Errata Notice on Schema Locations

(September 30, 2020)

This standard makes use of namespace locations with a form of http://www.scte.org/schemas/xyz/*, where “xyz” is the location of the specific schema being referenced. Due to limitations on the current SCTE•ISBE website, those specific locations are not available.

To find such schemas:

1. Go to the standards download page at https://www.scte.org/download-scteisbe-standards/
2. Search for the standard number (xyz in the above example)
3. Expand the listing by clicking the plus (+) button to the left of the standard number

The expanded list will show the desired schema.

This notice will be removed once the exact namespace values are functional.

SCTE · ISBE

STANDARDS

Digital Video Subcommittee

AMERICAN NATIONAL STANDARD

ANSI/SCTE 130-7 2020

Digital Program Insertion–Advertising Systems Interfaces
Part 7 – Message Transport

NOTICE

The Society of Cable Telecommunications Engineers (SCTE) / International Society of Broadband Experts (ISBE) Standards and Operational Practices (hereafter called "documents") are intended to serve the public interest by providing specifications, test methods and procedures that promote uniformity of product, interchangeability, best practices and ultimately the long-term reliability of broadband communications facilities. These documents shall not in any way preclude any member or non-member of SCTE•ISBE from manufacturing or selling products not conforming to such documents, nor shall the existence of such standards preclude their voluntary use by those other than SCTE•ISBE members.

SCTE•ISBE assumes no obligations or liability whatsoever to any party who may adopt the documents. Such adopting party assumes all risks associated with adoption of these documents, and accepts full responsibility for any damage and/or claims arising from the adoption of such documents.

Attention is called to the possibility that implementation of this document may require the use of subject matter covered by patent rights. By publication of this document, no position is taken with respect to the existence or validity of any patent rights in connection therewith. SCTE•ISBE shall not be responsible for identifying patents for which a license may be required or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Patent holders who believe that they hold patents which are essential to the implementation of this document have been requested to provide information about those patents and any related licensing terms and conditions. Any such declarations made before or after publication of this document are available on the SCTE•ISBE web site at http://www.scte.org.

All Rights Reserved

© Society of Cable Telecommunications Engineers, Inc. 2020
140 Philips Road
Exton, PA 19341

Table of Contents

List of Figures

Title **Page Number**

List of Examples

Title **Page Number**

List of Tables

Title **Page Number**

1. Scope

This document describes the Digital Program Insertion Advertising Systems Interfaces' transport protocols required for the exchange of messages defined in the individual parts of the SCTE 130 specification.

Note: Security issues surrounding the transport protocols defined herein have been purposely omitted and are considered outside of the scope of this document.

2. Normative References

The following documents contain provisions, which, through reference in this text, constitute provisions of this document. At the time of Subcommittee approval, the editions indicated were valid. All documents are subject to revision; and while parties to any agreement based on this document are encouraged to investigate the possibility of applying the most recent editions of the documents listed below, they are reminded that newer editions of those documents might not be compatible with the referenced version.

2.1. SCTE References

[SCTE130-2]	SCTE 130-2 2020: Digital Program Insertion—Advertising Systems Interfaces Part 2 - Core Data Elements

2.2. Standards from Other Organizations

[W3C – SOAP1.1]	W3C Ref: Simple Object Access protocol (SOAP) 1.1
[W3C – SOAP1.2]	W3C Ref: SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). April 27th, 2007
IETF STD05	Internet Protocol. September 1981. DARPA Internet Program Protocol Specification.
IETF STD07	Transmission Control Protocol. September 1981. DARPA Internet Program Protocol Specification.
[HTTP]	RFC 7231: Hypertext Transfer Protocol (HTTP/1.1)

2.3. Published Materials

- No normative references are applicable.

2.4. Normative Reference Acquisition

2.4.1. SCTE Standards: United States of America

URL: http://www.scte.org. Society of Cable Telecommunications Engineers Inc., 140 Philips Road, Exton, PA 19341; Telephone 800-542-5040; Facsimile: 610-363-5898; E-mail: standards@scte.org.

2.4.2. W3C Standards

URL: http://www.w3c.org. MIT, 32 Vassar Street, Room 32-G515, Cambridge, MA 02139, USA; Telephone: +1.617.258.5999.

2.4.3. IETF

URL: http://www.ietf.org. IETF Secretariat (c/o NeuStar, Inc.), 45000 Center Oak Plaza, Sterling, VA, 20166, Phone: 1.571.434.3500; Facsimile: 1.571.343.3535; E-mail: ietf-info@ietf.org.

3. Informative References

The following documents might provide valuable information to the reader but are not required when complying with this document.

3.1. SCTE References

[SCTE130-3]	SCTE 130-3: Digital Program Insertion—Advertising Systems Interfaces Part 3 – Ad Management Service (ADM) Interface
[SCTE130-4]	SCTE 130-4: Digital Program Insertion – Advertising Systems Interfaces Part 4 – Content Information Service (CIS) Interface
[SCTE130-5]	SCTE 130-5: Digital Program Insertion – Advertising Systems Interfaces Part 5 – Placement Opportunity Information Service (POIS) Interface
[SCTE130-6]	SCTE 130-6: Digital Program Insertion – Advertising Systems Interfaces Part 6 – Subscriber Information Service (SIS) Interface
[SCTE130-8]	SCTE 130-8: Digital Program Insertion – Advertising Systems Interfaces Part 8 – General Information Service (GIS) Interface

3.2. Standards from Other Organizations

[W3C – SOAP Part 0]	W3C Ref: SOAP Version 1.2 Part 0: Primer.

[W3C – WSDL]	W3C Ref: Web Services Description Language (WSDL) Version 1.1.
[W3C – DOM]	W3C Ref: Document Object Model (DOM) Level 3 Core.
[W3C – URI]	W3C Ref: URIs, URLs and URNs: Clarifications and Recommendations 1.0.
[WS-I – Basic Profile 1.1]	Web Services Interoperability Organization. Basic Profile Version 1.1.

3.3. Published Materials

- No informative references are applicable.

3.4. Informative Reference Acquisition

3.4.1. SCTE Standards: United States of America

URL: http://www.scte.org. Society of Cable Telecommunications Engineers Inc., 140 Philips Road, Exton, PA 19341; Telephone 800-542-5040; Facsimile: 610-363-5898; E-mail: standards@scte.org.

3.4.2. W3C Standards

URL: http://www.w3c.org. MIT, 32 Vassar Street, Room 32-G515, Cambridge, MA 02139, USA; Telephone: +1.617.258.5999.

