



Liberty ID-WSF SOAP Binding Specification

Version: 2.0-09

Editors:

Jeff Hodges, NeuStar, Inc.
John Kemp, Nokia Corporation
Robert Aarts, Trustgenix, Inc.
Greg Whitehead, Trustgenix, Inc.

Contributors:

Conor Cahill, America Online, Inc.
Darryl Champagne, IEEE-ISTO
Marc Hadley, Sun Microsystems, Inc.
Jukka Kainulainen, Nokia Corporation
Jonathan Sergent, Sun Microsystems, Inc.

Abstract:

This specification defines a SOAP binding for the Liberty Identity Web Services Framework (ID-WSF) and the Liberty Identity Services Interface Specifications (ID-SIS). It specifies use of the Web Services Addressing (WS-Addressing) SOAP extension, as well as provider declaration, processing context, consent claims, usage directives and a number of other optional headers.

Filename: draft-liberty-idwsf-soap-binding-v2.0-09.pdf

1

Notice

2 This document has been prepared by Sponsors of the Liberty Alliance. Permission is hereby granted to use the
3 document solely for the purpose of implementing the Specification. No rights are granted to prepare derivative works
4 of this Specification. Entities seeking permission to reproduce portions of this document for other uses must contact
5 the Liberty Alliance to determine whether an appropriate license for such use is available.

6 Implementation of certain elements of this document may require licenses under third party intellectual property
7 rights, including without limitation, patent rights. The Sponsors of and any other contributors to the Specification are
8 not, and shall not be held responsible in any manner for identifying or failing to identify any or all such third party
9 intellectual property rights. **This Specification is provided "AS IS", and no participant in the Liberty Alliance
10 makes any warranty of any kind, express or implied, including any implied warranties of merchantability,
11 non-infringement of third party intellectual property rights, and fitness for a particular purpose.** Implementors
12 of this Specification are advised to review the Liberty Alliance Project's website (<http://www.projectliberty.org>) for
13 information concerning any Necessary Claims Disclosure Notices that have been received by the Liberty Alliance
14 Management Board.

15 Copyright © 2005 Adobe Systems; America Online, Inc.; American Express Company; Amsoft Systems Pvt Ltd.;
16 Avatier Corporation; Axalto; Bank of America Corporation; BIPAC; BMC Software, Inc.; Computer Associates
17 International, Inc.; DataPower Technology, Inc.; Diversinet Corp.; Enosis Group LLC; Entrust, Inc.; Epok, Inc.;
18 Ericsson; Fidelity Investments; Forum Systems, Inc.; France Télécom; French Government Agence pour le
19 développement de l'administration électronique (ADAE); Gamefederation; Gemplus; General Motors; Giesecke &
20 Devrient GmbH; GSA Office of Governmentwide Policy; Hewlett-Packard Company; IBM Corporation; Intel
21 Corporation; Intuit Inc.; Kantega; Kayak Interactive; MasterCard International; Mobile Telephone Networks (Pty)
22 Ltd; NEC Corporation; Netegrity, Inc.; NeuStar, Inc.; Nippon Telegraph and Telephone Corporation; Nokia
23 Corporation; Novell, Inc.; NTT DoCoMo, Inc.; OpenNetwork; Oracle Corporation; Ping Identity Corporation;
24 Reactivity Inc.; Royal Mail Group plc; RSA Security Inc.; SAP AG; Senforce; Sharp Laboratories of America;
25 Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Supremacy Financial Corporation; Symlabs, Inc.;
26 Telecom Italia S.p.A.; Telefónica Móviles, S.A.; Trusted Network Technologies; Trustgenix; UTI; VeriSign, Inc.;
27 Vodafone Group Plc.; Wave Systems Corp. All rights reserved.

28 Liberty Alliance Project
29 Licensing Administrator
30 c/o IEEE-ISTO
31 445 Hoes Lane
32 Piscataway, NJ 08855-1331, USA
33 info@projectliberty.org

34 Contents

35	1. Introduction	5
36	2. Notation and Conventions	7
37	2.1. XML Namespaces	7
38	2.2. Terminology	7
39	2.3. Treatment of Boolean Values	9
40	2.4. String and URI Values	9
41	2.5. Time Values	10
42	3. Schema Particulars	11
43	3.1. Schema Declarations	11
44	3.2. "ID" Types	11
45	3.3. Status Types	11
46	3.3.1. Status Codes	11
47	3.4. SOAP Fault Types	12
48	4. SOAP Binding	14
49	4.1. SOAP Version	14
50	4.2. The SOAPAction HTTP Header	14
51	4.3. Ordinary ID-* Messages	14
52	4.4. ID-* Fault Messages	14
53	4.5. SOAP-bound ID-* Messages	15
54	5. Messaging-specific Header Blocks	18
55	5.1. The <wsu:Timestamp> element in the <wsse:Security> Header Block	18
56	5.2. The <wsa:MessageID> Header Block	18
57	5.2.1. <wsa:MessageID> Value Requirements	18
58	5.3. The <wsa:RelatesTo> Header Block	18
59	5.4. The <wsa:To> Header Block	18
60	5.5. The <wsa:Action> Header Block	19
61	5.6. The <wsa:ReplyTo> Header Block	19
62	5.7. The <wsa:FaultTo> Header Block	19
63	5.8. The <Sender> Header Block	19
64	5.9. The <InvocationIdentity> Header Block	20
65	5.10. The <TargetIdentity> Header Block	21
66	5.11. Messaging Processing Rules	22
67	5.11.1. Constructing and Sending a SOAP-bound ID-* Message	23
68	5.11.2. Receiving and Processing a SOAP-bound ID-* Message	25
69	5.12. Examples	28
70	6. Optional Header Blocks	31
71	6.1. The <ProcessingContext> Header Block	31
72	6.1.1. The ProcessingContextType Header Block Type	31
73	6.1.2. <ProcessingContext> Header Block Element	32
74	6.1.3. <ProcessingContext> Header Block Semantics and Processing Rules	32
75	6.2. The <Consent> Header Block	35
76	6.2.1. The consentType Header Block Type	35
77	6.2.2. <Consent> Header Block Element	36
78	6.3. The <CredentialsContext> Header Block	36
79	6.3.1. Overview	36
80	6.3.2. CredentialsContext Type and Element	37
81	6.3.3. CredentialsContext Example	37
82	6.3.4. Processing Rules	38
83	6.4. The <EndpointUpdate> Header Block	38
84	6.4.1. Overview	39
85	6.4.2. EndpointUpdate Type and Element	39
86	6.4.3. EndpointUpdate Examples	40

87	6.4.4. Processing Rules for the EndpointUpdate header	43
88	6.4.5. Processing Rules for the EndpointMoved SOAP Fault	44
89	6.5. The <Timeout> Header Block	44
90	6.5.1. Overview	44
91	6.5.2. Timeout Type and Element	45
92	6.5.3. Timeout Example	45
93	6.5.4. Processing Rules	46
94	6.6. The <UsageDirective> Header Block	47
95	6.6.1. Overview	47
96	6.6.2. UsageDirective Header Type and Element	47
97	6.6.3. Usage Directive Examples	48
98	6.6.4. Processing Rules	49
99	6.7. The <ApplicationEPR> Header Block	50
100	7. Security Considerations	52
101	8. Acknowledgements	53
102	Bibliography	54
103	A. liberty-idwsf-soap-binding-v2.0.xsd Schema Listing	56
104	B. liberty-idwsf-utility-v2.0.xsd Schema Listing	59
105	C. liberty-utility-v2.0.xsd Schema Listing	60
106	D. soap-envelope-1.1.xsd Schema Listing	62
107	E. ws-addr-1.0.xsd Schema Listing	65
108	F. wss-util-1.0.xsd Schema Listing	68

109 1. Introduction

110 The Liberty Identity Web Services Framework (ID-WSF) [[LibertyIDWSFOverview](#)] is designed so that "application
111 layer" messages or "services" messages utilizing the framework, referred to as *ID-* messages* in this specification, may
112 be mapped onto various transport or transfer protocols. Thus, they are designed to be conveyed in the data portion of
113 the underlying protocol's messages. ID-* messages do not intrinsically address specific aspects of message exchange
114 such as: to which system entity the message is to be sent, message correlation, the mechanics of message exchange,
115 or security context.

116 Examples of ID-* messages include the <DiscoveryLookupRequest> message of [[LibertyDisco](#)], and the
117 <Modify> message of [[LibertyIDPP](#)].

118 This specification defines a mapping of ID-* messages onto SOAP [[SOAPv1.1](#)], an XML-based [[XML](#)] messaging
119 protocol.

120 SOAP itself does not define the specific message exchange aspects mentioned above, but offers an *extensibility model*
121 that may be used to define message components that do address such message exchange specifics. SOAP extensibility
122 is effected by adding message components to the portion of the SOAP message called the *Header*. These message
123 components are referred to as *SOAP header blocks* [[SOAPv1.2](#)].

124 WS-Addressing SOAP Binding [[WSAv1.0-SOAP](#)] is a SOAP extension that defines a set of SOAP header blocks
125 that facilitate end-to-end addressing and message correlation. This specification profiles WSAv1.0-SOAP to address
126 specific aspects of ID-* message exchange functionality.

127 This specification also defines several optional SOAP header blocks relevant to ID-* message processing. They are:

128 • Processing Context:

129 An ID-* requester may need to express additional context for a given request, for example indicating that the
130 requester expects to make such requests in the future when the Principal may or may not be online. This
131 specification defines the <ProcessingContext> header block for this purpose.

132 • Consent Claims:

133 ID-WSF-based entities may wish to claim whether they obtained the Principal's consent for carrying out any
134 given operation, such as updating a Principal's Personal Profile entry [[LibertyIDPP](#)]. This specification defines the
135 <Consent> header block for this purpose.

136 • Credentials Context:

137 The receiver of an ID-* message might indicate that credentials supplied in the request did not meet its policy in
138 allowing access to the requested resource. The <CredentialsContext> header block allows such policies to be
139 expressed to the requester.

140 • Endpoint Update:

141 The <EndpointUpdate> header block allows a service to indicate that requesters should contact it on a different
142 endpoint or use a different security mechanism and credentials to access the requested resource.

143 • Timeout:

144 The <Timeout> header block is defined in this specification to allow the receiver of an ID-* message to indicate
145 that processing of the received message failed due to a timeout condition.

146 • Usage Directives:

147 ID-WSF-based entities may wish to indicate their policies for handling data at the time of data request, and entities
148 releasing data may wish to specify their policies for the subsequent use of data at the time of data release. This
149 specification defines the <UsageDirective> header block for this purpose.

150 Additionally, this specification defines how ID-* messages are bound into SOAP message bodies, and how the SOAP
151 header blocks implementing the above functionalities are bound into SOAP message headers.

152 Note that other specifications in the ID-WSF specification suite also define SOAP header blocks, for example
153 [[LibertySecMech](#)] and [[LibertyInteract](#)] , which may be used concurrently with the header blocks defined in this
154 specification. Header blocks specified in specifications outside of the ID-WSF specification suite may also be
155 composed with ID-WSF header blocks. An example is the <wsse:Security> header block as discussed in
156 [[LibertySecMech](#)] . However no further mention of doing such is made in this specification.

157 2. Notation and Conventions

158 This specification uses schema documents conforming to W3C XML Schema [Schema1] and normative text to
 159 describe the syntax and semantics of XML-encoded protocol messages.

160 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT",
 161 "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]:

162 “ they MUST only be used where it is actually required for interoperation or to limit behavior which
 163 has potential for causing harm (e.g., limiting retransmissions) ”

164 These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application
 165 features and behavior that affect the interoperability and security of implementations. When these words are not
 166 capitalized, they are meant in their natural-language sense.

167 2.1. XML Namespaces

168 This specification makes normative use of the XML namespace prefixes noted in Table 1.

169 **Table 1. XML Namespaces and Prefixes**

Prefix	Namespace
sb:	Represents the Liberty SOAP Binding namespace (v2.0): <code>urn:liberty:sb:2005-11</code> Note This is the point of definition of this namespace. This namespace is the default for instance fragments, type names, and element names in this document when a namespace is not explicitly noted.
idpp:	Represents the namespace defined in [LibertyIDPP].
is:	Represents the namespace defined in [LibertyInteract].
S:	Represents the SOAP namespace: <code>http://schemas.xmlsoap.org/soap/envelope/</code> This namespace is defined in [SOAPv1.1].
samlp2:	Represents the namespace defined in [SAMLCore2].
wsa:	Represents the WS-Addressing namespace: <code>http://www.w3.org/2005/08/addressing</code> This namespace is defined in [WSAv1.0].
wsse:	Represents the SOAP Message Security namespace: <code>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-secext-1.0.xsd</code> This namespace is defined in [wss-sms].
wsu:	Represents the SOAP Message Security Utility namespace: <code>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd</code> This namespace is defined in [wss-sms].
xs:	Represents the W3C XML schema namespace: <code>http://www.w3.org/2001/XMLSchema</code> This namespace is defined in [Schema1].

170 2.2. Terminology

171 This section defines key terminology used in this specification. Definitions for other Liberty-specific terms can be
172 found in [\[LibertyGlossary\]](#). See also [\[RFC2828\]](#) for overall definitions of security-related terms.

173 affiliation An *affiliation* is a set of one or more entities, described by *Provider IDs*, who may perform
174 Liberty interactions as a member of the set. An affiliation is referenced by exactly one
175 *Affiliation ID*, and is administered by exactly one entity identified by their Provider ID.
176 Members of an affiliation may invoke services either as a member of the affiliation—by virtue
177 of their Affiliation ID, or individually by virtue of their Provider ID [\[LibertyGlossary\]](#).

178 Affiliation ID An *Affiliation ID* identifies an *affiliation*. It is schematically represented by the
179 `affiliationID` attribute of the `<AffiliationDescriptor>` metadata element
180 [\[LibertyMetadata\]](#).

181 client A *role* assumed by a *system entity* which makes a request of another system entity, often
182 termed a *server* [\[RFC2828\]](#), i.e. a client is also a *sender*.

183 ID-* A shorthand designator referring to the Liberty ID-WSF, ID-FF, and ID-SIS specification
184 sets. For example, one might say that the former specification sets are all part of the Liberty
185 ID-* specification suite.

186 ID-* header block One of the header blocks defined in this specification, or defined in any of the other Liberty
187 ID-* specification suite.

188 ID-* message Equivalent to *ordinary ID-* message*.

189 ID-* fault message See [Section 4.4](#).

190 ID-SIS Liberty Identity Service Interface specification set.

191 ID-WSF Liberty Identity Web Services Framework specification set.

192 MEP see Message Exchange Pattern.

193 Message Exchange Pattern A [\[SOAPv1.2\]](#) term for the overall notion of various patterns of message exchange
194 between SOAP nodes. For example, request-reply and one-way are two *MEPs* used in this
195 specification.

196 message thread A *message thread* is an exchange of messages in a request-response *MEP* between two
197 *SOAP nodes*. All the messages of a given message thread are "linked" via each message's
198 `<wsa:RelatesTo>` header block value being set, by the sender, from the previous success-
199 fully received message's `<wsa:MessageID>` header block value.

200 Ordinary ID-* message See [Section 4.3](#).

201 processing context A *processing context* is the collection of specific circumstances under which a particular
202 processing step or set of steps take place.

203 processing context facet A *processing context facet* is an identified aspect, inherent or additive, of a *processing*
204 *context*.

