TS 101 321 V1.4.2 (1998-12)

Technical Specification

Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); Inter-domain pricing, authorization, and usage exchange

Reference

DTS/TIPHON-03004 (c8c00jdf.PDF)

Keywords

internet, network, interoperability, protocol,
telephony, IP

ETSI

Postal address

F-06921 Sophia Antipolis Cedex - FRANCE

Office address

650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16
Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Internet

secretariat@etsi.fr
Individual copies of this ETSI deliverable
can be downloaded from
http://www.etsi.org

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available **free of charge** from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://www.etsi.org/ipr).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by ETSI Project Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON).

Introduction

The contents of the present document are the result of contributions and discussions in Working Group 3. The structure of the present document and overall principles of the standard were agreed upon, but the details described in the text should be seen as a draft version for further study. The complete text is not yet approved.

1 Scope

The present document specifies a set of protocols and associated profiles to permit the exchange of inter-domain pricing, authorization, and settlement information between internet telephony operators. The protocols specified fulfil the essential requirements of such services, by providing appropriate functionality between multiple administrative domains in a secure manner. The specification also provides for non-standard extensions that permit co-operating parties to augment or replace the basic functionality.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.
- A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] American National Standards Institute. Accredited Standards Committee X9 Working Draft: American National Standard X9.42-1993: Public Key Cryptography for the Financial Services Industry: Management of Symmetric Algorithm Keys Using Diffie-Hellman. American Bankers Association, September 21, 1994.

[2] IETF RFC 1945 (May 1996): "Hypertext Transfer Protocol -- HTTP/1.0. T. Berners-Lee, R. Fielding & H. Frystyk".

[3] W3C Recommendation (10 February 1998): "*Extensible Markup Language (XML) 1.0*." (http://www.w3.org/TR/REC-xml).

[4] IETF RFC 2068 (January 1997): "Hypertext Transfer Protocol -- HTTP/1.1. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee".

[5] IETF RFC 2045 (November 1996): "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. N. Freed & N. Borenstein".

[6] Internet draft (March 1996): "*The SSL Protocol Version 3.0"* (http://www.netscape.com/eng/ssl3/ssl-toc.html) as amended by SSL 3.0 Errata (26 August 1996) (http://www.netscape.com/eng/ssl3/ssl-errata.html).

[7] ISO 4217 (1995): "Codes for the representation of currencies and funds".

[8] ISO 8601 (1988): "Data elements and interchange formats — Information interchange — Representation of dates and times".

[9] ITU-T Recommendation H.225.0 (1998): "Call Signalling Protocols and Media Stream Packetization for Packet Based Multimedia Communications Systems".

[10] ITU-T Recommendation H.245 (1998): "Control Protocol For Multimedia Communication".

[11] ITU-T Recommendation X.691 (1995): "Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)".

[12] ITU-T Recommendation E.164: "The international public telecommunication numbering plan".

[13] ITU-T Recommendation H.323 (1998): "Packet Based Multimedia Communications Systems".

[14] ITU-T Recommendation H.235 (1998): "Security and Encryption for H Series (H.323 and other H.245 based) Multimedia Terminals".

[15] National Institute of Standards and Technology, U.S. Department of Commerce NIST FIPS PUB 46-1 (January 1988): "Data Encryption Standard".

[16] National Institute of Standards and Technology, U.S. Department of Commerce NIST FIPS PUB 186 (18 May 1994): "Digital Signature Standard".

[17] National Institute of Standards and Technology, U.S. Department of Commerce NIST FIPS PUB 180-1 (31 May 1994): "Secure Hash Standard".

[18] IETF RFC 2311 (March 1998): "S/MIME Version 2 Message Specification. S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, L. Repka".

[19] IETF RFC 2268 (January 1998): "A Description of the RC2(r) Encryption Algorithm. R. Rivest".

[20] IETF RFC 1321 (April 1992): "The MD5 Message-Digest Algorithm. R. Rivest".

[21] PKCS #1 (RSA Laboratories): "RSA Encryption Standard. Version 1.5, November 1993".

[22] PKCS #7 (RSA Laboratories): "Cryptographic Message Syntax Standard. Version 1.5, November 1993".

[23] The Unicode Consortium. *The Unicode Standard*. Version 2.0.

[24] Dierks, Tim and Christopher Allen. *The TLS Protocol Version 1.0*. Work in progress.

[25] The Open Trading Protocol Consortium. Internet Open Trading Protocol Part 2: Specification. Version 0.9, 12 January 1998.

3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DH	Diffie-Hellman (key exchange)
DSA	Digital Signature Algorithm
DTD	Document Type Definition
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPSEC	Internet Protocol Security
MD5	Message Digest 5
MIME	Multipurpose Internet Mail Extensions
PIN	Personal Identification Number (e.g. for automated teller machines)
PKCS	Public Key Cryptography Standard
RAS	Registration Admission and Status
RC4	Rivest Cipher 4
RSA	Rivest Shamir Aldeman
SHA	Secure Hash Algorithm
S/MIME	Secure Multipurpose Internet Mail Extensions
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Uniform Resource Locator
UTC	Co-ordinated Universal Time
UTF	Universal Text Format
XML	Extensible Markup Language

4 Protocol Architecture

This clause introduces the protocol architecture for this specification. It identifies the major protocols used by communicating parties, and it outlines their relationship to each other. The clause also describes the overall format of messages exchanged by the protocols. The intent of this clause is to outline the framework for the standard’s protocols and message formats; later clauses detail specific profiles for these protocols and the specific message content.

4.1 Communication Protocols

As Figure 1 shows, conforming systems use a combination of the Hypertext Transfer Protocol (HTTP), and either the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) to transfer pricing, authorization, and usage information. As the figure indicates, these protocols are layered on top of the Transmission Control Protocol (TCP) for communication across Internet Protocol (IP) networks.

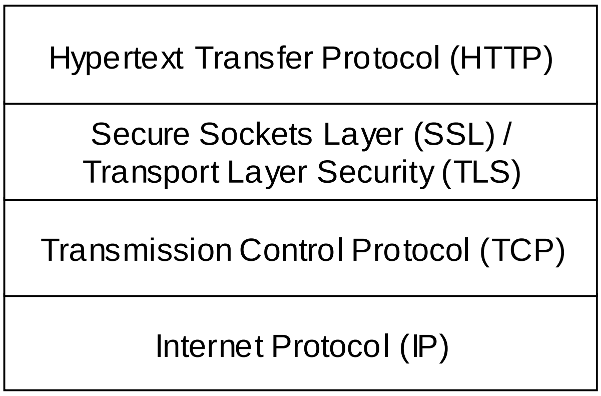
Hypertext Transfer Protocol (HTTP)
Secure Sockets Layer (SSL) / Transport Layer Security (TLS)
Transmission Control Protocol (TCP)
Internet Protocol (IP)

Figure 1: Protocol Architecture for Pricing, Authorization, and Usage Exchange

4.1.1 Secure Sockets Layer / Transport Layer Security

The Secure Sockets Layer and Transport Layer Security protocols add authentication and privacy to TCP connections. SSL is the standard protocol for securing web browsing. As such, it is widely deployed on the Internet and is distinguished by considerable operational experience. SSL also enjoys near universal support from firewalls and proxy servers. TLS is an updated version of SSL currently being developed within the Internet Engineering Task Force (IETF). TLS is heavily based on SSL and, although it is not strictly backwards compatible with SSL, systems supporting both TLS and SSL can automatically recognize either protocol and adapt as required to ensure interoperability.

NOTE: As other industry standard mechanisms for IP-based security (for example, IPSEC) reach maturity, later revisions to this specification may incorporate support for those mechanisms in addition to SSL/TLS. Such revisions to the security mechanisms may also permit the use of an unreliable transport such as UDP.

4.1.2 Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is the standard protocol for web-based communications. HTTP has been adopted for a wide variety of purposes including proxy services, bi-directional content delivery, database access, network management, and metering information. HTTP is by far the most widely used application protocol on the Internet, and is supported by all significant firewalls and proxy servers.

4.2 Message Format

To illustrate the overall format of this standard’s messages, Figure 2 shows an example message. As the figure indicates, the content within the HTTP message is formatted according to the standard for Multipurpose Internet Mail Extensions (MIME). The individual components of the message are a document conforming to the Extensible Markup Language

(XML) specification and a Secure MIME (S/MIME) digital signature. Note that the digital signature is optional and, if omitted, the message content consists solely of a XML document.

HTTP Header

```
POST scripts/settlements HTTP/1.0
content-type: multipart/signed;
    protocol="application/pkcs7-signature";
    micalg=sha1;
    boundary=bar
content-length: 844
```

Message Content

```
--bar
Content-Type: text/plain
Content-Length: 524

<?xml version=1.0?>
<Message messageId="123454321" random="12345678">
    <AuthorizationRequest componentId="9876567890">
        <Timestamp>
            1998-04-24T17:03:00Z
        </Timestamp>
        <CallId>
            1234432198766789
        </CallId>
        <SourceInfo type="e164">
            81458811202
        </SourceInfo>
        <DestinationInfo type="e164">
            4766841360
        </DestinationInfo>
        <Service/>
        <MaximumDestinations>
            5
        </MaximumDestinations>
    </AuthorizationRequest>
</Message>
```

Digital Signature

```
--bar
Content-Type: application/pkcs7-signature
Content-Length: 191

GhyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIG
fHfYT64VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756t
bB9HGTrfvbnjn8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfH
fYT6ghyHhHUujpfyF47GhIGfHfYT64VQbnj756

--bar--
```

Figure 2: Example Message Showing Overall Format

4.2.1 Multipurpose Internet Mail Extensions

All messages exchanged as part of this standard conform to the Multipurpose Internet Mail Extensions (MIME) specification. The MIME specification defines mechanisms to combine individual components of arbitrary format (e.g. text, graphics, audio information, binary data, etc.) into a single message. Originally designed for electronic mail, the MIME specification has been adapted for a variety of communication applications, including web browsing. MIME format is widely supported by existing firewalls and proxy servers.

4.2.2 Extensible Markup Language

The first part of each MIME message is a document conforming to the Extensible Markup Language (XML) standard. As an extension of the widely deployed Hypertext Markup Language (HTML), XML can be readily parsed by firewalls and proxy servers. Unlike HTML, though, XML is readily extensible and can easily support rich, structured data such as pricing and usage information.