3.4.3. WS-I Organization

URL: http://www.ws-i.org. Web Services Interoperability Organization.

4. Compliance Notation

shall	This word or the adjective "***required***" means that the item is an absolute requirement of this document.
shall not	This phrase means that the item is an absolute prohibition of this document.
forbidden	This word means the value specified shall never be used.
should	This word or the adjective "*recommended*" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighted before choosing a different course.
should not	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
may	This word or the adjective "*optional*" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.
deprecated	Use is permissible for legacy purposes only. Deprecated features may be removed from future versions of this document. Implementations should avoid use of deprecated features.

5. Abbreviations and Definitions

5.1. Abbreviations

SCTE	Society of Cable Telecommunications Engineers

5.2. Definitions

All [SCTE130-2] definitions and abbreviations are included herein. See [SCTE130-2] for additional information.

HTTP: (Hypertext Transfer Protocol)	The underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web Servers and browsers should take in response to various commands.
HTTPS: (HTTP over SSL or HTTP Secure)	Is the use of Secure Socket Layer (SSL) or Transport Layer Security (TLS) as a sublayer under regular HTTP application layering.
IP: (Internet Protocol)	A protocol by which data is sent from one computer to another computer over a network.
RPC: (Remote Procedure Call)	A protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.
SOAP: (Simple Object Access Protocol / Service Oriented Architecture Protocol)	A way for a program executing in one kind of operating system to communicate with a program executing in the same or another kind of operating system by using the World Wide Web's Hypertext Transfer Protocol (HTTP) and its Extensible Markup Language (XML) as the mechanisms for information exchange.

TCP: (Transmission Control Protocol)	A set of rules used along with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet.
WSDL: (Web Services Description Language)	An XML based general purpose language for describing interfaces, protocol bindings, and deployment details of network services.
CDATA: (Character DATA)	XML data that is not parsed. CDATA carries markup examples that would otherwise be interpreted as XML because of the tags.
DII: (Dynamic Invocation Interface)	A method of accessing web service resources through low level application programming interface (API) functions.
DOM: (Document Object Model)	A specification for a programming interface (API) from the W3C that allows programs and scripts to update the content, structure and style of HTML and XML documents.

6. Transport Overview (Informative)

This document describes implementation details for the Digital Program Insertion Advertising Systems Interfaces' message transport. Only one of the two transport mechanisms defined in this document is required for implementing SCTE 130 compliant services.

6.1. Document Organization

This document provides a detailed description of the required transport mechanisms. Subsequent sections focus on the description of the required transport mechanism and provides links to concrete implementation examples in the document appendices.

Section 7.0 explains this document's notational conventions and Section 8.0 identifies the processing conventions. Section 9.0 defines the XML namespace usage and the applicable XML semantics. Section 10.0 introduces reliable network delivery and section 11.1.0 introduces the SCTE 130 transport protocols followed by (INFORMATIVE) WEB SERVICES (SOAP), which contains non-normative information regarding the SOAP transport type.

7. Notational Conventions

7.1. Normative XML Schema

See [SCTE130-2] for information.

7.2. Document Conventions

This specification utilizes the same document conventions as [SCTE130-2]. See [SCTE130-2] for conventions and XML schema illustrations nomenclature explanations

8. Processing Conventions

8.1. Unknown/Unrecognized/Unsupported XML Elements and Attributes

See [SCTE130-2] for information.

9. XML Namespaces

This specification uses the 'trans' prefix, as described in Table 1, for the interface associated with the specific XML namespace URI that ***shall*** be used by all implementations. Table 1 lists the prefix, the corresponding namespace, and a description of the defining specification used herein.

Table 1: XML Namespace Declarations

Standard	XML Schema Prefix	XML Schema Elements	Value
2020 (latest)	core (SCTE 130-2)	Schema namespace	http://www.scte.org/schemas/130-2/2008a/core[1]
		Schema version attribute	20200321
		Schema filename	SCTE_130-2_core_20200321.xsd
	trans (This doc.)	Schema namespace	http://www.scte.org/schemas/130-7/2008/trans[2]
		Schema version attribute	20200603
		Schema filename	SCTE_130-7_trans_20200603.xsd
	xsd	Schema namespace	http://www.w3.org/2001/XMLSchema
	env	Schema namespace	http://schemas.xmlsoap.org/soap/envelope
	soap-env	Schema namespace	http://www.w3.org/2003/05/soap-envelope

[1] While this specification has a ratified year of 2020, the XML schema/XSD namespace has a year of 2008a, which is the year the XSD and this specification's syntax was initially ratified. All subsequent changes have been backwards compatible and thus, the namespace has not changed.
[2] While this specification has a ratified year of 2020, the XML schema/XSD namespace has a year of 2008, which is the year the XSD and this specification's syntax was initially ratified. All subsequent changes have been backwards compatible and thus, the namespace has not changed.

Unless otherwise stated, all references to XML elements illustrated in this document are from the 'trans' namespace. Elements from other namespaces will be prefixed with the name of the external namespace, e.g. <core:XXXX>.

10. Reliable Message Delivery

Message delivery, as defined in [SCTE130-2], describes the concept of reliable message acquisition through the use of positive message acknowledgement with the possibility of message retransmission. Each SCTE 130 top level request and/or notification message has a corresponding response and/or acknowledgement message. In addition, the specification allows for the concept of message retransmission via the @resend attribute.

When combined with a communication protocol such as TCP as defined in [IETF STD07], truly reliable message delivery between cooperating services can be achieved. Figure 1 illustrates the conceptual layering of services involved in reliable message delivery using TCP, with an additional layer for packet definition and routing.