205	provider	A <i>provider</i> is a Liberty-enabled entity that performs one or more of the provider roles
206		in the Liberty architecture, for example Service Provider or Identity Provider. See also
207		<i>Liberty-enabled Provider</i> in [LibertyGlossary]. Providers are identified in Liberty protocol
208		interactions by their <i>Provider IDs</i> or optionally their <i>Affiliation ID</i> if they are a member of an
209		affiliation(s) and are acting in that capacity.
210	Provider ID	A <i>Provider ID</i> identifies an entity known as a <i>provider</i> . It is schematically represented by the
211		<code>providerID</code> attribute of the <code><EntityDescriptor></code> metadata element [LibertyMetadata].
212	receiver	A <i>role</i> taken by a <i>system entity</i> when it receives a message sent by another system entity. See
213		also <i>SOAP receiver</i> in [SOAPv1.2].
214	role	A function or part performed, especially in a particular operation or process [Merriam-
215		Webster].
216	sender	A <i>role</i> donned by a <i>system entity</i> when it constructs and sends a message to another system
217		entity. See also <i>SOAP sender</i> in [SOAPv1.2].
218	server	A <i>role</i> performed by a <i>system entity</i> that provides a service in response to requests from other
219		system entities called <i>clients</i> [RFC2828]. Note that in order to provide a service to clients; a
220		server will often be both a <i>sender</i> and a <i>receiver</i> .
221	service request	A <i>service request</i> is another term for an <i>ordinary ID-* message</i> . Service request is also
222		loosely equivalent to a "SOAP-bound (ordinary) ID-* message".
223	SOAP-bound ID-* message	See Section 4.5.
224	SOAP header block	A [SOAPv1.2] term whose definition is: An [element] used to delimit data that logically
225		constitutes a single computational unit within the SOAP header. In [SOAPv1.1] these are
226		known as simply <i>SOAP headers</i> , or simply <i>headers</i> . This specification uses the SOAPv1.2
227		terminology.
228	SOAP message	In this specification, the term <i>SOAP message</i> refers to a message consisting of only a
229		<code><S:Envelope></code> element as defined in [SOAPv1.1]. It contains two top-level subelements:
230		<code><S:Header></code> and <code><S:Body></code> . This message is in turn mapped onto a lower-layer transport
231		or transfer protocol, typically HTTP [RFC2616].
232	SOAP node	A [SOAPv1.2] term describing <i>system entities</i> who are parties to SOAP-based message
233		exchanges that are, for purposes of this specification, also the ultimate destination of the
234		exchanged messages, i.e. <i>SOAP endpoints</i> . In [SOAPv1.1], SOAP nodes are referred to as
235		<i>SOAP endpoints</i> , or simply <i>endpoints</i> . This specification uses the SOAPv1.2 terminology.
236	system entity	An active element of a computer/network system. For example, an automated process or set
237		of processes, a subsystem, a person or group of persons that incorporates a distinct set of
238		functionality [SAMLGloss].

239 **2.3. Treatment of Boolean Values**

240 For readability, when an XML Schema type is specified to be `xsd:boolean`, this document discusses the values as
241 `TRUE` and `FALSE` rather than "1" and "0", which will exist in a document instance conforming to the SOAP Envelope
242 1.1 schema [[SOAPv1.1-Schema](#)].

243 **2.4. String and URI Values**

244 All string and URI [[RFC2396](#)] values in this specification have the types `string` (as a base type in this case) and
245 `anyURI` respectively, which are built in to the W3C XML Schema Datatypes specification [[Schema2](#)]. All strings
246 in ID-WSF messages **MUST** consist of at least one non-whitespace character (whitespace is defined in the XML
247 Recommendation [[XML](#)] section 2.3). Empty and whitespace-only values are disallowed. Also, unless otherwise
248 indicated in this specification, all URI values **MUST** consist of at least one non-whitespace character.

249 **Note**

250 Various element and/or attribute components of the schema described by this specification (see [Appendix A: SOAP](#)
251 [Binding Schema XSD v2.0](#) , below) may have further requirements placed on the values they may take on. For
252 example, see [Section 5.2.1: <wsa:MessageID> Value Requirements](#) .

253 **2.5. Time Values**

254 All time values in this specification have the type `dateTime`, which is built in to the W3C XML Schema Datatypes
255 specification [[Schema2](#)] and **MUST** be expressed in UTC form.

256 Senders and receivers **SHOULD NOT** rely on other applications supporting time resolution finer than milliseconds.
257 Implementations **MUST NOT** generate time instants that specify leap seconds.

258 3. Schema Particulars

259 This section addresses schema particulars such as which schemas this specification defines, describes, and depends
260 upon, as well as various underlying schema types.

261 3.1. Schema Declarations

262 This specification normatively defines and describes an XML schema which is constituted in the XML Schema
263 [Schema1] files ("Liberty ID-WSF SOAP Binding Schema v2.0", reproduced in [Appendix A](#)). In addition, the Liberty
264 ID-WSF SOAP Binding Schema file explicitly includes, in the XML Schema sense, the Liberty ID-WSF utility schema
265 file (reproduced in [Appendix B](#)).

266 Also, the Liberty ID-WSF SOAP Binding Schema files explicitly depend upon the SOAPv1.1 schema [[SOAPv1.1-](#)
267 [Schema](#)] (reproduced in [Appendix D](#)) and WSAv1.0 schema [[WSAv1.0-Schema](#)] (reproduced in [Appendix E](#)).

268 3.2. "ID" Types

269 The XML Schema [[Schema1](#)] type `xs:ID` is used in this specification to declare *ID* attributes on elements, such as
270 SOAP header blocks, that must be referenceable, say by an XML Signature. It should be noted that XML processors,
271 such as XML Signature verifiers, must be aware of the `xs:ID` type of these ID attributes in order resolve references to
272 the elements they identify. If the W3C `xml:id` recommendation is finalized before this specification goes final, all ID
273 attributes defined in this specification will be changed to `xml:id`. This change will allow XML processors to resolve
274 references to elements defined in this specification without requiring specific knowledge about the schema defined in
275 this specification.

276 3.3. Status Types

277 The `<Status>` element, of type `StatusType` complex type, is used in this specification to convey status codes and
278 related information. The schema fragment in [Figure 1](#), from the ID-WSF Utility schema ([Appendix B](#)), shows both
279 the `<Status>` element and `StatusType` complex type.

```
280
281 <xs:complexType name="StatusType">
282   <xs:annotation>
283     <xs:documentation>
284       A type that may be used for status codes.
285     </xs:documentation>
286   </xs:annotation>
287   <xs:sequence>
288     <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
289   </xs:sequence>
290   <xs:attribute name="code" type="xs:string" use="required"/>
291   <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
292   <xs:attribute name="comment" type="xs:string" use="optional"/>
293 </xs:complexType>
294
295 <xs:element name="Status" type="StatusType">
296   <xs:annotation>
297     <xs:documentation>
298       A standard Status type
299     </xs:documentation>
300   </xs:annotation>
301 </xs:element>
302
303
304
```

305 **Figure 1. status and StatusType Schema**

306 **3.3.1. Status Codes**

307 This section lists, in [Table 2](#), the values defined in this specification for the `code` attribute of the `<Status>` element.
308 Other specifications MAY define additional code attribute values.

309 **Table 2. Status Codes**

Code	Semantics	Suggested Fault Source
InvalidActor	There is an issue with the <code>actor</code> attribute on the indicated header block in the indicated message.	S:Client
InvalidMustUnderstand	There is an issue with the <code>mustUnderstand</code> attribute on the indicated header block in the indicated message.	S:Client
StaleMsg	The indicated inbound SOAP-bound ID-* message has a timestamp value outside of the receivers allowable time window.	S:Client
DuplicateMsg	The indicated inbound SOAP-bound ID-* message appears to be a duplicate.	S:Client
InvalidRefToMsgID	The indicated inbound SOAP-bound ID-* message appears to incorrectly refer to the preceding message in the message thread.	S:Client
ProviderIDNotValid	The receiver does not consider the claimed Provider ID to be valid.	S:Client
AffiliationIDNotValid	The receiver does not consider the claimed Affiliation ID to be valid.	S:Server
InvocationIdentityNotValid	The receiver does not consider the invocation identity to be valid.	S:Client
TargetIdentityNotValid	The receiver does not consider the target identity to be valid.	S:Client
IDStarMsgNotUnderstood	There was a problem with understanding/parsing the conveyed ID-* message.	S:Client
ProcCtxURINotUnderstood	The receiver did not understand the processing context facet URI.	S:Server
ProcCtxUnwilling	The receiver is unwilling to apply the sender's stipulated processing context.	S:Server
CannotHonourUsageDirective	The receiver is unable or unwilling to honor the stipulated usage directive.	S:Server
EndpointMoved	The request cannot be processed at this endpoint. This is typically used in conjunction with the <code><EndpointUpdate></code> header block to indicate the endpoint to which the request should be re-submitted.	S:Server
InappropriateCredentials	The sender has submitted a request that does not meet the needs of the receiver. The receiver may indicate credentials that are acceptable to them via a <code><CredentialsContext></code> or <code><EndpointUpdate></code> header block.	S:Client
ProcessingTimeout	The sender is indicating that processing of the request has failed due to the processing taking longer than the <code>maxProcessingTime</code> specified on the request <code><Timeout></code> header block.	S:Server

310 **3.4. SOAP Fault Types**

311 The SOAPv1.1 `Fault` and `Detail` complex types are used in this specification to convey processing exceptions.

312 The schema fragment in [Figure 2](#), extracted from [\[SOAPv1.1-Schema\]](#), defines the SOAPv1.1 `Fault` and `detail`
313 complex types, which define the `<S:Fault>` and `<detail>` elements, respectively.

314 **Note**

315 The `<S:Fault>` element is **not** intended to be used as a SOAP header block. Rather, it is designed to be conveyed in
316 the `<S:Body>` of a SOAP message.

```
317  
318 <xs:element name="Fault" type="tns:Fault"/>  
319  
320 <xs:complexType name="Fault" final="extension">  
321 <xs:annotation>  
322 <xs:documentation>  
323     Fault reporting structure  
324 </xs:documentation>  
325 </xs:annotation>  
326 <xs:sequence>  
327 <xs:element name="faultcode" type="xs:QName"/>  
328 <xs:element name="faultstring" type="xs:string"/>  
329 <xs:element name="faultactor" type="xs:anyURI" minOccurs="0"/>  
330 <xs:element name="detail" type="tns:detail" minOccurs="0"/>  
331 </xs:sequence>  
332 </xs:complexType>  
333  
334 <xs:complexType name="detail">  
335 <xs:sequence>  
336 <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>  
337 </xs:sequence>  
338 <xs:anyAttribute namespace="##any" processContents="lax"/>  
339 </xs:complexType>  
340  
341
```

342 **Figure 2. SOAP `Fault` and `detail` Types Schema**

343 4. SOAP Binding

344 This section defines the notion of *ID-* messages* and the overall, high-level considerations with respect to *binding*
345 them into *SOAP messages* for subsequent conveyance. The detailed processing rules are then given in
346 [Section 5.11: Messaging Processing Rules](#).

347 4.1. SOAP Version

348 This specification normatively depends upon SOAP version 1.1, as specified in [\[SOAPv1.1\]](#). Messages conformant to
349 this specification MUST also be conformant to [\[SOAPv1.1\]](#).

350 4.2. The SOAPAction HTTP Header

351 [\[SOAPv1.1\]](#) defines the SOAPAction HTTP header, and requires its usage on HTTP-bound SOAP messages. This
352 header may be used to indicate the "intent" of a SOAP message to the recipient.

353 Note

354 The value of the SOAPAction HTTP header SHOULD the same as the value of the <wsa:Action> header block (see
355 [Section 5.5: The <wsa:Action> Header Block](#)).

356 Also note that [\[WSDLv1.1\]](#) documents may be defined that specify the value of the SOAPAction header to be included
357 on messages sent to the service defined in WSDL.

358 4.3. Ordinary ID-* Messages

359 *Ordinary ID-* messages* are so-called "application layer" messages or "services" messages, of the forms defined in
360 the Liberty ID-WSF and ID-SIS specification sets or by other applications or services building on the Liberty ID-WSF
361 specifications. These messages as a class are characterized by being able to be correctly conveyed in the "Body" of a
362 SOAP [\[SOAPv1.1\]](#) message. See [Example 1](#). Such messages share the characteristic of needing to be mapped onto
363 an underlying transport or transfer protocol in order for them to be communicated between system entities.

```
364  
365     <idpp:Query>  
366     :  
367     <!-- various message-specific subelements may go here -->  
368     :  
369     </idpp:Query>  
370
```

371 **Example 1. A Specific ID-* Message: The <idpp:Query> Message**

372 4.4. ID-* Fault Messages

373 An *ID-* Fault Message* consists of a SOAP <S:Fault> element (see [Section 3.4: SOAP Fault Types](#)) containing a
374 <Status> element, with the attributes and attribute values of both elements configured as specified herein.

375 The <S:Fault> element's attributes and child elements MUST be tailored according to these rules:

376 1. The <S:Fault> element:

377 A. SHOULD contain a <faultcode> element whose value SHOULD be either "S:server" or "S:client".

378 **Note**

379 A <faultcode> of "S:server" indicates that the receiver believes that it has erred, whereas "S:client"
380 is intended for cases in which the receiver believes that the sender has erred. Such an indication should be
381 considered merely informational.

382 B. SHOULD contain a <faultstring> element. This string value MAY be localized.

383 C. SHOULD NOT contain a <S:faultactor> element.

384 2. The <S:Fault> element's <detail> child element MUST contain a <Status> element (see [Section 3.3: Status](#)
385 Types). The <Status> element:

386 A. MUST contain a code attribute set to the value as specified when the issuance of a ID-* Fault message
387 is indicated. Code attribute values defined in this specification are listed above in [Section 3.3.1](#). Other
388 specifications MAY define additional code attribute values.

389 B. MAY contain a ref attribute set to the value as specified in this specification when the issuance of a ID-*
390 Fault message is indicated.

391 C. MAY contain a comment attribute set to the value as specified in this specification when the issuance of a
392 ID-* Fault message is indicated. This string value MAY be localized.

393 3. Additionally, to aid in diagnostics, the header block or message body element referred to by the fault MAY be
394 included in the <S:Fault> element's <detail> element, after the <Status> element.

395 **4.5. SOAP-bound ID-* Messages**

396 ID-* messages are bound into SOAP messages, yielding *SOAP-bound ID-* messages*. This binding thus provides a
397 concrete means for ID-* message conveyance since [[SOAPv1.1](#)] specifies a binding to HTTP [[RFC2616](#)], which is
398 itself layered onto the ubiquitous [TLS/SSL]/TCP/IP protocol stack.

399 Although this binding is the only one given in this specification, other protocols could be used to convey ID-*
400 messages, with appropriateness depending on the protocol selected and the target operational context. This is not
401 discussed further in this specification.

402 **A SOAP-bound ID-* message is defined as:**

403 • having all required ID-* header blocks in its <S:Header> element, and,

404 • perhaps having other optional ID-* header blocks in its <S:Header> element, and,

405 • containing either an ordinary ID-* message, or an ID-* fault message, in its <S:Body> element. The former is
406 known as an *ordinary SOAP-bound ID-* message* (see [Example 2](#)), and the latter is known as a *SOAP-bound ID-**
407 *fault message* (see [Example 3](#)).

408 [Section 5.11: Messaging Processing Rules](#) specifies the detailed normative processing rules for constructing, sending,
409 and receiving SOAP-bound ID-* messages.

```
410
411 <S:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
412   xmlns:sb="..."
413   xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
414
415   <S:Header>
416
417     ...
418
419     <wsse:Security>
420       <wsu:Timestamp>
421         <wsu:Created>2005-06-17T04:49:17Z</wsu:Created>
422       </wsu:Timestamp>
423     </wsse:Security>
424
425     <wsa:MessageId>...</wsa:MessageId>
426
427     <wsa:To>...</wsa:To>
428
429     <wsa:Action>...</wsa:Action>
430
431     <!-- reference params from target EndpointReference -->
432
433     <sb:Sender providerID="..." affiliationID="..." />
434
435     <wsa:ReplyTo>
436       <wsa:Address>...</wsa:Address>
437     </wsa:ReplyTo>
438
439     ...
440
441   </S:Header>
442
443   <S:Body>
444
445     <idpp:Query> <!-- This is an ID-PP "Query" message bound -->
446       : <!-- into the <S:Body> of a SOAP message. -->
447       :
448     </idpp:Query>
449
450   </S:Body>
451
452 </S:Envelope>
```

453 **Example 2. An Ordinary SOAP-bound ID-* Message**


```

454
455 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
456   xmlns:sb="..."
457   xmlns:pp="urn:liberty:id-sis-pp:2003-08">
458
459   <S:Header>
460
461     ...
462
463     <wsse:Security>
464       <wsu:Timestamp>
465         <wsu:Created>2005-06-17T04:49:18Z</wsu:Created>
466       </wsu:Timestamp>
467     </wsse:Security>
468
469     <wsa:MessageId>...</wsa:MessageId>
470
471     <wsa:RelatesTo>...</wsa:RelatesTo>
472
473     <wsa:To>...</wsa:To>
474
475     <wsa:Action>...</wsa:Action>
476
477     <!-- reference params from FaultTo/ReplyTo EndpointReference -->
478
479     <sb:Sender providerID="..." />
480
481     ...
482
483   </S:Header>
484
485   <S:Body>
486
487     <S:Fault>
488       <faultcode>S:server</faultcode>
489       <faultstring>Server Error</faultstring>
490       <!-- <S:faultactor> should be absent -->
491
492       <detail>
493         <sb:Status code="SomeStatus"
494           ref="Foo"
495           comment="Bar" />
496       </detail>
497     </S:Fault>
498
499   </S:Body>
500
501 </S:Envelope>
502

```

503 **Example 3. A SOAP-bound ID-* Fault Message**

504 **5. Messaging-specific Header Blocks**

505 This section profiles the use of WS-Addressing SOAP Binding [WSAv1.0-SOAP] and WS-Security [wss-sms] header
506 blocks to implement the ID-* message exchange model.