4.2.3 Secure MIME

The second part of each MIME message, if present, is a digital signature conforming to the Secure Multipurpose Internet Mail Extensions (S/MIME). S/MIME format includes support for multiple digest and signing algorithms and for variable cryptographic strength (e.g. key lengths). S/MIME format is also self-identifying with respect to these parameters, so that a recipient can derive the necessary information for verifying the signature from the signature data.

NOTE: This does not imply that the recipient is guaranteed to be able to verify the signature, only that the recipient can tell what it needs to perform the verification. (So that, for example, the recipient may identify a signing algorithm that it does not support.)

5 Protocol Profiles

This clause specifies the profiles for the protocols required by this standard. It identifies the normative references to those protocols, as well as the specific versions, options, and extensions that this standard requires. The specific protocols described in this clause are the Secure Sockets Layer (SSL) and Transport Layer (TLS) protocols and the Hypertext Transfer Protocol (HTTP). The clause concludes by specifying the overall format of the messages conveyed through these protocols. The following clauses describe the message content in detail.

5.1 Secure Sockets Layer / Transport Layer Security

If secure authentication of the server is desired, or if confidentiality of the information exchanged between client and server is desired, the communication between the devices shall be secured using the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) as described in this clause.

5.1.1 Protocol Version

Conforming systems shall use version 3.0 of the Secure Sockets Layer protocol [6] to secure their communications. Systems may optionally use version 1.0 of the Transport Layer Security protocol [24] or later versions, provided those versions are capable of automatic recognition and fallback to SSL version 3.0, and provided the system supports that operation.

5.1.2 Client/Server Roles

When initiating a communication as part of this standard, the initiating system shall act as an SSL/TLS client while the responding system shall act as an SSL/TLS server.

5.1.3 Client Authentication

As SSL/TLS client authentication is ineffective in a network environment in which the communicating parties are separated by proxy servers, and since the S/MIME digital signature can provide client authentication, implementations conforming to this standard should not use SSL/TLS client authentication.

5.1.4 CipherSuites

Annex B documents the cryptographic algorithms required and recommended by this specification, including SSL/TLS ciphersuites.

5.2 Hypertext Transfer Protocol

5.2.1 Protocol Version

Conforming systems shall use version 1.0 of the Hypertext Transfer Protocol [2] as the base transfer protocol for their messages. Systems may optionally use HTTP version 1.1 [4], but shall be capable of automatic fallback to version 1.0 operation should their peer indicate support for version 1.0 only.

5.2.2 Client/Server Roles

When initiating a communication as part of this standard, the initiating system shall act as an HTTP client, while the responding system shall act as an HTTP server.

5.2.3 TCP Port

In the absence of a prior agreement between communicating parties, clients should send their requests to TCP port 443 if SSL or TLS is being used, and to TCP port 80 otherwise. Communicating parties may agree to communicate via other TCP ports.

5.2.4 HTTP Methods

Requests from clients to a server shall be in the form of HTTP request messages using the POST method. Responses from a server shall consist of valid HTTP response messages.

5.2.5 Uniform Resource Identifier

The uniform resource identifier included in the POST request is not specified in this standard, but rather is subject to prior agreement between the communicating parties.

5.2.6 HTTP Headers

The HTTP header of the POST method shall minimally consist of the request-line. All request-header and general-header fields are optional. If present, they shall conform to the HTTP standard [2]. The status-line for the HTTP responses shall be present in those responses, and it shall conform to the HTTP standard, including status-code and reason-phrase values. All response-header and general-header fields are optional. If present, they shall conform to the HTTP standard.

5.2.7 HTTP Entity Body

Each message (i.e. HTTP entity body) conveyed as part of this specification shall conform to the Multipurpose Internet Mail Extensions standard [5], and shall consist of exactly two parts, an Extensible Markup Language document and a Secure Multipurpose Internet Mail Extensions digital signature, as specified in the following two clauses. The highest level structure for each message shall conform to the multipart/signed syntax defined in S/MIME [18]. The message’s media type shall be "multipart/signed" with appropriate parameters (e.g. protocol of "application/pkcs7-signature" and micalg of "sha1.") The entity shall indicate the correct content-length value, as defined in the HTTP standard [2].

6 XML Content

This clause specifies the actual message format used to exchange pricing, authentication and authorization, and usage information. It outlines the overall XML document structure, lists the individual XML elements, and describes how those elements are combined into exchanges.

6.1 Document Structure

6.1.1 Multipurpose Internet Mail Extensions Conformance

As the first part of a Multipurpose Internet Mail Extensions (MIME) message, each message content shall conform to the MIME standard [5] as indicated below.

6.1.1.1 Content-Type

The message’s content-type shall be designated text/plain.

NOTE: It is anticipated that the Internet Engineering Task Force (IETF) will eventually define a MIME content-type for XML documents (e.g. text/xml). When such a definition is available, subsequent revisions of this standard may specify the use of that content-type instead of text/plain.

6.1.1.2 Content-Length

All messages shall indicate the correct content-length value as defined in the MIME standard.

6.1.1.3 Transfer Encoding

As XML documents can be carried within the HTTP protocol in their native format, no transfer encoding (e.g. quoted-printable or base-64) shall be used.

NOTE: Since XML content is carried in its native encoding, that encoding will not conform to the recommendations for the text/plain content-type. In particular, the XML encoding specifies the use of a single line-feed character (#xA) to indicate line ending rather than the return/line-feed pair. Although not the default encoding for text/plain, such an encoding does not violate the MIME standard.

6.1.2 XML Conformance

The actual message content itself shall conform to the XML standard [3]. As part of that conformance, systems shall follow the well-formedness and character encoding requirements as follows.

6.1.2.1 XML Version

Message content shall conform to version 1.0 of the XML standard, and shall indicate that version with the required XML prologue of `<?xml version="1.0">`.

6.1.2.2 Well-Formed Constraint

All messages shall be well-formed XML documents, as defined in the [3] standard. Messages may be valid XML documents as well, by referencing the appropriate XML document type definitions (DTDs). Strict validity (as defined by [3]) is not required, however.

NOTE: The terms "well-formed" and "valid" have specific meanings with the XML standard, and are used to indicate specific degrees of conformance to the standard.

6.1.2.3 Character Encoding

Messages may use any character set permitted by the XML standard. As specified in that standard, however, all implementations shall be capable of generating and interpreting UTF-8 and UTF-16 encodings. In the absence of explicit knowledge that the receiving system can support other character encodings, sends shall use UTF-8 or UTF-16 encoding [23].

6.1.3 XML Framework

All messages shall conform to the overall framework illustrated in Figure 3. As the figure shows, messages consist of a single root entity, which contains one or more components, each of which consists in turn of one or more elements. These elements may include XML attributes.

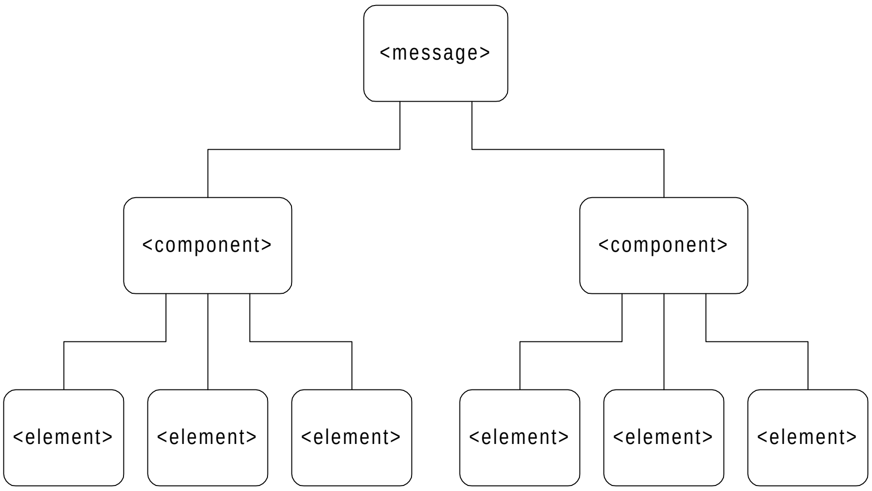

Figure 3: Overall XML Framework

6.1.3.1 Root Entity

```
<!DOCTYPE Message [
<!ELEMENT Message ( ( PricingIndication | PricingConfirmation | AuthorizationRequest |
    AuthorizationResponse | AuthorizationIndication | AuthorizationConfirmation | UsageIndication |
    UsageConfirmation | ReauthorizationRequest | ReauthorizationResponse )+ ) >
<!ATTLIST Message messageId ID    #REQUIRED
                  random    CDATA #REQUIRED >
...
]>
```

The root entity for each message shall be a `Message` element. It shall contain the components documented above, as well as a unique identifier attribute and a random attribute.

6.1.3.2 Random Attribute

Each `Message` element shall include a random attribute. This attribute's value shall be a random number, encoded as a decimal character string. The attribute ensures that each message contains some random content, and it therefore increases the security of the S/MIME digital signature.

NOTE: Systems should use a cryptographically strong random number generator as the source for this attribute's value. Standard library random number functions (such as from the ANSI C standard) are generally ***not*** cryptographically strong, and should not be used.

6.1.3.3 Identifier Attribute

Each message and each component within a message shall include a unique identifier for that element. The system that initiates communication (through either a request or an indication) shall ensure that the identifier is unique. The system that replies (through either a response or a confirmation) shall use the identifier value to associate messages and components in its response or confirmation with the corresponding elements in the request or indication. The identifier attribute consists of an arbitrary character string, and is indicated by the attribute names `messageId` or `componentId`, as appropriate.

6.1.3.4 Critical Attribute

All XML elements shall include a special attribute with the name `critical` that takes values of `"True"` or `"False"`. This attribute indicates whether or not processing of the message can safely proceed if the particular element is not supported by the receiver.

If a system receives a message containing a critical element that it does not support, that system shall not accept or process the message.

If the critical attribute is omitted from an element (and its parents), that element shall be treated as if the critical attribute was present with the value `"True"`.

The value of the critical attribute (whether explicit or implied) for an element shall apply to all subelements of the element unless a subelement explicitly indicates otherwise.

