SEMI E128-0414 (Reapproved 0519)
SPECIFICATION FOR XML MESSAGE STRUCTURES

This Standard was technically approved by the Information & Control Global Technical Committee. This edition was approved for publication by the global Audits and Reviews Subcommittee on December 17, 2018. Available at www.semiviews.org and www.semi.org in May 2019; originally published July 2003; previously published April 2014.

1 Purpose

1.1 This Standard specifies XML structures required to encode message header or 'envelope' information for synchronous and asynchronous messages. XML is used in software applications to encode information with tags that provide structure and data type information specified in XML Schemas. XML is also used to structure information needed to route and deliver messages between applications. A standard for message headers is required to avoid development and use of multiple incompatible protocols based on XML. This Specification will be referred to as 'The XML Messaging Specification.'

1.2 The XML Messaging Specification provides definitions of message headers needed for messages exchanged in an asynchronous or synchronous fashion. Synchronous message delivery is chosen when application communication requires message consumption and execution to be completed before a new message is allowed. Asynchronous messages are delivered independently without dependence on transport technology to correlate related messages. The rationale for choosing an asynchronous message delivery mechanism is based on the characteristics of the interaction between two software systems. Some of these reasons are that:

- Support for long-running Request/Reply interactions may not be practical in a synchronous session-oriented communication.
- Some messaging interactions require additional callback messages reporting on server activity which supplement the reply message.
- Clients may wish to interleave requests and replies to manage parallel or overlapping activities in multiple concurrent messaging conversations.

1.3 Headers defined for asynchronous messaging may also support synchronous interactions that rely on transport sessions for identification and correlation of related messages. However, there may be benefits of using the message header elements specified here even in synchronous interactions. The message header elements provide self describing information about messages that may be useful for debugging or tracking message traffic. Their use may also enable future migration to different message transports.

1.4 The XML Messaging Specification uses established, openly referenceable industry standards for XML messaging where possible. It only specifies extensions to existing industry standards when needed to meet the immediate requirements for messaging in a SEMI Standard application context. The intent of this Standard is not to replicate existing standards or offer competing specifications, but to align with and cite the usage of existing standards.

2 Scope

2.1 *In Scope* — The XML Messaging Specification will include within its scope the following:

2.1.1 *Use of the W3C Simple Object Access Protocol (SOAP)* — SOAP is used as a basic message foundation for SEMI messaging applications.

2.1.2 *Definition of SOAP Extensions* — This Specification defines SOAP extensions for header and body elements needed to support synchronous and asynchronous messaging interactions.

2.1.3 *Structures for Request/Reply Point-to-Point Message Dialogs* — This Specification addresses the message header data needed to support request/reply interactions with a request message and corresponding reply message.

2.1.4 *Structures for Asynchronous Point-to-Point Callback Messages* — This Specification addresses the message header data needed to enable atomic, callback messages that complement a request/reply dialog between two endpoints.

2.1.5 *Structures for Multicast Event Messages* — This Specification addresses the message header data needed to enable atomic, event messages distributed via a publication multicast to multiple subscribers.

SUPERSEDED

Documents Document Sharing Hub
Full document details: SEMI E128-0414-0519R

2.2 *Out of Scope* — The following topics are beyond the scope of this Specification. These topics may be addressed in separate specifications, but are not intended as future additions to this Standard.

2.2.1 *Message Security* — The Specification does not provide mechanisms for authentication of the message sender or receiver or for securing the content of messages from unauthorized access.

2.2.2 *Quality of Performance* — The XML Messaging Specification does not provide performance requirements or conformance testing for the performance characteristics of message transports. There may be environments in which XML messaging structures are not appropriate or effective for high performance communications. The definition of the environments where XML Messaging may or may not be appropriate is beyond the scope of this Document.

NOTICE: SEMI Standards and Safety Guidelines do not purport to address all safety issues associated with their use. It is the responsibility of the users of the Documents to establish appropriate safety and health practices, and determine the applicability of regulatory or other limitations prior to use.

3 Limitations

3.1 Protocol bindings are not specified in this Document. It is common for SOAP implementations to use the basic XML 1.0 binding to HTTP. However, other bindings are available. As noted in *SOAP Version 1.2 Part 0: Primer*, the MTOM (Message Transport Optimization Mechanism) specification defines a SOAP binding to HTTP that allows for selective optimization of SOAP message fields under certain circumstances. This includes transport of non-textual data as message attachments.

