledge field **MUST** be the same size as the corresponding sequence number.

#### 4.4.3.3 Time Stamp

**SAE name:** `time-stamp`

**Data type:** padded to processing granule, unsigned-32 or unsigned-64

**Valid settings:** 0 – n, where n is full magnitude of data type.

**Description:** Specifies a “system tick” from the transmitting NPE at the time the packet was queued for transmission. The unit of system ticks is vendor specific, but can be normalized by control plane software. The system tick value rolls over and continues to increment from zero.

### 4.4.4 Path Attributes

#### 4.4.4.1 Ingress Port Identifier

**SAE name:** `ingress-port-id`

**Data type:** padded to processing granule, unsigned

**Valid settings:** Determined by interoperating vendors to meet the requirements of the system architecture.

**Description:** Specifies the physical system ingress port that the packet was received on. Thus, this field typically corresponds to the ingress port of the network element on which the packet was received, which might be many NPEs away from the NPE receiving this field.

#### 4.4.4.2 Ingress Channel

**SAE name:** `ingress-channel`

**Data type:** padded to processing granule, unsigned

**Valid settings:** 0 – n, where n is the largest channel number.

**Description:** Specifies the physical ingress channel within the `ingress-port-id`. An `ingress-port-id` field **MUST** be present when this field is present in a message header.

#### 4.4.4.3 Egress Port Identifier

**SAE name:** `egress-port-id`

**Data type:** padded to processing granule, unsigned

**Valid settings:** Determined by interoperating vendors to meet the requirements of the system architecture

**Description:** Specifies the physical system egress port that the packet is requested to be output on by the transmitting NPE (i.e., final NPE in the system). Thus, this field typically corresponds to the destination port of the network element on which the packet is to be transmitted, which might be many NPEs away from the NPE receiving this field.

#### 4.4.4.4 Egress Channel

**SAE name:** `egress-channel`

**Data type:** padded to processing granule, unsigned

**Valid settings:** 0 – n, where n is the largest channel number

**Description:** Specifies the physical egress channel within the `egress-port-id`. A `egress-port-id` field **MUST** be present when this field is present in a message header.

## 4.4.5 Payload Attributes

### 4.4.5.1 Offset

**SAE name:** `offset`

**Data type:** padded to processing granule, unsigned

**Valid settings:** 0 – maximum network packet length supported by receiver.

**Description:** Specifies an offset, in bytes, into the message payload. Used either in isolation or in conjunction with `user-attribute` to reference a particular starting location in the payload to operate on.

### 4.4.5.2 User-attribute

**SAE name:** `user-attribute`

**Data type:** fixed-string

**Valid settings:** Determined by interoperating vendors to meet the requirements of the system architecture.

**Description:** Specifies an application-dependent value for the receiving NPE to OPTIONALLY utilize in conjunction with the preceding `offset` field for processing the message payload. For example, the `user-attribute` could be a “pointer” to a URL replacement string within the receiving NPE memory, or a hash index value for the NPE to use in its lookup function. This field is a string since the contents are of unknown format to the SAE. However, the size of each `user-attribute` must be fixed and known to the SAE.

## 4.4.6 Configuration Fields

When configuration fields are used within a message transfer, the message MAY NOT contain “real” network traffic. The data in a configuration message (both message header and message payload) provides NPE in-band information (e.g., configuration, status, statistics, alarms, timeouts) either going to, or coming from, an NPE. These configuration message transfers are peer-to-peer (i.e., neither NPE is the master), “fire and forget” messages. The NPE on the “source” side of a conveyance MAY transmit messages while the NPE of the “destination” side of the conveyance MUST receive the messages. The format of any data used for configuration within the message payload of any in-band message transfer is vendor dependant.

When a single (unidirectional) conveyance is used between two NPEs, the in-band message transfers MUST contain an `npe-operation` of type unidirectional request (to the destination NPE), or unidirectional event (from the source NPE). If a bi-directional message link (i.e., two unidirectional conveyances) is available between two NPEs, then additional `npe-operations` types MAY be used. These types include bi-directional request (to the destination NPE) and the various forms of bi-directional responses (from the source NPE) providing a success, or one of four error, indications in response to the originating request. While using bi-directional message links, the function of each unidirectional conveyance remains fundamentally the same. The message transfers remain peer-to-peer: either source MAY transmit in-band message transfers, and each destination MUST receive in-band message transfers.

NPE configuration messages compete with non-configuration messages for bandwidth on the physical data link. The factors influencing the impact of this competition include: the volume of NPE configuration information sent to or generated from an NPE, the priority of these NPE configuration messages verses “network traffic”, queuing of NPE configuration information, loss of configuration information messages, or loss of “network traffic”. Information about how this bandwidth competition is handled is not covered by this implementation agreement.

### 4.4.6.1 NPE Operation

**SAE name:** `npe-operation`

**Data type:** bit-vector-8

**Valid settings:** See Table 1 and Table 2

**Description:** Specifies a common set of operations that an NPE can “execute”, “execute” and then respond to, or asynchronously “execute/generate an event”. `npe-operation` is a bit vector that contains two pieces of information about the operation: the type and the code. The three high order bits define the “type” of `npe-operation` and the low order five bits define the specific operation “code”.

The high order portion of the `npe-operation` bit vector has been broken up into four **types** covering eight ranges: two for requests, one for events, and five for responses (see Table 1). When a single unidirectional conveyance is used, the two permitted `npe-operation` types are the unidirectional request (i.e., silently “execute” the received operation) and the unidirectional event (i.e., “execute/generate” an asynchronous event notification). When unidirectional conveyances are combined to form a bi-directional message link, two additional `npe-operation` types are also permitted. These types are: bi-directional request (i.e., “execute” and then respond to the received operation) and bi-directional responses (i.e., transmit a response, either OK or one of four general errors after “executing” the requested operation).

**Table 1. npe-operation Types**

NPE Operation Types (High Order 3 Bits)	Setting		NPE Operation Type Range			
	Hex	Binary	Range (low)		Range (high)	
Unidirectional Request	0	000	00	00000000	1F	00011111
Unidirectional Event	1	001	20	00100000	3F	00111111
Bi-directional Request	2	010	40	01000000	5F	01011111
Bi-directional Response OK	3	011	60	01100000	7F	01111111
Bi-directional Response Error: Unsupported Operation	4	100	80	10000000	9F	10011111
Bi-directional Response Error: Invalid Setting	5	101	A0	10100000	BF	10111111
Bi-directional Response Error: Read Only Access	6	110	C0	11000000	DF	11011111
Bi-directional Response Error: Other	7	111	E0	11100000	FF	11111111

The low order portion of the `npe-operation` bit vector defines the **code** for the individual operation (see Table 2). The codes that range from *hello* to *remove* (hexadecimal values 0 through 9, inclusive) MUST only be used for requests and responses. The codes that range from *reset\_complete* to *statistics* (hexadecimal values 1C through 1F) MUST only be used for events. Each code is placed into one of four groups: system, register, table, or event. An NPE messaging implementation that chooses to any code from a particular group MUST implement all required codes for that group (see the status column for each of the codes within the group). For example, to support the register group, the implementation MUST support the codes *read* and *write*. The implementation for code *read\_and\_init* SHOULD be supported. In addition, the format of the message MUST match the options available to the group. For example, a message format that supports the register group MUST contain `npe-register` and MUST NOT contain either `npe-table` or `npe-record`. If this type of error does occur, and a bi-directional message link exists, the receiving NPE MUST respond with a bi-directional response error of unsupported operation, otherwise the message MUST be dropped.