507 The messaging processing rules associated with the ID-* message exchange model are given in
508 [Section 5.11: Messaging Processing Rules](#).

509 Additional ID-* header blocks and their processing rules are defined below in [Section 6: Optional Header Blocks](#).

510 **Note**

511 Other ID-* specifications MAY define additional ID-* header blocks. [LibertyInteract] defines a header block, for
512 example.

513 **5.1. The <wsu:Timestamp> element in the <wsse:Security> Header** 514 **Block**

515 The <wsu:Timestamp> element and the <wsse:Security> header block are defined in [wss-sms]. When included
516 in a message, the <wsu:Timestamp> element provides a means for the sender to specify the time at which the message
517 was prepared for transmission and the time at which the message should expire.

518 **Note**

519 Depending on the security mechanisms in use [LibertySecMech], it may be necessary to include a <wsse:Security>
520 header block solely for the purpose of including the <wsu:Timestamp> element.

521 **5.2. The <wsa:MessageID> Header Block**

522 The <wsa:MessageID> header block is defined in [WSAv1.0-SOAP]. The value of this header block uniquely
523 identifies the message that contains it.

524 **5.2.1. <wsa:MessageID> Value Requirements**

525 Values of the <wsa:MessageID> header block MUST satisfy the following property:

526 Any party that assigns a value to a <wsa:MessageID> header block MUST ensure that there is
527 negligible probability that that party or any other party will accidentally assign the same identifier
528 to any other message.

529 The mechanism by which SOAP-based ID-* senders or receivers ensure that an identifier is unique is left to
530 implementations. In the case that a pseudorandom technique is employed, the probability of two randomly chosen
531 identifiers being identical MUST be less than 2^{-128} and SHOULD be less than 2^{-160} . The above requirement MAY
532 be met by applying Base64 [RFC2045] encoding to a randomly chosen value [RFC1750] 128 or 160 bits in length.

533 It is OPTIONAL for a <wsa:MessageID> value to be resolvable in principle to some resource. In the case that the
534 value is resolvable in principle (for example, it is in the form of a URI reference [RFC2396], it is OPTIONAL for the
535 identifier to be dereferenceable.

536 **5.3. The <wsa:RelatesTo> Header Block**

537 The <wsa:RelatesTo> header block is defined in [WSAv1.0-SOAP]. The value of this header block establishes a
538 relationship between the message that contains it and some other message. The type of relationship is specified in the
539 RelationshipType attribute.

540 **Note**

541 When the relationship is `http://www.w3.org/2005/03/addressing/reply`, the `RelationshipType` attribute
542 may be omitted.

543 **5.4. The <wsa:To> Header Block**

544 The `<wsa:To>` header block is defined in [WSAv1.0-SOAP]. The value of this header block specifies the intended
545 destination of the message.

546 **Note**

547 In the typical case that a WS-Addressing endpoint reference is used to address a message, the value of this header
548 block is taken from the `<wsa:Address>` of the endpoint reference. If the `<wsa:To>` header block is not present,
549 the value defaults to `http://www.w3.org/2005/03/addressing/role/anonymous`; so, when constructing a
550 message, the header block can be omitted if this is the value that would be used. This typically allows the `<wsa:To>`
551 header block to be omitted in responses during synchronous request-response message exchanges over HTTP.

552 **5.5. The <wsa:Action> Header Block**

553 The `<wsa:Action>` header block is defined in [WSAv1.0-SOAP]. The value of this header block uniquely identifies
554 the semantics implied by the message.

555 **Note**

556 The value of this header block SHOULD the same value as the SOAPAction HTTP header (see Section 4.2: The
557 SOAPAction HTTP Header).

558 **5.6. The <wsa:ReplyTo> Header Block**

559 The `<wsa:ReplyTo>` header block is defined in [WSAv1.0-SOAP]. The value of this header block, which is of the
560 WS-Addressing endpoint reference type, specifies the address to which a reply should be sent.

561 **Note**

562 If this header block is not present, then no reply will be sent. For synchronous request-response message exchanges
563 over HTTP, the `<wsa:Address>` value `http://www.w3.org/2005/03/addressing/role/anonymous` MAY be
564 used.

565 **5.7. The <wsa:FaultTo> Header Block**

566 The `<wsa:FaultTo>` header block is defined in [WSAv1.0-SOAP]. The value of this header block, which is of the
567 WS-Addressing endpoint reference type, specifies the address to which a fault should be sent, if one should arise in
568 the processing of the message. If not present, faults are sent to the address specified in the `<wsa:ReplyTo>` header
569 block (if present).

570 **5.8. The <Sender> Header Block**

571 This section defines the `<Sender>` header block. When included in a message, this header provides a means for
572 a sender to claim that it is a provider identified by a given `providerID` value. The sender may also claim that it is
573 a member of a given affiliation. Such claims are generally verifiable by receivers by looking up these values in the
574 sender's metadata [LibertyMetadata].

575 **Note**

576 The providerID claim MAY be used by the receiver as a hint to locate metadata for use in verifying the security of
577 the message (see [LibertyMetadata] and [LibertySecMech]). The mechanisms by which the receiver might locate or
578 establish trust in a provider's metadata are not covered here.

579 The receiver SHOULD ensure that the claims in the <Sender> header block are protected with adequate message
580 security to bind them to the message sender (see [LibertySecMech]).

581 The <Sender> header block defines the following attributes:

- 582 • providerID [Required] – The Provider ID of the sender.
- 583 • affiliationID [Optional] – The Affiliation ID of the sender, if any.
- 584 • id [Optional] – An attribute facilitating references to elements of this type. This attribute MUST be used when
585 the message is signed as described in [LibertySecMech], and the element instance is to be included as one of the
586 set of signed message components.
- 587 • S:mustUnderstand [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].
- 588 • S:actor [Optional] – The SOAP actor attribute [SOAPv1.1].

589 The schema fragment in Figure 3 defines the <Sender> header block.

```
590
591 <!-- sender header block -->
592
593
594 <xs:complexType name="SenderType">
595   <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
596   <xs:attribute name="affiliationID" type="xs:anyURI" use="optional"/>
597   <xs:attribute name="id" type="xs:ID" use="optional"/>
598   <xs:attribute ref="S:mustUnderstand" use="optional"/>
599   <xs:attribute ref="S:actor" use="optional"/>
600 </xs:complexType>
601
602 <xs:element name="Sender" type="SenderType"/>
603
604
605
606
```

607 **Figure 3. The <Sender> Header Block Schema**

```
608
609 <sb:Sender S:mustUnderstand="1"
610   id="A72139...381"
611   actor="http://schemas.../next"
612   providerID="http://example.com"
613   affiliationID="http://affiliation.com"/>
614
```

615 **Example 4. An instantiated <Sender> header block**

616 5.9. The <InvocationIdentity> Header Block

617 This section defines the <InvocationIdentity> header block. When included in a message, this header provides
618 a means for the sender to include an *identity token* (see [LibertySecMech]) that specifies an identity at the service that
619 is the invoker of the message. This identity may be different than the sender of the message.

620 Note

621 If no <InvocationIdentity> header block is present, then the invocation identity is typically obtained from the
622 security context of the message (see [LibertySecMech]).

623 The <InvocationIdentity> header block has a content model of any and defines the following attributes:

- 624 • id [Optional] – An attribute facilitating references to elements of this type. This attribute **MUST** be used when
625 the message is signed as described in [LibertySecMech], and the element instance is to be included as one of the
626 set of signed message components.
- 627 • S:mustUnderstand [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].
- 628 • S:actor [Optional] – The SOAP actor attribute [SOAPv1.1].
- 629 • wsa:IsReferenceParameter [Optional] – Indicates that the header block was included because it was included
630 as a reference parameter in the EPR used to construct the message [WSAv1.0-SOAP].

631 The schema fragment in Figure 4 defines the <InvocationIdentity> header block.

```
632  
633  
634 <!-- invocation identity header block -->  
635  
636 <xs:complexType name="InvocationIdentityType">  
637   <xs:sequence>  
638     <xs:any namespace="##any" processContents="lax" />  
639   </xs:sequence>  
640   <xs:attribute name="id" type="xs:ID" use="optional" />  
641   <xs:attribute ref="S:mustUnderstand" use="optional" />  
642   <xs:attribute ref="S:actor" use="optional" />  
643   <xs:attribute ref="wsa:IsReferenceParameter" use="optional" />  
644   <xs:anyAttribute namespace="##other" processContents="lax" />  
645 </xs:complexType>  
646  
647 <xs:element name="InvocationIdentity" type="InvocationIdentityType" />  
648  
649  
650  
651
```

652 **Figure 4. The <InvocationIdentity> Header Block Schema**

```
653  
654 <sb:InvocationIdentity S:mustUnderstand="1"  
655   id="A31739...293"  
656   actor="http://schemas.../next">  
657   ...  
658 </sb:InvocationIdentity>  
659  
660
```

661 **Example 5. An instantiated <InvocationIdentity> header block**

662 5.10. The <TargetIdentity> Header Block

663 This section defines the <TargetIdentity> header block. When included in a message, this header provides a
664 means for the sender to include an *identity token* (see [LibertySecMech]) that specifies an identity at the service that is
665 the target of the message. For example, to obtain profile attributes for a principal, a query message might be sent to a
666 profile service associated with the principal, including an identity token in the target identity header that specifies the
667 principal's identity at the profile service.

668 Note

669 If no <TargetIdentity> header block is present, then the invocation identity is typically used as the identity at the
670 service that is the target of the message.

671 The <TargetIdentity> header block has a content model of any and defines the following attributes:

- 672 • id [Optional] – An attribute facilitating references to elements of this type. This attribute MUST be used when
673 the message is signed as described in [LibertySecMech], and the element instance is to be included as one of the
674 set of signed message components.
- 675 • S:mustUnderstand [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].
- 676 • S:actor [Optional] – The SOAP actor attribute [SOAPv1.1].
- 677 • wsa:IsReferenceParameter [Optional] – Indicates that the header block was included because it was included
678 as a reference parameter in the EPR used to construct the message [WSAv1.0-SOAP].

679 The schema fragment in Figure 5 defines the The <TargetIdentity> header block.

```
680  
681  
682 <!-- target identity header block -->  
683  
684 <xs:complexType name="TargetIdentityType">  
685   <xs:sequence>  
686     <xs:any namespace="##any" processContents="lax"/>  
687   </xs:sequence>  
688   <xs:attribute name="id" type="xs:ID" use="optional"/>  
689   <xs:attribute ref="S:mustUnderstand" use="optional"/>  
690   <xs:attribute ref="S:actor" use="optional"/>  
691   <xs:attribute ref="wsa:IsReferenceParameter" use="optional"/>  
692   <xs:anyAttribute namespace="##other" processContents="lax"/>  
693 </xs:complexType>  
694  
695 <xs:element name="TargetIdentity" type="TargetIdentityType"/>  
696  
697  
698  
699
```

700 **Figure 5. The <TargetIdentity> Header Block Schema**

```
701  
702 <sb:TargetIdentity S:mustUnderstand="1"  
703   id="A31739...293"  
704   actor="http://schemas.../next">  
705   ...  
706 </sb:TargetIdentity>  
707
```

708 **Example 6. An instantiated <TargetIdentity> header block**

709 5.11. Messaging Processing Rules

710 Overall processing of SOAP-bound ID-* messages follows the rules of the SOAP processing model described in
711 [SOAPv1.1]; specifically, the SOAP mustUnderstand and actor attributes MAY be used to mandate header block
712 processing and target header blocks, respectively. Where applicable, specific processing rules for these attributes are
713 given in the overall processing rules defined below.

714 The system entity constructing and sending a SOAP-bound ID-* message is called the *sender* in the context of the act
715 of sending the message. The entity receiving this message is called the *receiver* in the context of the act of receiving
716 an individual message (see [Section 2.2: Terminology](#)).

717 Two *Message Exchange Patterns* (MEPs) are supported: one-way, and request-response. One-way is simply where a
718 sender sends a message to a receiver without necessarily expecting to receive an explicit response to the sent message.
719 Request-response is where a sender sends a message to a receiver and expects to receive an explicit response.

720 The processing rules are described below in terms of [Constructing and Sending a SOAP-bound ID-* Message](#) and
721 [Receiving and Processing a SOAP-bound ID-* Message](#) . A sender instigating a one-way message exchange will
722 perform only the steps outlined in the former section. A sender participating in a request-response message exchange
723 will perform the steps in the former section when sending a message, and the steps in the latter section when receiving
724 and processing the response. A receiver participating in a request-response exchange will do the reverse. Note that a
725 receiver of an asynchronous one-way message will perform the steps in the latter section.

726 **Note**

727 The label "ID-* header block(s)" is used to refer to at least one of, or all of, the following set of header blocks (the first
728 eight are defined in this specification, the remainder are defined in the cited specifications):

- 729 • <Sender>
- 730 • <InvocationIdentity>
- 731 • <TargetIdentity>
- 732 • <ProcessingContext>
- 733 • <Consent>
- 734 • <UsageDirective>
- 735 • <EndpointUpdate>
- 736 • <Timeout>
- 737 • <CredentialsContext>
- 738 • <wsa:MessageID> [[WSAv1.0](#)]
- 739 • <wsa:RelatesTo> [[WSAv1.0](#)]
- 740 • <wsa:To> [[WSAv1.0](#)]
- 741 • <wsa:Action> [[WSAv1.0](#)]
- 742 • <wsa:ReplyTo> [[WSAv1.0](#)]
- 743 • <wsa:FaultTo> [[WSAv1.0](#)]
- 744 • <wsse:Security> [[LibertySecMech](#)]
- 745 • <is:UserInteraction> [[LibertyInteract](#)]

746 Other specifications in the Liberty ID-* specification suite MAY define header block(s) not listed above. Nevertheless,
747 they should generally be considered a member of the above list when interpreting the processing rules in this section,
748 and explicitly considered where the processing rules refer to "ID-* header blocks" (see [Section 2.2: Terminology](#)).