4 Referenced Standards and Documents

4.1 *SEMI Standards and Safety Guidelines*

SEMI E120 — Specification for the Common Equipment Model (CEM)

SEMI E125 — Specification for Equipment Self Description (EqSD)

SEMI E132 — Specification for Equipment Client Authentication and Authorization

SEMI E134 — Specification for Data Collection Management

4.2 *ISO Standards*[1]

ISO/IEC 11578:1996 — Information Technology – Open Systems Interconnection – Remote Procedure Call

4.3 *NIST Standards*[2]

NIST FIPS PUB 180-1 — Secure Hash Algorithm (SHA-1)

4.4 *World Wide Web Consortium (W3C) Standards*[3]

Extensible Markup Language (XML) 1.0 (Fourth Edition); http://www.w3.org/TR/2006/REC-xml-20060816/

Namespaces in XML (Second Edition); http://www.w3.org/TR/2006/REC-xml-names-20060816/

XML Schema Part 0: Primer (Second Edition); http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/

XML Schema Part 1: Structures (Second Edition); http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

XML Schema Part 2: Datatypes (Second Edition); http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

SOAP Version 1.2 Part 0: Primer (Second Edition); http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) – W3C, 27 April 2007; http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[1] International Organization for Standardization, ISO Central Secretariat, BIBC II, Chemin de Blandonnet 8, CP 401, 1214 Vernier, Geneva, Switzerland; Telephone: +41.22.749.01.11, http://www.iso.org

[2] National Institute of Standards and Technology, 100 Bureau Drive, Stop 3460, Gaithersburg, MD 20899-3460, USA; Telephone: +1.301.975.6478, http://www.nist.gov

[3] World Wide Web Consortium, Massachusetts Institute of Technology (MIT) Laboratory for Computer Science, 200 Technology Square, Cambridge, MA 02139, USA; Telephone: +1.617.253.2613, Fax: +1.617.258.5999, http://www.w3.org

SUPERSEDED

Documents
Document Sharing Hub

Full document details: SEMI E128-0414-0519R

SOAP Version 1.2 Part 2: Adjuncts (Second Edition) – W3C, 27 April 2007; http://www.w3.org/TR/2007/REC-soap12-part2-20070427/

Web Services Description Language (WSDL) 1.1 (WSDL) — World Wide Web Consortium, 15 March 2002. This version of the Web Services Description Language Note is http://www.w3.org/TR/2001/NOTE-wsdl-20010315. The latest version of Web Services Description Language is available at http://www.w3.org/TR/wsdl.

4.5 *Other Documents*

Secure Sockets Layer (SSL) Protocol 3.0 (see http://wp.netscape.com/eng/ssl3/draft302.txt)

NOTICE: Unless otherwise indicated, all documents cited shall be the latest published versions.

5 Terminology

5.1 *Abbreviations and Acronyms*

5.1.1 *SOAP* — Simple Object Access Protocol

5.1.2 *WSDL* — Web Services Definition Language

5.1.3 *XML* — Extensible Markup Language

5.2 *Definitions*

5.2.1 *asynchronous messaging* — a style of communication based on the exchange of atomic messages separated in time and implemented with one-way message deliveries.

5.2.2 *callback message* — a message that communicates supplemental information resulting from performance of an action initiated by a related request/reply conversation.

5.2.3 *message document* — an XML document that contains the message envelope and encapsulated message header and message content.

5.2.4 *message envelope* — the encapsulating XML structures that define an overall framework for expressing what is in a message; who should deal with it, and whether it is optional or mandatory.

5.2.5 *notification* — a message that is sent to a consumer where no reply is expected.

5.2.6 *reply message* — a message that contains data resulting from the completion of an action initiated by a related request message.

5.2.7 *request message* — a message that contains necessary information to allow a server to perform a requested action on behalf of the requester.

5.2.8 *synchronous communication* — a style of communication where a message is sent and blocking is applied until the message reply is received; no other messages can be sent or received while waiting for the reply.

5.2.9 *X notification* — a message that is sent to a consumer were no reply is expected.

5.2.10 *X synchronous communication* — a style of communication where a message is sent and blocking is applied until the message reply is received; no other messages can be sent or received while waiting for the reply.

6 Requirements

6.1 *Simple Object Access Protocol (SOAP) Envelope* — Messages shall be enclosed in the Envelope element as specified in the SOAP 1.2 document. The SOAP Envelope shall include a Header element and a Body element. Within the Header element, message documents shall include an element named MessageHeader that contains data required for message preparation and transport as specified below.