**Table 2. npe-operation Codes and the Groups****NPE Operation Codes (Low Order 5 Bits)**

	Setting		NPE Operation Group		Legend
	Hex	Binary	Group	Status	
Hello	0	00000	System	R	X NOT Permitted
Reset	1	00001	System	R	
Read	2	00010	Register	R	O Optional R Required
Read_and_Init	3	00011	Register	O	
Write	4	00100	Register	R	
Initialize	5	00101	Table	O	
Get	6	00110	Table	R	
Set	7	00111	Table	O	
Insert	8	01000	Table	R	
Remove	9	01001	Table	R	
Reset_Complete	1C	11100	Event	R	
Alarm	1D	11101	Event	R	
Timeout	1E	11110	Event	R	
Statistics	1F	11111	Event	O	

Npe-operation codes are limited by the npe-operation type that is set in the bit vector. Tables 3 through 6 specify the valid npe-operation codes per type.

When an npe-operation is included in the message header, it may be followed by npe-table, npe-record, or npe-register (see Figure 15). The optional message payload accompanying these fields contains either the vendor specific data to write, insert, etc. to the NPE, the vendor specific data read, etc. from the NPE, or the vendor specific asynchronous event data for an alarm, statistics, etc. sent by the NPE. The payload columns of Table 3, Table 4, Table 5, and Table 6 also specify whether a vendor specific message payload MUST exist (R), MAY exist (O), or MUST NOT exist (X). When a message payload is present with an npe-operation, the payload MUST contain the vendor specific information that supports the npe-operation. Several message payload contents examples include: a 144-bit key followed by 32-bits of associated data for a TCAM to add an entry to a search table, one 16-byte record of a configuration file (a table) to write, or the contents read from a 32-bit register.

Valid npe-operation codes for type unidirectional request MAY be sent at any time to the receiving NPE and are listed in Table 3. The receiving NPE “executes” the operation using the remainder data in the message header and message payload, if any. The receiving NPE MUST NOT respond to this operation, even if a bi-directional message link is available. If the npe-operation fails to “execute” for any reason (e.g., invalid message header/payload contents or lack of NPE resources) the message MUST be discarded. Any error capabilities the NPE provides to record or report this error are not covered by this implementation agreement.

**Table 3. npe-operation Codes for Types: Unidirectional Requests****Unidirectional Requests**

	Payload	Legend
Hello	X	X NOT Permitted O Optional R Required
Reset	X	
Write	R	
Initialize	X	
Set	R	
Insert	R	
Remove	R	

Valid `npe-operation` codes for type unidirectional event MAY be sent at any time. Messages with these codes are sent asynchronously from the source NPE to the destination NPE. These codes are listed in Table 4. Vendor data associated with the event MAY be placed in the message payload. Unidirectional event messages are sent as determined by the vendor-dependant configuration/programming of the NPE and are not covered by this implementation agreement. The receiving NPE MUST NOT respond to a unidirectional event, even if a bi-directional link exists.

**Table 4. `npe-operation` Codes for Types: Unidirectional Events**

**Unidirectional Events**

	Payload		<b>Legend</b>
Reset_Complete	O		
Alarm	R	X	NOT Permitted
Timeout	R	O	Optional
Statistics	R	R	Required

Valid `npe-operation` codes for type Bi-directional Request MAY be sent at any time to the receiving NPE. These codes are listed in Table 5. The receiving NPE “executes” the operation using the remainder of the data in the message header and message payload, if any. After the operation completes within the receiving NPE, the receiving NPE MUST send a response on the opposite unidirectional message link back to the original sender.

**Table 5. `npe-operation` Codes for Types: Bi-directional Requests**

**Bi-directional Requests**

	Payload		<b>Legend</b>
Hello	X		
Reset	X	X	NOT Permitted
Read	X	O	Optional
Read_and_Init	X	R	Required
Write	R		
Initialize	X		
Get	X		
Set	R		
Insert	R		
Remove	R		

Valid `npe-operation` codes for any one of the five types of bi-directional responses (see Table 1) MUST only be sent back from the receiving NPE to the originating NPE in response to a bi-directional request. Any of the five bi-directional responses MUST NOT be sent unsolicited or as a response to a unidirectional request or a unidirectional event.

If the `npe-operation` completes successfully, the receiving NPE MUST return, to the originating NPE, a `npe-operation` type bi-directional response OK code. If a successful response contains a message payload, the message payload SHOULD contain the data read back after the operation completes. Otherwise, the message payload MAY be a copy from the request, if available.

Otherwise, when an error occurs with the operation, the receiving NPE MUST return the most relevant bi-directional response error code, as explained in the following list. If a failure response contains a message payload, the message payload SHOULD contain additional vendor-specific error information to help diagnose the problem.

1. Unsupported operation MUST be returned if the receiving NPE did not implement the requested `npe-operation`. Unsupported operation MUST also be returned if the `npe-operation` is a reserved setting. Unsupported (invalid) operation MUST also be returned if the `npe-operation` requested does not match the other configuration fields (or lack there of) in the remainder of the message format.

2. Invalid setting **MUST** be returned if the receiving NPE does not “have” the specified table, record within the table, or register specified by the remaining information within the message header.
3. Read-only access **MUST** be returned if the receiving NPE identifies the specified table, record within the table, or register specified by the remaining information within the message header to be a read-only “element” within the NPE. For example, a register *read\_and\_init* is requested on a read-only register; or a table *initialize* is requested for a read-only table.
4. Other **MUST** be returned if the receiving NPE encounters an error that is not covered by the prior three error cases. For example, the receiving NPE might have exhausted the number of entries allowed for the specified table prior to the *insert*; or the table might already be empty prior to the *remove*; or the entry data may not be valid for a *set*.

**Table 6. Npe-operation Codes for Types: Bi-directional Responses**

**Bi-directional Responses**

	Payload		<b>Legend</b>
Hello	X		
Reset	X	X	NOT Permitted
Read	R	O	Optional
Read_and_Init	R	R	Required
Write	O		
Initialize	O		
Get	R		
Set	O		
Insert	O		
Remove	O		

**Hello** (to the destination NPE) `npe-operation` code allows messages to be sent to the receiving NPE without any request for “work” to be performed. The receiving NPE **MUST** interpret this message to mean that the sending NPE is still functional. On a bi-directional message link, if a bi-directional request - hello is sent, the receiving NPE **MUST** return a bi-directional response OK - hello. If the hello operation round trip is completed, each NPE **MUST** interpret the messages to mean that the other NPE is alive and functional. When the `npe-operation` code is set to hello, the remainder of the message header **MUST NOT** contain any other configuration fields or a message payload.

**Reset** (to the destination NPE) `npe-operation` code allows the sending NPE to force the receiving NPE to reset and initialize itself. When received, the destination NPE **MUST** discard all other “work” in progress and then reset and initialize. If a bi-directional request - reset is sent, the receiving NPE **MUST**, as its final effort, return a bi-directional response OK - reset on the bi-directional message link informing the original sending NPE that the reset and initialize process is about to begin. When the `npe-operation` code is set to reset, the remainder of the message header **MUST NOT** contain any other configuration fields or a message payload.