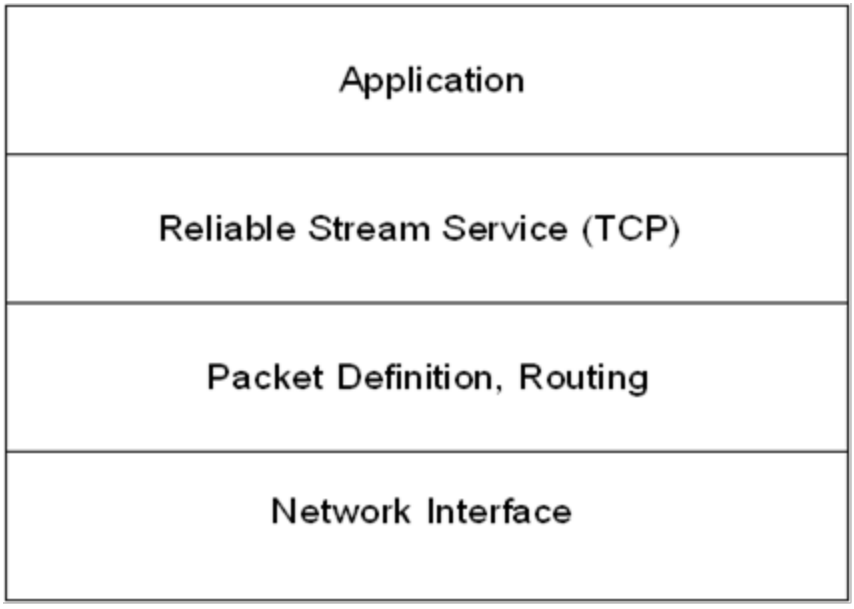

Figure 1: Network Layering

Each of the transport mechanisms described herein depend on the use of the reliable stream service (TCP) at some layer within their transport implementation. Thus, the use of TCP, or more specifically, TCP along with IP for packet definition and routing, becomes the de facto standard for message delivery within this specification.

11. SCTE 130-7 Message Transport Types

The following sections describe two SCTE 130-7 specified transport types for message delivery between cooperating SCTE 130 services.

SCTE 130 service implementations *may* implement the SOAP protocol, described in section 11.2 or the HTTP protocol, described in section 11.3.

11.1. Transport Types

The core:Address element described in [SCTE130-2] *may* contain an @type attribute that identifies the transport type associated with the specified address. The @type attribute ***shall*** be used and ***shall*** appear exactly as it does in Table 2 for the application selected transport protocol.

Table 2: Address Types

Transport Type	Description
SOAP1.1	SOAP transport protocol identifier for [W3C – SOAP1.1]
SOAP1.2	SOAP transport protocol identifier for [W3C – SOAP1.2]
HTTP	Hypertext Transport Protocol for [HTTP]
…	User defined and outside the scope of this specification. The string ***shall*** be prefixed with the text “private:”.

```
<core:Callout>
  <core:Address type="SOAP1.1">http://10.250.32.50/scte130</core:Address>
</core:Callout>
```

Example 1: Address Type Usage

Example 1 illustrates the use of the @type attribute within an core:Address element. In this example, the SOAP1.1 transport type has been specified as the protocol to be used when communicating with the URI contained within the core:Address element.

```
<core:Callout>
  <core:Address
type="private:HighSpeedTS">http://10.250.30.22/scte130</core:Address>
</core:Callout>
```

Example 2: Private Address Type

Example 2 illustrates the use of the ‘private’ keyword within the @type attribute value to allow for the use of an additional transport type other than those defined in Table 2.

A description of the private:HighSpeedTS transport type or any other private transport protocol is outside the scope of this document.

11.2. SOAP Transport

Web services, and SOAP in particular, cover a very broad range of implementation styles and techniques. SOAP originally stood for Simple Object Access Protocol, and more recently Service Oriented Architecture Protocol, but is now simply SOAP. The original acronym was dropped with version 1.2 of the standard, which became a W3C recommendation on June 24, 2003, as it was considered to be misleading.

All SCTE 130 Part-7 implementations *may* implement [W3C – SOAP1.1] and *may* implement [W3C – SOAP1.2].

Each interface described in the SCTE 130 specification is supported by WSDL definitions. Information Service oriented SCTE 130 services, like the CIS, SIS and POIS, ***shall*** contain two (2) port sections

within a single WSDL document. This separation of port definitions within information service WSDLs allows for the separation of client-side service endpoints from the server-side service endpoints. An example of this includes the cis:ContentNotification service endpoint, which *may* be implemented by [SCTE130-4] clients but not by CIS servers.

Other SCTE 130 services that are not information service oriented ***shall*** use a single WSDL port definition to define the services available at a particular service endpoint.

See Appendix A.2.1 for a brief description of WSDL file components.

SCTE 130 web-service implementations ***shall*** use the Document/Literal binding style for all SOAP bindings, as outlined in the [WS-I – Basic Profile 1.1]. See Appendix A.1.4 for additional information on the advantages of the Document/Literal binding style, and for a detailed description of all available binding styles.

11.2.1. WSDL Target Namespace URI Format

This document defines the WSDL target namespace URI format for the SOAP transport service interfaces associated with the SCTE 130 specification. This URI format ***shall*** be used by all implementations applying this specification.

WSDL target namespace URIs ***shall*** have the following structure:

[<prefix>/<part-#>/<version>/<interface name>]

The WSDL target namespace URIs for the separate parts within the SCTE 130 specification ***shall*** contain the following elements separated by the standard URI path separation character '/'.

<prefix> – The prefix element for all WSDL target namespaces ***shall*** contain the URI fragment [http://www.scte.org/wsdl].

<part-#> - The part number element ***shall*** contain a reference to the SCTE 130 part number for which the namespace has been defined.

<version> - The version number element ***shall*** contain a value which indicates the particular version of the WSDL target namespace.

<interface name> - The interface name element ***shall*** contain a reference to the particular interface of the SCTE 130 specification for which the WSDL target namespace has been defined.

The interface name element is a refinement of the <part-#> element and is used to identify individual interfaces within the same SCTE 130 specification part.

An example of the WSDL target namespace URI for [SCTE130-4] is illustrated in Example 3.

```
http://www.scte.org/wsdl/130-4/2011/cis
```

Example 3: Part-4 WSDL Target Namespace URI

Example 4 illustrates the WSDL target namespace URIs for [SCTE130-3].

```
http://www.scte.org/wsdl/130-3/2013/adm
http://www.scte.org/wsdl/130-3/2013/ads
```

Example 4: WSDL URI Target Namespaces for [SCTE130-3]

In Example 4, the part number of the two WSDL target namespace URIs are both the same since each target namespace comes from [SCTE130-3]. The interface name for each namespace is different and identifies the separate interfaces within [SCTE130-3]. This separation allows for separate implementations for the ADM and ADS to be built from the same specification.

11.2.2. Fault Notification

SOAP fault messages are the mechanism by which SOAP applications report errors 'upstream' to nodes earlier in the message path. The intended use of SOAP faults within this specification is for errors that are unique to the SOAP stack implementation only.

Errors that occur at the application level ***shall*** use the core:StatusCode element as described in [SCTE130-2] and not the SOAP fault mechanism described herein to communicate errors.

