International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

J.380.7

(11/2011)

SERIES J: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

Digital transmission of television signals

Digital program insertion – Advertising systems interfaces – Message transport

Recommendation ITU-T J.380.7

ITU-T

Recommendation ITU-T J.380.7

Digital program insertion – Advertising systems interfaces – Message transport

Summary

Recommendation ITU-T J.380.7 describes the digital program insertion advertising systems interfaces' transport protocols required for the exchange of messages defined in the ITU-T J.380.x series of Recommendations.

NOTE – Security issues surrounding the transport protocols defined herein have been purposely omitted and are considered outside of the scope of this Recommendation.

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T J.380.7	2011-11-13	9

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

Table of Contents

List of Figures

List of Tables

Introduction

Recommendation ITU-T J.380.7 describes implementation details for the digital program insertion advertising systems interfaces' message transport. Two transport mechanisms are described herein. Only one of the two transport mechanisms defined in this Recommendation is required for implementing ITU-T J.380 compliant services. The second transport protocol defined herein is optional, but is a highly recommended second choice for all implementations.

Recommendation ITU-T J.380.7 provides a detailed description of two separate transport mechanisms. Each subsequent clause focuses on the description of a particular transport mechanism, and provides links to concrete implementation examples in the appendices.

Clause 5 explains this Recommendation's notational conventions and identifies the processing conventions, and defines the XML namespace usage and the applicable XML semantics. Clause 6 introduces reliable network delivery and clause 7 introduces the ITU-T J.380 transport protocols followed by Appendix I, which contain non-normative information regarding the SOAP and TCP transport types.

Recommendation ITU-T J.380.7

Digital program insertion – Advertising systems interfaces – Message transport

1 Scope

This Recommendation describes the digital program insertion advertising systems interfaces' transport protocols required for the exchange of messages defined in the parts which make up the ITU-T J.380.x series of Recommendations.

NOTE – Security issues surrounding the transport protocols defined herein have been purposely omitted and are considered outside of the scope of this Recommendation.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T J.380.2]	Recommendation ITU-T J.380.2 (2011), *Digital program insertion – Advertising systems interfaces – Core data elements*.
[ITU-T J.380.3]	Recommendation ITU-T J.380.3 (2011), *Digital program insertion – Advertising systems interfaces – Management interface*.
[ITU-T J.380.4]	Recommendation ITU-T J.380.4 (2011), *Digital program insertion – Advertising systems interfaces – Content information service*.
[ITU-T J.380.x]	Recommendation ITU-T J.380.x-series (2011), *Digital program insertion – Advertising systems interfaces.*
[IETF RFC 791]	IETF RFC 791 (1981) *Internet Protocol*.
[IETF RFC 793]	IETF RFC 793 (1981) *Transmission Control Protocol*.
[SCTE 130-7 Schema]	ANSI/SCTE 130-7-2010, *Digital Program Insertion–Advertising Systems Interfaces Part 7 – Message Transport s*chema file.
[W3C – SOAP1.1]	W3C Note (2000), *Simple Object Access protocol (SOAP) 1.1*.
[W3C – SOAP1.2]	W3C Recommendation (2007), *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*.
[XMLSchemaP1]	W3C Recommendation (2004), *XML Schema Part 1: Structures (Second Edition).* <http://www.w3.org/TR/xmlschema-1/>
[XMLSchemaP2]	W3C Recommendation (2004), *XML Schema Part 2: Datatypes (Second Edition).* <http://www.w3.org/TR/xmlschema-2/>

3 Definitions

Throughout this Recommendation the terms below have specific meanings. Because some of the terms which have very specific technical meanings are defined in other parts of [ITU-T J.380.x], the reader is referred to the original source for their definition. For terms defined by this standard, brief definitions are given below.

All [ITU-T J.380.2] definitions are included herein. See [ITU-T J.380.2] for additional information.

3.1 Terms defined elsewhere

None.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 character DATA (CDATA): XML data that is not parsed. CDATA carries markup examples that would otherwise be interpreted as XML because of the tags.

3.2.2 document object model (DOM): 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.

3.2.3 dynamic invocation interface (DII): A method of accessing web service resources through low level application programming interface (API) functions.

3.2.4 HTTP over SSL or HTTP secure (HTTPS): This is the use of Secure Socket Layer (SSL) or Transport Layer Security (TLS) as a sub-layer under regular HTTP application layering.

3.2.5 hypertext transfer protocol (HTTP): 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.

3.2.6 Internet protocol (IP): A protocol by which data is sent from one computer to another computer over a network.

3.2.7 remote procedure call (RPC): 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.

3.2.8 simple object access protocol/service oriented architecture protocol (SOAP): 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.

3.2.9 transmission control protocol (TCP): A set of rules used along with IP to send data in the form of message units between computers over the Internet.