6.1.3.5 Extensions

Organizations may define additional elements components beyond those documented in this standard. To ensure that no two organizations use the same named element, all private extensions shall have, as a prefix to their name, an officially registered Internet domain name belonging to the defining party. That domain name may be separated from the element name by a colon. If, for example, the organization that owns the Internet domain name acme.com wishes to add an element named `PrivateOption`, the tags delineating that element will be `<acme.com:PrivateOption>` and `</acme.com:PrivateOption>`. As noted above, the `critical` attribute may be used to indicate to a recipient whether or not it can safely ignore a private extension that it does not understand or support.

6.2 Components

Components are the main elements within the each message. The `<Message>` element shall contain at least one and may contain more than one component. The components defined in this revision of the standard include pairs to effect pricing exchange (`PricingIndication` and `PricingConfirmation`), obtain authorization (`AuthorizationRequest` and `AuthorizationResponse`), verify authorization (`AuthorizationIndication` and `AuthorizationConfirmation`), refresh authorization (`ReauthorizationRequest` and `ReauthorizationResponse`) and report usage (`UsageIndication` and `UsageConfirmation`).

6.2.1 PricingIndication

```
<!ELEMENT PricingIndication ( Timestamp, SourceInfo, DestinationInfo, Currency, Amount, Increment,
    Unit, Service, ValidAfter, ValidUntil ) >
<!ATTLIST PricingIndication componentId ID #REQUIRED >
```

A `PricingIndication` component identifies the price for a particular service. It is composed of several elements, as indicated above.

For example, the following XML content states that the cost of basic telephone calls from the United States (country code 1) to France (country code 33) is US$ 0,50 per minute, effective immediately and indefinitely.

```
<PricingIndication componentId="1234567890">
    <Timestamp>
        1997-05-02T19:03:00Z
    </Timestamp>
    <SourceInfo type="e164prefix">
        1
    </SourceInfo>
    <DestinationInfo type="e164prefix">
        33
    </DestinationInfo>
    <Currency>
        USD
    </Currency>
    <Amount>
        0.5
    </Amount>
    <Increment>
        60
    </Increment>
    <Unit>
        s
    </Unit>
    <Service/>
    <ValidAfter/>
    <ValidUntil/>
</PricingIndication>
```

6.2.2 PricingConfirmation

```
<!ELEMENT PricingConfirmation ( Timestamp, Status ) >
<!ATTLIST PricingConfirmation componentId ID #REQUIRED >
```

A PricingConfirmation component indicates acceptance or rejection of the corresponding PricingIndication. The componentId attribute associates the confirmation with the indication. The only elements within the PricingConfirmation are the Timestamp and Status elements.

6.2.3 AuthorizationRequest

```
<!ELEMENT AuthorizationRequest ( Timestamp, CallId*, SourceInfo, SourceAlternate*,
    DestinationInfo, DestinationAlternate*, Service, MaximumDestinations ) >
<!ATTLIST AuthorizationRequest componentId ID #REQUIRED >
```

An AuthorizationRequest asks for authorization to use resources. In the context of basic Internet telephony service, it asks for authorization to complete a phone call. The call, for example, may be identified by E.164 numbers [12] in the SourceInfo and DestinationInfo elements. The requesting system may leave the choice of peer endpoints up to the authorizing server, or it may specify the peer endpoint itself in a DestinationAlternate element.

The client may provide one or more CallId elements in the request. A single CallId element implies that the client wishes to use that call identifier for all possible destinations returned in the AuthorizationResponse. Multiple CallId elements imply that the client wishes each potential destination to have its own call identifier. If the client wishes to specify multiple call identifiers, the number of CallId elements shall be equal to the value of the MaximumDestinations element.

NOTE 1: The use of the AuthorizationRequest message is not intended to be a replacement for, or supersede any H.323 [13] Registration Admission and Status (RAS) messages. Although the elements of the AuthorizationRequest are similar to information elements in RAS messages, AuthorizationRequest is intended for use in the bulk transfer of authorization tokens or other scenarios in which RAS signalling would not be appropriate.

NOTE 2: This standard permits either party in an authorization exchange to determine the call routing information (e.g. identifying the peer endpoint for the call). If the client wishes to explicitly specify call routing, it does so by including one or more DestinationAlternate elements (e.g. of type "transport" containing the IP address and port number of the peer endpoint) in the AuthorizationRequest. If the server is performing call routing, it returns the information in the AuthorizationResponse. The DestinationAlternate elements are advisory during the request. The server should attempt to use one of the endpoints specified, but it is free to substitute a different set of endpoints if it is unable to settle the call using those specified.

NOTE 3: There are actually three separate authentication/authorization processes that are involved in the authorization exchange. As a point of clarification, the three processes rely on the following mechanisms:

a) The client authenticates the authorization server as part of the SSL/TLS handshake.

b) The authorization server authenticates the client by validating the digital signature of the request.

c) The authorization server may provide, as part of the AuthorizationResponse, a token which will authorize the client to a peer endpoint or gatekeeper.

6.2.4 AuthorizationResponse

```
<!ELEMENT AuthorizationResponse ( Timestamp, Status, TransactionId, Destination* ) >
<!ATTLIST AuthorizationResponse componentId ID #REQUIRED >
```

An AuthorizationResponse component returns authorization information corresponding to an AuthorizationRequest. It includes a Timestamp, a Status element, a TransactionId, and zero or more

Destination elements. The response includes a componentId attribute to associate it with the appropriate AuthorizationRequest.

NOTE: If a client has expressed its own call routing preferences in the AuthorizationResponse (e.g. with DestinationAlternate elements of type transport), then the server should make every attempt to honour those preferences by returning appropriate Destination elements. The server may also include alternative destinations in its response.

6.2.5 AuthorizationIndication

```
<!ELEMENT AuthorizationIndication ( Timestamp, Role, CallId, SourceInfo, SourceAlternate*,
    DestinationInfo, DestinationAlternate*, Service, Token* ) >
<!ATTLIST AuthorizationIndication componentId ID #REQUIRED >
```

An AuthorizationIndication asks for verification of previously issued authorization, typically by asking for verification of an authorization token. Because tokens may be opaque to the terminating endpoint, that endpoint may not be able to determine the originator of a particular token. It is therefore acceptable to pass an entire sequence of tokens from a setup message in this component, and it is acceptable to send simultaneous AuthorizationIndication messages to multiple servers. The server shall be capable of recognizing authorization tokens in addition to validating them.

NOTE: Even though this specification defines messages for validating authorization tokens, it does not require their use. In particular, some tokens may be constructed so that they can be completely verified in the peer system (e.g. through the use of digital signatures).

6.2.6 AuthorizationConfirmation

```
<!ELEMENT AuthorizationConfirmation ( Timestamp, Status, ValidAfter, ValidUntil ) >
<!ATTLIST AuthorizationConfirmation componentId ID #REQUIRED >
```

An AuthorizationConfirmation component indicates whether or not an authorization is valid. It includes a Timestamp, a Status element and validation time limits. The confirmation also includes a componentId attribute to associate it with the appropriate AuthorizationIndication.

6.2.7 UsageIndication

```
<!ELEMENT UsageIndication ( Timestamp, Role, TransactionId, CallId, SourceInfo, SourceAlternate*,
    DestinationInfo, DestinationAlternate*, UsageDetail* ) >
<!ATTLIST UsageIndication componentId ID #REQUIRED >
```

The UsageIndication component reports resource usage. In the context of basic Internet telephony service, this is typically call duration.

6.2.8 UsageConfirmation

```
<!ELEMENT UsageConfirmation ( Timestamp, Status ) >
<!ATTLIST UsageConfirmation componentId ID #REQUIRED >
```

A UsageConfirmation component indicates acceptance or rejection of the corresponding UsageIndication. The componentId attribute associates the confirmation with the indication. The only elements within the UsageConfirmation are the Timestamp and Status elements.

6.2.9 ReauthorizationRequest

```
<!ELEMENT ReauthorizationRequest ( Timestamp, Role, CallId, SourceInfo?, SourceAlternate*,
    DestinationInfo?, DestinationAlternate*, TransactionId, UsageDetail*, Token* ) >
<!ATTLIST ReauthorizationRequest componentId ID #REQUIRED >
```

A ReauthorizationRequest component requests a reauthorization of a previously authorized service. A client may use this message, for example, if a previous authorization has expired.

6.2.10 ReauthorizationResponse

```
<!ELEMENT ReauthorizationResponse ( Timestamp, Status, TransactionId, Destination* ) >
<!ATTLIST ReauthorizationResponse componentId ID #REQUIRED >
```

A ReauthorizationResponse component indicates acceptance or rejection of the corresponding ReauthorizationRequest. The componentId attribute associates the response with the request.

6.3 Elements

This subclause defines the individual elements that make up each message component.

6.3.1 Amount

```
<!ELEMENT Amount (#PCDATA)>
<!ATTLIST Amount critical (True | False) "True">
```

The Amount element identifies a numeric value and is often associated with the Increment and Unit elements, as well as the Currency element. Amounts are expressed using the period (.) as a decimal separator and with no punctuation as the thousands separator. The following excerpt, for example, expresses a rate of 50 cents (U.S.) per minute.

```
<Currency>
    USD
</Currency>
<Amount>
    0.5
</Amount>
<Increment>
    60
</Increment>
<Unit>
    s
</Unit>
```

6.3.2 AuthorityURL

```
<!ELEMENT AuthorityURL (#PCDATA)>
<!ATTLIST AuthorityURL critical (True | False) "True">
```

The AuthorityURL element identifies a uniform resource locator (URL) by which authorization may be verified or refreshed.

6.3.3 CallId

```
<!ELEMENT CallId (#PCDATA)>
<!ATTLIST CallId encoding (cdata | base64) "cdata"
                 critical (True | False)   "True">
```

The CallId element contains a call's H.323 CallId value, and is thus used to uniquely identify individual calls. To convey its binary value, a call identifier may either be encoded using XML CDATA format and appropriate escape sequences, or it may be encoded with base64 encoding as per the [5] standard. An encoding attribute indicates the method selected, and the default value for that attribute is XML CDATA.