**Read** (register) `npe-operation` code allows the sending NPE to request a copy of the register data stored in the specified location to be returned from the receiving NPE. Read operations, when present, **MUST** only be sent on a bi-directional message link. The requesting NPE sends a bi-directional request - read to the receiving NPE. The message payload of the request **MUST** be empty. The receiving NPE accesses the register data and returns a bi-directional response OK - read or the relevant bi-directional response error back across the opposite message link to the originating NPE.

**Read\_and\_Init** (register) `npe-operation` code allows the same functionality as the read operations plus the specified register data is returned to its initial value (usually zeros) once the current value is copied into the message payload. The receiving NPE **SHOULD** perform an atomic read-and-set of the requested data. If a failure occurs, and a bi-directional response error is returned and the specified data **SHOULD NOT** be altered.

**Write** (register) `npe-operation` code allows the sending NPE to overwrite data stored in the specified register location. The requesting NPE sends a unidirectional request or a bi-directional request to the receiving NPE. The message payload **MUST** contain the vendor specific register data to be written. The receiving NPE **MUST** use the data in the message payload to replace the current contents of the specified register location, unless there is an error in the remaining contents of the message header or payload.

**Initialize** (complete table, or single record within table) `npe-operation` code allows the sending NPE to select an entry within the receiving NPE to be returned to its initial value and/or state. The behavior is vendor dependant, but several examples include: if a record within a table is initialized, it is returned to its initial value after reset. If a table is initialized, it may reserve several internal resources and then initialize these resources to known values. A bi-directional response error: read-only access **MUST** be returned on a bi-directional message link if the specified element can not be updated.

**Get** (table entry, or record within table) `npe-operation` code, when used on a table, allows an existing entry to be read from the specified table. The receiving NPE is responsible to “find” the specified entry within the table and to return its contents in the vendor-dependant data in the message payload. A `get npe-operation` code, when used on a record in a table, allows the specified record to be read and placed in the vendor-dependant data in the message payload. Get operations **MAY** only be sent on a bi-directional message link. The requesting NPE sends a bi-directional request - `get` to the receiving NPE. The message payload of the request **MUST** be empty. The receiving NPE accesses the table entry or record data and returns a bi-directional response `OK - get` or the relevant bi-directional response error back across the opposite message link to the originating NPE.

**Set** (table entry, or record within table) `npe-operation` code, when used on a table, allows an existing entry (or some of the data within the existing entry) to be changed within the specified table. The receiving NPE is responsible to “find” the specified entry within the table and to alter it according to the vendor-dependant data in the message payload. A `set npe-operation` code, when used on a record in a table, allows the specified record to be replaced with the vendor-dependant data in the message payload. The behavior is vendor dependant, but several examples of failures where a bi-directional response error - other **SHOULD** be returned on a bi-directional message link include: the data in the message payload is not the same size as the table entry or the record within the table, or the new data would cause the table entry to duplicate an existing entry in the table. Duplicate records within a table **MAY** be an error, this is vendor dependant. A bi-directional response error - read-only access **MUST** be returned on a bi-directional message link if the specified element can not be updated.

**Insert** (entry into table) `npe-operation` code allows new entries to be added to the specified table. This is different from a `set npe-operation` code because the insert code does not specify a record number within the table, just the table itself. The receiving NPE is responsible to find the “correct” location for the entry within the table. The receiving NPE **MUST** allow redundant insert `npe-operations` to be performed on the same table to provide a means for the sending NPE to retry inserting an entry when it is not clear whether a prior attempt was successful. The error behavior is vendor dependant, but several examples of failures where a bi-directional response error - other **SHOULD** be returned on a bi-directional message link include: the table is full, or the entry contents are invalid.

**Remove** (entry from table) `npe-operation` code allows existing entries to be deleted from the specified table. The receiving NPE is responsible to “find” the specified entry within the table and remove it. This **SHOULD** cause any associated vendor resources to be released for future use. The receiving NPE **MUST** allow redundant remove `npe-operations` to be performed on the same table to provide a means for the sending NPE to retry deleting an entry when it is not clear whether a prior attempt was successful. The error behavior is vendor dependant, but several examples of failures where a bi-directional response error - other **SHOULD** be returned on a bi-directional message link include: the table is empty, or the entry contents are invalid.

**Reset\_Complete** (from the source NPE) `npe-operation` code allows the reset NPE to provide an indication that is initialized and is now fully functional. The NPE **MUST NOT** send a enidirectional event - `reset_complete` code unless all of its vendor-specific self-diagnostics have passed. The NPE **MUST**

NOT send a unidirectional event - reset\_complete code after any other in-band messages have been sent.

**Alarm** (from the source NPE) `npe-operation` code allows the NPE to send out an asynchronous message to inform the destination NPE that an alarm condition has occurred. Examples of unidirectional event - alarm conditions include: loss of signal on another interface, lack of internal resources, error handler invocation, or an external memory failure. The message payload MUST contain detailed information about the alarm. This information is vendor dependant.

**Timeout** (from the source NPE) `npe-operation` code allows the NPE to send notification of a timeout to the destination NPE. Examples of a unidirectional event - timeout include: an entry in table n has expired, or a bi-directional response from an earlier b-directional request has not been received. The message payload MUST contain detailed information about the timeout. This information is vendor dependant.

**Statistics** (from the source NPE) `npe-operation` code allows the NPE to send periodic statistics information to the destination NPE. Examples of a unidirectional event - statistics include: table entry n processed m packets during the last 60 seconds, or register x has recorded y packets dropped during the last second. The message payload MUST contain detailed information about the statistics. This information is vendor dependant.

**Reserved** operations (see Table 7) SHOULD not be used. These settings remain available for future expansion of this implementation agreement. Reserved `npe-operation` use is outside of the scope of this version of the implementation agreement. The NPF requests that vendors submit their use of these settings to the NPF to allow future versions of this implementation agreement to convert excepted operations to `npe-operations`, extending the capabilities of this implementation agreement while allowing a greater degree of vendor interoperability.

#### Reserved NPE Operations

Range (low)		Range (high)	
Hex	Binary	Hex	Binary
0A	00001010	1B	00011011
2A	00101010	3B	00111011
4A	01001010	5B	01011011
6A	01101010	7B	01111011
8A	10001010	9B	10011011
AA	10101010	BB	10111011
CA	11001010	DB	11011011
EA	11101010	FB	11111011

Table 7: Reserved `npe-operation` Ranges

#### 4.4.6.2 NPE Table

**SAE name:** `npe-table`

**Data type:** unsigned-16

**Valid settings:** Determined by the NPE vendor

**Description:** Specifies the "table" within the NPE that the `npe-operation` is targeting. For example, if the NPE supports a firmware load, table 100 might reference the NPEs firmware file, and a following `npe-record` would specify which record within the table (file). The `npe-table` number alone provides enough information for a receiving NPE to insert, remove, or get an entry within the table.

The message payload, when used without `npe-record`, is either the full table or one entry in the table. The size, bit order, number of fields within a table entry, and meaning of each table entry is vendor dependent.