3.2.10 Web services description language (WSDL): An XML based general purpose language for describing interfaces, protocol bindings, and deployment details of network services.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

All [ITU-T J.380.2] abbreviations are included herein. See [ITU-T J.380.2] for additional information. Further to those:

CDATA Character Data

DII	Dynamic Invocation Interface
DOM	Document Object Model
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP over SSL or HTTP Secure
IP	Internet Protocol
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol / Service Oriented Architecture Protocol
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
WSDL	Web Services Description Language

5 Conventions

5.1 Notational conventions

5.1.1 Normative XML schema

See [ITU-T J.380.2] for information.

5.1.2 Document conventions

This specification utilizes the same document conventions as [ITU-T J.380.2]. See [ITU-T J.380.2] for conventions and XML schema illustrations nomenclature explanations.

5.2 Processing conventions

Unknown/Unrecognized/Unsupported XML elements and attributes. See [ITU-T J.380.2] for information.

5.3 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 and the corresponding namespace, and gives a description of the defining specification used herein.

Table 1 – XML namespace declarations

Prefix	Namespace	Description
core	http://www.scte.org/schemas/130-2/2008a/core	See [ITU-T J.380.2]
trans	http://www.scte.org/schemas/629-7/2008/trans	ITU-T J.380.7
env	http://schemas.xmlsoap.org/soap/envelope	See [W3C – SOAP1.1]
soap-env	http://www.w3.org/2003/05/soap-envelope	See [W3C – SOAP1.2]
xsd	http://www.w3.org/2001/XMLSchema	See [XMLSchemaP1] and [XMLSchemaP2]

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>.

6 Reliable message delivery

Message delivery, as defined in [ITU-T J.380.2], describes the concept of reliable message acquisition through the use of positive message acknowledgement with the possibility of message retransmission. Each [ITU-T J.380.x] 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, 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.

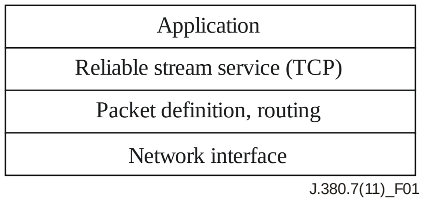

Figure 1 – Network layering

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.

7 ITU-T J.380.7 message transport types

The following clauses describe three ITU-T J.380.7 specified transport types for message delivery between cooperating ITU-T J.380 services.

ITU-T J.380 service implementations shall implement the SOAP protocol, described in clause 7.2, and should implement the TCP protocol described in clause 7.3.

7.1 Transport types

The core:Address element described in [ITU-T J.380.2] 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]
TCP	TCP transport protocol identifier
…	User defined and outside the scope of this Recommendation. The string shall be prefixed with the text "private:".

```
<core:Callout>
  <core:Address type="SOAP1.1">http://10.250.32.50/J.380</core:Address>
</core:Callout>
```

Figure 2 – Example 1: Address type usage

Figure 2 illustrates the use of the @type attribute within a 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:HTTP">http://10.250.30.22/J.380</core:Address>
</core:Callout>
```

Figure 3 – Example 2: Private address type

Figure 3 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:HTTP transport type or any other private transport protocol is outside the scope of this document.

7.2 SOAP transport

Web services and SOAP in particular, cover a very broad range of implementation styles and techniques. SOAP originally stood for s*imple object access protocol*, and more recently s*ervice 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 ITU-T J.380.7 implementations shall implement [W3C – SOAP1.1] and should implement [W3C – SOAP1.2].

Each interface described in [ITU-T J.380.x] is supported by WSDL definitions. Information service oriented ITU-T J.380 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 [ITU-T J.380.4] clients but not by CIS servers.

Other ITU-T J.380 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 clause I.2.1 for a brief description of WSDL file components.

ITU-T J.380 web-service implementations shall use the Document/literal binding style for all SOAP bindings, as outlined in [b-WS-I-Basic Profile 1.1]. See clause I.1.4 for additional information on the advantages of the Document/literal binding style, and for a detailed description of all available binding styles.

7.2.1 WSDL target namespace URI format

This document defines the WSDL target namespace URI format for the SOAP transport service interfaces associated with [ITU-T J.380.x]. This URI format shall be used by all implementations applying this Recommendation.

WSDL target namespace URIs shall have the following structure:

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

The WSDL target namespace URIs for the separate parts within [ITU-T J.380.x] 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 ITU-T J.380 part number for which the namespace has been defined (e.g., 3 for ITU-T J.380.3).

<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 ITU-T J.380 Recommendation 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 ITU-T J.380 Recommendation part.

An example of the WSDL target namespace URI for [ITU-T J.380.4] is illustrated in Figure 4.

```
http://www.scte.org/wsdl/130-4/2008a/cis
```

Figure 4 – Example 3: Part-4 WSDL target namespace URI

Figure 5 illustrates the WSDL target namespace URIs for [ITU-T J.380.3].