6.3.4 Code

```
<!ELEMENT Code (#PCDATA)>
<!ATTLIST Code critical (True | False) "True">
```

The `Code` element contains the numeric value that uniquely and unambiguously indicates the sender's response to a request or indication. It is usually paired with a `Description` element within a `Status` element. The `Code` content consists of three numeric digits in the form `NNN`. The first (most significant) digit indicates the success or failure of the operation, subsequent digits provide greater detail. Values defined by this standard include the following.

NOTE: Subsequent revisions to this standard may add other code values, but will always preserve the meaning of a most significant 2 as success, and any other most significant digit as failure.

`2`*`xx`* = operation successful
`200` = success (no other information)
`201` = information created (no previous values)
`210` = updated information accepted (previous values replaced)
`4xx` = client error
`400` = bad request (generic problem interpreting message)
`401` = unauthorized
`410` = character encoding not supported
`411` = parsing unsuccessful
`412` = critical element not supported
`420` = generic security problem (no other information available)
`421` = signature invalid
`422` = cryptographic algorithm not supported
`423` = certificate invalid
`424` = certificate revoked
`425` = encryption required
`5`*`xx`* = server error
`500` = internal server error
`501` = not implemented
`503` = service not available
`510` = transient problem in server
`520` = long term problem in server
`530` = time problem
`531` = valid time too soon
`532` = time interval too small
`999` = generic failure (no other information available)

6.3.5 Currency

```
<!ELEMENT Currency (#PCDATA)>
<!ATTLIST Currency critical (True | False) "True">
```

The `Currency` element defines the financial currency in use for the parent element. It is represented according to the notation of [7]. In addition, the following two definitions not included in [7] may be used:

ECU	European Currency Unit
SDR	Special Drawing Rights

6.3.6 Description

```
<!ELEMENT Description (#PCDATA)>
<!ATTLIST Description critical (True | False) "False">
```

The `Description` element provides a textual description for a response, and is typically paired with a `Code` element as part of a `Status` parent element. The `Description` element is for informational purposes only, as the `Code` element defines the behaviour. Suggested values for the Description element are the descriptions given above for each Code value.

6.3.7 Destination

```
<!ELEMENT Destination ( DestinationInfo?, DestinationAlternate*, DestinationSignalAddress, Token*,
    ValidAfter?, ValidUntil?, UsageDetail*, AuthorityURL*, CallId )>
<!ATTLIST Destination critical (True | False) "True">
```

The Destination element is the parent element for call routing information, and it is returned by servers in an AuthorizationResponse messages. As the above definition shows, as many as nine different types of subelements may comprise a Destination element.

6.3.8 DestinationAlternate

```
<!ELEMENT DestinationAlternate (#PCDATA)>
<!ATTLIST DestinationAlternate type     ( e164 | h323 | url | email | transport |
                                          international | national | network |
                                          subscriber | abbreviated | e164prefix )    #REQUIRED
                               critical ( True | False )                             "True"       >
```

The DestinationAlternate element contains secondary identification of the destination. This information provides an alternative to the DestinationInfo element. DestinationAlternate uses the same notation as DestinationInfo.

6.3.9 DestinationInfo

```
<!ELEMENT DestinationInfo (#PCDATA)>
<!ATTLIST DestinationInfo type     ( e164 | h323 | url | email | transport |
                                     international | national | network |
                                     subscriber | abbreviated | e164prefix )         #REQUIRED
                          critical ( True | False )                                  "True"       >
```

The DestinationInfo element gives the primary identification of the destination, or called party, for a call. The element includes a type attribute, and can take one of several forms depending on the value of that attribute.

The following list indicates the contents of the element, given each possible attribute type.

e164	full E.164 telephone number [12] containing numeric digits only (i.e. no punctuation)
h323	H.323 identifier
url	Uniform Resource Locator [13]
email	electronic mail address [13]
transport	transport address is the form of name:nn where name is the domain name (or IP address enclosed in square brackets) and :nn is an (optional) TCP or UDP port number, (e.g. [172.16.1.1]:112)
international	international party number [13]
national	national party number [13]
network	network specific party number [13]
subscriber	subscriber party number [13]
abbreviated	abbreviated party number [13]
e164prefix	initial (most significant) digits of an E.164 number [12] with no punctuation

6.3.10 DestinationSignalAddress

```
<!ELEMENT DestinationSignalAddress (#PCDATA)>
<!ATTLIST DestinationSignalAddress critical (True | False) "True">
```

The DestinationSignalAddress element identifies the call signalling address for the destination. It is represented as name:nn, where name is a domain name or an IP address enclosed in square brackets. The :nn is optional and indicates a

TCP port number. For example, call signalling to device gateway.operator.com at TCP port number 112 is represented as follows.

```
<DestinationSignalAddress>
    gateway.operator.com:112
</DestinationSignalAddress>
```

6.3.11 Increment

```
<!ELEMENT Increment (#PCDATA)>
<!ATTLIST Increment critical (True | False) "True">
```

The `Increment` element indicates the number of units being accounted. It is typically used in combination with the `Amount` and `Unit` elements. The following excerpt, for example, expresses a duration of 5½ minutes.

```
<Amount>
    5.5
</Amount>
<Increment>
    60
</Increment>
<Unit>
    s
</Unit>
```

6.3.12 MaximumDestinations

```
<!ELEMENT MaximumDestinations (#PCDATA)>
<!ATTLIST MaximumDestinations critical (True | False) "True">
```

The `MaximumDestinations` element appears in the `AuthorizationRequest` component to indicate the maximum number of potential destinations the client wishes to receive in the response.

6.3.13 Role

```
<!ELEMENT Role (#PCDATA)>
<!ATTLIST Role critical (True | False) "True">
```

The `Role` element indicates the role of the system generating a message. It shall contain one of the following values.

`source`	message generated by source
`destination`	message generated by destination
`other`	message generated by systems other than source or destination

6.3.14 Service

```
<!ELEMENT Service EMPTY>
<!ATTLIST Service critical (True | False) "True">
```

The `Service` element indicates a type of service being priced, authorized, or reported. An empty `Service` element indicates basic Internet telephony service, which is the only service type defined by this version of this specification.

NOTE: Later revisions of this standard are expected to specify more enhanced service definitions to represent quality of service, availability, payment methods, etc.

6.3.15 SourceAlternate

```
<!ELEMENT SourceAlternate (#PCDATA)>
<!ATTLIST SourceAlternate type     ( e164 | h323 | url | email | transport |
                                     international | national | network |
                                     subscriber | abbreviated | e164prefix )        #REQUIRED
                          critical ( True | False )                                 "True"       >
```

The `SourceAlternate` element contains secondary identification of the source of a call. It conforms to the same notation as the `DestinationInfo` element.

6.3.16 SourceInfo

```
<!ELEMENT SourceInfo (#PCDATA)>
<!ATTLIST SourceInfo type     ( e164 | h323 | url | email | transport |
                                international | national | network | subscriber |
                                abbreviated | e164prefix )                           #REQUIRED
                     critical ( True | False )                                       "True"       >
```

The `SourceInfo` element contains the primary identification of the source of a call. It uses the same notation as the `DestinationInfo` element.

6.3.17 SourceSignalAddress

```
<!ELEMENT SourceSignalAddress (#PCDATA)>
<!ATTLIST SourceSignalAddress critical (True | False) "True">
```

The `SourceSignalAddress` element identifies the call signalling address of the source of a call. It uses the same notation as the `DestinationSignalAddress` element.

6.3.18 Status

```
<!ELEMENT Status ( Code, Description? ) >
<!ATTLIST Status critical (True | False) "True">
```

The `Status` element reports the results of a response or confirmation. It is composed of a `Code` element and an optional `Description` element.

For example, the following excerpt indicates a successful response or confirmation.

```
<Status>
    <Code>
        000
    </Code>
    <Description>
        success (no other information)
    </Description>
</Status>
```

6.3.19 Timestamp

```
<!ELEMENT Timestamp (#PCDATA)>
<!ATTLIST Timestamp critical (True | False) "True">
```

The Timestamp element indicates the time at which the component was generated. It is represented by a restricted form of [8] format. In particular, time is always represented in co-ordinated universal time (UTC) using the notation `YYYY-MM-DDThh:mm:ssZ` where

`YYYY` = four-digit year (for example, 1998)
`MM` = two-digit month (01=January, etc.)
`DD` = two-digit day of month (01 through 31)
`T` = indicates division between date and time
`hh` = two-digit hour (00 through 23)
`mm` = two-digit minute (00 through 59)
`ss` = two-digit second (00 through 59)
`Z` = indicates co-ordinated universal time

For example, exactly 3:03 P.M. on May 2, 1997, Eastern Daylight Time in the United States, is represented as

```
<Timestamp>
    1997-05-02T19:03:00Z
</Timestamp>
```

6.3.20 Token

```
<!ELEMENT Token (#PCDATA)>
<!ATTLIST Token encoding (cdata | base64) "cdata"
                critical (True | False)   "True"  >
```

The `Token` element conveys an H.235 security token. To convey its binary value, a token may either be encoded using XML CDATA format and appropriate escape sequences, or it may be encoded with base64 encoding as per the [5] standard. An encoding attribute indicates the method selected, and the default value for that attribute is XML CDATA.

6.3.21 TransactionId

```
<!ELEMENT TransactionId (#PCDATA)>
<!ATTLIST TransactionId critical (True | False) "True">
```

The `TransactionId` element contains an integer, decimal valued identifier assigned to a specific authorized transaction. It is represented without any punctuation (e.g. no thousands separator).

6.3.22 Unit

```
<!ELEMENT Unit (#PCDATA)>
<!ATTLIST Unit critical (True | False) "True">
```

The `Unit` element indicates the units by which pricing is measured or usage recorded. It shall contain one of the following values.

`s`	seconds
`pkt`	packets (datagrams)
`byte`	bytes

6.3.23 UsageDetail

```
<!ELEMENT UsageDetail ( Service, Amount, Increment, Unit ) >
<!ATTLIST UsageDetail critical (True | False) "True">
```

The `UsageDetail` element collects information describing the usage of a service. Individual transactions may combine multiple `UsageDetail` elements as part of their usage report. This capability supports both parallel services (e.g. audio and video streams during a video conference) and serial services (e.g. low bit-rate codec switching to a higher quality codec as network conditions deteriorate).