749 **5.11.1. Constructing and Sending a SOAP-bound ID-* Message**

750 The sender MUST follow these processing rules when constructing and sending an outgoing SOAP-bound ID-*
751 message (hereafter referred to as the *outgoing message*):

- 752 1. The outgoing message MUST satisfy the rules given in [Section 4: SOAP Binding](#).
- 753 2. The outgoing message MUST satisfy the rules given in [[WSAv1.0-SOAP](#)].
- 754 3. The outgoing message MUST include exactly one `<wsa:MessageID>` header block in the `<S:Header>` child
755 element of the `<S:Envelope>` element and its value SHOULD be set according to the rules presented in
756 [Section 5.2.1: <wsa:MessageID> Value Requirements](#) .
- 757 4. If the sender is participating in a request-response MEP and is
 - 758 A. sending a request message, the outgoing message MUST include exactly one `<wsa:ReplyTo>` header block
759 and at most one `<wsa:FaultTo>` header block (if the `<wsa:FaultTo>` header block is not included, faults
760 will be delivered to the `<wsa:ReplyTo>` endpoint)
 - 761 B. responding to a prior-received request message, the outgoing message MUST include exactly one
762 `<wsa:RelatesTo>` header block with `RelationshipType` equal to `http://www.w3.org/2005/03/`
763 `addressing/reply` in the `<S:Header>` child element of the `<S:Envelope>` element (note that this is
764 the default `RelationshipType` and so the attribute MAY be omitted). The value of this header block MUST
765 be set to the value of the `<wsa:MessageID>` header block from the prior-received message.
- 766 5. The outgoing message MUST include exactly one `<wsse:Security>` header block. The `<wsse:Security>`
767 header block MUST include a `<wsu:Timestamp>` element. The `<wsu:Timestamp>` element MUST include a
768 `<wsu:Created>` element, the value of which SHOULD be set to the time at which the message is prepared for
769 transmission. This value MUST conform to the rules presented in [Section 2.5: Time Values](#).
770 If no clock is available to the message sender then a time value of `1970-01-01T00:00:00Z` SHOULD be used.
- 771 6. If the sender is acting in the role of a Liberty provider, the message MUST include exactly one `<Sender>` header
772 block in the `<S:Header>` child element of the `<S:Envelope>` element. The attributes of this `<Provider>`
773 header block MUST be set as follows:
 - 774 A. `providerID` MUST be present and SHOULD be set to a value appropriate for the sender to claim
775 [[LibertyMetadata](#)].
 - 776 B. `affiliationID` MAY be present. If so, it SHOULD be set to a value appropriate for the sender to claim
777 [[LibertyMetadata](#)].
- 778 7. The sender MAY include an `<InvocationIdentity>` header block, as needed, to identify the invocation
779 identity of the message. The sender MUST NOT include more than one `<InvocationIdentity>` header block.
- 780 8. The sender MAY include a `<TargetIdentity>` header block, as needed, to identify the target identity of the
781 message. The sender MUST NOT include more than one `<TargetIdentity>` header block.
- 782 9. The sender MAY include other ID-* header blocks in the message, in addition to those enumerated above,
783 as required by the overall messaging and processing context. For example, the sender may include a
784 `<wsse:Security>` header block [[LibertySecMech](#)].
- 785 10. The sender adds either:

- 786 A. an ordinary ID-* message (as described in [Section 4.3: Ordinary ID-* Messages](#); see [Example 2](#)), or,
787 B. an ID-* fault message (as **prescribed** in [Section 4.4: ID-* Fault Messages](#); see [Example 3](#)),
788 to the SOAP-bound ID-* message's <S:Body> element.

789 11. The sender also performs any needed additional preparation of the message, for example including other header
790 blocks, and signing some or all of the message elements, and then sends the message to the receiver. See
791 [Section 5.12: Examples](#) .

792 **5.11.2. Receiving and Processing a SOAP-bound ID-* Message**

793 The receiver of a SOAP-bound ID-* message, either ordinary or fault, **MUST** perform the following processing steps
794 on the ID-* header blocks of the incoming SOAP-bound ID-* message.

795 **Note**

796 Although the steps below are explicitly arranged and numbered sequentially, the intent is **not** to strictly define a specific
797 overall processing algorithm in terms of having implementations follow these steps in exactly the same sequence on a
798 per-header-block basis. However, all specified tests **MUST** be applied as appropriate to all ID-* header blocks in the
799 incoming SOAP-bound ID-* message.

800 1. Processing common to **all** received ID-* header blocks:

801 A. The `S:actor` attribute **MAY** be present. If present, its value **SHOULD** be
802 "`http://schemas.xmlsoap.org/soap/actor/next`" or some other previously agreed upon (out-of-
803 band) value.

804 B. The `S:mustUnderstand` attribute **MAY** be present. If present, its value **SHOULD** be `TRUE`.

805 C. If the foregoing tests ([1.A](#) and [1.B](#)) hold true, processing continues with step [2](#).

806 D. Otherwise, the receiver **MAY** respond to the sender with a SOAP-bound ID-* Fault message (per [Section 4.4:](#)
807 [ID-* Fault Messages](#)) with the <Status> element configured with:

808

- a `code` attribute with a value of:

809

- "`InvalidActor`" if the failed test is [1.A](#),

810

- "`InvalidMustUnderstand`" if the failed test is [1.B](#),

811

- and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

812 The <S:Fault> **SHOULD** contain a <S:faultcode> of `S:Client`.

813 The receiver **MAY** discard the incoming message. The receiver is finished processing this incoming message
814 at this point.

815 2. Processing specific to the <wsa:MessageID> and <wsa:RelatesTo> header blocks and the
816 <wsu:Timestamp> element in the <wsse:Security> header block:

817 A. A single <wsse:Security> header block **MUST** be present in the header of the message.
818 The <wsse:Security> header block **MUST** include a <wsu:Timestamp> element. The
819 <wsu:Timestamp> element **MUST** include a <wsu:Created> element.

820 B. The value of the <wsu:Created> element **SHOULD** be within an appropriate offset from local time
821 expressed in UTC. Absent other guidance, a value of 5 minutes **MAY** be used.

822 If the <wsu:Timestamp> element includes an <wsu:Expires> element, the time at the receiver **MUST**
823 be before that time.

- 824 **Note**
- 825 Certain classes of client devices, such as consumer electronics, often do not have correctly set clocks. These
826 processing rules may be relaxed for messages received from such devices.
- 827 C. A single `<wsa:MessageID>` header block **MUST** be present in the header of the message.
- 828 D. If the `<wsa:RelatesTo>` header block with `RelationshipType` equal to `http://www.w3.org/2005/03/`
829 `addressing/reply` is present, and if the receiver is participating in a request-response MEP with the
830 sending party, then the value of the `<wsa:RelatesTo>` header block **SHOULD** match the value of the
831 `<wsa:MessageID>` header block of a message previously sent by the receiver to the sender of the now
832 incoming message.
- 833 E. If the foregoing tests (2.A through 2.D) hold true, processing continues with step 3 .
- 834 F. Otherwise, the receiver **MAY** respond to the sender with a SOAP-bound ID-* Fault message (per Section 4.4)
835 with the `<Status>` element configured with:
- 836
 - a `code` attribute with a value of:
- 837
 - "IDStarMsgNotUnderstood" if the failed test is 2.A or 2.C.
- 838
 - "sb:StaleMsg" if the failed test is 2.B,
- 839
 - "sb:InvalidRefToMsgID" if the failed test is 2.D,
- 840
 - and a `ref` attribute with its value taken from the `messageID` value of the incoming message.
- 841 The `<S:Fault>` **SHOULD** contain a `<S:faultcode>` of `S:Client`.
- 842 The receiver **MAY** discard the incoming message. The receiver is finished processing this incoming message
843 at this point.

844 **Note**

845 This specification does not include specific processing rules designed to ensure reliable message delivery or
846 to prevent message replay. Services building on this specification should expect that clients may re-transmit
847 messages for which no reply has been received.

848 3. At this point, the receiver of the message MAY cease processing the message, and indicate to the sender that the
849 message should be re-submitted to a different endpoint, according to the rules specified in [Section 6.4.5.1](#)

850 4. Processing specific to the <Sender> header block:

851 A. Verify that any declared `providerID` or `affiliationID`, are valid. The receiver SHOULD perform this
852 verification and validation against metadata (see [\[LibertyMetadata\]](#)).

853 The declared `providerID` and `affiliationID` MUST NOT be used to establish a security context for further
854 processing of the message on their own, but must be validated by an adequate security mechanism as
855 specified in [\[LibertySecMech\]](#).

856 B. If the foregoing test (4.A) holds true, processing continues with step 6.

857 C. Otherwise, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per [Section 4.4](#))
858 with the <Status> element configured with:

859 • a `code` attribute with a value of:

860 • "ProviderIDNotValid", or,

861 • "AffiliationIDNotValid", as appropriate (if both the claimed Provider ID and the Affiliation
862 ID
863 are deemed invalid, then the returned code SHOULD be "AffiliationIDNotValid"),

864 • and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

865 The <S:Fault> SHOULD contain a <S:faultcode> of S:Client.

866 The receiver MAY discard the incoming message. The receiver is finished processing this incoming message
867 at this point.

868 5. Processing specific to the <InvocationIdentity> header block:

869 A. Verify that any provided invocation identity token is valid (see [\[LibertySecMech\]](#)) and, if appropriate, that
870 the identity specified by the token is known.

871 B. If the foregoing test (5.A) holds true, processing continues with step 6.

872 C. Otherwise, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per [Section 4.4](#))
873 with the <Status> element configured with:

874 • a `code` attribute with a value of:

875 • "InvocationIdentityNotValid"

876 • and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

877 The <S:Fault> SHOULD contain a <S:faultcode> of S:Client.

878 The receiver MAY discard the incoming message. The receiver is finished processing this incoming message
879 at this point.

- 880 6. Processing specific to the <TargetIdentity> header block:
- 881 A. Verify that any provided target identity token is valid (see [LibertySecMech]) and, if appropriate, that the
882 identity specified by the token is known.
- 883 B. If the foregoing test (6.A) holds true, processing continues with step 7.
- 884 C. Otherwise, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per Section 4.4)
885 with the <Status> element configured with:
- 886 • a code attribute with a value of:
- 887 • "TargetIdentityNotValid"
- 888 • and a ref attribute with its value taken from the messageID value of the incoming message.
- 889 The <S:Fault> SHOULD contain a <S:faultcode> of S:Client.
- 890 The receiver MAY discard the incoming message. The receiver is finished processing this incoming message
891 at this point.
- 892 7. All remaining ID-* header blocks SHOULD be processed at this point. See appropriate sections in this and other
893 specifications for the processing rules associated with these header blocks and the manner of reporting any issues
894 with this processing. If there are no issues with these header blocks, then processing continues with step 8 below,
895 otherwise the receiver is finished processing this incoming message at this point.
- 896 **Note**
- 897 It should be noted that the receiver MAY return an sb:InappropriateCredentials based on their process-
898 ing of the <wsse:Security> header block, under conditions specified below in Section 6.4 and Section 6.3, in
899 addition to other conditions such as an expired credential (see [LibertySecMech]).
- 900 8. If the incoming message's applicable header blocks have passed all specified and applicable tests, the incoming
901 message SHOULD be dispatched for further processing as appropriate.
- 902 If the message contained in the encompassing SOAP message's <S:Body> element is not dispatchable, the
903 receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per Section 4.4) with the <Status>
904 element configured with:
- 905 • a code attribute with a value of:
- 906 • "IDStarMsgNotUnderstood"
- 907 • and a ref attribute with its value taken from the messageID value of the incoming message.
- 908 Receivers MUST be able to avoid ID-* fault message "loops". For example, if the incoming message is conveying
909 an ID-* fault message, and there is some issue with one or more of its ID-* header blocks, the receiver should not
910 issue a SOAP-bound ID-* Fault message in response.

911 **Note**

912 Other specifications conforming to this binding that specify ordinary ID-* messages and their processing, such as
 913 [LibertyIDPP] or [LibertyDisco], MAY define <Status> element code attribute values in addition to the ones defined
 914 in Section 3.3.1 of this document. These code attribute values SHOULD be used to signal to the sender any issues
 915 with the incoming ordinary ID-* message found by the receiver. This specification does not define any such conditions
 916 other than the one described above in 8, and they are not further discussed in this document.

917 **5.12. Examples**

918 **Example 7** illustrates a SOAP-bound ID-* message conveying a Personal Profile (ID-PP) Modify request message
 919 [LibertyIDPP].

```

920
921     <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
922       xmlns:sb="urn:liberty:sb:2005-11"
923       xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
924
925     <S:Header>
926
927       ...
928
929     <wsse:Security>
930       <wsu:Timestamp>
931         <wsu:Created>2005-06-17T04:49:17Z</wsu:Created>
932       </wsu:Timestamp>
933     </wsse:Security>
934
935     <wsa:MessageID>uuid:efefefef-aaaa-ffff-cccc-eeeeffffbbbbb</wsa:MessageID>
936
937     <wsa:To>http://spwsp.com/idpp</wsa:To>
938
939     <wsa:Action>urn:liberty:id-sis-pp:2003-08:Modify</wsa:Action>
940
941     <!-- reference params from target EndpointReference -->
942
943     <sb:Sender providerID="http://spwsc.com" affiliationID="http://affiliation.com"/>
944
945     <wsa:ReplyTo>
946       <wsa:Address>http://www.w3.org/2005/03/addressing/role/anonymous</wsa:Address>
947     </wsa:ReplyTo>
948
949     ...
950
951   </S:Header>
952
953   <S:Body>
954
955     <idpp:Modify>
956       : <!-- this is an ID-PP Modify message -->
957     </idpp:Modify>
958
959   </S:Body>
960
961 </S:Envelope>
962
  
```

963 **Example 7. A SOAP-bound ID-* Request Message**

964 **Example 8** illustrates a SOAP-bound ID-* response to the message in the previous example, which conveyed an ID-PP
 965 Modify message. Note how the <wsa:RelatesTo> header value references the <wsa:MessageID> in the example
 966 above.

```
967
968 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
969   xmlns:sb="urn:liberty:sb:2005-11"
970   xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
971
972   <S:Header>
973
974     ...
975
976     <wsse:Security>
977       <wsu:Timestamp>
978         <wsu:Created>2005-06-17T04:49:18Z</wsu:Created>
979       </wsu:Timestamp>
980     </wsse:Security>
981
982     <wsa:MessageID>uuid:aaaaeeee-efef-fefe-efef-abababababab</wsa:MessageID>
983     <wsa:RelatesTo>uuid:efefefef-aaaa-ffff-cccc-eeeeffffbba</wsa:RelatesTo>
984
985     <wsa:Action>urn:liberty:id-sis-pp:2003-08:ModifyResponse</wsa:Action>
986
987     <sb:Sender providerID="http://spwsp.com" />
988
989     ...
990
991   </S:Header>
992
993   <S:Body>
994
995     <idpp:ModifyResponse>
996       :      <!-- this is an ID-PP ModifyResponse message -->
997     </idpp:ModifyResponse>
998
999   </S:Body>
1000 </S:Envelope>
1001
1002
```

1003

Example 8. A SOAP-bound ID-* Response Message

1004 **6. Optional Header Blocks**

1005 The optional header blocks described in this specification are:

- 1006 • <ProcessingContext>
- 1007 • <Consent>
- 1008 • <CredentialsContext>
- 1009 • <EndpointUpdate>
- 1010 • <Timeout>
- 1011 • <UsageDirective>
- 1012 • <ApplicationEPR>

1013 The following sections describe these optional ID-* header blocks along with their specific processing rules.

1014 **Note**

1015 Whenever an optional header block appears in a SOAP-bound ID-* message, the processing rules specific to that
1016 header block (which are given in this section, below) **MUST** be used in combination with the messaging processing
1017 rules given above in [Section 5.11: Messaging Processing Rules](#). This applies whether the message is being constructed
1018 and sent, or being received and processed.

1019 **6.1. The <ProcessingContext> Header Block**

1020 This section defines the <ProcessingContext> header block. This header block may be employed by a sender to
1021 signal to a receiver that the latter should add a specific additional facet to the overall *processing context* in which
1022 any action(s) are invoked as a result of processing any ID-* message also conveyed in the overall SOAP-bound ID-*
1023 message. The full semantics of this header block are described below in [Section 6.1.3: <ProcessingContext>](#)
1024 Header Block Semantics and Processing Rules .

1025 Processing context facets are denoted by URIs. URIs are assigned to denote specific processing context facets. This
1026 specification defines several such URIs below in [Section 6.1.3.2](#).

1027 **6.1.1. The ProcessingContextType Header Block Type**

1028 The ProcessingContextType content model is anyURI. It defines the following attributes:

- 1029 • id [Optional] – identifies a <ProcessingContext> header block instance. This attribute **MUST** be used when
1030 the message is signed as described in [[LibertySecMech](#)], and the element instance is to be included as one of the
1031 set of signed message components.
- 1032 • S:mustUnderstand [Optional] – The SOAP mustUnderstand attribute [[SOAPv1.1](#)].
- 1033 • S:actor [Optional] – The SOAP actor attribute [[SOAPv1.1](#)].