```
http://www.scte.org/wsdl/130-3/2008a/adm
http://www.scte.org/wsdl/130-3/2008a/ads
```

Figure 5 – Example 4 – WSDL URI target namespaces for [ITU-T J.380.3]

In Figure 5, the part number of the two WSDL target namespace URIs are both the same since each target namespace comes from [ITU-T J.380.3]. The interface name for each namespace is different and identifies the separate interfaces within [ITU-T J.380.3]. This separation allows for separate implementations for the ADM and ADS to be built from the same specification.

7.2.2 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 Recommendation 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 [ITU-T J.380.2] and not the SOAP fault mechanism described herein to communicate errors.

7.2.3 SOAP 1.1 fault message

The XML schema for the SOAP 1.1 fault message is illustrated in Figure 6.

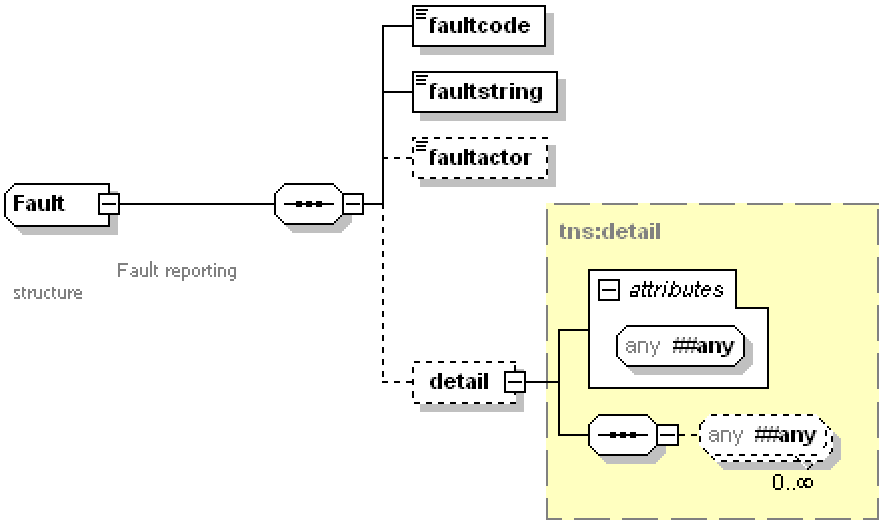

Figure 6 – SOAP 1.1 fault XML schema

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 that occur 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 (clause 7.3.3) within the env:detail element.

Figure 7 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</core: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>
```

Figure 7 – Example 5: SOAP 1.1 fault with ExceptionFaultReport

In Figure 7, 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.

7.2.4 SOAP 1.2 fault message

The XML schema for the SOAP 1.2 fault message is illustrated in Figure 8.

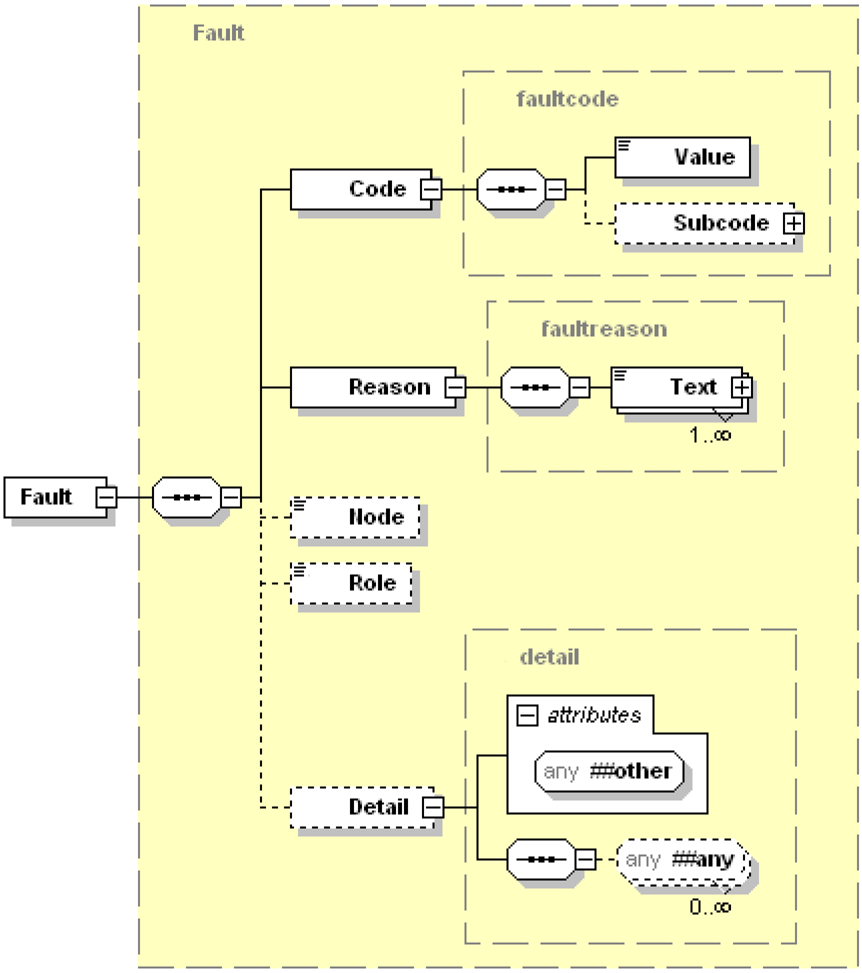

Figure 8 – SOAP 1.2 fault XML schema

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.