6.2 *SEMI XML Messaging Namespace* — All XML items defined in this Specification belong to the reserved SEMI namespace ‘urn:semi-org:xsd.E128.V0706.xms’. The following XML Namespace declaration shall be used to specify the namespace.

```
<xsd:schema xmlns="urn:semi-org:xsd.E128.V0706.xms"
xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:xms="urn:semi-org:xsd.E128.V0706.xms"
```


SUPERSEDED
Documents Document Sharing Hub
Full document details: SEMI E128-0414-0519R

```
targetNamespace="urn:semi-org:xsd.E128.V0706.xms"
elementFormDefault="qualified">
```

6.3 *Use of SOAP Messages for Synchronous Request/Reply Interactions* — The XML Messaging Specification defines specific SOAP header elements when using synchronous interactions. The asynchronous SOAP Header Extensions specified in this Document are not required for a synchronous exchange, but may be useful even in those cases. A standard adopting a synchronous use of SOAP shall clearly indicate whether additional header extensions are required for conformant implementations.

6.4 *SOAP Header Entry for Messaging* — The SOAP message shall have an entry contained in the SOAP-ENV:Header element named **MessageHeader** that contains elements as defined in this section. The MessageHeader element shall be defined with a mustUnderstand = '1' attribute that requires all conformant implementations to process the header element or return a 'MustUnderstand' fault to the message sender.

6.4.1 *From* — The **From** element shall identify the application that sent this message. The format of the value in the **From** element shall be a valid URI. The domain of the value in the **From** element may vary by transport binding. If the message is a Reply Message, the From element shall contain the sender of the Reply Message (not the sender of the original Request Message).

6.4.2 *To* — The **To** element shall identify the application that is the intended recipient of this message. The format of the value in the **To** element shall be a valid URI. The domain of the value in the **To** element may vary by transport binding. If the message is a Reply Message, the To element shall contain the recipient of the Reply Message (not the recipient of the original Request Message).

6.4.3 *MessageType* — The **MessageType** element shall identify the role of the message in a conversation. A message shall be either a request, reply or callback message.

6.4.4 *Request Message* — A **request** message shall originate from an endpoint in the client role and shall establish a new RequestId. The RequestId shall be unique within the domain of all such Ids generated by the originating client. The request message body contains the information needed by the server to enable it to perform a requested action on behalf of the client.

6.4.4.1 *Reply Message* — A **reply** message shall originate from an endpoint in the server role and shall echo the RequestId received from a client in a prior request message as a CorrelationId. The reply message body may contain information that conveys the completion of an action initiated by a related request message. It may instead include a body entry for a Fault that communicates the reason for a failure. In other cases the Fault may provide information such as warnings that supplement the normal content of the body.

6.4.4.2 *Callback Message* — A **callback** message shall originate from an endpoint in the server role and shall echo the RequestId received from a client in a prior request message. A callback message body contains unspecified information that relates to the requested action initiated by a related request message. A callback message may precede the reply message (a leading callback message), or it may follow the reply message (a trailing callback message). The callback message shall also contain an element for the EventIndex if the server will send more than one callback message. The body of a callback message may also contain a Fault element if there is a processing error that needs to be conveyed to the client. The Fault element borrows the form of SOAP 1.1 Faults in order to report faults. Do not confuse this usage with true SOAP Faults.

6.4.5 *RequestId* — The **RequestId** element shall be used in a request message to identify it as the first of a series of messages that belong to a single interaction. The value of RequestId shall be generated by a client with the initial request message and then shall be echoed back as the **CorrelationId** in subsequent callback messages or reply messages from the responding server. The XML Schema shall specify a choice between RequestId and CorrelationId. RequestId shall be the choice when MessageType is 'Request'.

6.4.6 *CorrelationId* — The **CorrelationId** element shall be used in each reply or callback message to identify it as part of an interaction that was initiated by a prior request message. The value of CorrelationId shall be identical to the **RequestId** from the initial request message. The same CorrelationId value shall be echoed back in all callback messages or reply messages sent from the server in response to the client's initial request message. The XML Schema shall specify a choice between RequestId and CorrelationId. CorrelationId shall be chosen when MessageType is 'Reply' or 'Callback'.

SUPERSEDED

Documents
Document Sharing Hub
Full document details: SEMI E128-0414-0519R

6.4.7 *Action* — The **Action** element shall be a string that serves to identify the unit of functionality invoked by the request message. An Action value shall be unique to the recipient of the message. In a simple web service binding, the Action value will be the operation name for the web service (i.e., ActivatePlan or GetSEMIObjType as defined in SEMI E134 and SEMI E125).