#### 4.4.6.3 NPE Record

**SAE name:** `npe-record`

**Data type:** unsigned-32

**Valid settings:** Determined by the NPE vendor, but is relative to the `npe-table`.

**Description:** Specifies the “record” within an `npe-table` that the `npe-operation` is targeting. For example, if the NPE supports a firmware load, record 0 might be the first 256 bytes of the firmware file.

The meaning of the message payload, which is the record, size, bit order, number of fields within the record, is vendor dependent. To increase the depth of interoperability, the format of several “well known” records could be defined, such as a 144-bit key and 32-bits of associated data.

#### 4.4.6.4 NPE Register

**SAE name:** `npe-register`

**Data type:** unsigned-32

**Valid settings:** 0x0000\_0000 through 0xFFEF\_FFFF: Determined by the NPE vendor.

0xFFFF\_0000 through 0xFFFF\_FFFF: Reserved by the NPF.

**Description:** Specifies the “register” within the NPE that the `npe-operation` is targeting. For example, the specific register “address” or “number” within the receiving NPE to read, read\_and\_init, or write.

The message payload is the contents of the register (when required). The size and bit-order are vendor dependent. To increase the depth of interoperability, the format of several “well known” register formats could be specified, such as a 32-bit, big-endian register format.

### 4.4.7 Integrity Checks

#### 4.4.7.1 Integrity Checksum

**SAE name:** `checksum-16`

**Data type:** ones-complement-16

**Description:** A value used to determine the integrity of all bytes in the message up to the integrity field. The algorithm used is the 1’s compliment checksum described in RFC1071 and RFC1141. When present, this field always placed at the end of the message header or message trailer, or both

#### 4.4.7.2 Integrity CRC32

**SAE name:** `crc-32`

**Data type:** bit-vector-32

**Description:** A value used to determine the integrity of all bytes in the message up to the integrity field. The algorithm used is the cyclic-redundancy check defined in ISO 3309. When present, always placed at the end of the message header or message trailer, or both.

## Annexes

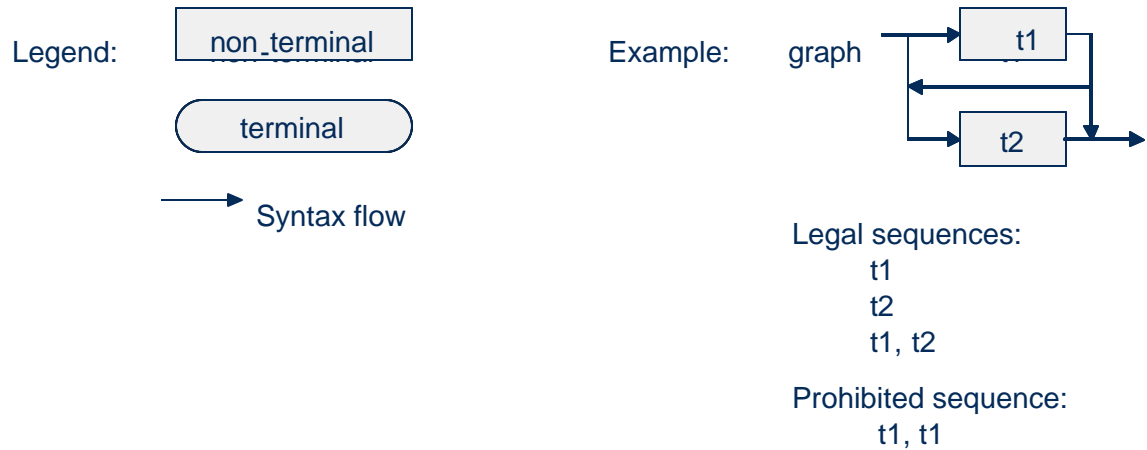
### A Glossary

Term	Definition
Conveyance	A NPE-to-NPE interconnect that carries messages.
Conveyance address	A field in the conveyance packet used to identify a service endpoint of the conveyance. E.g., Conveyance addresses can be used to identify destination or sources of conveyance payloads containing messages.
Conveyance channel	A conveyance and conveyance address. An instance of a conveyance channel is unidirectional.
Conveyance granule	The minimum, logical, allocation of bandwidth (in bytes) for a particular conveyance
Implicit field	A message field which does not appear in the physical instantiation of the message either because it has a constant value or because it is available from the conveyance (or another lower) layer.
Message	A message header, message payload, and message trailer. The message payload and message trailer are OPTIONAL.
Message field	One or more consecutive slots representing a single message header or trailer attribute.
Message granule	The larger of the processing or conveyance granule, when the message contains a trailer, and simply the conveyance granule otherwise
Message header	One or more ordered message fields delivered together preceding the message payload (if present) across a single conveyance using a conveyance address associated with messages.
Message payload	Zero or more bytes of data relevant to the message header or trailer.
Message trailer	One or more ordered message fields delivered together after the message payload across a single conveyance using a conveyance address associated with messages.
Network Coprocessor Element (NCE)	One type of NPE
Network Element	A single chassis, or box, containing one or more NPEs interconnected via conveyances.
Network Processing Element (NPE)	A device that either consumes, produces, or consumes and messages.
Network Processor (NPU)	One type of NPE
Processing granule	The minimum, logical, computational access (in bytes) for a message, as defined by the SAE based on the requirements of the NPEs
Reserved	A reserved field, slot, or bit MUST be transmitted as zero and

	ignored on reception.
Service Access Point (SAP)	An application endpoint of NPF Messaging
Slot	One byte (8 bits) of a message field.
System architectural entity (SAE)	A logical software-based entity that collects system and NPE capabilities and then configures conveyance and SAP settings in each NPE and finally defines message formats per-conveyance assigns message fields to slots, .
x	The conveyance granule in bits
y	The processing granule in bits
z	The message granule in bits

## B Syntax Graph Notation Guide (Informative)

The syntax graphs, as used in this document, contain three elements as shown in Figure 34.