Table 4 – SOAP 1.2 FaultCode values

Error	Description
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.

Figure 9 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</core: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>
```

Figure 9 – Example 6: SOAP 1.2 fault with ExceptionFaultReport

7.2.5 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 [ITU-T J.380.3] 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 messages 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.

7.2.6 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 clause 7.2.5. 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.

7.2.7 Endpoint addressing (Normative)

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:

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

An example of a SOAP end-point address is illustrated in Figure 10.

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

Figure 10 – 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 Figure 15 is considered to be a 'default' service endpoint and shall support all of the message types for the associated service.

All ITU-T J.380.7 SOAP transport implementations shall support the HTTP transfer protocol and may support HTTPS.

See [ITU-T J.380.2] for details on IPv4 and IPv6 addressing formats.

7.2.7.1 Connection management

7.2.7.1.1 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 of the scope of this Recommendation. Implementations may choose to set specific time limits on message transactions and utilize the resend functionality described in [ITU-T J.380.2] to reconcile incomplete transactions.

7.2.7.2 Service channel termination

Service channels between ITU-T J.380 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 [ITU-T J.380.2] 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, ITU-T J.380 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 behaviour of an ADS upon receipt of an adm:ADSDeregistrationNotification is beyond the scope of this Recommendation. 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.

7.2.8 Discovery

Automatic discovery of SOAP transport services for ITU-T J.380 services is outside of the scope of this Recommendation. 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 ITU-T J.380 service implementation.

7.3 TCP transport

The transmission control protocol (TCP) provides a reliable, in-order delivery of a stream of bytes. TCP software libraries use the underlying IP layer and provide a simple interface, i.e., socket layer interface, to applications by hiding most of the underlying packet structures, rearranging out-of-order packets, minimizing network congestion, and re-transmitting lost packets.

TCP offers a highly scalable alternative ITU-T J.380 transport. To deploy TCP as a transport for [ITU-T J.380.x], a general message structure, referred to as the TCPMessageHeader, shall be utilized to encapsulate the ITU-T J.380 message. The TCPMessageHeader is responsible for identifying the message boundary, notifying the message originator of faults and managing the TCP connection.

7.3.1 TCP message header

TCP is a byte stream protocol. To detect ITU-T J.380 message boundaries within the TCP stream, a general message structure is necessary. All TCP ITU-T J.380 transmissions consist of a fixed-length header (the TCPMessageHeader) immediately followed by the ITU-T J.380 XML message payload.

The TCPMessageHeader shall consist of an 8 octet fixed-length header. The format of this header shall be as follows in Figure 11.

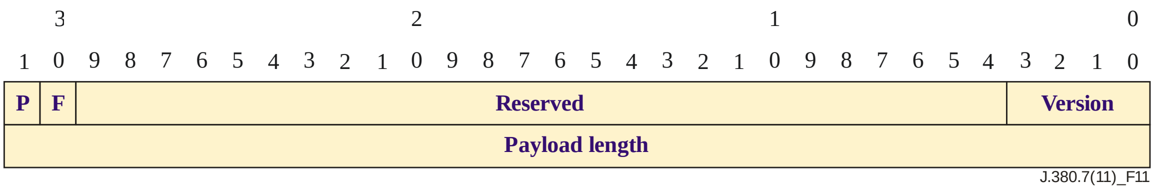

Figure 11 – TCPMessageHeader

– A 1-bit "***P***" field indicates that the TCPMessageHeader contains "Private" information. A value of 1b shall indicate the header contains private information where the F-bit, the Reserved bits and the Version bits are privately defined and the definition is outside the scope of this Recommendation. The 32-bit payload length shall remain present as defined by the standard. A value of 0b indicates the header conforms to the ITU-T J.380 transport standard defined herein. Figure 12 below illustrates the structure of the TCPMessageHeader when the 1-bit '***P***' field is set to a value of 1b.