6.4.8 *ReplyExpected* — The **ReplyExpected** element is optional. When present, the ReplyExpected element shall indicate whether the client expects to receive a reply message by a 'true' value. When present with a value of 'false', the ReplyExpected element shall indicate that the client does not expect to receive a reply message. If not present, the server receiving the request shall assume that a reply is expected (equivalent to a 'true' value). A service shall be prepared to return a reply for every request, and a client shall be prepared to receive replies even if this element is set to 'false'. ReplyExpected is intended only for optimizing conversations where the client intends to ignore reply messages.

6.4.9 *MessageId* — The **MessageId** element shall be optional in the MessageHeader to define a string value to uniquely identify the message. MessageId need not be included in the MessageHeader if there is no value established for uniquely identifying each message by the selected message transport. If present, MessageId shall be globally unique.

6.4.10 *EventIndex* — The **EventIndex** element shall be optional in the MessageHeader of a callback message. EventIndex shall not be included in request or reply messages. Within the MessageHeader of a callback message, EventIndex is optional only if there is just one callback message in an interaction, but required if there are a series of callbacks. EventIndex is used to identify this message's position in relation to multiple callback messages resulting from the original request.

6.4.10.1 *EventIndex Sub-Elements* — EventIndex shall contain sub-elements for Position, and either Total or Finished as specified in the following paragraphs. Either Total or Finished shall be present, but not both.

6.4.10.2 *Position* — The **Position** element shall indicate this message's position in a stream of callback messages. Position shall be greater than or equal to 1. Position shall be less than or equal to Total, when Total is present.

6.4.10.3 *Total* — The **Total** element shall indicate the final count of callback messages to be delivered in this interaction (i.e., all messages sharing a common RequestId/CorrelationId). Total shall have the same value in each callback message in the sequence.

6.4.10.4 *Finished* — The **Finished** element shall indicate whether this is the last in a sequence of callback messages. Finished shall be present in callback messages for interactions in which the total number of callback messages cannot be known in advance. A value of 'false' shall indicate that additional callback messages will be sent. A value of 'true' shall indicate that no further callback messages will be sent.

6.4.10.5 *Default Interpretation if EventIndex is not Present* — If EventIndex is not present, the client receiving the callback message shall interpret it as equivalent to a EventIndex element with Position = 1 and Total = 1, indicating a single callback message from server to client.

6.4.11 *AsyncHeader* — The XML schema element in Figure 1 defines the contents of the header elements required for asynchronous messaging.

SUPERSEDED

Documents Document Sharing Hub

Full document details: SEMI E128-0414-0519R

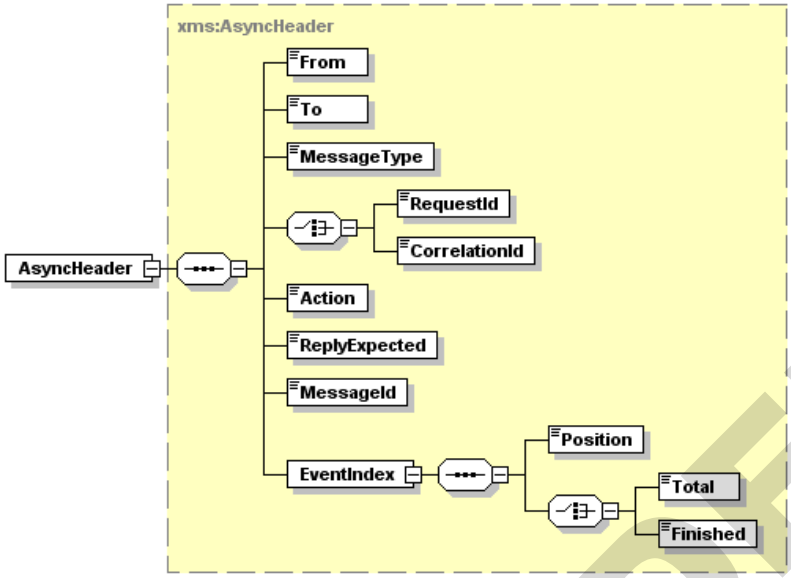

Figure 1
Asynchronous Message Header

6.4.12 *SyncHeader* — The **SyncHeader** element defines the SOAP header elements that are required for sychronous messaging. Three elements are required: **From**, **To** and **SessionId**. The **From** and **To** headers are defined in §§ 6.4.1 and 6.4.2 respectively.