The `UsageDetail` element may also be present in either an `AuthorizationResponse` or a `ReauthorizationResponse`. In that case, it indicates a limit to the authorization. For example, an `AuthorizationResponse` that includes a `UsageDetail` of (amount=3, increment=60, unit=s) indicates that the authorization is valid for no more than 3 minutes of service.

6.3.24 ValidAfter

```
<!ELEMENT ValidAfter (#PCDATA)>
<!ATTLIST ValidAfter critical (True | False) "True">
```

The `ValidAfter` element identifies the time and date after which the component's information shall be effective or valid. It is encoded using the same notation as the `Timestamp` element. If this element is empty, component information is assumed to be valid as soon as it is received.

6.3.25 ValidUntil

```
<!ELEMENT ValidUntil (#PCDATA)>
<!ATTLIST ValidUntil critical (True | False) "True">
```

The `ValidUntil` element identifies the time and date after which the component's information is no longer effective or valid. It is encoded using the same notation as the `Timestamp` element. If this element is empty, component information is assumed to be effective indefinitely, or until it is explicitly modified with new information.

7 Signature Format

If present, the digital signature shall conform to the application/pkcs7-signature format specified in the Secure Multipurpose Internet Mail Extensions (S/MIME) standard [18]. This clause specifies how that signature is created, including the canonicalization procedure, signature algorithm, and transfer encoding.

7.1 Canonical Form

Digital signatures described in this specification are signatures of the entire, transmitted XML document, beginning with (and including) the leftmost "<" of the XML declaration and ending with (and including) the rightmost ">" of the end tag of the root XML entity. Furthermore, conforming implementations should construct their XML documents using the following procedures to arrive at a canonical form. Such a form will enable the reconstruction of an XML document (and the verification of a digital signature) from the abstract information of the document.

NOTE 1: The following procedure borrows heavily from the Internet Open Trading Protocol (IOTP) specification [25].

1) Isolate the element to be signed. For this standard, that entity consists of the entire XML document, beginning with the leftmost "<" of the XML declaration and ending with (and including) the rightmost ">" of the end tag of the root XML entity.

2) Convert all characters in the element to canonical form for the character encoding.

3) Apply all external XML entities and all character and entity references in the element so that they are completely resolved.

4) Exclude comments and processing instructions.

5) Reduce all attributes to their canonical form using the attribute type in the Document Type Definition (DTD). Replace all single and double quotes present in attributes with `&#39;` and `&#34;` respectively so that attributes can be enclosed in double quotes.

6) Create attributes, using their default value, which are not present in the original but have default values in the DTD.

7) Sort the original and generated attributes in ascending attribute name order according to character encoding of the attribute name.

8) For whitespace inside markup but not inside attribute values, generate it as minimally as possible. Specifically, (1) remove non-essential whitespace, and (2) represent required whitespace by a single space character.

9) Generate the content of all start tags using only the element name and the attributes as described above. If the element is an empty element, then generate it using the single empty tag format (i.e. a trailing slash). Generate end tags using only element name with no added whitespace.

10) Remove all whitespace in the element content.

11) Assemble start tags, end tags, empty tags, CDATA sections, and text sections in the same order as the original document.

NOTE 2: The above procedure results in a canonicalized XML document rather than a canonicalized MIME text part. In particular, the line ending is encoded as a single line-feed character (#xA) rather than the S/MIME convention of a return/line-feed pair.

7.2 Signature Algorithms

Annex B provides a list of cryptographic algorithms required and recommended by this specification, including S/MIME signature algorithms.

7.3 Transfer Encoding

Unlike some electronic mail protocols, HTTP is capable of transferring raw binary data. Consequently, no transfer encoding (e.g. quoted-printable or base-64) shall be used for the signature.

8 Protocol Behaviour

This clause specifies the relationship between the message components. It defines message sequencing, interdependence of messages, and exception handling.

8.1 Message Sequencing

This standard specifies a simple client/server protocol. All protocol exchanges shall be initiated by clients, who send one or more of the five client components in a single message. The server shall reply with its own single message, which contains one server component for each client component. Figure 4 shows the five different component pairs.

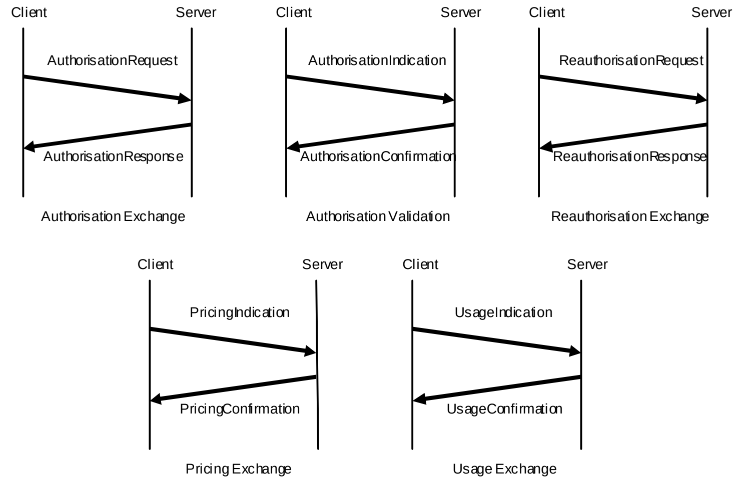

Figure 4: Component Pairs

When multiple components are combined in a single message, each component shall be treated independently of all others in the message. Logically, this treatment shall have the same effect as if each component is carried in its own message.

In addition, each component exchange shall be considered independent of all others; there shall be no dependence between message exchanges. This is true even of Authorization exchanges. Specifically, both AuthorizationIndication / Confirmation and ReauthorizationRequest / Response exchanges may refer to authorization previously obtained through means other than an AuthorizationRequest and AuthorizationResponse.

8.2 Exception Handling

Systems conforming to this standard should use all levels of error handling available in this specification.

8.2.1 Transmission Control Protocol

If TCP indicates that communication cannot be established, or that communication has failed during a transmission, the communicating parties should not assume the delivery of any partial information.

8.2.2 Secure Socket Layer / Transport Layer Security

If SSL or TLS indicates that compatible encryption parameters cannot be established, the communicating parties should not assume the delivery of any partial information. In addition, if SSL/TLS is unable to successfully authenticate the server, the client should not proceed with the transfer of any information.

8.2.3 Hypertext Transfer Protocol

Communicating parties should treat HTTP status codes as defined in [2]. In particular, unless the status code is in the range 200-299, the client should not assume delivery of any information.

8.2.4 Status Element

XML `Code` elements within responses and confirmations should be treated appropriately according to the definitions of subclause 6.3.3. The associated `Description` element should be used solely for informational purposes.

Annex A (normative): Document Type Definition

This Annex contains the complete XML document type definition for the messages described in this standard. The information is repeated from Clause 6, but is collected in one place for convenience of reference.

```
<!DOCTYPE Message [
<!ELEMENT Message ( ( PricingIndication | PricingConfirmation |
          AuthorizationRequest | AuthorizationResponse |
          AuthorizationIndication | AuthorizationConfirmation |
          UsageIndication | UsageConfirmation | ReauthorizationRequest |
          ReauthorizationResponse )+ ) >
<!ATTLIST Message messageId ID    #REQUIRED
                  random    CDATA #REQUIRED >
<!ELEMENT PricingIndication ( Timestamp, SourceInfo, DestinationInfo,
          Currency, Amount, Increment, Unit, Service, ValidAfter,
          ValidUntil ) >
<!ATTLIST PricingIndication componentId ID #REQUIRED >
<!ELEMENT PricingConfirmation ( Timestamp, Status ) >
<!ATTLIST PricingConfirmation componentId ID #REQUIRED >
<!ELEMENT AuthorizationRequest ( Timestamp, CallId*, SourceInfo,
          SourceAlternate*, DestinationInfo, DestinationAlternate*,
          Service, MaximumDestinations ) >
<!ATTLIST AuthorizationRequest componentId ID #REQUIRED >
<!ELEMENT AuthorizationResponse ( Timestamp, Status, TransactionId,
          Destination* ) >
<!ATTLIST AuthorizationResponse componentId ID #REQUIRED >
<!ELEMENT AuthorizationIndication ( Timestamp, Role, CallId, SourceInfo,
          SourceAlternate*, DestinationInfo, DestinationAlternate*, Service,
          Token* ) >
<!ATTLIST AuthorizationIndication componentId ID #REQUIRED >
<!ELEMENT AuthorizationConfirmation ( Timestamp, Status, ValidAfter,
          ValidUntil ) >
<!ATTLIST AuthorizationConfirmation componentId ID #REQUIRED >
<!ELEMENT UsageIndication ( Timestamp, Role, TransactionId, CallId,
          SourceInfo, SourceAlternate*, DestinationInfo,
          DestinationAlternate*, UsageDetail* ) >
<!ATTLIST UsageIndication componentId ID #REQUIRED >
<!ELEMENT UsageConfirmation ( Timestamp, Status ) >
<!ATTLIST UsageConfirmation componentId ID #REQUIRED >
<!ELEMENT ReauthorizationRequest ( Timestamp, Role, CallId, SourceInfo?,
          SourceAlternate*, DestinationInfo?, DestinationAlternate*,
          TransactionId, UsageDetail*, Token* ) >
<!ATTLIST ReauthorizationRequest componentId ID #REQUIRED >
<!ELEMENT ReauthorizationResponse ( Timestamp, Status, TransactionId, Destination* ) >
<!ATTLIST ReauthorizationResponse componentId ID #REQUIRED >
<!ELEMENT Amount (#PCDATA)>
<!ATTLIST Amount critical (True | False) "True">
<!ELEMENT AuthorityURL (#PCDATA)>
<!ATTLIST AuthorityURL (True | False) "True">
<!ELEMENT CallId (#PCDATA)>
<!ATTLIST CallId encoding (cdata | base64) "cdata"
                 critical (True | False)   "True">
<!ELEMENT Code (#PCDATA)>
<!ATTLIST Code critical (True | False) "True">
<!ELEMENT Currency (#PCDATA)>
<!ATTLIST Currency critical (True | False) "True">
<!ELEMENT Description (#PCDATA)>
<!ATTLIST Description critical (True | False) "False">
<!ELEMENT Destination ( DestinationInfo?, DestinationAlternate*,
          DestinationSignalAddress, Token*, ValidAfter?, ValidUntil?,
          UsageDetail*, AuthorityURL*, CallId ) >
<!ATTLIST Destination critical (True | False) "True">
<!ELEMENT DestinationAlternate (#PCDATA)>
<!ATTLIST DestinationAlternate type ( e164 | h323 | url | email | transport |
                                      international | national | network |
                                      subscriber | abbreviated | e164prefix
                                    ) #REQUIRED
                               critical ( True | False ) "True" >
<!ELEMENT DestinationInfo (#PCDATA)>
<!ATTLIST DestinationInfo type ( e164 | h323 | url | email | transport |
                                 international | national | network |
                                 subscriber | abbreviated | e164prefix
                               ) #REQUIRED
                          critical ( True | False ) "True" >
<!ELEMENT DestinationSignalAddress (#PCDATA)>
<!ATTLIST DestinationSignalAddress critical (True | False) "True">
<!ELEMENT Increment (#PCDATA)>
<!ATTLIST Increment critical (True | False) "True">
<!ELEMENT MaximumDestinations (#PCDATA)>
<!ATTLIST MaximumDestinations critical (True | False) "True">
<!ELEMENT Role (#PCDATA)>
<!ATTLIST Role critical (True | False) "True">
<!ELEMENT Service EMPTY>
<!ATTLIST Service critical (True | False) "True">
<!ELEMENT SourceAlternate >
<!ATTLIST SourceAlternate type ( e164 | h323 | url | email | transport |
                                 international | national | network |
                                 subscriber | abbreviated | e164prefix
                               ) #REQUIRED
                          critical ( True | False ) "True" >
<!ELEMENT SourceInfo >
<!ATTLIST SourceInfo type ( e164 | h323 | url | email | transport |
                            international | national | network | subscriber |
                            abbreviated | e164prefix
                          ) #REQUIRED
                     critical ( True | False ) "True" >
<!ELEMENT SourceSignalAddress (#PCDATA)>
<!ATTLIST SourceSignalAddress critical (True | False) "True">
<!ELEMENT Status ( Code, Description? ) >
<!ATTLIST Status critical (True | False) "True">
<!ELEMENT Timestamp (#PCDATA)>
<!ATTLIST Timestamp critical (True | False) "True">
<!ELEMENT Token (#PCDATA)>
<!ATTLIST Token encoding (cdata | base64) "cdata"
                critical (True | False)   "True"  >
<!ELEMENT TransactionId (#PCDATA)>
<!ATTLIST TransactionId critical (True | False) "True">
<!ELEMENT Unit (#PCDATA)>
<!ATTLIST Unit critical (True | False) "True">
<!ELEMENT UsageDetail ( Service, Amount, Increment, Unit ) >
<!ATTLIST UsageDetail critical (True | False) "True">
<!ELEMENT ValidAfter (#PCDATA)>
<!ATTLIST ValidAfter critical (True | False) "True">
<!ELEMENT ValidUntil (#PCDATA)>
<!ATTLIST ValidUntil critical (True | False) "True">
]>
```