1034 The following schema fragment defines the ProcessingContext header block type:

```

1035
1036     <xs:complexType name="ProcessingContextType">
1037     <xs:simpleContent>
1038     <xs:extension base="xs:anyURI">
1039     <xs:attribute name="id" type="xs:ID" use="optional"/>
1040     <xs:attribute ref="S:mustUnderstand" use="optional"/>
1041     <xs:attribute ref="S:actor" use="optional"/>
1042     </xs:extension>
1043     </xs:simpleContent>
1044     </xs:complexType>
1045

```

1046 **Figure 6. The ProcessingContext Header Block Type Schema**

1047 6.1.2. <ProcessingContext> Header Block Element

1048 The <ProcessingContext> schema element is given in [Figure 7](#).

```

1049
1050     <xs:element name="ProcessingContext" type="ProcessingContextType"/>
1051

```

1052 **Figure 7. The <ProcessingContext> Element Schema**

```

1053
1054 <S:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
1055           xmlns:sb="..."
1056           xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
1057
1058   <S:Header>
1059     ...
1060
1061     <sb:ProcessingContext S:mustUnderstand="1">
1062       urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
1063     </sb:ProcessingContext>
1064
1065     ...
1066
1067   </S:Header>
1068
1069   <S:Body>
1070
1071     <idpp:Modify> <!-- This is an ID-PP "Modify" message bound -->
1072       :      <!-- into the <Body> of a SOAP message.      -->
1073     </idpp:Modify>
1074
1075   </S:Body>
1076
1077 </S:Envelope>
1078
1079

```

1080 **Example 9. A SOAP-bound ID-* Message with an Attached <ProcessingContext> Header Block**

1081 6.1.3. <ProcessingContext> Header Block Semantics and Processing Rules

1082 This section first describes the overall semantics of the <ProcessingContext> header block, then defines two
 1083 *processing context facet URIs*, and concludes with defining specific processing rules.

1084 **6.1.3.1. <ProcessingContext> Header Block Semantics**

1085 The overall semantic of the <ProcessingContext> header block is:

1086 The <ProcessingContext> header block MAY be employed by a sender, who is acting in a web
1087 services client (WSC) role, to signal to a receiver, who is acting in a web services provider (WSP)
1088 role, that the latter should add a specific *processing context facet* to the overall *processing context*
1089 (see [Section 2.2: Terminology](#)) in which the *service request* is evaluated.

1090 The specific processing context facet being conveyed by the <ProcessingContext> header block is identified by
1091 the header block's URI element value.

1092 Such URIs are known as *processing context facet URIs*. An example of a processing context facet that may be signaled
1093 by such a URI is whether the principal should be considered to be online or not.

1094 An ID-WSF or ID-SIS WSP receiving a service request containing a <ProcessingContext> header block with
1095 one of the above processing context facet URIs SHOULD process the conveyed ID-* message **with the indicated**
1096 **processing context facet in force**. Thus the ID-* message's processing as well as any applicable access management
1097 policies are exercised within an overall processing context which includes the processing context facet. Finally, an
1098 indication of success or failure of the ID-* message processing is returned to the sender, in the same manner as would
1099 be done if the ID-* message had been sent without the attendant <ProcessingContext> header block.

1100 The above completely describes the semantic of this header block itself, and further description of particular effects
1101 on processing must be made in descriptions of processing context facet URIs. Such a description is given in the next
1102 section.

1103 **Note**

1104 Whether or not a receiver honors a <ProcessingContext> header block is a matter of local policy at the
1105 receiver, as is whether or not a receiver honors any given request from any given sender. For example, the
1106 <ProcessingContext> header block functionality has security implications in the sense of possibly facilitating
1107 an adversary to probe a receiver's behavior given adversary-chosen inputs. For these reasons, whether or not the
1108 <ProcessingContext> header block functionality is enabled on the part of a receiver with respect to a particular
1109 sender should be a matter of business-level agreement between the receiver and the sender.

1110 **6.1.3.2. Processing Context Facet URIs: PrincipalOnline, PrincipalOffline, and Simulate**

1111 Three processing context facet URIs are defined below for use with the <ProcessingContext> header block:

1112 urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
1113 Conduct the processing of the ID-* message as if the Principal is offline.

1114 urn:liberty:sb:2003-08:ProcessingContext:PrincipalOnline
1115 Conduct the processing of the ID-* message as if the Principal is online.

1116 urn:liberty:sb:2003-08:ProcessingContext:Simulate
1117 *Simulate* the processing of the ID-* message.

1118 If the sender includes a <UserInteraction> header block in addition to the <ProcessingContext> header block
1119 in the SOAP-bound ID-* request message, the receiver and sender MUST appropriately initiate the indicated user
1120 interaction (see [\[LibertyInteract\]](#)), and incorporate information supplied by the user as a part of the resultant user
1121 interaction, into the appropriate data and/or policy stores.

1122 **Note**

1123 Any processing context facet that was conveyed in the <ProcessingContext> header block MUST NOT be enforced
1124 during such a user interaction. Rather, it applies only to the processing of the ID-* message itself.

1125 In summary, the overall intended side-effect of using the above-defined processing context facets is for the receiver to
1126 evaluate applicable policy, and return a putative indication of success or failure to the sender. This provides WSCs the
1127 capability to make an ID-WSF or ID-SIS service request and ascertain whether it will be successful or not—without
1128 the service request actually being carried out. Additionally, it facilitates carrying out any user interaction that may be
1129 indicated by the current combination of service request context and applicable policy. This will, for example, facilitate
1130 some WSCs to fashion more "user friendly" experiences.

1131 **6.1.3.3. Defining New Processing Context Facet URIs**

1132 The rightmost portions of the processing context facet URIs after the "ProcessingContext:" component are referred
1133 to as *processing context facet identifiers*. For example, whether the Principal is online or not is a facet of a request
1134 context. New processing context facet identifiers MAY be defined in other specifications, for example in ID-SIS data
1135 service specifications. An ID-SIS data service may define as many levels of request context identifiers as necessary
1136 to address the application's needs.

1137 **6.1.3.4. Sender Processing Rules**

1138 A sender MAY include a <ProcessingContext> header block in a SOAP-bound ID-* message. The sender MUST
1139 include a processing context facet URI in the <ProcessingContext> header block. The sender then sends the ID-*
1140 SOAP-bound message to an ID-WSF or ID-SIS service-hosting node (AKA the receiver).

1141 A sender MAY indicate that it believes either that the Principal is currently "online" or "offline" when it sends a
1142 message by specifying one of the two processing context facet URIs:

- 1143 • urn:liberty:sb:2003-08:ProcessingContext:PrincipalOnline
- 1144 • urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline

1145 The sender will typically receive a response from the receiver indicating success or failure or will receive a SOAP
1146 fault indicating a processing error with the SOAP-bound ID-* message. Note that in the case of a "successful" request
1147 *simulation*, the service will not return any result data other than an indication of success or failure to the sender.

1148 **6.1.3.5. Receiver Processing Rules**

1149 The receiver of a request containing a <ProcessingContext> header block MUST examine the included processing
1150 context facet URI. If it is known to the data service, then the data service MUST attempt to process the data service
1151 request, represented by the ID-* message, in an overall processing context including the processing context facet as
1152 indicated by the conveyed processing context facet URI, and return an indication of success or failure to the sender.

1153 If the data service request is malformed or has some other issue that would normally cause the receiver to issue a
1154 SOAP fault, the receiver SHOULD do so.

1155 If the receiver is asked to simulate processing of the request (by the inclusion of the
1156 urn:liberty:sb:2003-08:ProcessingContext:Simulate facet URI), and they are both able and willing to
1157 honor that processing context, then the receiver MUST evaluate the conveyed ID-* message, but MUST NOT actually
1158 perform the operation. That is, the receiver MUST NOT make actual changes to underlying ID-* service datastore,
1159 and it MUST NOT return any data as a result of evaluating the ID-* message.

1160 If the sender includes a <UserInteraction> header block, in addition to the <ProcessingContext> header block,
1161 then both participants MUST initiate the indicated user interaction (see [LibertyInteract] appropriately, and incorporate
1162 information supplied by the user as a part of the interaction into appropriate data and/or policy stores, even if

1163 the `urn:liberty:sb:2003-08:ProcessingContext:Simulate` URI is specified in a `<ProcessingContext>`
1164 header.

1165 In the event the receiver does not understand the included processing context facet URI, the receiver MAY respond
1166 with a SOAP-bound ID-* fault message (per [Section 4.4: ID-* Fault Messages](#)) with the `<Status>` element configured
1167 with:

1168 • a `code` attribute with a value of:

1169 • `"ProcCtxURINotUnderstood"`

1170 • and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

1171 In the event the receiver is not willing to enforce a stipulated processing context, the receiver MAY respond with a
1172 SOAP-bound ID-* fault message (per [Section 4.4: ID-* Fault Messages](#)) with the `<Status>` element configured with:

1173 • a `code` attribute with a value of:

1174 • `"ProcCtxUnwilling"`

1175 • and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

1176 **Note**

1177 The receiver MAY reference multiple `<ProcessingContext>` headers in the `<detail>` of the fault response (in
1178 accordance with the rules specified in [Section 4.4](#)).

1179 **6.2. The `<Consent>` Header Block**

1180 This section defines the `<Consent>` header block. This header block is used to explicitly claim that the Principal
1181 consented to the present interaction.

1182 **6.2.1. The `consentType` Header Block Type**

1183 The `<Consent>` header block element MAY be employed by either a sender or a receiver. For example, the Principal
1184 may be using a Liberty-enabled client or proxy (common in the wireless world), and in that sort of environment the
1185 mobile operator may cause the Principal's terminal (AKA: cell phone) to prompt the principal for consent for some
1186 interaction.

1187 The `consentType` header block type has the following attributes:

1188 • `uri` [Required] – A URI indicating that the Principal's consent was obtained.

1189 Optionally, the URI MAY identify a particular *Consent Agreement Statement* defining the specific nature of the
1190 consent obtained.

1191 This specification defines one well-known URI Liberty implementors and deployers MAY use to indicate positive
1192 Principal consent was obtained with respect to whatever ID-* interaction is underway or being initiated. This URI
1193 is known as the "Principal Consent Obtained" URI (PCO). The value of this URI is:

1194 `urn:liberty:consent:obtained`

1195 This URI does not correspond to any particular Consent Agreement Statement. Rather, it simply states that consent
1196 was obtained. The full meaning and implication of this will need to be derived from the execution context.

1197 • `timestamp` [Optional] – For denoting the time at which the sender obtained Principal consent with the POC.

1198 • id [Optional] – identifies a <Consent> header block instance. This attribute MUST be used when the message is
1199 signed as described in [LibSecMech], and the element instance is to be included in the signed message components.

1200 • S:mustUnderstand [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].

1201 • S:actor [Optional] – The SOAP actor attribute [SOAPv1.1].

1202 The schema fragment in Figure 8 defines the Consent header block type.

```
1203
1204 <xs:complexType name="consentType">
1205   <xs:attribute name="uri" type="xs:anyURI" use="required"/>
1206   <xs:attribute name="timestamp" type="xs:dateTime" use="optional"/>
1207   <xs:attribute name="id" type="xs:ID" use="optional"/>
1208   <xs:attribute ref="S:mustUnderstand" use="optional"/>
1209   <xs:attribute ref="S:actor" use="optional"/>
1210 </xs:complexType>
1211
```

1212 **Figure 8. The Consent Header Block Type Schema**

1213 6.2.2. <Consent> Header Block Element

1214 The schema fragment in Figure 9 defines the <Consent> element:

```
1215
1216 <xs:element name="Consent" type="consentType"/>
1217
```

1218 **Figure 9. The <Consent> Element Schema**

```
1219
1220 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1221   xmlns:sb="..."
1222   xmlns:idpp="urn:liberty:id-sis-pp:2003-08" >
1223
1224   <S:Header>
1225
1226     ...
1227
1228     <sb:Consent id="A124395732495743"
1229       uri="urn:liberty:consent:obtained"
1230       timestamp="2112-03-15T11:12:10Z"/>
1231
1232     ...
1233
1234   </S:Header>
1235
1236   <S:Body>
1237
1238     <idpp:Modify> <!-- This is an ID-PP "Modify" message bound -->
1239       : <!-- into the <Body> of a SOAP message. -->
1240     </idpp:Modify>
1241
1242   </S:Body>
1243
1244 </S:Envelope>
1245
```

1246 **Example 10. A SOAP-bound ID-* Message with an Attached <Consent> Header Block**

1247 **6.3. The <CredentialsContext> Header Block**

1248 **6.3.1. Overview**

1249 It may be necessary for an entity receiving an ID-* message to indicate the type of credentials that should be used by
1250 the sender in submitting a message.

1251 **6.3.2. CredentialsContext Type and Element**

1252 Receivers of an ID-* message MAY add <CredentialsContext> elements to the SOAP header of their response.

1253 The element is based upon the `CredentialsContextType` which is defined as:

- 1254 • `samlp2:RequestedAuthnContext` [Optional] – a container that allows the expression of a requested authenti-
1255 cation context (see [[SAMLCore2](#)]).
- 1256 • `SecurityMechID` [Optional] – A set of elements that specify ID-WSF security mechanism URIs (see [[Liberty-](#)
1257 [SecMech](#)]).
- 1258 • `id` [Optional] – An attribute facilitating references to elements of this type. This attribute MUST be used when
1259 the message is signed as described in [[LibertySecMech](#)], and the element instance is to be included as one of the
1260 set of signed message components.
- 1261 • `S:mustUnderstand` [Optional] – The SOAP `mustUnderstand` attribute [[SOAPv1.1](#)].
- 1262 • `S:actor` [Optional] – The SOAP `actor` attribute [[SOAPv1.1](#)].

1263 The following schema fragment describes the <CredentialsContext> header block.

```
1264  
1265  
1266 <!-- credentials context header block -->  
1267  
1268 <xs:complexType name="CredentialsContextType">  
1269   <xs:sequence>  
1270     <xs:element ref="samlp2:RequestedAuthnContext" minOccurs="0" />  
1271     <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded" />  
1272   </xs:sequence>  
1273   <xs:attribute name="id" type="xs:ID" use="optional" />  
1274   <xs:attribute ref="S:mustUnderstand" use="optional" />  
1275   <xs:attribute ref="S:actor" use="optional" />  
1276 </xs:complexType>  
1277  
1278 <xs:element name="CredentialsContext" type="CredentialsContextType" />  
1279  
1280  
1281
```

1282 **Figure 10. The <CredentialsContext> Header Block Element and Type Schema**

1283 **6.3.3. CredentialsContext Example**

```

1284
1285 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1286     xmlns:sb="urn:liberty:sb:2005-11"
1287     xmlns:lib="urn:liberty:iff:2003-08">
1288
1289 <S:Header>
1290
1291     ...
1292
1293 <!-- Says that the sender would like credentials that include RequestAuthnContext
1294     as specified -->
1295
1296 <sb:CredentialsContext mustUnderstand="1">
1297
1298     <sampl2:RequestedAuthnContext>
1299         ...
1300     </sampl2:RequestedAuthnContext>
1301
1302 </sb:CredentialsContext>
1303
1304     ...
1305
1306 </S:Header>
1307
1308 <S:Body>
1309
1310 <!-- a fault in the body indicates that the WSP's policy requires
1311 different (perhaps "stronger") credentials than were originally
1312 provided in the request -->
1313
1314 <S:Fault>
1315     <faultcode>S:Server</faultcode>
1316     <faultstring>
1317         Your request contained inappropriate credentials.
1318     </faultstring>
1319     <detail>
1320         <sb:Status code="InappropriateCredentials"/>
1321         <wsse:Security id="a6352...564"/>
1322     </detail>
1323 </S:Fault>
1324 </S:Body>
1325
1326 </S:Envelope>
1327
    
```

1328 **Example 11. A CredentialsContext Header Offered in Response to a Request with Inappropriate Credentials.**

1329 **6.3.4. Processing Rules**

1330 **6.3.4.1. Sender Processing Rules**

1331 A sender including this header **MUST** specify at least one RequestAuthnContext or one SecurityMechID.

1332 The SecurityMechID elements **SHOULD** be listed in order of preference by the sender.

1333 **6.3.4.2. Receiver Processing Rules**

1334 The receiver of a <CredentialsContext> header containing one or more SecurityMechID elements **SHOULD**
 1335 use the highest-listed (first) SecurityMechID that it supports in future requests to the sender of this header.

1336 The receiver of a <CredentialsContext> header containing a RequestAuthnContext element **SHOULD** use
 1337 credentials that conform to the policies specified therein in any future requests to the sender of this header (where
 1338 credentials are required).

1339 **6.4. The <EndpointUpdate> Header Block**

1340 **6.4.1. Overview**

1341 It may be necessary for an entity receiving an ID-* message to indicate that messages from the sender should be
1342 directed to a different endpoint, or that they wish a different credential to be used than was originally specified
1343 by the entity for access to the requested resource. The <EndpointUpdate> allows a message receiver to indicate
1344 that a new endpoint or new credentials should be employed by the sender of the message on any subsequent
1345 messages. This header block may be used in conjunction with the <sb:InappropriateCredentials> and
1346 <sb:EndpointMoved> faults, to indicate that the current message processing failed for those reasons, and should
1347 be submitted with the changes noted in any accompanying <EndpointUpdate> header block.