**Figure 34** Syntax graph symbols and example

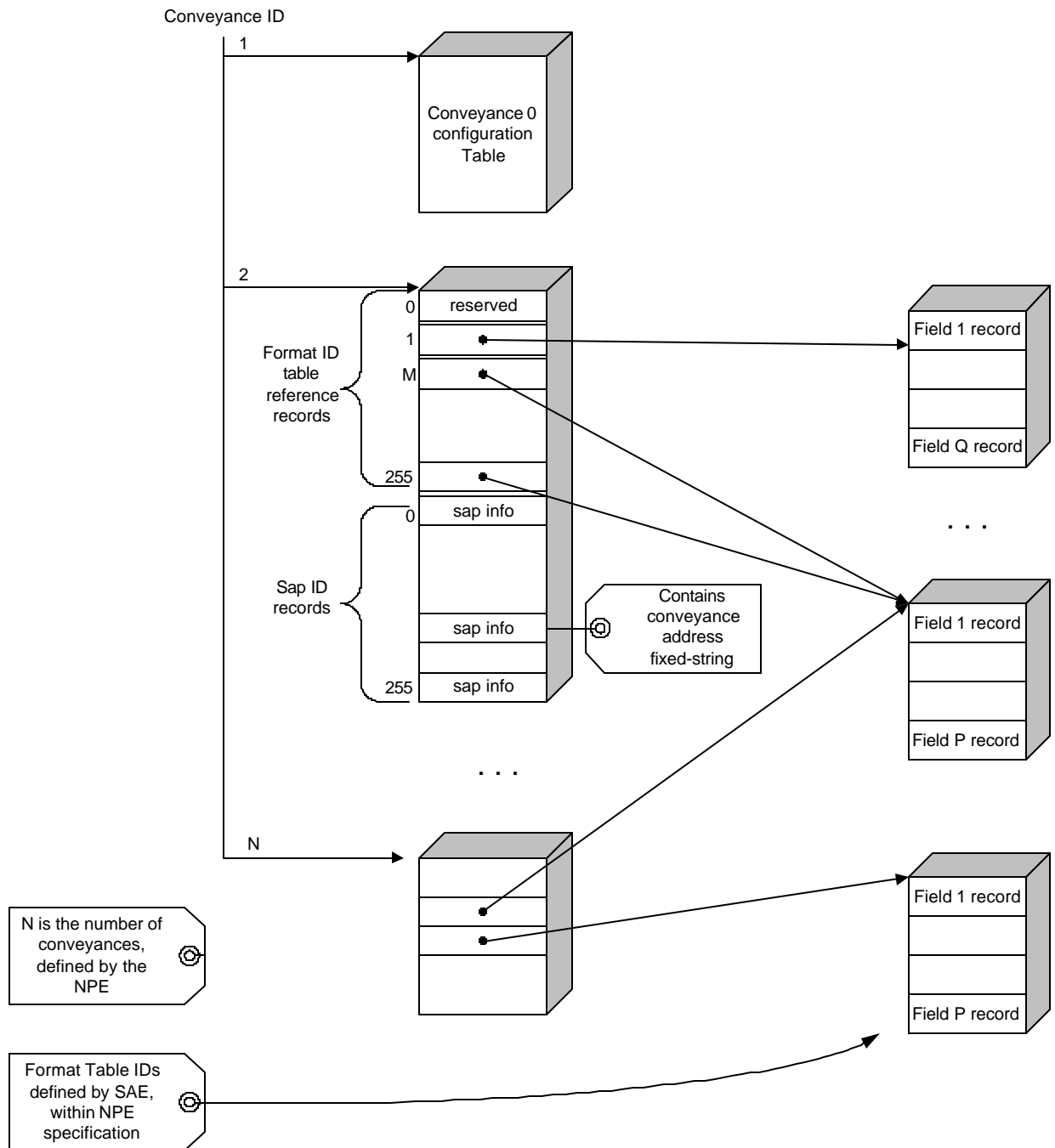
Non-terminal entries in the syntax are represented with square-corner rectangles. These symbols refer to other syntax graphs.

Terminal entries in the syntax are represented with rounded-corner rectangles. Terminal symbols represent tokens in the syntax.

Legal syntactic flow is represented with unidirectional lines. The “flow” is always unidirectional, regardless of where a line is entered. For example, as shown in Figure 34, the syntax graph has two non-terminal nodes, t1 and t2. The three legal interpretations of this graph are: t1, t1 followed by t2, and t2. The sequence: t1 followed by t1 is not expressed by the syntax graph because that would require the flow to traverse the line entering t2 in a direction opposite (“up” in this case) of its legal direction (“down”).

## **C In-band Configuration (informative)**

When used with a `format-id` of zero (0), an `npe-operation` can be used for in-band configuration by reading and writing the appropriate per-conveyance configuration tables. The format of the actual messages for such configuration information is not currently specified in this document, but Figure 35 shows the conceptual format for said configuration information. This conceptual representation of in-band configuration information does not imply any particular implementation. It is acceptable for an NPE participating in in-band configuration to physically layout its configuration information in a manner different from that shown in Figure 35. Indeed, the conceptual model was chosen specifically with a maximum of information available (i.e., maximum flexibility) so that transformations to other representations (e.g., direct mappings from conveyances to message formats) are trivial.



**Figure 35 In-band configuration per-conveyance table format (conceptual)**

*Conveyance IDs*

At the outer-most level, all in-band configuration information is specified per-conveyance. An NPE participating in in-band configuration has some number (*N*) of conveyances -- note, this includes both input and output conveyances -- and the configuration information is addressed via these conveyance numbers, as determined by the NPE vendor.

*Conveyance Configuration Tables*

The configuration for each conveyance is represented with a *Conveyance Configuration Table*. Within each conveyance configuration table are references to 255 message format specifications, one per valid `format-id` (value zero is used for in-band configuration itself) and 256 SAP information records, one per `sap-id`. The table layout has the message format specification references first, indexed by

`format-id` including a reserved record for `format-id` with value of zero, followed by the SAP information records, indexed by `sap-id`.

When a conveyance configuration table references a message format specification this indicates that the conveyance will accept or transmit -- depending on direction of the conveyance -- the referenced message format. It is to be noted that an NPE MAY choose not to physically implement references to these message format specifications but instead collapse and duplicate message format specifications directly into the conveyance configuration table. Such a mapping is acceptable, but is an implementation issue beyond the scope of this implementation agreement.

When a conveyance configuration table contains a valid SAP information record this indicates the association between the conveyance-specific addressing scheme and the messaging `sap-id`.

#### *Format ID Tables*

Message format specifications, which are indirectly referred to in the conveyance configuration tables, are represented with *format ID tables*. A format ID table contains one or more records; each record contains enough information to identify a field in the message. For example, field type, field size, field location and field value, each when appropriate.

Format ID table numbers are allocated by the SAE but will adhere to certain restrictions specified by the NPE. For example, an NPE may restrict the legal values for format ID table numbers, or may restrict the total number of format ID tables, and the SAE will honor such restrictions. Specifying the SAE as the allocating entity for format ID tables allows maximum flexibility in this conceptual model. For example, format ID tables can be shared across both `format-ids` within a single conveyance, and across conveyances on a single NPE.

#### *SAP Information Records*

A conveyance configuration table contains 256 *SAP information records*. Each SAP record is indexed by `sap-id`, i.e., with the values 0 through 255. Each SAP information record contains a string -- a value opaque to messaging -- which is the conveyance's address for the associated `sap-id`. The format of this string is dependent on the conveyance and is beyond the scope of this implementation agreement. The NPE can use this mapping to associate conveyance addresses with the messaging layer, or even messaging layer applications, as appropriate.

## D NPE Capabilities Grammar (normative)

To enable automated conformance checking tools, as well as universal SAE tools, a grammar is specified for the messaging requirements (e.g., input fields and formats, and output fields and formats, and conveyances) of an NPE. Each NPE vendor **MUST** supply an instance of this grammar along with each conforming NPE. No NPE would ever be required to parse this grammar. Instead, the SAE, which can be human or general-purpose software, would parse this grammar. Thus, this grammar is specified in a human-readable format that **MAY** be parsed by a tool.

At a conceptual level, this grammar specifies the input or output (depending on direction of the conveyance) message format capabilities for each conveyance of the NPE. Each format contains the inputs that are necessary for the NPE to operate or the outputs that the NPE is capable of generating. These inputs and outputs are represented by field name, size (in slots), and location within the message (i.e., header or trailer). In addition to the system specification (not specified in this implementation agreement), these input and output field names, sizes, and locations enable the SAE to match the necessary inputs of downstream NPEs with the outputs of upstream NPEs. The system designer needs to know (or be provided) which NPEs need to communicate, and what those NPEs need to communicate. Such information is beyond the scope of this implementation agreement; the grammar specified in this annex is local to a single NPE.