11.2.3. SOAP 1.1 Fault Message

The XML schema for the SOAP 1.1 Fault message is illustrated in Figure 2:

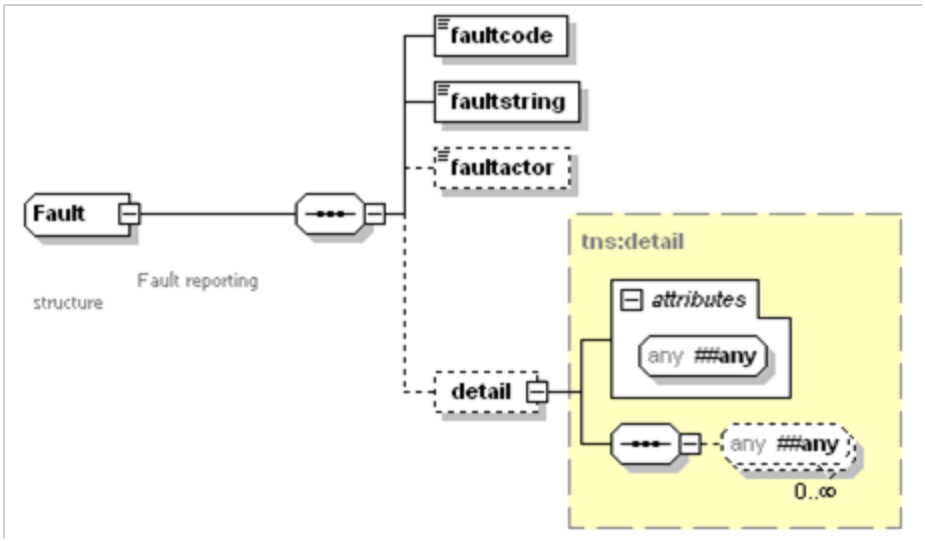

Figure 2: SOAP 1.1 Fault XML Schema

The SOAP 1.1 Fault message defines the following attributes and elements.

env:faultcode [Required, xsd:QName] – A code for identifying the fault. Codes that can be carried in the env:faultcode element include those defined in Table 3 and ***shall*** appear in the env:faultcode exactly as presented in this table.

Table 3: SOAP 1.1 Faultcode Values

Error	Description
VersionMismatch	The faulting node found an invalid element information item instead of the expected env:Envelope element information item. The namespace, local name or both did not match the expected env:Envelope element information item.

MustUnderstand	An immediate child element of the SOAP env:Header element. marked as mustUnderstand='1', was not understood by the receiving system.
Client	The message was incorrectly formed or contained incorrect information.
Server	The message could not be processed for reasons attributable to the processing of the message rather than to the contents of the message itself. For example, processing could include communicating with an upstream SOAP node, which did not respond. The message could succeed if resent at a later point in time.

env:faultstring [Required, xsd:string] – A human readable explanation of the fault. Note: While this element is required by the [W3C – SOAP1.1] specification, the type is xsd:string and can be zero length.

env:faultfactor [Optional, xsd:anyURI] – Information about who caused the fault to happen. Recipients of the soap:Fault message that do not represent the ultimate destination for the soap:Fault, must include the env:faultFactor element indicating the actual source of the fault.

env:detail [Optional] – Holds application specific error information related to the env:Body element.

For errors occurring within the SOAP stack, the SOAP fault message ***shall*** be used to communicate the error condition back to the initiator. Provided that support for the inclusion of application data within the env:detail element of the env:fault message is supported by the implementation SOAP stack, the respondent *should* include an ExceptionFaultReport (section 11.3.3) within the env:detail element.

Example 5 illustrates the use of the env:detail element to provide the initiator with additional information concerning the fault.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope"
  xmlns:core="http://www.scte.org/schemas/130-2/2008a/core"
  xmlns:trans="http://www.scte.org/schemas/130-7/2008/trans"/>
   <env:Header/>
   <env:Body>
      <env:Fault>
         <env:faultcode>env:Client</env:faultcode>
         <env:faultstring>java.lang.Exception: Failed to process
ServiceCheckRequest. Required attribute missing.
         </env:faultstring>
         <env:detail>
           <trans:ExceptionFaultReport>
             <core:StatusCode class="1" detail="1">
                <core:Note>Parse error. Required attribute identity missing</Note>
             </core:StatusCode>
             <trans:ErrantMessage>
               <![CDATA[<core:ServiceCheckRequest
                 messageId="BEE48AE6-62E7-2DF0-6611-13417C776E58"
                 system="10.250.30.22" version="1.0"/>]]>
             </trans:ErrantMessage>
```