1348 **Note**

1349 The use of this header block allows the sender of the message to convey updates to security tokens, essentially
1350 providing a token renewal mechanism. This is not discussed further in this specification.

1351 **6.4.2. EndpointUpdate Type and Element**

1352 Receivers of an ID-* message may add an <EndpointUpdate> element to the SOAP header of their response.

1353 This element is based upon the EndpointUpdateType which is an extension of wsa:EndpointReferenceType
1354 that adds the following attributes:

1355 • updateType [Optional] – A URI attribute indicating whether the update should be interpreted as completely
1356 superseding the endpoint reference used to send the current request (the default) or whether it should be interpreted
1357 as a partial update.

1358 urn:liberty:sb:2005-11:EndpointUpdate:Complete
1359 A complete update.

1360 urn:liberty:sb:2005-11:EndpointUpdate:Partial
1361 A partial update. The complete updated endpoint reference is constructed according to the processing rules below.

1362 • id [Optional] – An attribute facilitating references to elements of this type. This attribute MUST be used when
1363 the message is signed as described in [LibertySecMech], and the element instance is to be included as one of the
1364 set of signed message components.

1365 • S:mustUnderstand [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].

1366 • S:actor [Optional] – The SOAP actor attribute [SOAPv1.1].

1367 The following schema fragment describes the <EndpointUpdate> header block.

```
1368
1369
1370 <!-- epr update header block -->
1371
1372 <xs:complexType name="EndpointUpdateType">
1373   <xs:complexContent>
1374     <xs:extension base="wsa:EndpointReferenceType">
1375       <xs:attribute name="updateType" type="xs:anyURI" use="optional"/>
1376       <xs:attribute name="id" type="xs:ID" use="optional"/>
1377       <xs:attribute ref="S:mustUnderstand" use="optional"/>
1378       <xs:attribute ref="S:actor" use="optional"/>
1379     </xs:extension>
1380   </xs:complexContent>
1381 </xs:complexType>
1382
1383 <xs:element name="EndpointUpdate" type="EndpointUpdateType"/>
1384
1385
1386
```

1387 **Figure 11. The <EndpointUpdate> Header Block Element and Type Schema**

1388 6.4.3. EndpointUpdate Examples


```
1389
1390
1391     1. Service responds to a request, indicating a new security mechanism and credential
1392
1393 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1394     xmlns:sb="urn:liberty:sb:2005-11"
1395     xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
1396
1397   <S:Header>
1398
1399     ...
1400
1401   <sb:EndpointUpdate mustUnderstand="1" updateType="urn:liberty:sb:2004-04:Partial">
1402     <wsa:Address>urn:liberty:sb:2005-11:EndpointUpdate:NoChange</wsa:Address>
1403     <wsa:Metadata>
1404       <ds:SecurityContext>
1405         <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</ds:SecurityMechID>
1406         <wsse:SecurityTokenReference>
1407           <wsse:Embedded>
1408             <wsse:BinarySecurityToken xmlns:wsse="..." wsu:Id="..."
1409               ValueType="anyNSprefix:ServiceSessionContext">
1410               ZjgzOWZlNzgyZTk1ZWU3OWEyMTRlODVmNGZkYzE4MmQ2ZDNhMzc3Nwo=
1411             </wsse:BinarySecurityToken>
1412           </wsse:Embedded>
1413         </wsse:SecurityTokenReference>
1414       </ds:SecurityContext>
1415     </wsa:Metadata>
1416   </sb:EndpointUpdate>
1417
1418     ...
1419   </S:Header>
1420
1421   <S:Body>
1422
1423     <idpp:QueryResponse>
1424       ...
1425     </idpp:QueryResponse>
1426
1427   </S:Body>
1428
1429 </S:Envelope>
1430
```

```
1431
1432
1433     2. The client sends a new request, using the contents of the EndpointUpdate
1434
1435 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1436     xmlns:sb="urn:liberty:sb:2005-11"
1437     xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
1438
1439   <S:Header>
1440
1441     ...
1442
1443   <wsse:Security xmlns:wsse="...">
1444
1445     <wsse:BinarySecurityToken xmlns:wsse="..." wsu:Id="bst"
1446       ValueType="anyNSprefix:ServiceSessionContext">
1447       ZjgzOWZlNzgyZTk1ZWU3OWEyMTRlODVmNGZkYzE4MmQ2ZDZhMzc3Nwo=
1448     </wsse:BinarySecurityToken>
1449
1450   </wsse:Security>
1451
1452     ...
1453
1454 </S:Header>
1455
1456 <S:Body>
1457
1458   <idpp:Query>
1459     ...
1460   </idpp:Query>
1461
1462 </S:Body>
1463
1464 </S:Envelope>
1465
```

1466 **Example 12. An EndpointUpdate Specifying a ServiceSessionContext Token, and the TLS Bearer Security Mechanism.**

```
1467
1468 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1469           xmlns:sb="urn:liberty:sb:2005-11">
1470
1471   <S:Header>
1472
1473     ...
1474
1475     <sb:EndpointUpdate mustUnderstand="1" updateType="urn:liberty:sb:2004-04:Partial">
1476       <wsa:Address>http://example.com:443/s oap</wsa:Address>
1477     </sb:EndpointUpdate>
1478
1479     ...
1480
1481   </S:Header>
1482
1483   <S:Body>
1484
1485     <!-- a fault in the body indicates that the request corresponding
1486     to this response should be re-submitted to the endpoint -->
1487
1488     <S:Fault>
1489
1490       <faultcode>S:Server</faultcode>
1491
1492       <faultstring>
1493         You must resubmit this request to the new endpoint.
1494       </faultstring>
1495
1496       <detail>
1497         <sb:Status code="EndpointMoved"/>
1498       </detail>
1499
1500     </S:Fault>
1501
1502   </S:Body>
1503
1504 </S:Envelope>
1505
```

1506 **Example 13. An EndpointUpdate Specifying an Updated Address.**

1507 **6.4.4. Processing Rules for the EndpointUpdate header**

1508 **6.4.4.1. Sender Processing Rules**

1509 The receiver of an ID-* message MAY add an <EndpointUpdate> header block to their response.

1510 If updateType is not present or has the value urn:liberty:sb:2005-11:EndpointUpdate:Complete, the
1511 <wsa:EndpointUpdate> MUST be a completely specified endpoint reference.

1512 If updateType has the value urn:liberty:sb:2005-11:EndpointUpdate:Partial, the <wsa:EndpointUpdate>
1513 MAY omit any direct children of <wsa:ReferenceParameters> or <wsa:Metadata> that have not changed from
1514 the original endpoint reference used to send the current request. Similarly, any extension elements that have not
1515 changed MAY be omitted. If the address has not changed, then the URI
1516 urn:liberty:sb:2005-11:EndpointUpdate:NoChange

1517 MAY be used in the <wsa:Address> value to indicate that the original address should continue to be used.

1518 **Note**

1519 The expressiveness of partial updates is limited. In particular, updates to `<wsa:ReferenceParameters>` and
1520 `<wsa:Metadata>` are done based on the qualified names of the direct children of those containers. If any child
1521 with a matching name is provided in the update, then all children with that name in the original are replaced. It is also
1522 impossible, with a partial update, to remove an element; elements may only be added or replaced.

1523 **6.4.4.2. Receiver Processing Rules**

1524 The receiver of an `<EndpointUpdate>` header SHOULD use the specified endpoint reference values to address any
1525 future requests to the sender of the header (where the endpoint reference used to address the request that resulted in
1526 the response containing the header would have been used), until newer information is obtained through this or some
1527 other mechanism or the updated information expires. If the updated information has a shorter lifetime than the current
1528 information (that it updates), then the current information SHOULD be retained as a fallback for when the updated
1529 information expires.

1530 If `updateType` is not present or has the value `urn:liberty:sb:2005-11:EndpointUpdate:Complete`, the
1531 `<wsa:EndpointUpdate>` is a completely specified endpoint reference.

1532 If `updateType` has the value `urn:liberty:sb:2005-11:EndpointUpdate:Partial`, the `<wsa:EndpointUpdate>`
1533 is a partially specified endpoint reference. The following steps are used to construct a complete endpoint reference
1534 from the endpoint reference that was used to address the request that resulted in the response containing this header:

- 1535 1. Take the `<wsa:Address>` from the `<wsa:EndpointUpdate>`. If the value is `urn:liberty:sb:2005-11:`
1536 `EndpointUpdate:NoChange`, then take the `<wsa:Address>` from the original endpoint reference.
- 1537 2. Take the `<wsa:ReferenceParameters>` from the `<wsa:EndpointUpdate>`, if present. Then, if
1538 `<wsa:ReferenceParameters>` is present in the original endpoint reference, take each direct child from
1539 that element that does not match an element already taken from the update (comparing the namespace qualified
1540 names of the elements).
- 1541 3. Take the `<wsa:Metadata>` from the `<wsa:EndpointUpdate>`, if present. Then, if `<wsa:Metadata>` is
1542 present in the original endpoint reference, take each direct child from that element that does not match an element
1543 already taken from the update (comparing the namespace qualified names of the elements).
- 1544 4. Take any extension elements from the `<wsa:EndpointUpdate>`, if present. Then, if any extension elements are
1545 present in the original endpoint reference, take each one that does not match an element already taken from the
1546 update (comparing the namespace qualified names of the elements).

1547 **6.4.5. Processing Rules for the EndpointMoved SOAP Fault**

1548 **6.4.5.1. Sender Processing Rules**

1549 The receiver of an ID-* message MAY issue a SOAP Fault indicating that the endpoint to which this message was
1550 submitted has permanently changed.

1551 Once the receiver has sent this fault response, no further processing of the message should take place.

1552 If the receiver chooses to send the fault response, then it SHOULD also include an `<EndpointUpdate>` header,
1553 indicating the new endpoint which should be used to re-submit this message, and any further messages directed to the
1554 responding service.

1555 **6.4.5.2. Receiver Processing Rules**

1556 If the receiver of this fault response also received an <EndpointUpdate> header in the response, it MAY re-submit
1557 the failed request to any endpoint specified in that header, but it SHOULD provide a different <wsa:MessageID>
1558 header block value in the request.

1559 **6.5. The <Timeout> Header Block**

1560 **6.5.1. Overview**

1561 A requesting entity may wish to indicate that they would like a request to be processed within some specified amount
1562 of time. Such an entity would indicate their wish via the <Timeout> header block.

1563 **6.5.2. Timeout Type and Element**

1564 Senders of ID-* messages MAY add a <Timeout> element to the SOAP header of their request.

1565 This element is based upon the TimeoutType which is defined as:

- 1566 • maxProcessingTime [Required] – an integer specifying (in seconds) the maximum amount of time the sender
1567 wishes the receiver to spend in processing their request
- 1568 • id [Optional] – An attribute facilitating references to elements of this type. This attribute MUST be used when
1569 the message is signed as described in [LibertySecMech], and the element instance is to be included as one of the
1570 set of signed message components.
- 1571 • S:mustUnderstand [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].
- 1572 • S:actor [Optional] – The SOAP actor attribute [SOAPv1.1].

1573 The following schema fragment describes the <Timeout> header block.

```
1574  
1575  
1576 <!-- timeout header block -->  
1577  
1578 <xs:complexType name="TimeoutType">  
1579   <xs:attribute name="maxProcessingTime" type="xs:integer" use="required"/>  
1580   <xs:attribute name="id" type="xs:ID" use="optional"/>  
1581   <xs:attribute ref="S:mustUnderstand" use="optional"/>  
1582   <xs:attribute ref="S:actor" use="optional"/>  
1583 </xs:complexType>  
1584  
1585 <xs:element name="Timeout" type="TimeoutType"/>  
1586  
1587  
1588
```

1589 **Figure 12. The <Timeout> Header Block Element and Type Schema**

1590 **6.5.3. Timeout Example**

```

1591
1592 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1593         xmlns:sb="urn:liberty:sb:2005-11"
1594         xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
1595
1596 <S:Header>
1597     ...
1598
1599
1600 <sb:Timeout mustUnderstand="1" id="timeout.123"
1601         maxProcessingTime="7"/>
1602
1603     ...
1604
1605 </S:Header>
1606
1607 <S:Body>
1608
1609 <idpp:Query>
1610     ...
1611 </idpp:Query>
1612
1613 </S:Body>
1614
1615 </S:Envelope>
1616
  
```

1617 **Example 14. Example of a Request with Timeout Specified**

```

1618
1619 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1620         xmlns:sb="urn:liberty:sb:2005-11">
1621
1622 <S:Header>
1623     ...
1624
1625 </S:Header>
1626
1627 <S:Body>
1628
1629 <S:Fault>
1630
1631 <S:faultcode>
1632     S:Server
1633 </S:faultcode>
1634
1635 <S:detail>
1636
1637 <sb:Status code="ProcessingTimeout"/>
1638
1639 <!-- Reference the specified Timeout header, if it was supplied
1640     by the sender -->
1641
1642 <sb:Timeout id="timeout.123"/>
1643
1644 </S:detail>
1645 </S:Fault>
1646
1647 </S:Body>
1648
1649 </S:Envelope>
1650
1651
  
```

1652 **Example 15. Example of a Timed-out Response**

1653 **6.5.4. Processing Rules**

1654 **6.5.4.1. Receiver Processing Rules**

1655 The receiver of a `<Timeout>` header SHOULD NOT begin processing of a message (beyond processing of the
1656 SOAP headers as noted in this specification) if it expects that such processing would exceed the value specified in
1657 the `maxProcessingTime` attribute.

1658 The receiver MUST respond to the message within the number of seconds specified in the `maxProcessingTime`
1659 attribute.

1660 If the receiver is unable to complete processing within the number of seconds specified in the `maxProcessingTime` at-
1661 tribute of the `<Timeout>` header, then they MUST respond with a SOAP Fault with a `code` of `ProcessingTimeout`.

1662 **Note**

1663 If the sender of a message does not include a `<Timeout>` header, but the receiver wishes to indicate to the sender
1664 that server processing failed due to a timeout, then the receiver MAY respond with a SOAP Fault with a `code` of
1665 `ProcessingTimeout`.

1666 **6.6. The `<UsageDirective>` Header Block**

1667 This section defines the ID-* usage directive facilities.

1668 **6.6.1. Overview**

1669 Participants in the ID-WSF framework may need to indicate the privacy policy associated with a message. To facilitate
1670 this, senders, acting as either a client or a server, may add one or more `<UsageDirective>` header blocks to the
1671 SOAP Header of the message being sent. A `<UsageDirective>` appearing in a SOAP-based ID-* request message
1672 expresses *intended usage*. A `<UsageDirective>` appearing in a response expresses *how* the receiver of the response
1673 is to use the response data. A `<UsageDirective>` in a response message containing no ID-WSF response message
1674 data, a fault response for example, may be used to express policies acceptable to the responder.

1675 **6.6.2. `UsageDirective` Header Type and Element**

1676 Senders MAY add a `<UsageDirective>` element to the SOAP header. This element is based upon the
1677 `UsageDirectiveType` which is defined as:

- 1678 • `ref` [Required] – An attribute referring to an element of the SOAP-based ID-* message to which the usage directive
1679 applies.
- 1680 • `id` [Optional] – An attribute facilitating references to elements of this type. This attribute MUST be used when
1681 the message is signed as described in [[LibertySecMech](#)], and the element instance is to be included as one of the
1682 set of signed message components.
- 1683 • `S:mustUnderstand` [Optional] – The SOAP `mustUnderstand` attribute [[SOAPv1.1](#)].
- 1684 • `S:actor` [Optional] – The SOAP `actor` attribute [[SOAPv1.1](#)].
- 1685 • `<element>(s)` [Optional] – Elements, comprising an instance of some policy expression language, whose
1686 purpose is to express the actual policy the usage directive is conveying. The `ref` attribute above points at the
1687 element in the overall SOAP-based ID-* message to which the usage directive applies.