### Top-level Grammar

The grammar has three parts as shown in Figure 36. First, a version number is provided. The version number is specific to the version of the messaging specification supported by the NPE. The current version number is 1.0. Second, the contents (i.e., field types) of one or more message formats are provided. NPEs specify the fields of their message formats (i.e., *what* is either required on input or possibly generated on output from the NPE), but not the exact format of the message (i.e., *how* these fields are placed into the message). An exception to this rule is that fixed-function NPEs **MAY** specify the actual format, thus restricting how the SAE places fields in the message. Each NPE has one or more *format-content-specifications*, each of which can be restricted to only a subset of conveyances as specified in the *conveyance-specification*. The *format-content-specifications* are separated from the *conveyance-specifications* so that the same format can be easily re-used between two or more conveyances.

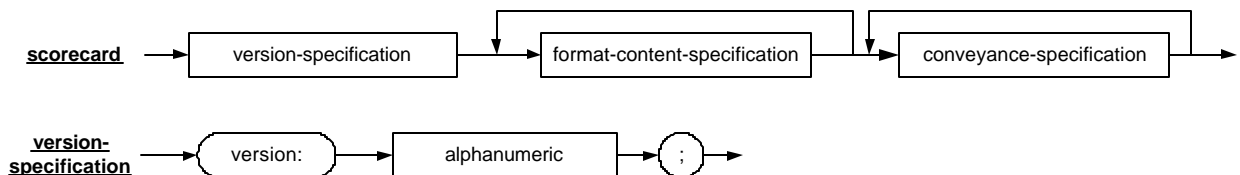
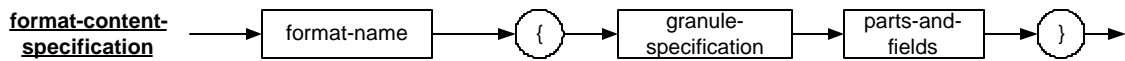


Figure 36 Basic components of the SAE grammar

The third part of the top-level grammar is one or more conveyances specifications. Each *conveyance-specification* consists of an identification of the conveyance, properties about the conveyance (e.g., conveyance granule), a list of *format-content-specifications* that are available on the conveyance, and SAP configuration mappings.

### Format Content Specification

Figure 37 shows the syntax graph for a single *format-content-specification*.

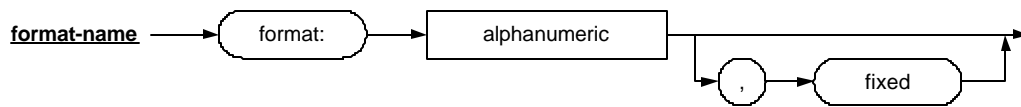


**Figure 37** *Format-content-specification* syntax graph

All *format-content-specifications* MUST contain the same three parts: a name, a (processing) granule specification, and a message parts and fields specification. Each of these parts is explained as follows:

*Format Name*

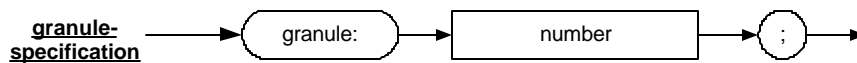
As shown in Figure 38, the name of a *format-content-specification* is a string (represented by the *alphanumeric* syntax graph shown in Figure 45), followed, optionally, by the string 'fixed'. The name has no meaning to the SAE except to match formats to those conveyances that support the format. The two legal ways to form the name of a format are: fixed and non-fixed. The only difference between the two is the additional 'fixed' string following the format name string. A fixed-format, as is explained later, specifies the exact placement of fields within the message format.



**Figure 38** *Format name* syntax graph

*Processing Granule Specification*

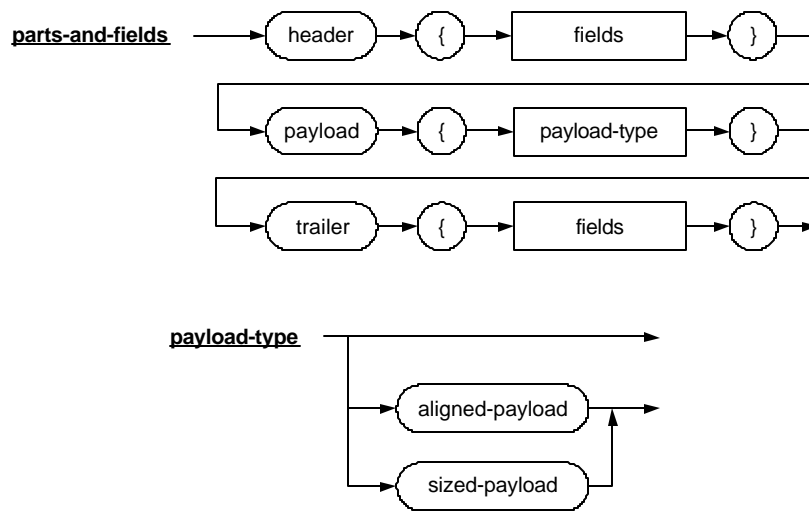
Every format contains a processing granule specification, the syntax graph for which is shown in Figure 39. The processing granule is specified as a number with the unit of bytes.



**Figure 39** *Granule-specification* syntax graph

*Parts and Fields Specification*

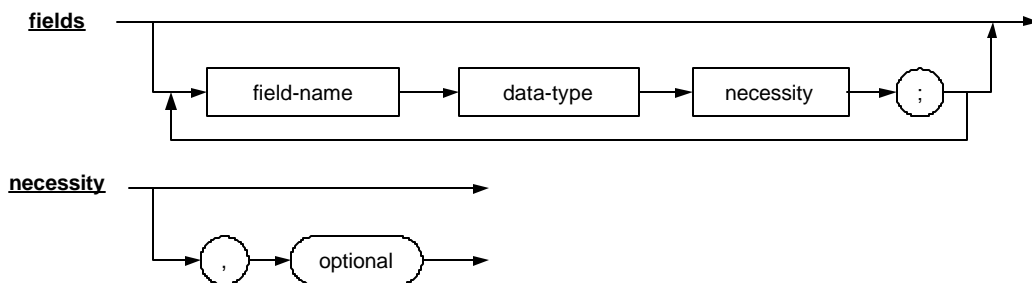
All *format-content-specifications* MUST specify the required parts of the message, as well as the fields in the message. As shown in Figure 40, the parts include the header, payload, and trailer. For headers and trailers, the fields that are used (i.e., required on input, generated on output) are provided. For the payload, the type of payload—no present, aligned, or sized-- is specified.