```
            </trans:ExceptionFaultReport>
          </env:detail>
        </env:Fault>
    </env:Body>
</env:Envelope>
```

Example 5: SOAP 1.1 Fault with ExceptionFaultReport

In Example 5, an ExceptionFaultReport encapsulates an invalid core:ServiceCheckRequest message received at a service endpoint.

In this example the core:ServiceCheckRequest is missing the required @identity attribute and will not validate properly. The faulty message has been added to an ExceptionFaultReport and returned to the initiator in a env:Fault message.

11.2.4. SOAP 1.2 Fault Message

The XML schema for the SOAP 1.2 Fault message is illustrated in Figure 3.

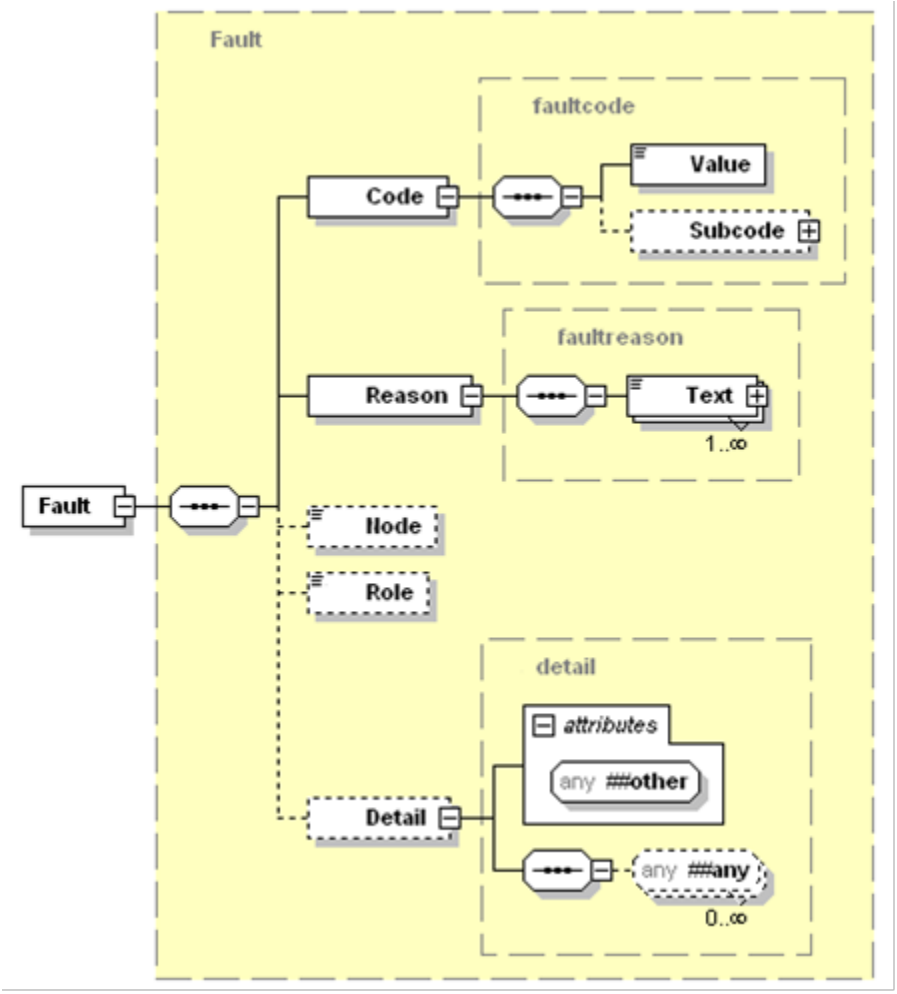

Figure 3: SOAP 1.2 Fault XML Schema

The SOAP 1.2 Fault message defines the following attributes and elements.

soap-env:Code [Required, soap-env:faultcode] – The code element contains a mandatory element tns:Value and an optional element soap-env:subCode. The soap-env:Value element contains a code for identifying the fault. Codes that can be carried in the soap-env:Value element include those defined in Table 4.

Table 4: SOAP 1.2 FaultCode Values

Error	Description
VersionMismatch	The faulting node found an invalid *element information item* instead of the expected Envelope element information item. The namespace, local name or both did not match the *envelope element information item* required by this recommendation.
MustUnderstand	An immediate child element of the SOAP env:Header element. marked as mustUnderstand='1', was not understood by the receiving system.
DataEncodingUnknown	A SOAP header block or SOAP body child element information item targeted at the faulting SOAP node is scoped with a data encoding that the faulting node does not support.
Sender	The message was incorrectly formed or did not contain the appropriate information in order to succeed. For example, the message could lack the proper authentication of payment information. It is generally an indication that the message is not to be resent without change.
Receiver	The message could not be processed for reasons attributable to the processing of the message rather than to the contents of the message itself. For example, processing could include communicating with an upstream SOAP node, which did not respond. The message could succeed if resent at a later point in time.

soap-env:reason [Required, soap-env:faultreason] – The Reason element information item is intended to provide a human-readable explanation of the fault.

soap-env:node [Optional, xsd:anyURI] – The Node element information item is intended to provide information about which SOAP node on the SOAP message path caused the fault to happen.

soap-env:role [Optional, xsd:anyURI] – The Role element information item identifies the role the node was operating in at the point the fault occurred.

soap-env:Detail [Optional, soap-env:detail] – The Detail element information item is intended for carrying application specific error information.

Example 6 illustrates the carriage of an ExceptionFaultReport within an soap-env:Detail element of a SOAP 1.2 Fault message.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<soap-env:Envelope
  xmlns:soap-env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:core="http://www.scte.org/schemas/130-2/2008a/core"
  xmlns:trans="http://www.scte.org/schemas/130-7/2008/trans"/>
   <soap-env:Header/>
   <soap-env:Body>
      <soap-env:Fault>
```

```
        <soap-env:Code>
            <soap-env:Value>soap-env:Sender</soap-env:Value>
        </soap-env:Code>
        <soap-env:Reason>
            <soap-env:Text xml:lang="en">
                Failed to parse message
            </soap-env:Text>
        </soap-env:Reason>
        <soap-env:detail>
          <trans:ExceptionFaultReport>
            <core:StatusCode class="1" detail="1">
              <core:Note>Parse error. Required attribute identity missing</Note>
            </core:StatusCode>
            <trans:ErrantMessage>
              <![CDATA[<core:ServiceCheckRequest
                messageId="BEE48AE6-62E7-2DF0-6611-13417C776E58"
                system="10.250.30.22" version="1.0"/>]]>
            </trans:ErrantMessage>
          </trans:ExceptionFaultReport>
        </soap-env:detail>
      </soap-env:Fault>
    </soap-env:Body>
</soap-env:Envelope>
```

Example 6: SOAP 1.2 Fault with ExceptionFaultReport

11.2.5. Message Ordering and Parallel Connections (Informative)

SOAP messaging is based on a request/response mechanism, which guarantees that only one request can be outstanding on any single connection at one time. Problems with message ordering can occur when multiple connections to a single endpoint are utilized to improve throughput. As an example, consider the transmission of two adm:PlacementStatusNotification messages to a single ADS endpoint. Each message contains a set of events that are all related, but only one message contains a adm:PlacementStatusEvent with the @type attribute set to ‘endAll’. (See [SCTE130-3] for a complete description of the @type attribute carried within the adm:PlacementStatusEvent element). If the messages are transmitted in the correct order but arrive in reverse order, the receiving ADS system *may* process the ‘endAll’ event before processing the remaining events in the other message.

There are several message types that can produce message ordering issues when used in conjunction with multiple parallel endpoints. These message examples are outlined in Table 5.

Table 5: Order Sensitive Message Types

Message Type	Order Sensitivity Problem Description
adm:PlacementStatusNotification	May cause the receiver to process @type ‘endAll’ before processing all required events.
adm:PlacementUpdateNotification	May cause the receiver to process placement updates in the wrong order.
cis:ContentNotification	May cause the receiver to process content notifications in the wrong order.

11.2.5.1. Message Ordering Solutions

Message ordering issues *may* be resolved by grouping messages with related transactions together for transmission on a single endpoint. As an example, consider the example given in section 11.2.5. The two adm:PlacementStatusNotifcation messages given in this example are related together through the context of the events carried in each message. Transmitting each adm:PlacementStatusNotification on the same physical connection eliminates messaging ordering issues that *may* occur when using separate physical endpoints.

11.2.6. Endpoint Addressing

SOAP endpoint addresses are expressed as URLs (Uniform Resource Locator). A URL is defined as a URI (Uniform Resource Identifier) that, in addition to identifying a resource, provides means of acting upon or obtaining a representation of the resource by describing its primary access mechanism or network.