– A 1-bit "***F***" field represents a "Fault" message generated by the responder. A fault is signalled when processing of the XML payload experiences an error. A value of 1b shall indicate a fault was generated by the respondent and that an ExceptionFaultReport element is contained within the message payload. See clause 7.3.3 for additional information. A value of 0b shall indicate that an ITU-T J.380 message (not an ExceptionFaultReport element) is contained within the payload. An initiator shall always set this field to the value 0b.

– "***Reserved***" indicates these bits are reserved for future use. A reserved bit shall have the value 0b.

– A 4-bit "***Version***" field specifies the TCPMessageHeader version. This version is specific to the TCPMessageHeader; hence, this field shall not be associated with any other version fields within [ITU-T J.380.x]. For this revision of this Recommendation, this field's value shall be "1" (i.e., 0001b). The value 0000b and the values 0002b through 1111b are reserved for future use and shall not be used.

– A 4-octet "***Payload Length***" field shall contain the length of the payload (i.e., the J.380 XML payload message length) in octets which follow this field (i.e., the 8 octet header length is not included in this value).

– The header shall be transmitted using network byte ordering [IETF RFC 791]. See clause 7.3.4 for additional details on network byte ordering.

Figure 12 illustrates the structure of the TCPMessageHeader when used to transport private information in the 'F', 'Reserved' and 'Version' fields.

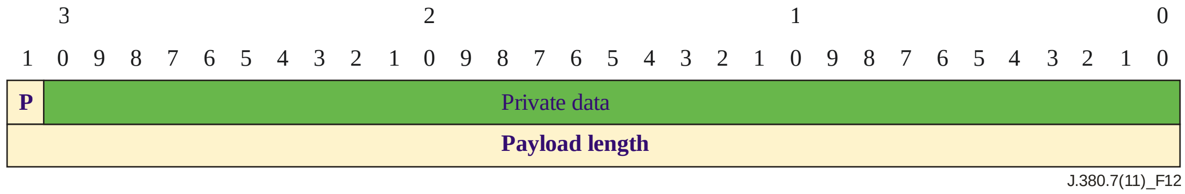

Figure 12 – TCPMessageHeader with private data

7.3.2 TCP fault notification

Much like the SOAPFault message described in clause 7.2.1 above, the TCP fault notification mechanism provides the implementer with a mechanism to respond to errors occurring prior to application involvement in message processing. An example is the receipt of a message within the TCPMessageHeader payload that cannot be properly parsed or is undefined to the application. e.g., a adm:PlacementRequest message arrives at a CIS implementation.

If a responder fails to parse an ITU-T J.380 message or receives an unknown message, the responder shall communicate the error condition to the initiator by first setting the fault indicator (F-bit) within the TCPMessageHeader to '1b' and inserting an ExceptionFaultReport message into the body of the response. See clause 7.3.3 for additional information on the ExceptionFaultReport message.

On receipt of a 'fault' message, the initiator should invoke the appropriate fault handler, which may then take the appropriate actions. For example, the fault handler may log the fault message and forward the failing message's identifier and error code to the upper application for additional processing. The implementation behaviour is outside the scope of this Recommendation.

7.3.3 ExceptionFaultReport

The ExceptionFaultReport message is specifically designed to allow a responder to inform the initiator of exceptions that have occurred within the transport handling layers of a remote application.

The XML schema definition for this message is illustrated in Figure 13.

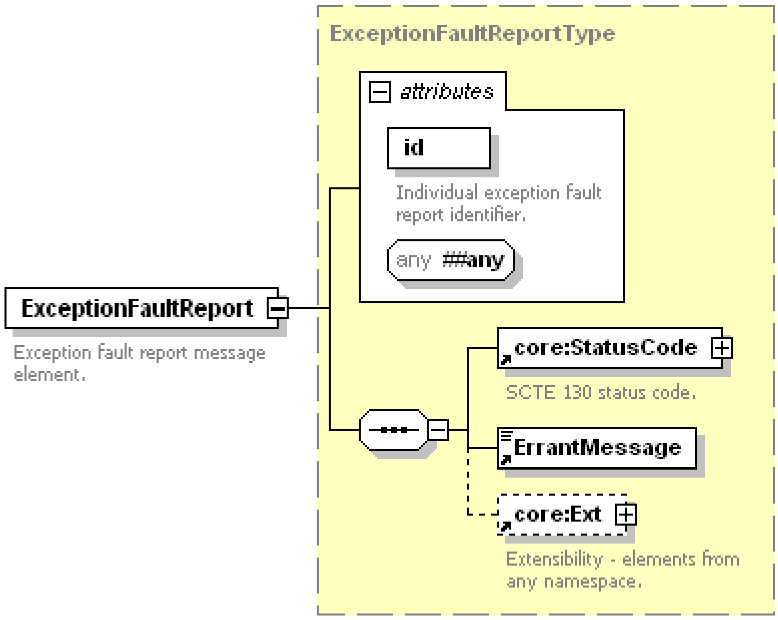

Figure 13 – ExceptionFaultReport XML schema

The ExceptionFaultReport message defines the following attributes and elements.

@id [Required] – A fault identification attribute. This value allows for the fault identification on the respondent. This value should be a GUID, but may be some other implementation specific value. For example, the identifier may be a simple counter.

@##any [Optional] – Any additional attribute from any namespace.

core:StatusCode [Required] – An core:StatusCode element for communicating status information to the initiator. See [ITU-T J.380.2] for additional information.

ErrantMessage [Required, core:nonEmptyStringType] – The ErrantMessage element is defined as a core:nonEmptyStringType. This element is used in the context of the ExceptionFaultReport to return the original initiator message that was involved in the fault. Specifically, the ErrantMessage element shall contain a CDATA section that can be used to transport the original initiator message, regardless of whether the original message is well formed or not.

core:Ext [Optional] – A container for any additional elements from any namespace. See [ITU-T J.380.2] for additional information.

Figure 14 illustrates how the ExceptionFaultReport message notifies an initiator of a failure by the respondent to properly process a badly formed core:ServiceCheckRequest message.