Annex B (normative): Cryptographic Algorithms

For convenience and ease of reference, this annex provides the specification of cryptographic algorithms referenced in the standard. Cryptographic algorithms are essential to SSL/TLS communications, S/MIME signatures, and token formats.

B.1 SSL/TLS CipherSuites

Systems conforming to this standard may negotiate any mutually supported SSL/TLS CipherSuite using the standard SSL/TLS handshake mechanisms. To ensure maximum interoperability, all compliant systems should support the SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA CipherSuite. In environments where triple DES data encryption is not desired or is otherwise unavailable, systems should support the SSL_DHE_DSS_WITH_DES40_CBC_SHA CipherSuite. If no data encryption is desired or available, systems should use the SSL_RSA _WITH_NULL_SHA CipherSuite.

NOTE: To take maximum advantage of deployed cryptographic software and services, systems should also support the SSL_RSA_WITH_RC4_128_MD5 and SSL_RSA_EXPORT_WITH_RC4_40_MD5 CipherSuites.

B.2 S/MIME Signatures

Communicating systems may use any digest and signing algorithms defined by the S/MIME standard [18]. To ensure interoperability, all compliant systems shall support the Secure Hash Algorithm (SHA) [17] for message digests, and the Digital Signature Algorithm (DSA) [16] for signing.

NOTE: To take maximum advantage of deployed cryptographic software, systems should also support the Message Digest 5 (MD5) digest algorithm [20] and the Rivest Shamir Aldeman (RSA) signature algorithm [21].

B.3 Tokens

Tokens used for authorization, call progress, and call completion may be signed and encrypted. For message digest, signing, and encryption, implementations may use any algorithm defined by the Public Key Cryptography Standard #7 (PKCS7) [22]. To ensure interoperability, systems should support Secure Hash Algorithm [17], Diffie-Hellman key exchange [1], Digital Signature Algorithm [16], and the Data Encryption Standard [15].

NOTE: To take maximum advantage of deployed cryptographic software, systems should also support the Message Digest 5 [20], Rivest Shamir Aldeman [21], and Rivest Cipher 2 [19] algorithms.

Annex C (normative): Enhanced Usage Reports

For further study.

Annex D (informative): Token Formats

This annex presents suggested formats for authorization tokens passed as part of the H.323 exchange between endpoints and/or gatekeepers. These tokens may also be exchanged as part of the authorization information defined in this standard. The annex describes suggested cryptographic encodings as well as suggested contents for these authorization tokens, and it provides a method for carrying the tokens within H.323 messages.

D.1 Cryptographic Encoding

Depending on the business roles and network environment, tokens may be digitally signed and/or encrypted. If digitally signed, tokens should conform to the Public Key Cryptography Standard # 7 (PKCS7) specification [22] for signed-data content. If encrypted, tokens should conform to the PKCS7 specification for enveloped-data content. If tokens are both signed and encrypted, they should be constructed in two phases. First, the token should be digitally signed, resulting in PKCS7 signed-data content. The resulting signed content should then be encrypted, resulting in PKCS7 enveloped-data content.

NOTE: In order to provide confidentiality of the signature information, PKCS7 signed-and-enveloped content should not be used.

Annex B documents the cryptographic algorithms recommended by this specification, including digest, signing, symmetric encryption, and asymmetric encryption algorithms.

D.2 Token Content

This clauses suggests two different token formats. The ASN.1 format relies on information elements defined in the H.323 standards, and is appropriate for native H.323 systems. The XML format is consistent with the body of this specification, and may be used where ASN.1 encoding/decoding is not available or desired.

D.2.1 ASN.1 Format

The actual content of the token may conform to the following ASN.1 specifications, as augmented by ASN.1 constructs from the H.323 standards [13], [9], [14], and [10]. The tokens should be encoded using the Packed Encoding Rules (Aligned) of X.691 [11].

```
ETSI-TIPHON-TOKENS DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
IMPORTS
        AliasAddress,
        CallIdentifier,
        ReleaseCompleteReason,
        TimeStamp
    FROM H323-MESSAGES;
ServiceDescription :: = CHOICE
{
    basicTelephony      NULL,
    vendorSpecific      NonStandardParameter,
    ...
}
MeasureUnits :: CHOICE
{
    seconds             NULL,
    packets             NULL,
    bytes               NULL,
    vendorSpecific      NonStandardParameter,
    ...
}
UsageData :: = SEQUENCE
{
    amount              INTEGER (0..4294967295)),
    units               MeasureUnits,
    increment           INTEGER (1..4294967295),
    vendorSpecific      NonStandardParameter,
    ...
}
AuthorizationToken :: = SEQUENCE
{
    random              INTEGER (1..4294967295),
    sourceInfo          SEQUENCE OF AliasAddress,
    destinationInfo     SEQUENCE OF AliasAddress,
    callId              CallIdentifier OPTIONAL,
    validAfter          TimeStamp OPTIONAL,
    validUntil          TimeStamp OPTIONAL,
    transactionId       OCTET STRING (SIZE(16)) OPTIONAL,
    serviceAuthorized   SEQUENCE OF SEQUENCE
    {
        service         ServiceDescription,
        limit           UsageData OPTIONAL
    }                   OPTIONAL,
    authorityURL        SEQUENCE OF IA5String (SIZE(1..512)) OPTIONAL,
    vendorSpecific      NonStandardParameter OPTIONAL,
    ...
}
END     -- of ASN.1
```

D.2.2 XML Format

Authorization tokens may also be constructed in XML as a standalone XML document. The root element of such a document shall be `TokenInfo`, and it shall conform to the following construction.

```
<!DOCTYPE TokenInfo [
<!ELEMENT TokenInfo ( SourceInfo?, SourceAlternate*, DestinationInfo?, DestinationAlternate*,
                      CallId*, ValidAfter?, ValidUntil?, TransactionId?, UsageDetail*,
                      AuthorityURL* ) >
<!ATTLIST TokenInfo random #REQUIRED >
...
]>
```

The individual elements of the document, including any private or future extensions, shall conform to the body of this specification. The following text illustrates an example XML token.

```
    <TokenInfo random="12345678">
        <SourceInfo type="e164">81458811202</SourceInfo>
        <DestinationInfo type="e164">4766841360</DestinationInfo>
        <CallId encoding=base64>Aadf981lasdfA8adooif7c3nnsa89</CallId>
        <ValidAfter>1998-04-24T17:01:01Z</ValidAfter>
        <ValidUntil>1998-04-24T17:11:01Z</ValidUntil>
        <TransactionId>6772374</TransactionId>
        <UsageDetail>
            <Service/>
            <Amount>24</Amount>
            <Increment>3600</Increment>
            <Unit>s</Unit>
        </UsageDetail>
    </TokenInfo>
```

D.3 Token Carriage

In H.323-based communications, these tokens should be carried as a `NonStandardParameter` within H.235 [14] `ClearToken` objects. Object identifier values for the required `nonStandardIdentifier` shall be assigned by ETSI and listed, in this clause, before final publication of this specification.