**Figure 40** *Parts-specification syntax graph*

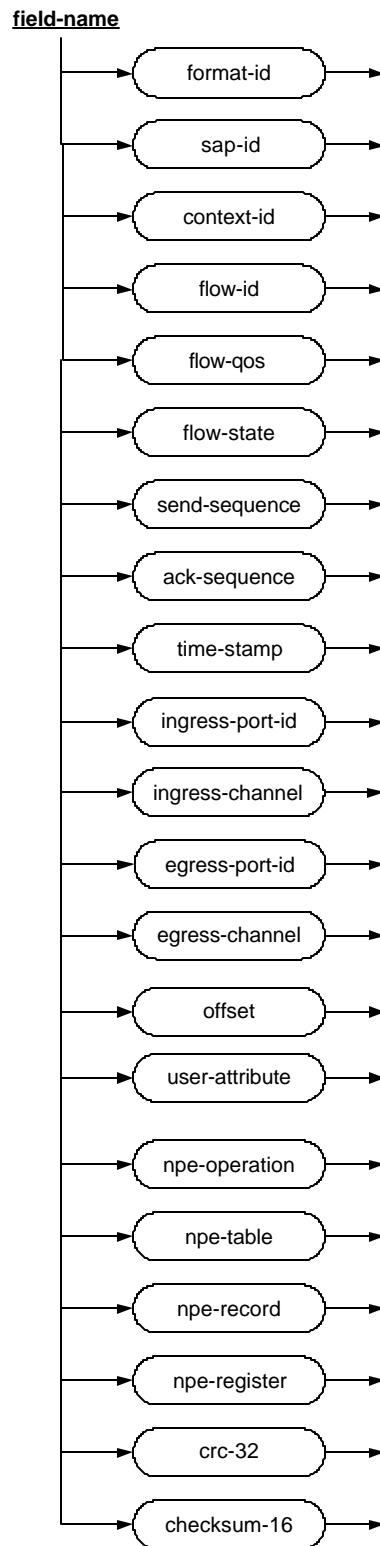
For the header and trailer parts, the list the message fields used by the specification is specified. As shown in Figure 41, each field is specified by its SAE name (i.e., a string), the data type for that field (see Figure 43), and the “necessity” of the field. The necessity of a field includes the ability to specify whether the field is mandatory or optional in the message. A field is considered mandatory unless the optional string is supplied, in which case the field is considered optional. For example, on input, and NPE specifies message fields without which the function of the NPE would be rendered impossible. Such fields are either mandatory, but never optional. On output from an NPE, however, some fields might be possible to generate, but if the downstream NPE does not consume them, these fields do not need to be generated. Such fields would be considered optional. Also, a field that may appear in both the header or trailer can be specified in both the header and trailer fields and as optional in both places.

The specified fields within the *format-content-specification* MUST be able of forming a legal message as defined by the syntax graphs of Section 4. If this specification is not a fixed-format, it is RECOMMENDED that the list follows the ordering constraints specified in Section 4. If this specification is a fixed-format, the specification of these fields MUST be in the order they appear in the message and thus MUST follow the ordering constraints specified in Section 4.



**Figure 41** *Fields-specification and related syntax graphs*

The field names are from the list of standard message fields in Section 4.4, which also specifies the semantic meaning of each field. These names are shown in Figure 42.



**Figure 42 Field name syntax**

The data type associated with each field is either implicit or explicit. The explicit data types come directly from the containers available for message fields and are identified by name. Implicit data types are identified by name as well, but include an additional fixed value (represented by a number). These data types are enumerated in the data-type syntax graph shown in Figure 43.

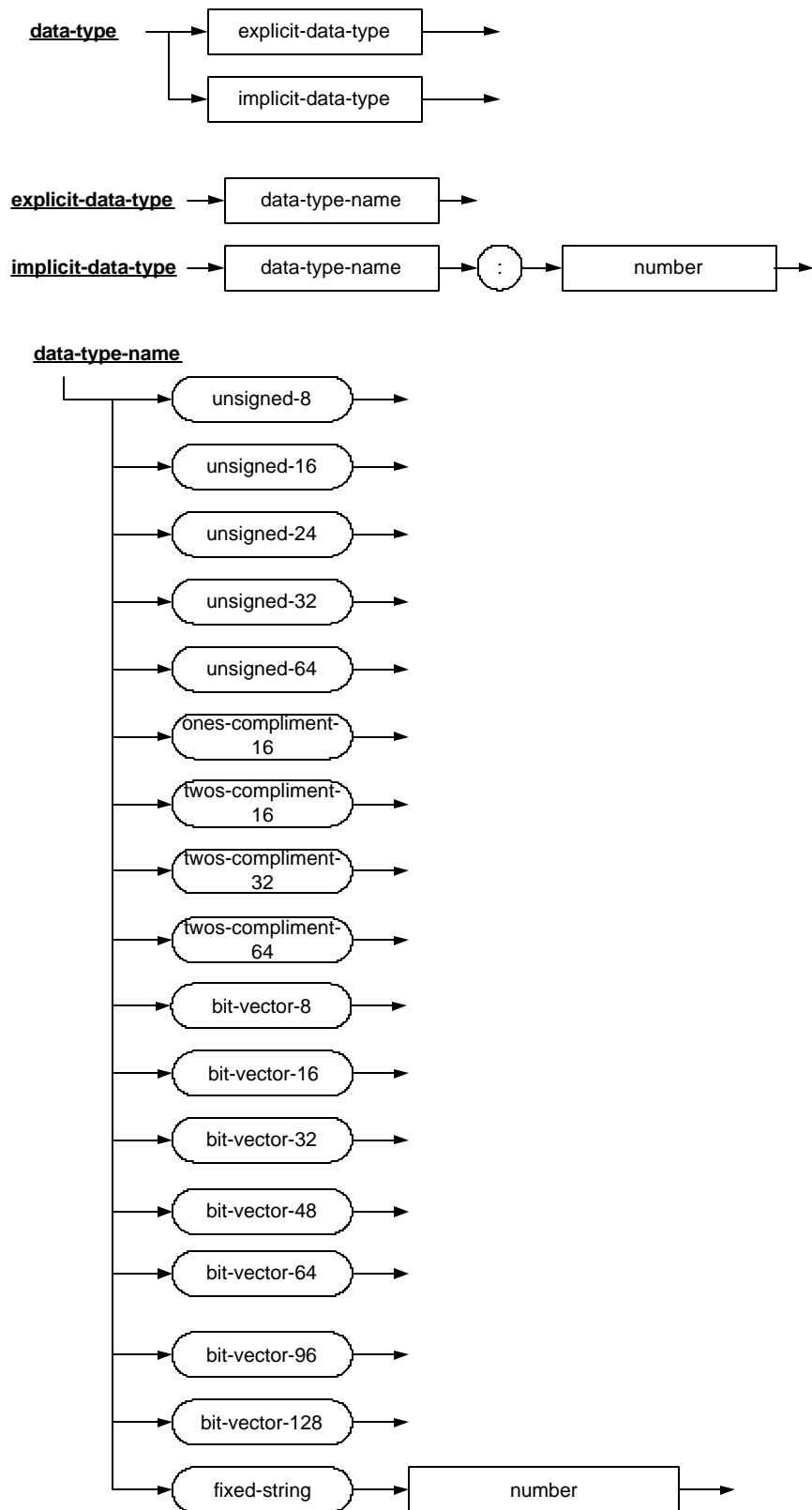
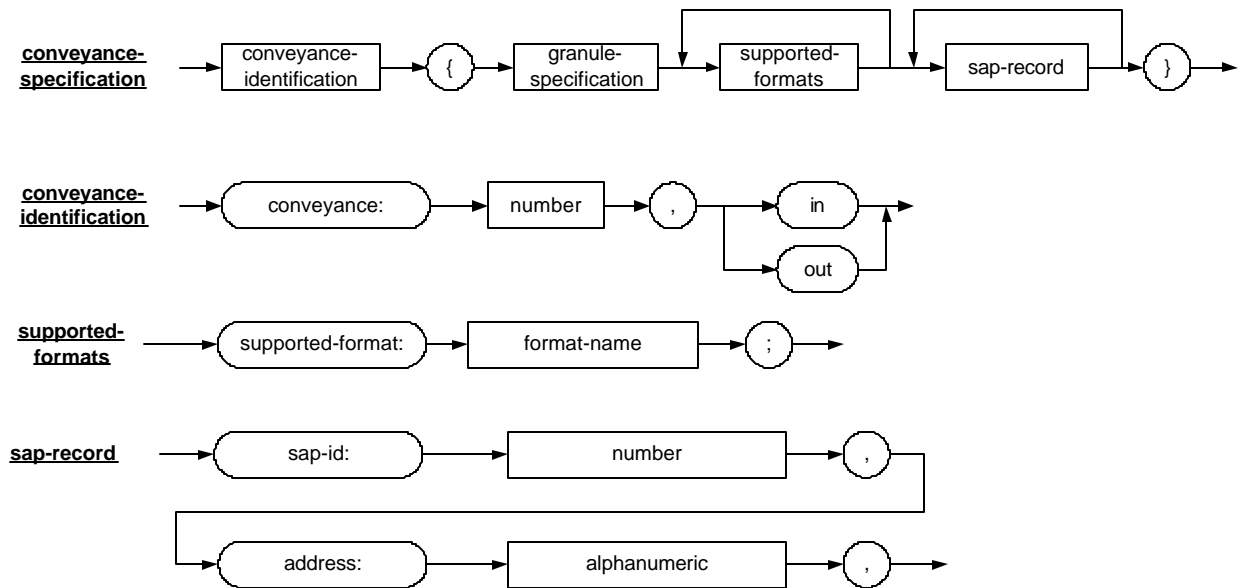