```
<ExceptionFaultReport>
  <core:StatusCode class="1" detail="1">
    <Note>Parse error. Required attribute identity missing.</Note>
  </core:StatusCode>
  <ErrantMessage>
      <![CDATA[
      <core:ServiceCheckRequest
        messageId="BEE48AE6-62E7-2DF0-6611-13417C776E58"
        system="10.250.30.22" version="1.1"  />
     ]]>
  </ErrantMessage>
</ExceptionFaultReport>
```

Figure 14 – Example 8: ExceptionFaultReport

In this example, the core:ServiceCheckRequest message enclosed within the ErrantMessage element does not contain the required @identity attribute. This error may have been detected during a validation process performed on the message payload before delivery to the application layer.

7.3.4 Byte ordering (network byte ordering)

Unlike the SOAP protocol described above in clause 7.2, the TCP protocol does not automatically handle the transmission of multi-byte data structures in a way that guarantees proper presentation at the receiver. TCP is a byte stream oriented transport mechanism which delivers individual bytes to a receiver in exactly the same order in which they were sent. This presents a problem for cooperating systems using different byte ordering schemes.

Networking protocols, like TCP, use big-endian byte ordering for exchanging multi-byte integers. Hence, big-endian byte ordering has become synonymous with network byte ordering. The ITU-T J.380 TCP transport shall use big-endian byte order for the network byte order. See [IETF RFC 791] for additional details on network byte ordering.

To pass the TCPMessageHeader, a binary structure containing multi-byte integers, the endpoints must utilize the same byte ordering on the wire and be prepared to swap byte ordering at both the transmitter as well as the receiver. Thus, before sending a message over TCP, the transmitting host converts the multi-byte header to network byte order from host byte ordering. On receipt of the transmission, the respondent must also convert the multi-byte header back to host byte order from network byte order.

7.3.5 Message ordering and parallel connections

Unlike the SOAP transport, a single TCP connection can support multiple outstanding request/notification messages simultaneously. This allows a client to use a single TCP address for all transmissions to a particular endpoint. Using a TCP connection in this fashion will require the client to properly order response/acknowledgement messages for execution. This ordering shall be done using the @messageId and @messageRef attributes of the request/response or notification/acknowledgement message pair.

If an implementation chooses to use multiple parallel TCP connections to the same endpoint, then the same message ordering issues, as outlined in clause 7.2.5, apply, and the implementation shall group related message transmission together on the same connection.

7.3.6 Endpoint addressing

TCP endpoint addresses are expressed as URLs (uniform resource locator). A URL is defined as a URI (uniform resource identifier) that, in additional to identifying a resource, provides means of acting upon or obtaining a representation of the resource by describing its primary access mechanism or network.

TCP URLs should have the following structure:

```
[network address>:[port]]
```

An example of a TCP end-point address is illustrated in Figure 15.

```
<core:Callout>
  <core:Address type="TCP">10.250.30.22:5659</core:Address>
</core:Callout>
```

Figure 15 – Example 9: TCP URL

In this example, the protocol is identified as 'TCP', the network address is '10.250.30.22' and the port is '5659'. This single address can be used to service all or some part of the total number of messages that can be addressed to an ITU-T J.380 server.

Because the optional @message attribute has been omitted from the core:Callout element, this endpoint described in Figure 15 is considered to be a 'default' service endpoint and shall support all of the message types for the associated service.

See [ITU-T J.380.2] for details on IPv4 and IPv6 addressing formats.

7.3.7 Connection management

Unlike web services, multiple outstanding request/notification messages may be outstanding at any one time on a single TCP connection. This allows the initiator to utilize a single TCP endpoint for communication with a remote implementation instead of managing a connection pool of web service connections for parallel activities. Furthermore, using a single TCP connection eliminates message ordering issues that may arise when multiple connections are open to the same endpoint.

Applications shall ensure that proper matching between request and response messages is maintained. This shall be done by properly matching the @messageId of a request/notification message with the @messageRef attribute of the corresponding response/acknowledgement message.

7.3.7.1 Message timeliness

In the TCP transport environment the message exchange transaction may be completely asynchronous. Individual request/response and notification/acknowledgement transactions are not required to be completed before the next message transaction may be initiated.

Timeliness of the response/acknowledgement message for a message transaction in the TCP environment is implementation specific. Specific details on how to handle message timeliness issues is outside of the scope of this Recommendation. Implementations may choose to set specific time limits on message transactions and utilize the resend functionality described in [ITU-T J.380.2] to reconcile incomplete transactions with respondents.

7.3.7.2 Service channel termination

See clause 7.2.7.2.

7.3.8 Discovery

Automatic discovery of TCP transport services for ITU-T J.380 services is outside of the scope of this Recommendation. TCP transport endpoints should be determined through the use of a single 'well known' endpoint that may resolve to a default service endpoint for an ITU-T J.380 service.

Appendix I

Web Services (SOAP)

(This appendix does not form an integral part of this Recommendation.)

I.1 Basic description

Web services and (SOAP) in particular, cover a very broad range of implementation styles and techniques. SOAP originally stood for s*imple object access protocol*, and more recently s*ervice 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 Recommendation.

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

I.1.1 RPC/encoded

In this binding style, the WSDL description of the web service is straight forward and easy to understand. Figure I.1 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 …./>
```

Figure I.1 – Example 10: RPC/encoded WSDL

An example SOAP message for the service described by the previous WSDL is illustrated in Figure I.2.

```
<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>
```

Figure I.2 – Example 11: 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 straightforward 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 [b-WS-I-Basic Profile 1.1] compliant.