SOAP URLs *should* have the following structure:

[http://<network address>:[port]/<resource name>]

An example of a SOAP end-point address is illustrated in Example 7.

```
<core:Callout>
  <core:Address type="SOAP1.1">http://10.250.30.22/ADSServer</core:Address>
</core:Callout>
```

Example 7: SOAP URL

In this example, the transport type is identified as ‘SOAP1.1’, the protocol type is HTTP, the network address is ‘10.250.30.22’ and the port is the default value of ‘80’. The resource name in this example is ‘ADSServer’.

Because the optional @message attribute has been omitted from the core:Callout element, the endpoint in **Error! Reference source not found.** is considered to be a ‘default’ service endpoint and ***shall*** support all of the message types for the associated service.

All SCTE 130-7 SOAP transport implementations ***shall*** support the HTTP transfer protocol and *may* support HTTPS.

See [SCTE130-2] for details on IPv4 and IPv6 addressing formats.

11.2.7. Connection Management

11.2.7.1. Message Timeliness

In the SOAP transport environment, the message exchange transaction is completely synchronous. Individual Request/Response and Notification/Acknowledgement transactions must be completed before the next message transaction can be executed.

Timeliness of the response/acknowledgement message for message transactions in the SOAP environment is implementation specific. Specific details on how to handle message timeliness issues is

outside the scope of this specification. Implementations *may* choose to set specific time limits on message transactions and utilize the resend functionality described in [SCTE130-2] to reconcile incomplete transactions.

11.2.7.2. Service Channel Termination

Service channels between SCTE 130 service implementations are considered logical connections and thus do not require a physical connection between services to remain active in order for the service channel to be considered intact. See [SCTE130-2] for complete details on the definition of a logical service channel and the normal life cycle associated with service channels.

Error conditions or other problems *may* create scenarios in which the viability of a service channel is in question. In this case, SCTE 130 service implementations *should* use the existing set of list registration and deregistration messages to either reaffirm service channel viability or to negotiate the tear down of any existing service channels.

For ADS to ADM communications, the core:ServiceCheckRequest message *should* be used to test connectivity between systems and the adm:ListADSRegistrationRequest message *should* be used by an ADS implementation to determine whether the expected service channel is still in a valid state.

Because the ADM cannot determine the viability of a service channel with an ADS, ADM instances recovering from error conditions *should* use the adm:ADSDeregistrationNotification message to force a service channel tear down between the ADM and ADS. The behavior of an ADS upon receipt of an adm:ADSDeregistrationNotification is beyond the scope of this document. It is reasonable to assume that ADS instances interested in maintaining a service channel with an ADM would attempt to re-establish a service channel connection with an ADM by issuing new core:ADSRegistrationRequest messages to the ADM.

For GIS derived service communications, the same course of action *should* be followed as outlined above, but with substitutions for the appropriate GIS derived messages for ListRegistration and Deregistration.

11.2.8. Discovery

Automatic discovery of SOAP transport services for SCTE 130 services is outside the scope of this document. SOAP transport endpoints *should* be determined through the use of a single ‘well known’ endpoint that *may* resolve to the published SOAP WSDL for the required SCTE 130 service implementation.

11.3. HTTP Transport

Support for the HTTP transport was added in the 2020 revision.

SCTE 130 messages *should* be carried as payload using the HTTP [HTTP]. (HTTPS *may* also be used and is outside the scope of this specification.)

11.3.1. Error Notification

Errors *may* be indicated using either the HTTP Status Code field or within the SCTE 130 response message using the core:StatusCode element. The recommended and preferred way is using the SCTE 130

response message. The actual error notification method is left up to the implementor to decide and is outside the scope of this specification.

11.3.2. Message Ordering and Parallel Connections (Informative)

Similar to section 11.2.5, the implementor must be aware and consider the implications of message order and parallel connections. See section 11.2.5 for addition information.

11.3.3. Endpoint Addressing

HTTP endpoint addresses are expressed as URLs similar to SOAP. See section 11.2.6 for additional information.

A HTTP endpoint address is illustrated in Example 8. The difference between a SOAP endpoint and an HTTP endpoint is the core:Address element's @type attribute is set to the value "HTTP".

```
<core:Callout>
  <core:Address type="HTTP">http://10.250.30.22/ADSServer</core:Address>
</core:Callout>
```

Example 8: HTTP URL

11.3.4. Connection Management

HTTP is similar to SOAP. See section 11.2.7 for additional information.

11.3.5. Discovery

Automatic discovery of HTTP transport services for SCTE 130 services is outside the scope this document.

11.4. TCP Transport

The TCP transport was removed from the standard in the 2020 revision.

Appendix A (INFORMATIVE) WEB SERVICES (SOAP)

A.1 Basic Description

Web Services and (SOAP) in particular, cover a very broad range of implementation styles and techniques. SOAP originally stood for Simple Object Access Protocol, and more recently Service Oriented Architecture Protocol, but is now simply SOAP. The original acronym was dropped with version 1.2 of the standard, which became a W3C recommendation on June 24, 2003, as it was considered to be misleading.

A WSDL document describes a Web Service. A WSDL binding describes how the service is bound to a messaging protocol, particularly the SOAP messaging protocol. A WSDL SOAP binding can be either a Remote Procedure Call (RPC) style binding or a document style binding. A SOAP binding can also have an encoded use or a literal use. This results in at least four (4) binding style or use models:

- RPC/encoded
- RPC/literal
- Document/encoded
- Document/literal

A fifth style is the Document/literal (wrapped). Unfortunately, this style has little support outside of a single vendor and thus shall not be expanded upon in this document.

Each of the styles listed above has a distinct set of advantages and disadvantages, which are outlined in the following sections.

A.1.1 RPC/encoded

In this binding style, the WSDL description of the web service is straight forward and easy to understand. Example 9 illustrates a snippet of a WSDL document for this style.

```
<message name="myMethodRequest">
        <part name="x" type="xsd:int"/>
        <part name="y" type="xsd:float"/>
</message>

<message name="empty"/>

<portType name="PT">
        <operation name="myMethod">
                <input message="myMethodRequest"/>
                <output message="empty"/>
        </operation>
</portType>

<binding …./>
```

Example 9: RPC/encoded WSDL

An example SOAP message for the service described by the previous WSDL is illustrated in Example 10.

```
<soap:envelope>
        <soap:body>
                <myMethod>
                        <x xsi:type="xsd:int">5</x>
                        <y xsi:type="xsd:float">5.0</y>
                </myMethod>
        </soap:body>
</soap:envelope>
```

Example 10: RPC/encoded SOAP message

There are a number of things to notice about the SOAP messages and the WSDL for RPC/encoded style web services:

Strengths:

- The WSDL is very straight forward and easy to understand. (Counter point) WSDL was not designed for human readability but for machine consumption.
- The operation name appears in the actual SOAP message. This allows the receiver to easily map the message into the correct method. (Counter point) This is only an advantage for the SOAP stack implementer.