Figure 43 Data-type syntax graph

### Conveyance Specification

As shown in Figure 44, each conveyance specification contains a conveyance identification, a (conveyance) granule specification, a list of supported formats, and a list of SAP records.



**Figure 44** Conveyance-specification and related syntax graphs

*Conveyance Identification*

A conveyance is identified by number, which is local to the definition of the NPE, as well as the direction (ingress or egress) of the conveyance. An interface supporting both directions is modeled as two unidirectional conveyances within the scope of this implementation agreement, and, subsequently, within this grammar.

*Conveyance Granule Specification*

Following the identification of a conveyance, the conveyance granule is defined. The conveyance granule is specified as a number with the unit of bytes.

*Supported Formats*

Following the conveyance granule specification, the list of supported *format-content-specifications* is provided. This list consists of one or more names (i.e., strings) from the *format-content-specification* list presented previously.

*SAP Record*

Finally, for each SAP using messaging, a SAP-to-conveyance-address mapping MUST be supplied. Each such mapping requires the `sap-id`, a number between 0 and 255, and a conveyance address, a string, the format of which is beyond the scope of this implementation agreement.

**Miscellaneous Syntax Graphs**

Figure 45 shows several syntax graphs that support the previously discussed syntax graphs for NPE capabilities. A number is simply a string of digits (i.e., 0 – 9). An alphanumeric is a series of digits, letters (a – z, A – Z) and certain symbols (i.e., `_`, `@`, `#`, `$`, and `%`).

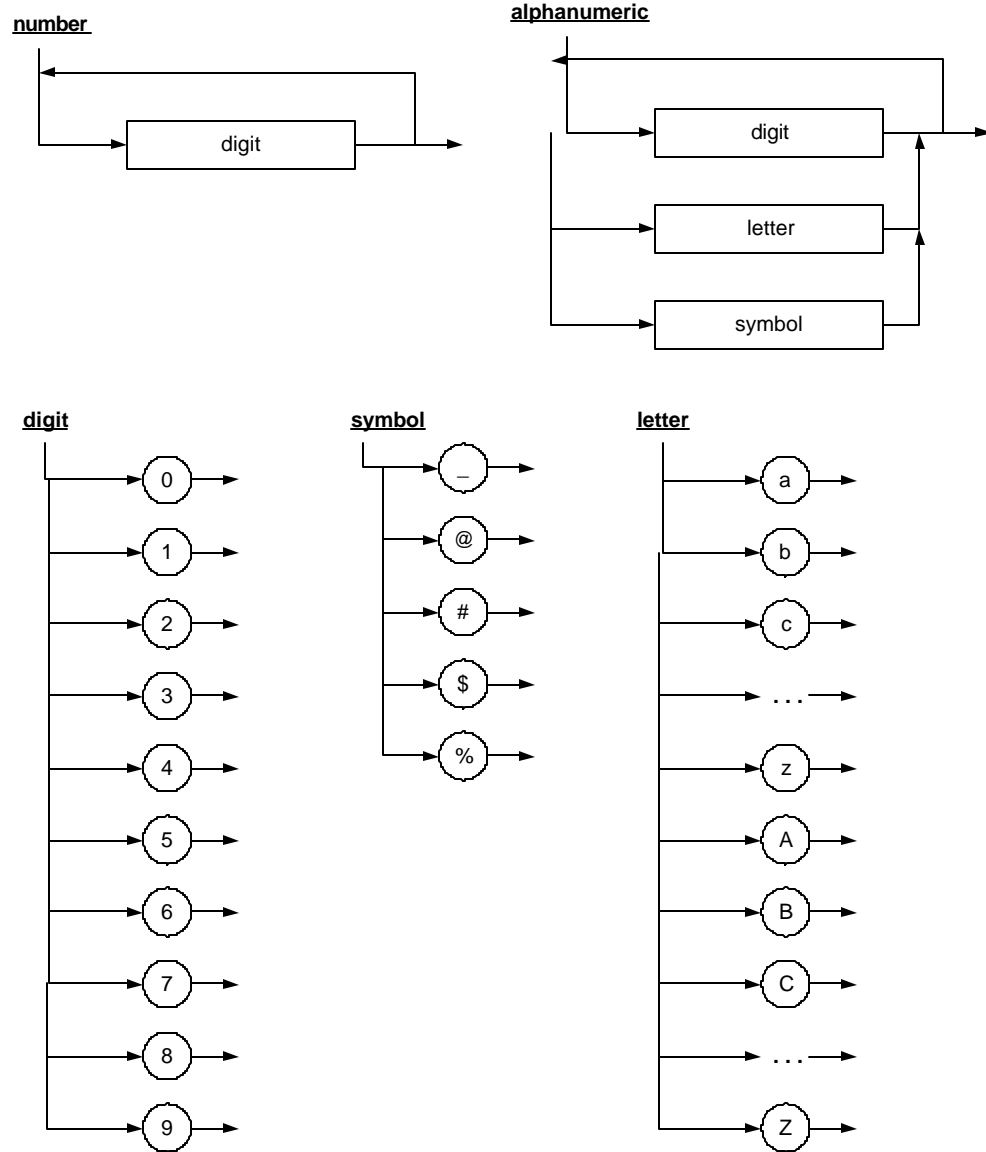


Figure 45 Miscellaneous syntax graphs supporting the NPE Capabilities grammar