I.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 Figure I.3.

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

Figure I.3 – Example 12: 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 [b-WS-I-Basic Profile 1.1] compliant.

Weaknesses:

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

I.1.3 Document/encoded

This method is not [b-WS-I-Basic Profile 1.1] compliant and thus not recommend for ITU-T J.380 message transport.

I.1.4 Document/literal

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

```
<types>
  <schema>
    <element name="xElement" >
      <complexType 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>
```

Figure I.4 – Example 13: Document//literal WSDL

An example SOAP message for the service described by the previous Document/literal style WSDL is provided in Figure I.5.

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

Figure I.5 – Example 14: 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 tougher 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.

– [b-WS-I-Basic Profile 1.1] 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.

I.1.5 Conclusion

The previous sections were provided to give the reader a basic understanding of the web service (SOAP) landscape. For [ITU-T J.380.x], 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 clause 7.2 for additional details.

Reasons for this choice include the following:

– **Message extensibility**: ITU-T J.380 messaging was designed with maximum extensibility in mind. Each ITU-T J.380 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**: ITU-T J.380 messaging is based on well-defined XML Schema models. The Document/literal style of SOAP messaging allows for the direct reference of the ITU-T J.380 XML schemas within the WSDL file.

– **Interface robustness**: Changes to existing ITU-T J.380 messages will not break Document/literal style SOAP message interfaces. The same cannot be said for RPC/encoded/literal style interfaces.

– **Simplicity**: ITU-T J.380 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.

I.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 Recommendation assumes the use of WSDL version 1.1. See [b-W3C-WSDL] for additional details.

I.2.1 WSDL file structure

WSDL files that support Document/literal web services contain a minimum of seven 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 Figure I.6.

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

Figure I.6 – Example 15: 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 Figure I.7.

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

Figure I.7 – Example 16: 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 [b-WS-I-Basic Profile 1.1] compliance when used in the Document/literal mode.

Example <portType> and <operation> elements are illustrated in Figure I.8.

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

Figure I.8 – Example 17: 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 Figure I.9.

```
<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>
```

Figure I.9 – Example 18: 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 Figure I.10.

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

Figure I.10 – Example 19: 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.

I.2.2 Web service client

There are a large number of tools available for the creation of client side web service resources. Where tools are not available, service implementers will need to create a client side code directly. This section describes a complete dynamic invocation interface (DII) web service client for use with ITU-T J.380 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();
  }
}
```

Figure I.11 – Example 20: DII client

Figure I.11 contains a complete working example of a DII web services client. The error handling code has been removed to reduce the size of the example.

I.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. Figure I.12 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/629-
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;
}
```

Figure I.12 – Example 21: SOAPElement creation

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

I.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 Figure I.13.

```
POST /axis/services/DVS629MessageService 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/629-2/2008a/core"/>
   </env:Body>
</env:Envelope>
```

Figure I.13 – Example 22: 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 Figure I.14.

```
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/629-3/2008a/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/629-2/2008a/core">
            <core:StatusCode class="0" detail="0">
               <core:Note>Hello World.</core:Note>
            </core:StatusCode>
         </core:ServiceCheckResponse>
      </soapenv:Body>
   </soapenv:Envelope>
```

Figure I.14 – Example 23: 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.

Appendix B

TCP

(This appendix does not form an integral part of this Recommendation.)

II.1 Basic description

The Transmission Control Protocol (TCP) provides a reliable, in-order delivery of a stream of bytes. TCP software libraries use the underlying IP network and provide a simple interface, i.e., socket layer interface, to applications by hiding most of the underlying packet structures, rearranging out-of-order packets, minimizing network congestion and re-transmitting lost packets.

TCP is a durable and mature protocol and is widely available as a standard library package for most programming languages.

II.2 Usage

The purpose of this section is to familiarize the reader with the basic concepts of exchanging messages using the TCP transport protocol described in clause 7.3. All of the example code snippets are rendered using the Java programming language.

II.2.1 Consumer socket creation

Figure II.1 illustrates the basic steps required for the creation of the consumer side socket.

```
public void createSocket(int port) {

  ServerSocket socket = new ServerSocket(port);
  Socket            conn   = null;

  Try {
    conn = serverSocket.accept();
    conn.setTcpNoDelay(true);
  } catch (SocketException se) {
    Log.error("Failed to accept socket connection from client");
  }
  ….
}
```

Figure II.1 – Example 24: Server side socket creation

The code in Figure II.1 illustrates the creation of the consumer side socket and then blocks in the accept() method. Once a producer attempts to connect, a bi-directional socket is created and the option to disable Nagel's algorithm is set. At this point, the consumer is ready to accept messages from a producer.