1688 The schema fragments in [Figure 13](#) and [Figure 14](#) defines the <UsageDirective> header type and element.

```
1689 <element name="UsageDirective" type="UsageDirectiveType" />
1690
1691
```

1692 **Figure 13. The <UsageDirective> Header Block Element Schema**

```
1693 <complexType name="UsageDirectiveType">
1694   <sequence>
1695     <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded" />
1696   </sequence>
1697   <attribute name="ref" type="reference" use="required" />
1698   <attribute name="id" type="xs:ID" use="optional" />
1699   <attribute ref="S:mustUnderstand" use="optional" />
1700   <attribute ref="S:actor" use="optional" />
1701 </complexType>
1702
1703
```

1704 **Figure 14. The UsageDirective Header Block Type Schema**

1705 6.6.3. Usage Directive Examples

1706 [Example 16](#) illustrates a SOAP-based ID-* message, containing a <UsageDirective> header block, and
1707 conveying a Personal Profile (ID-PP) Modify message [[LibertyIDPP](#)]. The <UsageDirective> header block
1708 contains a usage directive expressed in a policy language identified by the cot: namespace and the URI
1709 <http://cot.example.com/policies/eu-compliant> , and applying to the ID-PP Query message identified by
1710 the id of datarequest001.


```
1711
1712     <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1713       xmlns:sb="urn:liberty:sb:2005-11"
1714       xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1715
1716     <S:Header>
1717
1718       ...
1719
1720       <sb:UsageDirective
1721         id="directive1000"
1722         ref="#datarequest001"
1723         S:mustUnderstand="1">
1724
1725         <cot:PrivacyPolicyReference
1726           xmlns:cot="http://cot.example.com/isf">
1727           http://cot.example.com/policies/eu-compliant
1728         </cot:PrivacyPolicyReference>
1729
1730       </sb:UsageDirective>
1731
1732       ...
1733
1734     </S:Header>
1735
1736     <S:Body>
1737
1738       <pp:Query id="datarequest001" xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1739         <pp:ResourceID>data:d8ddw6dd7m28v6 28</pp:ResourceID>
1740         <pp:QueryItem>
1741           <pp:Select>/pp: IDPP/pp: IDPPAddressCard< /pp:Select>
1742         </pp:QueryItem>
1743       </pp:Query>
1744
1745     </S:Body>
1746
1747   </S:Envelope>
1748
```

1749 **Example 16. A Usage Directive on a Request for the Address of a Principal.**

1750 6.6.4. Processing Rules

1751 6.6.4.1. Sender Processing Rules

1752 The sender of a SOAP-based ID-* message with a <UsageDirective> header block MUST ensure that the value
1753 of the ref attribute is set to the value of the id of the appropriate element in the message. The sender SHOULD
1754 ensure that the <UsageDirective> is integrity-protected. The protection mechanism, if utilized, SHOULD be in
1755 accordance with those defined in [LibertySecMech].

1756 6.6.4.2. Receiver Processing Rules

1757 A receiver of a SOAP-based ID-* message with an attached <UsageDirective> header block MUST check the
1758 actor attribute and determine if it, the receiver, is the actor the header block is targeted at. If so, the receiver MUST
1759 check the mustUnderstand attribute. If set to TRUE the receiver MUST process the contents. If the attribute is absent
1760 or set to FALSE the receiver SHOULD attempt to process the content of the <UsageDirective> header block.

1761 A receiver that processes the contents of a <UsageDirective> header block SHOULD verify the integrity of the
1762 header block – that is, it should verify any digital signatures that list the header block in its manifest [XMLDsig]. The
1763 receiver MUST verify that the ref attribute refers to an element in the message. That receiver MUST further process
1764 the message according to the policy expressed by the children elements of the <UsageDirective> header block.

1765 Those children elements will be imported from a foreign namespace, and MUST be parsed and interpreted according
1766 to the applicable schema and processing rules of that foreign namespace.

1767 A receiver that cannot process a <UsageDirective> with mustUnderstand="TRUE" MUST respond with a
1768 <S:Fault>. The s:Fault MUST contain a <S:Detail> element which in turn MUST contain a <Status> element
1769 with its code attribute set to CannotHonourUsageDirective. The <Status> element SHOULD possess a ref
1770 attribute with its value set to the value of the id attribute of the offending <UsageDirective> header block in the
1771 request message.

1772 A receiver that cannot honor a non-mandatory (with mustUnderstand="FALSE") <UsageDirective> must re-
1773 spond according to the contained policy. In addition, in this case the receiver MAY respond with a SOAP-based ID-*
1774 message that includes a <Status> element with its code attribute set to CannotHonourUsageDirective. This
1775 <Status> element instance SHOULD include a ref attribute with its value set to the value of the id attribute of the
1776 <UsageDirective> header block in the request message that could not be honored.

1777 In this case, the receiver MAY include one or more new <UsageDirective> header blocks in its response message,
1778 each expressing a policy that the receiver would have been able to honor. The ref attribute of these headers SHOULD
1779 be set to the value of the <wsa:MessageID> header block in the request.

1780 **6.7. The <ApplicationEPR> Header Block**

1781 This section defines the <ApplicationEPR> header block. This header may be included in a message zero or more
1782 times and provides a means for a sender to specify application endpoints that may be referenced from the SOAP Body
1783 of the message.

1784 The <ApplicationEPR> header block is an extension of <wsa:EndpointReferenceType> that adds the following
1785 attributes:

- 1786 • id [Optional] – An attribute facilitating references to elements of this type. This attribute MUST be used when
1787 the message is signed as described in [LibertySecMech], and the element instance is to be included as one of the
1788 set of signed message components.
- 1789 • S:mustUnderstand [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].
- 1790 • S:actor [Optional] – The SOAP actor attribute [SOAPv1.1].

1791 The schema fragment in Figure 15 defines the The <ApplicationEPR> header block.

```
1792  
1793  
1794 <!-- application epr header block -->  
1795  
1796 <xs:complexType name="ApplicationEPRTYPE">  
1797   <xs:complexContent>  
1798     <xs:extension base="wsa:EndpointReferenceType">  
1799       <xs:attribute name="id" type="xs:ID" use="optional"/>  
1800       <xs:attribute ref="S:mustUnderstand" use="optional"/>  
1801       <xs:attribute ref="S:actor" use="optional"/>  
1802     </xs:extension>  
1803   </xs:complexContent>  
1804 </xs:complexType>  
1805  
1806 <xs:element name="ApplicationEPR" type="ApplicationEPRTYPE"/>  
1807  
1808  
1809
```

1810 **Figure 15. The <ApplicationEPR> Header Block Schema**

```
1811
1812     <sb:ApplicationEPR S:mustUnderstand="1"
1813       id="NotifyTo-001"
1814       actor="http://schemas.../next">
1815       <wsa:Address>...</wsa:Address>
1816     </sb:ApplicationEPR>
1817
```

1818 **Example 17. An instantiated <ApplicationEPR> header block**

1819 7. Security Considerations

- 1820 • The header blocks specified in this document should be integrity-protected using the mechanisms detailed in
1821 [[LibertySecMech](#)].
- 1822 • Header blocks should be signed in accordance with [[LibertySecMech](#)]. The receiver of a message containing a
1823 signature that covers specific header blocks should verify the signature as part of verifying the integrity of the
1824 header block.
- 1825 • Metadata [[LibertyMetadata](#)] should be used to the greatest extent possible to verify message sender identity claims.
- 1826 • Message senders and receivers should be authenticated to one another via the mechanisms discussed in [[Liberty-](#)
1827 [SecMech](#)].
- 1828 • To prevent message replay, receivers should maintain a message cache, and check received `messageID` values
1829 against the cache.

1830 **8. Acknowledgements**

1831 The members of the Liberty Technical Expert group, especially Greg Whitehead, Jonathan Sergent, Xavier Serret,
1832 and Conor Cahill, provided valuable input to this specification. The docbook source code for this specification was
1833 hand set to the tunes of The Sugarcubes, King Crimson, Juliana Hatfield, Smashing Pumpkins, Evanescence, Mad at
1834 Gravity, Elisa Korenne, The Breeders, fIREHOSE, Polly Jean Harvey, Jimi Hendrix, and various others.

1835 Bibliography

1836 Normative

- 1837 [LibertyDST] Kellomäki, Sampo, Kainulainen, Jukka, eds. "Liberty ID-WSF Data Services Template," Version 2.1-
1838 08, Liberty Alliance Project (21 Sep 2005). <http://www.projectliberty.org/specs>
- 1839 [LibertyIDFFOverview] Wason, Thomas, eds. "Liberty ID-FF Architecture Overview," Version 1.2-errata-v1.0,
1840 Liberty Alliance Project (12 September 2004). <http://www.projectliberty.org/specs>
- 1841 [LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-
1842 04, Liberty Alliance Project (21 Sep 2005). <http://www.projectliberty.org/specs>
- 1843 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 2.0-02,
1844 Liberty Alliance Project (25 November 2004). <http://www.projectliberty.org/specs>
- 1845 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions
1846 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
1847 SIS Standard, Organization for the Advancement of Structured Information Standards [http://docs.oasis-
open.org/security/saml/v2.0/saml-core-2.0-os.pdf](http://docs.oasis-
1848 open.org/security/saml/v2.0/saml-core-2.0-os.pdf)
- 1849 [LibertySecMech] Ellison, Gary, Hirsch, Frederick, Madsen, Paul, eds. "Liberty ID-WSF Security Mechanisms Core,"
1850 Version v2.0-12, Liberty Alliance Project (22 September 2005). <http://www.projectliberty.org/specs>
- 1851 [RFC2045] Freed, N., Borenstein, N., eds. (November 1996). "Multipurpose Internet Mail Extensions
1852 (MIME) Part One: Format of Internet Message Bodies ," RFC 2045., Internet Engineering Task
1853 Force <http://www.ietf.org/rfc/rfc2045.txt> [November 1996].
- 1854 [RFC2119] Bradner, S., eds. "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
1855 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt> [March 1997].
- 1856 [RFC2396] Berners-Lee, T., Fielding, R., Masinter, L., eds. (August 1998). "Uniform Resource Identifiers (URI):
1857 Generic Syntax," RFC 2396, The Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2396.txt>
1858 [August 1998].
- 1859 [RFC2828] Shirey, R., eds. (May 2000). "Internet Security Glossary," RFC 2828., Internet Engineering Task Force
1860 <http://www.ietf.org/rfc/rfc2828.txt> [May 2000].
- 1861 [SAMLGloss] Hodges, Jeff, Maler, Eve, eds. (05 November 2002). "Glossary for the OASIS Security Assertion
1862 Markup Language (SAML)," SAML V1.0, OASIS Standard, Organization for the Advancement of Structured
1863 Information Standards <http://www.oasis-open.org/specs/index.php#samlv1.0>
- 1864 [Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May
1865 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium
1866 <http://www.w3.org/TR/xmlschema-1/>
- 1867 [Schema2] Biron, Paul V., Malhotra, Ashok, eds. (May 2002). "XML Schema Part 2: Datatypes," Recommendation,
1868 World Wide Web Consortium <http://www.w3.org/TR/xmlschema-2/>
- 1869 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David , Kakivaya, Gopal, Layman,
1870 Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C
1871 Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 1872 [SOAPv1.1-Schema] "SOAP 1.1 Envelope schema," W3C W3C Note <http://schemas.xmlsoap.org/soap/envelope/>

- 1873 [WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Meredith,
1874 Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001).
1875 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- 1876 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C.M., Maler, Eve, eds. (Oct 2000). "Extensible
1877 Markup Language (XML) 1.0 (Second Edition)," Recommendation, World Wide Web Consortium
1878 <http://www.w3.org/TR/2000/REC-xml-20001006>
- 1879 [XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and
1880 Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlsig-core>
- 1881 [WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, eds. World Wide Web
1882 Consortium W3C Candidate Recommendation (17 August 2005). [http://www.w3.org/TR/2005/CR-ws-addr-
1883 core-20050817/](http://www.w3.org/TR/2005/CR-ws-addr-core-20050817/)
- 1884 [WSAv1.0-SOAP] "WS-Addressing 1.0 SOAP Binding," Gudgin, Martin, Hadley, Marc, eds. World Wide Web
1885 Consortium W3C Candidate Recommendation (17 August 2005). [http://www.w3.org/TR/2005/CR-ws-addr-
1886 soap-20050817/](http://www.w3.org/TR/2005/CR-ws-addr-soap-20050817/)
- 1887 [WSAv1.0-Schema] "WS-Addressing 1.0 Schema," W3C W3C Working Draft [http://dev.w3.org/cvsweb/~checkout~/2004/ws/addressing-
1888 addr.xsd](http://dev.w3.org/cvsweb/~checkout~/2004/ws/addressing-addr.xsd)
- 1889 [wss-sms] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (January, 2004). "Web
1890 Services Security: SOAP Message Security," OASIS Standard V1.0 [OASIS 200401], Organization for the
1891 Advancement of Structured Information Standards [http://docs.oasis-open.org/wss/2004/01/oasis-200401-
1892 wss-soap-message-security-1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)

1893 Informational

- 1894 [LibertyDisco] Beatty, John, Sergent, Jonathan, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification,"
1895 Version 2.0-12, Liberty Alliance Project (21 Sep 2005). <http://www.projectliberty.org/specs>
- 1896 [LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0-03, Liberty Alliance Project (29 Aug
1897 2005). <http://www.projectliberty.org/specs>
- 1898 [LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services
1899 Framework Overview," Version 1.0-errata-v1.0, Liberty Alliance Project (12 September 2004).
1900 <http://www.projectliberty.org/specs>
- 1901 [LibertyIDPPP] Kellomäki, Sampo, eds. "Liberty Identity Personal Profile Service Specification," Version 1.1-02,
1902 Liberty Alliance Project (13 September, 2005). <http://www.projectliberty.org/specs>
- 1903 [Merriam-Webster] "Merriam-Webster Dictionary," <http://www.merriam-webster.com/>
- 1904 [RFC1750] Eastlake, D., Crocker, S., Schiller, J., eds. (December 1994). "Randomness Recommendations for
1905 Security," RFC 1750, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc1750.txt> [August 2003].
- 1906 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., eds. (June
1907 1999). "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, The Internet Engineering Task Force
1908 <http://www.ietf.org/rfc/rfc2616.txt> [June 1999].
- 1909 [SOAPv1.2] "SOAP Version 1.2 Part 1: Messaging Framework," Gudgin, Martin, Hadley, Marc, Mendelsohn,
1910 Noah, Moreau, Jean-Jacques, Nielsen, Henrik Frystyk, eds. World Wide Web Consortium W3C Proposed
1911 Recommendation (07 May 2003). <http://www.w3.org/TR/2003/PR-soap12-part1-20030507/>