Weaknesses:

- The type encoding info (xsi:type="xsd:int") is overhead which degrades performance.
- This SOAP message cannot be easily validated as the message has not been defined in XML Schema.
- This style is not [WS-I Basic Profile] compliant.

A.1.2 RPC/literal

In this binding style, the WSDL is essentially the same as in the RPC/encoded style. There are small changes to the binding section that indicate that the binding is now literal instead of encoded.

The RPC/literal SOAP message is also different, illustrated here in Example 11.

```
<soap:envelope>
        <soap:body>
                <myMethod>
                        <x>5</x>
                        <y>5.0</y>
                </myMethod>
        </soap:body>
</soap:envelope>
```

Example 11: RPC/literal SOAP message

There are a number of things to notice about the RPC/literal WSDL and SOAP message:

Strengths:

- The WSDL is still straightforward.
- The operation name still appears in the message
- The type encoding information is eliminated from the message.
- RPC/literal is [WS-I Basic Profile] compliant.

Weaknesses:

- The SOAP message still cannot be easily validated since the message is not described by XML Schema.

A.1.3 Document/encoded

This method is not [WS-I Basic Profile] compliant and thus not recommend for SCTE 130 message transport.

A.1.4 Document/literal

The WSDL for the Document/literal style changes considerably from the RPC/literal style:

```
<types>
        <schema>
                <element name="xElement" >
                        <compleType name="xElementType">
                            <sequence>
                                <element type="xsd:int" name="x"/>
                                <element type="xsd:float" name="y"/>
                            </sequence>
                        </complexType>
                </element>
        </schema>
</types>

<message name="myMethodRequest">
        <part name="request" element="xElement"/>
</message>

<message name="myMethodResponse"/>

<portType name="PT">
        <operation name="myMethod">
                <input message="myMethodRequest"/>
                <output message="myMethodResponse"/>
```

```
        </operation>
</portType>
```

Example 12: Document//literal WSDL

An example SOAP message for the service described by the previous Document/literal style WSDL is provided in Example 13.

```
<soap:envelope>
        <soap:body>
                <xElement>
                        <x>5</x>
                        <y>5.0</y>
                </xElement>
        </soap:body>
</soap:envelope>
```

Example 13: Document/literal SOAP message

There are several things to note about the Document/literal style and associated SOAP messages:

Strengths:

- There is no type encoding info in the SOAP message.
- The entire message can be validated. Everything within the SOAP body is defined by XML Schema.
- The entire document contained within the SOAP body is passed verbatim to the target method.

Weaknesses:

- The WSDL is more complicated. (Counter point) WSDL was made for machine consumption.
- The operation name is not in the SOAP message. (Counter point) This does make it more difficult for the SOAP stack implementer, but allows for the definition of business documents in XML without the need to include method name information in the XML or as part of the Schema.
- [WS-I Basic Profile] only allows one child in the SOAP body element. Document/literal does not eliminate this weakness, but, as in this example, it should be evident that this issue can be easily avoided.

A.1.5 Conclusion

The previous sections were provided to give the reader a basic understanding of the Web Service (SOAP) landscape. For SCTE 130, the choice of style/usage model will have an impact on interoperability of cooperating services and extensibility of the message structure. For these reasons, the choice of style/usage model is restricted to Document/literal only. See section 11.2 for additional details.

Reasons for this choice include the following:

- Message Extensibility: SCTE 130 messaging was designed with maximum extensibility in mind. Each SCTE 130 top level message contains an extension element that allows for the addition of

elements from other namespaces. The Document/literal style allows for the extensibility of messages.

- Message validation: SCTE 130 messaging is based on well defined XML Schema models. The Document/literal style of SOAP messaging allows for the direct reference of the SCTE 130 XML Schemas within the WSDL file.
- Interface Robustness: Changes to existing SCTE 130 messages will not break Document/literal style SOAP message interfaces. The same cannot be said for RPC/encoded/literal style interfaces.
- Simplicity: SCTE 130 messages are delivered in whole to the receiving service interface as DOM document elements. Parsing and validation of the elements are available to the service implementation and not buried within the SOAP stack.

A.2 Usage

The purpose of this section is to familiarize the reader with the structure of WSDL files that support Document/literal web services. Additionally, this section provides an example of a dynamic invocation web service client that does not require advanced tool support.

Note: This document assumes the use of WSDL version 1.1. See [W3C – WSDL] for additional details.

A.2.1 WSDL File Structure

WSDL files that support Document/literal web services contain a minimum of seven (7) separate elements used for the definition of a web service.

- Types: A container for data type definitions using a type system like XSD.
- Message: An abstract, typed definition of the data being communicated.
- Operation: An abstract description of an action supported by the service.
- PortType: An Abstract set of operations supported by one or more endpoints.
- Binding: A concrete protocol and data format specification for a particular port type.
- Port: A single endpoint defined as a combination of a binding and a network address.
- Service: A collection of related endpoints.

An example <types> element is illustrated in Example 14.

```
<types>
  <xsd:schema targetNamespace='http:// …. /adm'>
     <xsd:element name="PlacementRequest" type="adm:PlacementReq…"/>
     <xsd:element name="PlacementResponse" type="adm:PlacementRes…"/>
  </xsd:schema>
</types>
```

Example 14: WSDL Types Element

The <types> element contains the XML Schema definitions for the elements utilized in subsequent WSDL document sections. The entire context of the XML Schema does not have to be contained within the <types> section. The schema can be imported from another source into this section.

An example <message> element is illustrated in Example 15.

```
<message name='PlacementRequest'>
    <part element='adm:PlacementRequest' name='request'/>
</message>
<message name='PlacementResponse'>
    <part element='adm:PlacementResponse' name='response'/>
</message>
```

Example 15: WSDL Message Element

The <message> element section defines the parts or parameters that will be passed to the receiving web service method. In this example, a single part has been defined for each message. By definition, WSDL allows for multiple parts to be passed into a single receiving web service method, which is fine for RPC/literal or encoded web services, but will break [WS-I Basic Profile] compliance when used in the Document/literal mode.