6.4.12.1 *SessionId* — The SessionId element value is generated by a server application that granted a client application to establish a session. The value of SessionId in reply or callback messages shall be identical to the SessionId from the initial request message, if SessionId was present in the initial request message. SessionIds allow clients to identify messages that can be treated as a single transaction for purposes of commit/rollback, authentication, or closing. The session identifier is required to be present in all requests. The session identifier shall be empty during the Establish Session operation when requesting a session and in the response when a session request is denied. The session identifier is generated by the server during the Establish Session operation, and must be a UUID that is guaranteed unique across space and time, see documentation referenced in ¶ 4.1 for specific details on syntax and format of a UUID. This provides the most transparent way of sending the session identifier without affecting the input/output parameters of the individual operations. If SSL is used, the session identifier need not be encrypted and shall be included in the SOAP header unmodified. Figure 2 shows the XML schema diagram for the SyncHeader schema element. The session identifier is modeled as the SessionID of XML type xsd:string in the SyncHeader.

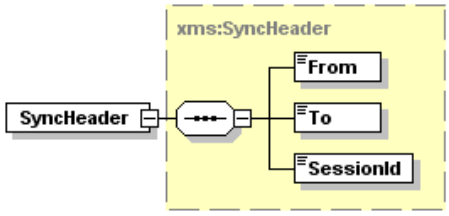

Figure 2
SyncHeader Element

6.4.13 *SyncHashHeader* — The **SyncHashHeader** element defines the SOAP headers that are used for notification messages when Synchronous communication is used. Three elements are required: **From**, **To** and **SessionIdHashed**. The **From** and **To** headers are defined in §§ 6.4.1 and 6.4.2 respectively.

SUPERSEDED

Documents
Document Sharing Hub
Full document details: SEMI E128-0414-0519R

6.4.13.1 *SessionIdHashed* — The **SessionIdHashed** element is generated by an application that sends a notification message to a consumer application. The value of SessionIdHashed is based on a based 64 Binary type. This value is calculated or computed by first applying SHA-1 hash function to the session identifier or the value of SessionID send originally by the server that granted a session, see documentation referenced in ¶ 4.1 for specific details. The session identifier must be converted to UTF-8 prior to applying the hash to get the correct value. The notification messages use the SyncHashHeader which include the session identifier in a hash format because sending the sessionID in plaintext is not recommended. This Specification defines a hashed session identifier to be included in the SOAP header as shown in Figure 3 in the XML Schema diagram for the SyncHashHeader schema element. The hashed session identifier is modeled as the XML element SessionIDHash of type xsd:base64Binary in the SyncHashHeader.[4]

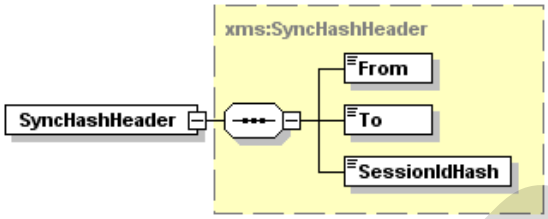

Figure 3
SyncHashHeader Schema Element

6.5 *Asynchronous SOAP Body Extensions* — Any exceptions that occur shall be communicated with a Fault element inside the SOAP body. The Fault element borrows the structure of the SOAP Fault. Do not confuse this usage with true SOAP Faults.

6.5.1 *Fault* — A **Fault** element shall be inserted into the SOAP body element as a body entry to relay error status from the server back to the client. A body entry is a top level element within the Body element of the SOAP envelope. This Specification borrows the 'Fault' definition from SOAP 1.1 § 4.4, which should not be confused with true SOAP Faults.

6.5.2 *faultcode* — The faultcode element shall be present in a fault element. The values specified in each specification defining the errors shall be used to indicate the nature of the error that occurred.

- 'Client' indicates that the request message was not correct and could not be processed by the server.
- 'Server' indicates that the request message was valid, but the server could not complete the requested action.

6.5.2.1 *faultstring* — The faultstring element shall be present to describe the cause of the fault in human understandable text.

6.5.2.2 *faultactor* — The use of the faultactor element is optional.

6.5.2.3 *detail* — The detail element is optional. It may be used to describe the cause of a fault that resulted from processing the body of the request message.

6.6 *Asynchronous XML Message Examples* — The examples of XML messages presented in this section apply only to asynchronous messages and are intended to help the reader understand the structure of messages that conform to this Specification.

6.6.1 *Example SOAP Envelope* — The SOAP envelope illustrated in Figure 4 contains the required Header and Body elements. It also includes the MessageHeader element specified in this Standard. The contents of these elements are intentionally left empty in this example.

[4] Note that the user code that composes the SyncHashHeader object should provide the SessionIDHash value as the SHA-1 encoding of the UTF-8 version of the sessionID string. The base64Binary conversion is normally handled within the SOAP messaging system.

SUPERSEDED
Documents
Document Sharing Hub
Full document details: SEMI E128-0414-0519R