Annex E (informative): Example Messages

This annex provides complete example messages for several information exchanges. The messages included are full and complete, but subject to the following modifications for readability:

- The content-type field of the HTTP header is shown as several lines. In actual implementations, that field will be constrained to a single line in conformance with HTTP specifications.
- Extra whitespace is included within the example XML documents. In actual implementations, this whitespace shall be removed before the digital signature is generated (according to the constraints of clause 7), and is not likely to be included in the actual transferred data.
- The digital signatures within the example messages do not represent the actual digital signature for the example content. True signatures would likely result in unprintable characters.

E.1 Pricing Exchange

The following two messages convey pricing information between two parties. The first message provides three separate price indications. Taken together, the three indications represent prices for basic Internet telephony service (thus the empty `<Service>` element) of ½ DM/minute to Berlin (country code 49, national destination code 30) 1 DM/minute elsewhere in Germany (country code 49), and 2 DM/minute elsewhere in the world. In all three cases no constraints are placed on the calling party (thus the empty `<SourceInfo>` elements).

```
POST scripts/settlements HTTP/1.0
content-type: multipart/signed;
    protocol="application/pkcs7-signature";
    micalg=sha1;
    boundary=bar
content-length: 1968

--bar
Content-Type: text/plain
Content-Length: 1647

<?xml version=1.0?>
<Message messageId="987654321" random="12345678">
    <PricingIndication componentId="1234567890">
        <Timestamp>
            1998-04-20T19:03:00Z
        </Timestamp>
        <SourceInfo type="e164prefix"/>
        <DestinationInfo type="e164prefix"/>
        <Currency>
            DEM
        </Currency>
        <Amount>
            2
        </Amount>
        <Increment>
            60
        </Increment>
        <Unit>
            s
        </Unit>
        <Service/>
        <ValidAfter/>
        <ValidUntil/>
    </PricingIndication>
    <PricingIndication componentId="1234567891">
        <Timestamp>
            1998-04-20T19:03:02Z
        </Timestamp>
        <SourceInfo type="e164prefix"/>
        <DestinationInfo type="e164prefix">
            49
        </DestinationInfo>
        <Currency>
            DEM
        </Currency>
        <Amount>
```

```
            1
        </Amount>
        <Increment>
            60
        </Increment>
        <Unit>
            s
        </Unit>
        <Service/>
        <ValidAfter/>
        <ValidUntil/>
    </PricingIndication>
    <PricingIndication componentId="1234567892">
        <Timestamp>
            1998-04-20T19:03:05Z
        </Timestamp>
        <SourceInfo type="e164prefix"/>
        <DestinationInfo type="e164prefix">
            4930
        </DestinationInfo>
        <Currency>
            DEM
        </Currency>
        <Amount>
            0.5
        </Amount>
        <Increment>
            60
        </Increment>
        <Unit>
            s
        </Unit>
        <Service/>
        <ValidAfter/>
        <ValidUntil/>
    </PricingIndication>
</Message>

--bar
Content-Type: application/pkcs7-signature
Content-Length: 191

GhyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT64VQpfyF467GhIGfHfYT6jH77n8HH
GghyHhHUujhh756tbB9HGTrfvbnjn8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--bar--
```

The following example shows a confirmation in reply to the above message. In the confirmation, all pricing information is accepted.

```
HTTP/1.0 200 OK
content-type: multipart/signed;
    protocol="application/pkcs7-signature";
    micalg=sha1;
    boundary=bar
content-length: 1401

--bar
Content-Type: text/plain
Content-Length: 1080

<?xml version=1.0?>
<Message messageId="987654321" random="12345678">
    <PricingConfirmation componentId="1234567890">
        <Timestamp>
            1998-04-20T19:04:00Z
        </Timestamp>
        <Status>
            <Code>
                201
            </Code>
            <Description>
                new pricing created
            </Description>
        </Status>
    </PricingConfirmation>
    <PricingConfirmation componentId="1234567891">
        <Timestamp>
            1998-04-20T19:04:22Z
        </Timestamp>
        <Status>
            <Code>
                210
            </Code>
            <Description>
                revised pricing accepted
            </Description>
        </Status>
    </PricingConfirmation>
    <PricingConfirmation componentId="1234567892">
        <Timestamp>
            1998-04-20T19:04:45Z
        </Timestamp>
        <Status>
            <Code>
                210
            </Code>
            <Description>
                revised pricing accepted
            </Description>
        </Status>
    </PricingConfirmation>
</Message>

--bar
Content-Type: application/pkcs7-signature
Content-Length: 191

GhyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT64VQpfyF467GhIGfHfYT6j
H77n8HHGghyHhHUujhJh756tbB9HGTrfvbnjn8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT
6ghyHhHUujpfyF47GhIGfHfYT64VQbnj756

--bar--
```

E.2 Authorization Exchange

The authorization exchange shows one party requesting and receiving authorization to access another party’s devices. In particular, the requestor wishes to complete a phone call in which the calling party is at +81 45 881 1202 and the called party is at +47 66 84 13 60.

```
POST scripts/settlements HTTP/1.0
content-type: multipart/signed;
    protocol="application/pkcs7-signature";
    micalg=sha1;
    boundary=bar
content-length: 920

--bar
Content-Type: text/plain
Content-Length: 600

<?xml version=1.0?>
<Message messageId="123454321" random="12345678">
    <AuthorizationRequest componentId="9876567890">
        <Timestamp>
            1998-04-24T17:03:00Z
        </Timestamp>
        <CallId encoding=”base64”>
            YT64VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756t
        </CallId>
        <SourceInfo type="e164">
            81458811202
        </SourceInfo>
        <DestinationInfo type="e164">
            4766841360
        </DestinationInfo>
        <Service/>
        <MaximumDestinations>
            5
        </MaximumDestinations>
    </AuthorizationRequest>
</Message>

--bar
Content-Type: application/pkcs7-signature
Content-Length: 191

GhyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT64VQpfyF467GhIGfHfYT6j
H77n8HHGghyHhHUujhJh756tbB9HGTrfvbnjn8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT
6ghyHhHUujpfyF47GhIGfHfYT64VQbnj756
```

```
--bar--
```

The reply to this request returns two separate authorizations, each representing a different peer system that is capable of completing the call. For each destination, the response authorizes up to 24 hours of service for the call.

```
HTTP/1.0 200 OK
content-type: multipart/signed;
    protocol="application/pkcs7-signature";
    micalg=sha1;
    boundary=bar
content-length: 2120

--bar
Content-Type: text/plain
Content-Length: 1799

<?xml version=1.0?>
<Message messageId="123454321" random="12345678">
    <AuthorizationResponse componentId="9876567890">
        <Timestamp>
            1998-04-24T17:03:01Z
        </Timestamp>
        <Status>
            <Code>
                200
            </Code>
            <Description>
                success
            </Description>
        </Status>
        <TransactionId>
            67890987
        </TransactionId>
        <Destination>
            <DestinationSignalAddress>
                [172.16.1.2]:112
            </DestinationSignalAddress>
            <Token encoding="base64">
                YT64VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756t
                HGTrfvbnjn8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfH
                6ghyHhHUujpfyF47GhIGfHfYT64VQbnj
            </Token>
            <ValidAfter>
                1998-04-24T17:01:01Z
            </ValidAfter>
            <ValidUntil>
                1998-04-24T17:11:01Z
            </ValidUntil>
            <CallId encoding=”base64”>
                YT64VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756t
            </CallId>
            <UsageDetail>
                <Service/>
                <Amount>
                    24
                </Amount>
                <Increment>
                    3600
                </Increment>
                <Unit>
                    s
                </Unit>
            </UsageDetail>
        </Destination>
        <Destination>
            <DestinationSignalAddress>
                [10.0.1.2]:112
            </DestinationSignalAddress>
            <Token encoding="base64">
                F467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tYT64VQpfy
                8HHGTrfvhJhjH776tbB9HG4VQbnj756HGTrfvbnjn7GhIGfH
                ujpfyF47GhIGfHfYT64VQbnj6ghyHhHU
            </Token>
            <ValidAfter>
                1998-04-24T17:01:02Z
            </ValidAfter>
            <ValidUntil>
                1998-04-24T17:11:02Z
            </ValidUntil>
            <CallId encoding=”base64”>
```

```
                YT64VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756t
            </CallId>
            <UsageDetail>
                <Service/>
                <Amount>
                    24
                </Amount>
                <Increment>
                    3600
                </Increment>
                <Unit>
                    s
                </Unit>
            </UsageDetail>
        </Destination>
    </AuthorizationResponse>
</Message>

--bar
Content-Type: application/pkcs7-signature
Content-Length: 191

GhyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT64VQpfyF467GhIGfHfYT6
jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnjn8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHf
YT6ghyHhHUujpfyF47GhIGfHfYT64VQbnj756

--bar--
```

E.3 Usage Exchange

The final examples demonstrate the exchange of usage information. The first message reports a call duration of 10 minutes.

```
POST scripts/settlements HTTP/1.0
content-type: multipart/signed;
    protocol="application/pkcs7-signature";
    micalg=sha1;
    boundary=bar
content-length: 1246

--bar
Content-Type: text/plain
Content-Length: 926

<?xml version=1.0?>
<Message messageId="123454321" random="12345678">
    <UsageIndication componentId="13579990">
        <Timestamp>
            1998-04-24T22:03:00Z
        </Timestamp>
        <Role>
            source
        </Role>
        <TransactionId>
            67890987
        </TransactionId>
        <CallId encoding=”base64”>
            YT64VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756t
        </CallId>
        <SourceInfo type="e164">
            81458811202
        </SourceInfo>
        <DestinationInfo type="e164">
            4766841360
        </DestinationInfo>
        <DestinationAlternate type="transport">
            [10.0.1.2]:112
        </DestinationAlternate>
        <UsageDetail>
            <Service/>
            <Amount>
                10
            </Amount>
            <Increment>
                60
            </Increment>
            <Unit>
                s
            </Unit>
        </UsageDetail>
    </UsageIndication>
</Message>

--bar
Content-Type: application/pkcs7-signature
Content-Length: 191

GhyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT64VQpfyF467GhIGfH
fYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnjn8HHGTrfvhJhjH776tbB9HG4VQbnj756
7GhIGfHfYT6ghyHhHUujpfyF47GhIGfHfYT64VQbnj756

--bar--
```