For clarity this code has been reduced to its most simplistic form. In order for the consumer to accept additional connections on the socket, the code within the try/catch block will need to be moved to a separate thread of execution and placed into a looping construct.

II.2.2 Producer connection establishment

Figure II.2 illustrates the basic steps required for a producer to successfully connect to an existing consumer.

```
public void connectToService(String host, int port, long timeout) {

  Socket conn = new Socket();  // Create new socket connection
  conn.bind(null);                      // Bind to local port

  conn.connect(new InetSocketAddress(host. Port, timeout);

  conn.setKeepAlive(true);    // Keep the socket alive during inactivity
  conn.setTcpNoDelay(true);   // Disable Nagel's algorithm
  ….
}
```

Figure II.2 – Example 25: Client connection

Figure II.2 illustrates the creation of the producer side socket and the connection of that socket to a remote consumer. Several options are set on the socket to keep the socket alive and to disable Nagel's algorithm.

For clarity, no error or exception handling code has been included in this example.

II.2.3 Exchanging messages

Once a TCP transport connection has been established with a remote service, the producer is ready to start exchanging messages with the service consumer.

Figure II.3 illustrates how the producer may transmit a message to the consumer.

```
public void transmitMessage(byte[] message,
                                               Socket connection) {
  // Create a data output stream

  DataOutputStream out = null;
  out = DataOutputStream(connection.getOutputStream());

  // Create the TCPMessageHeader

  int headerInfo = 0;                                // TCPHeader 1st  4 octets
  int payloadLength = message.length;  // TCPHeader 2nd 4 octets

  // Populate the 'P', 'F' and 'Version' fields

  headerInfo |= ((isPrivate() ? 1 : 0) << 31); // Shift and assign
  headerInfo |= ((isFault() ? 1 : 0) << 30);    // Shift and assign
  headerInfo |= getHeaderVersion();             // Assign value of 1

  // Write out the TCPHeader components

  out.writeInt(headerInfo);         // Private, Fault, Reserved, Version
  out.writeInt(payloadLength);  // Payload length

  // Write the payload or fault message out to the byte stream

  if(isFault()) {
    out.write(getFaultMessageAsByteArray());
  } else {
    out.write(payload);
  }
  ….
}
```

Figure II.3 – Example 26: Message transmission

Figure II.4 illustrates how a consumer may intercept the TCPMessageHeader and finally read the payload off of the TCP transport byte stream.

```
public byte[] retrieveMessage(Socket connection) {

  // Create a data input stream

  DataInputStream in = null;
  in = DataInputStream(connection.getInputStream());

  // Read the TCPMessageHeader

  int headerInfo     = in.readInt(); // TCPHeader 1st  4 octets
  int payloadSize  = in.readInt(); // TCPHeader 2nd 4 octets

  // Extract the Private, Fault and Version values

  boolean isPrivate = (((headerInfo >> 31) & 0x01) == 1 ? true : false);
  boolean isFault    = (((headerInfo >> 30) & 0x01) == 1 ? true : false);
  int version            = (headerInfo & 0x0f);

  // Read the payload

  int bytesRead = 0;
  byte[] payload = new byte[payloadSize];

  while(bytesRead != payloadSize) {
    bytesRead += in.read(payload,      // Buffer
                                                         bytesRead,  // Starting byte in buffer
                                                         (payloadSize - bytesRead)); // Read amount
  }

  // Handle Fault and Private flags
    ….

  // Return the payload

  return payload;
}
```

Figure II.4 – Example 27: Consumer socket reader

Bibliography

[b-ITU-T J.380.5] Recommendation ITU-T J.380.5 (2011), *Digital program insertion – Advertising systems interfaces – Placement opportunity information service*.

[b-ITU-T J.380.6] Recommendation ITU-T J.380.6 (2011), *Digital program insertion – Advertising systems interfaces – Subscriber information service*.

[b-ITU-T J.380.8] Recommendation ITU-T J.380.8 (2011), *Digital program insertion – Advertising systems interfaces – General information service*.

[b-W3C-DOM] W3C Recommendation (2004), *Document Object Model (DOM) Level 3 Core Specification*.

[b-W3C-SOAP Part 0] W3C Recommendation (2007), *SOAP Version 1.2 Part 0: Primer*.

[b-W3C-URI] W3C Note (2001), *URIs, URLs and URNs: Clarifications and Recommendations 1.0.*

[b-W3C-WSDL] W3C Note (2001), *Web Services Description Language (WSDL) 1.1*.

[b-WS-I-Basic Profile 1.1] WS-I (2004), *Basic Profile Version 1.1*, Web Services-Interoperability Organization.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	**Cable networks and transmission of television, sound programme and other multimedia signals**
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2012