Example <portType> and <operation> elements are illustrated in Example 16.

```
<portType name='ADMMessageServer'>
    <operation name='processPlacementRequest'>
        <input message='tns:PlacementRequest'/>
        <output message='tns:PlacementResponse'/>
    </operation>
</portType>
```

Example 16: WSDL PortType and Operation Elements

The <portType> element section defines the methods available on the web service. Note that in Document/literal form, method names are not provided in the actual SOAP body. The web service implementation itself is responsible for matching the incoming message body with a method defined in this section.

Each <operation> element describes the input parameters for each method as well as the output element. Each of these elements is a reference back to the one of the <message> elements defined above.

An example <binding> element is illustrated in Example 17.

```
<binding name='ADSMessageServerBinding' type='tns:ADSMessageServer'>
    <soap:binding style='document' transport='http://……/soap/http'/>
    <operation name="processPlacementRequest">
        <soap:operation soapAction=''/>
        <input>
            <soap:body use='literal' namespace='http://…../adm'/>
        </input>
        <output>
            <soap:body use='literal' namespace='http://…../adm'/>
        </output>
    </operation>
```

```
</binding>
```

Example 17: WSDL Binding element

The <binding> section contains the actual SOAP binding and a reference to the previously defined operation. In this example the ‘style’ attribute is set to ‘document’ and the ‘use’ attribute within the <soap:body> element is set to ‘literal’.

Example <service> and <port> elements are illustrated in Example 18.

```
<service>
    <port binding=’tns:ADSMessageServiceBinding’ name=’ADSPort’>
        <soap:address location=’http://10.250.30.22:8080/ADSServer’/>
    </port>
</service>
```

Example 18: WSDL Service and Port Elements

Finally, the <service> element contains the port binding and the physical address where the web service will be made available.

A.2.2 Web Service Client

There are a large number of tools available for creation of client side web service resources. Where tools are not available, service implementers will need to create client side code directly. This section describes a complete dynamic invocation interface (DII) web service client for use with SCTE 130 web services.

The following DII example code is written in the Java programming language and uses libraries from the Apache AXIS 1.4 tool set.

```
import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;

import javax.xml.rpc.ServiceException;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConnection;
import javax.xml.soap.SOAPConnectionFactory;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;

import org.w3c.dom.Element;
public class WSClient {

    private URL url  = null;
```

```
    private SOAPConnectionFactory conFactory = null;
    private SOAPConnection            connection = null;

    public WSClient(URL url) {
            this.url  = url;
    }

    /**
     * This method returns the entire SOAPEnvelope
     */
    public SOAPElement invoke(SOAPElement message) {
            MessageFactory mf       = MessageFactory.newInstance();
            SOAPMessage   request = mf.createMessage();

            Request.getSOAPBody().addChildElement(message);


            If (connection == null) {
                    conFactory = SOAPConnectionFactory.newInstance();
                    connection = conFactory.createConnection();
            }

            SOAPMessage response = connection.call(request, url);

            return response.getSOAPBody().getParentElement();
    }
}
```

Example 19: DII Client

Example 19 contains a complete working example of a DII web services client. Error handling code has been removed to reduce the size of the example.

A.2.3 Creating SOAP Messages

Tools are also available for the creation of specific language bindings for SOAP messages using the individual XML Schemas as source documents. Where tools are not available, service implementers may need to manually create messages directly. Example 20 illustrates how a complete core:ServiceCheckRequest message can be constructed using standard Java SOAP libraries.

```
public SOAPElement getSOAPElement() {

    SOAPFactory  factory   = SOAPFactory.newInstance();
    SOAPElement message = null;

    message = factory.createElement("ServiceCheckRequest",
                                    "core",
```

```
                                        "http://www.scte.org/schemas/130-
2/2008a/core");

    message.setAttribute("messageId",  "my message id");
    message.setAttribute("version",      "version 1.1");
    message.setAttribute("identity",     "my identity");
    message.setAttribute("system",       "my system");

    return message;
}
```

Example 20: SOAPElement Creation

Note that in this example the extension element "Ext" has been left out of the message.

A.2.4 SOAP Message Examples

The physical message that is carried over the transport medium, in this case SOAP wrapped in HTTP, is illustrated in Example 21.

```
POST /axis/services/SCTE130MessageService HTTP/1.1
SOAPAction: ""
Content-Type: text/xml; charset=UTF-8
Authorization: Basic YWRtaW46YWRtaW5pc3RyYXRvcg==
User-Agent: Java/1.5.0_11
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 334

<env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'>
  <env:Header/>
  <env:Body>
    <core:ServiceCheckRequest identity='D7200AFF-2510-7A6B-624C-
59BED5689A28' messageId='D09666AF-3C6D-3AB8-9521-B2275FB5F6B6'
system='10.250.30.22' version='1.1' xmlns:core='http://www.scte.org/schemas/130-
2/2008a/core'/>
  </env:Body></env:Envelope>
```

Example 21: ServiceCheckRequest Message

The previous example contains a live message, indicated by the GUIDs used for the identity and messageId attributes. The entire top portion of the message starting with the word "POST" and extending to the "Content-Length: 334" is the HTTP envelope. The SOAP envelope begins with the XML "<env:Envelope….>" element. Note that no additional information is being carried in the envelope header and that the SOAP body contains a complete <core:ServiceCheckRequest> message. This is a typical Document/literal message, which does not contain the remote method name.

The response to the previous core:ServiceCheckRequest is illustrated in Example 22.

```
HTTP/1.1 200 OK
Content-Type: text/xml;charset=utf-8
Transfer-Encoding: chunked
Date: Fri, 02 Nov 2007 18:00:15 GMT
Server: Apache-Coyote/1.1

<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:adm="http://www.scte.org/schemas/130-3/2008a/adm">
    <soapenv:Body>
      <core:ServiceCheckResponse messageId="id" version="1.1" identity="identity"
system="system" messageRef="'D09666AF-3C6D-3AB8-9521-B2275FB5F6B6"
xmlns:core="http://www.scte.org/schemas/130-2/2008a/core">
        <core:StatusCode class="0" detail="0">
          <core:Note>Hello World.</core:Note>
        </core:StatusCode>
      </core:ServiceCheckResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Example 22: ServiceCheckResponse Message

The ServiceCheckResponse message returned from the service implementation is wrapped in an HTTP response envelope indicating a successful transport response <HTTP/1.1 200 OK>. The <core:ServiceCheckResponse> message is embedded within the response SOAP envelope and body.