To complete the exchange, the server responds with a UsageConfirmation.

```
HTTP/1.0 200 OK
content-type: multipart/signed;
    protocol="application/pkcs7-signature";
    micalg=sha1;
    boundary=bar
content-length: 724

--bar
Content-Type: text/plain
Content-Length: 404

<?xml version=1.0?>
<Message messageId="123454321" random="12345678">
    <UsageConfirmation componentId="13579990">
        <Timestamp>
            1998-04-24T22:44:00Z
        </Timestamp>
        <Status>
            <Code>
                201
            </Code>
            <Description>
                new usage information created
            </Description>
        </Status>
    </UsageConfirmation>
</Message>

--bar
Content-Type: application/pkcs7-signature
Content-Length: 191

GhyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT64VQpfyF467GhIGfHfYT6
jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnjn8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHf
YT6ghyHhHUujpfyF47GhIGfHfYT64VQbnj756

--bar--
```

Annex F (informative): Billing Format Conversion

A wide variety of billing formats are currently in use within telecommunications and data networking systems. One of the advantages of using Extensible Markup Language within this specification is that it allows extremely easy conversion of usage information to and from other formats. As a demonstration of that flexibility, this annex includes example software to covert from `<UsageIndication>` components to a proprietary call detail record format of one particular, commercially-available system—the VocalTec Internet Telephony Gateway version 3.1. The software is written in Java, and relies on an XML parser available from Microsoft at http://www.microsoft.com/xml. Both the gateway and XML parser are selected solely to demonstrate the ease of the format conversion; this annex does not imply the suitability or fitness of either product for any purpose.

```
import com.ms.xml.parser.*;
import com.ms.xml.om.Document;
import com.ms.xml.om.Element;
import com.ms.xml.util.XMLOutputStream;
import com.ms.xml.util.Name;
import com.ms.xml.om.ElementEnumeration;
import com.ms.xml.om.ElementImpl;

import java.util.Enumeration;
import java.io.*;
import java.io.PrintStream;
import java.net.*;

import java.util.*;

import com.ms.xml.util.Name;
import com.ms.xml.om.ElementFactory;

class CDR
{

    public static void main(String args[])
    {
        // "fix" the version to 2 digits to workaround a bug in the XML parser
        Properties props = new Properties ();
        props = System.getProperties ();
        props.put ("java.version", "1.1");
        System.setProperties (props);

        parseArgs (args);
        if (fileName == null)
            printUsage (System.out);
        else
        {
            URL url = createURL (fileName);

            Document d = null;
            try
            {
                d = new Document ();
                d.setCaseInsensitive (true);
                d.setLoadExternal (true);
                d.load (url);
            }
            catch (ParseException e)
            {
                System.out.println (e);
            }

            if (d != null)
            {
                loadValues (d);

                // type
                cdr = pad ("Voice", 10);

                // date
                cdr += pad (((timestamp == null) ? "  --" :
                    timestamp.substring (5, 10) + "-" + timestamp.substring (2, 4)), 10);

                // time
                cdr += pad (((timestamp == null) ? "  --" :
                    timestamp.substring (11, 19)), 10);

                // duration
                cdr += pad (((unit != null && unit.equals ("s") && amount != null &&
                    increment != null) ? "" + Integer.parseInt (amount) *
                    Integer.parseInt (increment) : "  --"), 10);

                // username
                cdr += pad ("User Name", 30);

                // dialed number
                cdr += pad (destinationinfo, 30);

                // line
                cdr += pad ("1", 10);

                // remote name
                cdr += pad ("Remote Name", 30);

                // disconnect reason
                cdr += pad ("Disconnect Reason", 40);

                // remote IP
                cdr += pad ("000.000.000.000", 20);

                // remote code
                cdr += pad ("0000", 15);

                // call type
                cdr += pad ((role != null && role.equals ("destination") ? "IPVOD" : "OPVOD"), 15);

                // user ID
                cdr += pad ("000000", 10);

                // fax pages
                cdr += pad ("", 13);

                // remote fax ID
                cdr += pad ("", 25);

                // local fax ref
                cdr += pad ("", 16);

                // remote fax ref
                cdr += pad ("", 16);

                // fax transfer mode
                cdr += pad ("", 18);

                // E.164 number
                cdr += pad (destinationinfo, 24);

                System.out.println (cdr);
            }
        }
        System.exit(0);
    }

    static void printUsage (PrintStream o)
    {
        o.println ("Usage:  java CDR filename");
        o.println ();
    }

    static void parseArgs (String args[])
    {
        if (args.length > 0)
            fileName = args [0];
    }

    static String pad (String s, int num)
    {
        String str;
        int len;
        if (s == null)
        {
            str = "  --";
            len = 4;
        }
        else
        {
            str = s;
            len = s.length ();
            if (len > num)
            {
                str = "  --";
```

```
                len = 4;
            }
        }
        for (int i = len; i < num; i++)
            str += " ";
        return str;
    }

    static URL createURL (String fileName)
    // Microsoft method
    {
        URL url = null;
        try
        {
            url = new URL (fileName);
        }
        catch (MalformedURLException ex)
        {
            File f = new File (fileName);
            try
            {
                String path = f.getAbsolutePath();
                String fs = System.getProperty("file.separator");
                if (fs.length() == 1)
                {
                    char sep = fs.charAt (0);
                    if (sep != '/')
                        path = path.replace (sep, '/');
                        if (path.charAt (0) != '/')
                            path = '/' + path;
                }
                path = "file://" + path;
                url = new URL (path);
            }
            catch (MalformedURLException e)
            {
                System.out.println ("Cannot create url for: " + fileName);
                System.exit(0);
            }
        }
        return url;
    }

    static void loadValues (Element e)
    {
        String attName = "--NULL--";
        String tagName;

        if (e.getTagName () != null)
        {
            attName = e.getText ();
            tagName = e.getTagName ().toString ();

            if (tagName.equals ("TIMESTAMP"))
                timestamp = attName;

            if (tagName.equals ("DESTINATIONINFO"))
                destinationinfo = attName;

            if (tagName.equals ("AMOUNT"))
                amount = attName;

            if (tagName.equals ("INCREMENT"))
                increment = attName;

            if (tagName.equals ("UNIT"))
                unit = attName;

            if (tagName.equals ("ROLE"))
                role = attName;
        }

        for (ElementEnumeration en = new ElementEnumeration (e); en.hasMoreElements(); )
        {
            Element child = (Element) en.nextElement();
            loadValues (child);
        }
    }

    static String fileName, timestamp, destinationinfo, amount, increment, unit, role, cdr;
}
```

Annex G (informative): XML Overview

As an aid to those readers unfamiliar with Extensible Markup Language, this annex offers a brief overview of the syntax of XML documents. Key components of XML structure are document definitions, element declarations, and attribute declarations.

NOTE: This annex is neither authoritative nor complete. A formal definition of XML may be found in [3].

G.1 Document Definition

XML documents are defined using the following format:

```
<!DOCTYPE DocumentName [DocumentStructureDefinition] >
```

`DocumentName` is the name of the XML document, and `DocumentStructureDefnition` is a series of element, attribute, and entity declarations. Those declarations are described below. An example document definition is:

```
<!DOCTYPE Z [
    <!ELEMENT Y (X, W) >
    <!ATTLIST Y ('a' | 'b' | 'c') 'a' #REQUIRED>
    <!ELEMENT X CDATA >
    <!ELEMENT W CDATA>
]>
```

G.2 Element Declaration

XML elements are declared using the following format:

```
<!ELEMENT ElementName (ElementContents)>
```

`ElementName`, as expected, is the name of the element being declared, while `ElementContents` specifies the permissible contents of the element. Elements may include child elements, in which case the `ElementContents` part defines their order and number within the parent element, and element may include character data.

Child elements within the `ElementContents` part of a declaration may be designated as a sequence of child elements or a choice of child elements. The comma (,) separates individual child elements in a sequence, while the vertical bar (|) separates child elements in a choice. For example, `<!ELEMENT Y (X, W)>` indicates that element `Y` consists of a child element `X` followed by a child element `W`. Similarly, `<!ELEMENT Z (X | W)>` indicates that element `Z` consists of either a child element `X` or a child element `W`.

The element declaration indicates the number of child elements permitted by a character following the child element's name. No character indicates that exactly one child element, a question mark (?) indicates zero or one child element, an asterisk (*) indicates zero or more child elements, and a plus (+) indicates one or more child elements. For example,

```
<!ELEMENT X (A, B?, C*, D+)>
```

defines element `X` as consisting of the sequence:

- exactly one child element `A;`
- an optional child element `B` (zero or one);
- an optional series of child element `C`s (zero or more);
- at least one, and possible multiple, child element `D`s (one or more).

G.3 Attribute Declaration

XML attributes are associated with elements; they are declared using the format:

```
<!ATTLIST ElementName AttributeName1 DeclaredValue1 DefaultValue1
                      AttributeName2 DeclaredValue2 DefaultValue2
                                           ...
                      AttributeNameN DeclaredValueN DefaultValueN>
```

`ElementName` identifies the XML element with which the attributes may be used. `AttributeName1`, `AttributeName2`, … are the names of the attributes defined for the element. Each attribute has either a list of permissible values or a data type, which the above construction labels `DeclaredValue`. The final part of each attribute's declaration is `DefaultValue`, which indicates the default value for the attribute, as well as constraints on its presence.

Explicit, permissible values for attributes are separated by the vertical bar (|). So that, for example the `Boolean` attribute for element `X` may be declared as

```
<!ATTLIST X Boolean (True | False) >
```

Note that no default value is specified in the above declaration. If a default value is specified, then the DefaultValue section may appear as one of:

`#REQUIRED` some explicit value for the attribute shall be included each time the attribute is used

`#IMPLIED` if no explicit value for the attribute is used, then the application may infer a default value

`'value'` if no explicit value for the attribute is used, then the attribute should be assumed to have the value '`value`'

`#FIXED 'value'` the attribute shall have, and can only have, the value '`value`'

For example, if the `Boolean` attribute should be assumed to be `True` unless otherwise, explicitly indicated, the following declaration would apply.

```
<!ATTLIST X Boolean (True | False) 'True' >
```

Note that no quotation marks are used in the `DeclaredValue` part, while the `DefaultValue` part does enclose the value in quotations.

History

Document history		
V1.4.2	December 1998	Publication

ISBN 2-7437-2739-X
Dépôt légal : Décembre 1998