1912 A. liberty-idwsf-soap-binding-v2.0.xsd Schema Listing

```
1913
1914 <?xml version="1.0" encoding="UTF-8"?>
1915 <xs:schema targetNamespace="urn:liberty:sb:2005-11"
1916 xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1917 xmlns:sb="urn:liberty:sb:2005-11"
1918 xmlns:samlp2="urn:oasis:names:tc:SAML:2.0:protocol"
1919 xmlns:wssu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
1920 xmlns:wsa="http://www.w3.org/2005/08/addressing"
1921 xmlns:xs="http://www.w3.org/2001/XMLSchema"
1922 xmlns="urn:liberty:sb:2005-11"
1923 elementFormDefault="qualified"
1924 attributeFormDefault="unqualified">
1925
1926 <!-- Author: John Kemp -->
1927 <!-- Last editor: $Author: dchampagne $ -->
1928 <!-- $Date: 2005/09/23 18:42:43 $ -->
1929 <!-- $Revision: 1.1 $ -->
1930
1931 <xs:import
1932 namespace="http://schemas.xmlsoap.org/soap/envelope/"
1933 schemaLocation="soap-envelope-1.1.xsd"/>
1934
1935 <xs:import
1936 namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecur
1937 ity-utility-1.0.xsd"
1938 schemaLocation="wss-util-1.0.xsd"/>
1939
1940 <xs:import
1941 namespace="urn:oasis:names:tc:SAML:2.0:protocol"
1942 schemaLocation="saml-schema-protocol-2.0.xsd"/>
1943
1944 <xs:import
1945 namespace="http://www.w3.org/2005/08/addressing"
1946 schemaLocation="ws-addr-1.0.xsd"/>
1947
1948 <xs:include schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
1949
1950 <xs:annotation>
1951 <xs:documentation>
1952 Liberty ID-WSF SOAP Binding Specification Extension XSD
1953 </xs:documentation>
1954 <xs:documentation>
1955 The source code in this XSD file was excerpted verbatim from:
1956
1957 Liberty ID-WSF SOAP Binding Specification
1958 Version 2.0-09
1959 22 September 2005
1960
1961 Copyright (c) 2005 Liberty Alliance participants, see
1962 http://www.projectliberty.org/specs/idwsf_2_0_r2_copyrights.php
1963 </xs:documentation>
1964 </xs:annotation>
1965
1966 <!-- sender header block -->
1967
1968
1969 <xs:complexType name="SenderType">
1970 <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
1971 <xs:attribute name="affiliationID" type="xs:anyURI" use="optional"/>
1972 <xs:attribute name="id" type="xs:ID" use="optional"/>
1973 <xs:attribute ref="S:mustUnderstand" use="optional"/>
1974 <xs:attribute ref="S:actor" use="optional"/>
1975 </xs:complexType>
1976
1977 <xs:element name="Sender" type="SenderType"/>
```



```
1978
1979
1980 <!-- invocation identity header block -->
1981
1982 <xs:complexType name="InvocationIdentityType">
1983   <xs:sequence>
1984     <xs:any namespace="##any" processContents="lax"/>
1985   </xs:sequence>
1986   <xs:attribute name="id" type="xs:ID" use="optional"/>
1987   <xs:attribute ref="S:mustUnderstand" use="optional"/>
1988   <xs:attribute ref="S:actor" use="optional"/>
1989   <xs:attribute ref="wsa:IsReferenceParameter" use="optional"/>
1990   <xs:anyAttribute namespace="##other" processContents="lax"/>
1991 </xs:complexType>
1992
1993 <xs:element name="InvocationIdentity" type="InvocationIdentityType"/>
1994
1995
1996 <!-- target identity header block -->
1997
1998 <xs:complexType name="TargetIdentityType">
1999   <xs:sequence>
2000     <xs:any namespace="##any" processContents="lax"/>
2001   </xs:sequence>
2002   <xs:attribute name="id" type="xs:ID" use="optional"/>
2003   <xs:attribute ref="S:mustUnderstand" use="optional"/>
2004   <xs:attribute ref="S:actor" use="optional"/>
2005   <xs:attribute ref="wsa:IsReferenceParameter" use="optional"/>
2006   <xs:anyAttribute namespace="##other" processContents="lax"/>
2007 </xs:complexType>
2008
2009 <xs:element name="TargetIdentity" type="TargetIdentityType"/>
2010
2011
2012 <!-- credentials context header block -->
2013
2014 <xs:complexType name="CredentialsContextType">
2015   <xs:sequence>
2016     <xs:element ref="sampl2:RequestedAuthnContext" minOccurs="0"/>
2017     <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
2018   </xs:sequence>
2019   <xs:attribute name="id" type="xs:ID" use="optional"/>
2020   <xs:attribute ref="S:mustUnderstand" use="optional"/>
2021   <xs:attribute ref="S:actor" use="optional"/>
2022 </xs:complexType>
2023
2024 <xs:element name="CredentialsContext" type="CredentialsContextType"/>
2025
2026
2027 <!-- epr update header block -->
2028
2029 <xs:complexType name="EndpointUpdateType">
2030   <xs:complexContent>
2031     <xs:extension base="wsa:EndpointReferenceType">
2032       <xs:attribute name="updateType" type="xs:anyURI" use="optional"/>
2033       <xs:attribute name="id" type="xs:ID" use="optional"/>
2034       <xs:attribute ref="S:mustUnderstand" use="optional"/>
2035       <xs:attribute ref="S:actor" use="optional"/>
2036     </xs:extension>
2037   </xs:complexContent>
2038 </xs:complexType>
2039
2040 <xs:element name="EndpointUpdate" type="EndpointUpdateType"/>
2041
2042
2043 <!-- timeout header block -->
2044
```

```

2045     <xs:complexType name="TimeoutType">
2046         <xs:attribute name="maxProcessingTime" type="xs:integer" use="required"/>
2047         <xs:attribute name="id" type="xs:ID" use="optional"/>
2048         <xs:attribute ref="S:mustUnderstand" use="optional"/>
2049         <xs:attribute ref="S:actor" use="optional"/>
2050     </xs:complexType>
2051
2052     <xs:element name="Timeout" type="TimeoutType"/>
2053
2054
2055 <!-- processing context header block -->
2056
2057 <xs:complexType name="ProcessingContextType">
2058     <xs:simpleContent>
2059         <xs:extension base="xs:anyURI">
2060             <xs:attribute name="id" type="xs:ID" use="optional"/>
2061             <xs:attribute ref="S:mustUnderstand" use="optional"/>
2062             <xs:attribute ref="S:actor" use="optional"/>
2063         </xs:extension>
2064     </xs:simpleContent>
2065 </xs:complexType>
2066
2067 <xs:element name="ProcessingContext" type="ProcessingContextType"/>
2068
2069 <!-- consent header block -->
2070
2071 <xs:complexType name="ConsentType">
2072     <xs:attribute name="uri" type="xs:anyURI" use="required"/>
2073     <xs:attribute name="timestamp" type="xs:dateTime" use="optional"/>
2074     <xs:attribute name="id" type="xs:ID" use="optional"/>
2075     <xs:attribute ref="S:mustUnderstand" use="optional"/>
2076     <xs:attribute ref="S:actor" use="optional"/>
2077 </xs:complexType>
2078
2079 <xs:element name="Consent" type="ConsentType"/>
2080
2081 <!-- usage directive header block -->
2082
2083 <xs:complexType name="UsageDirectiveType">
2084     <xs:sequence>
2085         <xs:any namespace="##other" processContents="lax"
2086             maxOccurs="unbounded"/>
2087     </xs:sequence>
2088     <xs:attribute name="id" type="xs:ID" use="optional"/>
2089     <xs:attribute name="ref" type="xs:IDREF" use="required"/>
2090     <xs:attribute ref="S:mustUnderstand" use="optional"/>
2091     <xs:attribute ref="S:actor" use="optional"/>
2092 </xs:complexType>
2093
2094 <xs:element name="UsageDirective" type="UsageDirectiveType"/>
2095
2096 <!-- application epr header block -->
2097
2098 <xs:complexType name="ApplicationEPRTType">
2099     <xs:complexContent>
2100         <xs:extension base="wsa:EndpointReferenceType">
2101             <xs:attribute name="id" type="xs:ID" use="optional"/>
2102             <xs:attribute ref="S:mustUnderstand" use="optional"/>
2103             <xs:attribute ref="S:actor" use="optional"/>
2104         </xs:extension>
2105     </xs:complexContent>
2106 </xs:complexType>
2107
2108     <xs:element name="ApplicationEPR" type="ApplicationEPRTType"/>
2109 </xs:schema>
2110
2111

```

2112 **B. liberty-idwsf-utility-v2.0.xsd Schema Listing**

```
2113
2114     <?xml version="1.0" encoding="UTF-8"?>
2115 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2116     elementFormDefault="qualified"
2117     attributeFormDefault="unqualified"
2118     version="2.0-01">
2119
2120     <xs:include schemaLocation="liberty-utility-v2.0.xsd"/>
2121
2122     <xs:annotation>
2123         <xs:documentation>
2124             Liberty Alliance Project utility schema. A collection of common
2125             IDentity Web Services Framework (ID-WSF) elements and types.
2126             This schema is intended for use in ID-WSF schemas.
2127
2128             This file intended for inclusion, rather than importation, into other schemas.
2129             This version: 2004-12
2130
2131             Copyright (c) 2005 Liberty Alliance participants, see
2132             http://www.projectliberty.org/specs/idwsf\_2\_0\_r2\_copyrights.php
2133
2134         </xs:documentation>
2135     </xs:annotation>
2136 </xs:schema>
2137
2138
```

2139 C. liberty-utility-v2.0.xsd Schema Listing

```
2140
2141     <?xml version="1.0" encoding="UTF-8"?>
2142 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2143     elementFormDefault="qualified"
2144     attributeFormDefault="unqualified"
2145     version="2.0-01">
2146     <xs:annotation>
2147         <xs:documentation>
2148 Liberty Alliance Project utility schema. A collection of common
2149 elements and types for use with independent Liberty XML Schema documents.
2150
2151 This file intended for inclusion, rather than importation, into other schemas.
2152 This version: 2004-12
2153
2154         Copyright (c) 2004 Liberty Alliance participants, see
2155         http://www.projectliberty.org/specs/idff\_copyrights.html
2156
2157     </xs:documentation>
2158 </xs:annotation>
2159 <xs:simpleType name="IDType">
2160     <xs:annotation>
2161         <xs:documentation>
2162             This type should be used to provide IDs to components
2163             that have IDs that may not be scoped within the local
2164             xml instance document.
2165         </xs:documentation>
2166     </xs:annotation>
2167     <xs:restriction base="xs:string"/>
2168 </xs:simpleType>
2169 <xs:simpleType name="IDReferenceType">
2170     <xs:annotation>
2171         <xs:documentation>
2172             This type can be used when referring to elements that are
2173             identified using an IDType.
2174         </xs:documentation>
2175     </xs:annotation>
2176     <xs:restriction base="xs:string"/>
2177 </xs:simpleType>
2178 <xs:complexType name="StatusType">
2179     <xs:annotation>
2180         <xs:documentation>
2181             A type that may be used for status codes.
2182         </xs:documentation>
2183     </xs:annotation>
2184     <xs:sequence>
2185         <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
2186     </xs:sequence>
2187     <xs:attribute name="code" type="xs:string" use="required"/>
2188     <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
2189     <xs:attribute name="comment" type="xs:string" use="optional"/>
2190 </xs:complexType>
2191
2192 <xs:element name="Status" type="StatusType">
2193     <xs:annotation>
2194         <xs:documentation>
2195             A standard Status type
2196         </xs:documentation>
2197     </xs:annotation>
2198 </xs:element>
2199
2200 <xs:complexType name="EmptyType">
2201     <xs:annotation>
2202         <xs:documentation> This type may be used to create an empty element </xs:documentation>
2203     </xs:annotation>
2204     <xs:complexContent>
```

```
2205     <xs:restriction base="xs:anyType" />
2206   </xs:complexContent>
2207 </xs:complexType>
2208 <xs:element name="Extension" type="extensionType">
2209   <xs:annotation>
2210     <xs:documentation>
2211       An element that contains arbitrary content extensions
2212       from other namespaces
2213     </xs:documentation>
2214   </xs:annotation>
2215 </xs:element>
2216 <xs:complexType name="extensionType">
2217   <xs:annotation>
2218     <xs:documentation>
2219       A type for arbitrary content extensions from other namespaces
2220     </xs:documentation>
2221   </xs:annotation>
2222   <xs:sequence>
2223     <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded" />
2224   </xs:sequence>
2225 </xs:complexType>
2226 </xs:schema>
2227
2228
```

2229 D. soap-envelope-1.1.xsd Schema Listing

```
2230
2231     <?xml version='1.0' encoding='UTF-8' ?>
2232
2233 <!-- Schema for the SOAP/1.1 envelope
2234
2235     This schema has been produced using W3C's SOAP Version 1.2 schema
2236     found at:
2237
2238     http://www.w3.org/2001/06/soap-envelope
2239
2240     Copyright 2001 Martin Gudgin, Developmentor.
2241
2242     Changes made are the following:
2243     - reverted namespace to http://schemas.xmlsoap.org/soap/envelope/
2244     - reverted mustUnderstand to only allow 0 and 1 as lexical values
2245     - made encodingStyle a global attribute 20020825
2246
2247     Further changes:
2248
2249     - removed default value from mustUnderstand attribute declaration - 20030314
2250
2251     Original copyright:
2252
2253     Copyright 2001 W3C (Massachusetts Institute of Technology,
2254     Institut National de Recherche en Informatique et en Automatique,
2255     Keio University). All Rights Reserved.
2256     http://www.w3.org/Consortium/Legal/
2257
2258     This document is governed by the W3C Software License [1] as
2259     described in the FAQ [2].
2260
2261     [1] http://www.w3.org/Consortium/Legal/copyright-software-19980720
2262     [2] http://www.w3.org/Consortium/Legal/IPR-FAQ-20000620.html#DTD
2263 -->
2264
2265
2266 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2267     xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
2268     targetNamespace="http://schemas.xmlsoap.org/soap/envelope/" >
2269
2270
2271     <!-- Envelope, header and body -->
2272
2273     <xs:element name="Envelope" type="tns:Envelope" />
2274
2275     <xs:complexType name="Envelope" >
2276         <xs:sequence>
2277             <xs:element ref="tns:Header" minOccurs="0" />
2278             <xs:element ref="tns:Body" minOccurs="1" />
2279             <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax" />
2280         </xs:sequence>
2281         <xs:anyAttribute namespace="##other" processContents="lax" />
2282     </xs:complexType>
2283
2284
2285     <xs:element name="Header" type="tns:Header" />
2286
2287     <xs:complexType name="Header" >
2288         <xs:sequence>
2289             <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax" />
2290         </xs:sequence>
2291         <xs:anyAttribute namespace="##other" processContents="lax" />
2292     </xs:complexType>
2293
2294
```

```

2295 <xs:element name="Body" type="tns:Body" />
2296
2297 <xs:complexType name="Body" >
2298   <xs:sequence>
2299     <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded" processContents="lax" />
2300   </xs:sequence>
2301   <xs:anyAttribute namespace="##any" processContents="lax" >
2302     <xs:annotation>
2303       <xs:documentation>
2304         Prose in the spec does not specify that attributes are allowed on the Body element
2305       </xs:documentation>
2306     </xs:annotation>
2307   </xs:anyAttribute>
2308 </xs:complexType>
2309
2310
2311 <!-- Global Attributes. The following attributes are intended to be usable via
2312      qualified attribute names on any complex type referencing them. -->
2313
2314 <xs:attribute name="mustUnderstand" >
2315   <xs:simpleType>
2316     <xs:restriction base='xs:boolean'>
2317       <xs:pattern value='0|1' />
2318     </xs:restriction>
2319   </xs:simpleType>
2320 </xs:attribute>
2321
2322 <xs:attribute name="actor" type="xs:anyURI" />
2323
2324 <xs:simpleType name="encodingStyle" >
2325   <xs:annotation>
2326     <xs:documentation>
2327       'encodingStyle' indicates any canonicalization conventions followed in
2328       the contents of the containing element. For example, the value
2329       'http://schemas.xmlsoap.org/soap/encoding/' indicates the
2330       pattern described in SOAP specification
2331     </xs:documentation>
2332   </xs:annotation>
2333   <xs:list itemType="xs:anyURI" />
2334 </xs:simpleType>
2335
2336 <xs:attribute name="encodingStyle" type="tns:encodingStyle" />
2337
2338
2339 <xs:attributeGroup name="encodingStyle" >
2340   <xs:attribute ref="tns:encodingStyle" />
2341 </xs:attributeGroup>
2342
2343
2344 <xs:element name="Fault" type="tns:Fault" />
2345
2346 <xs:complexType name="Fault" final="extension" >
2347   <xs:annotation>
2348     <xs:documentation>
2349       Fault reporting structure
2350     </xs:documentation>
2351   </xs:annotation>
2352   <xs:sequence>
2353     <xs:element name="faultcode" type="xs:QName" />
2354     <xs:element name="faultstring" type="xs:string" />
2355     <xs:element name="faultactor" type="xs:anyURI" minOccurs="0" />
2356     <xs:element name="detail" type="tns:detail" minOccurs="0" />
2357   </xs:sequence>
2358 </xs:complexType>
2359
2360 <xs:complexType name="detail">
2361   <xs:sequence>

```

```
2362         <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded" processContents="lax" />
2363     </xs:sequence>
2364     <xs:anyAttribute namespace="##any" processContents="lax" />
2365 </xs:complexType>
2366
2367 </xs:schema>
2368
2369
```


2370 E. ws-addr-1.0.xsd Schema Listing

```
2371
2372     <?xml version="1.0" encoding="utf-8"?>
2373 <!DOCTYPE xs:schema PUBLIC "-//W3C//DTD XMLSCHEMA 200102//EN" "http://www.w3.org/2001/XMLSchema.dtd">
2374
2375 <!--
2376     W3C XML Schema defined in the Web Services Addressing 1.0 specification
2377     http://www.w3.org/TR/ws-addr-core
2378
2379     Copyright © 2005 World Wide Web Consortium,
2380
2381     (Massachusetts Institute of Technology, European Research Consortium for
2382     Informatics and Mathematics, Keio University). All Rights Reserved. This
2383     work is distributed under the W3C® Software License [1] in the hope that
2384     it will be useful, but WITHOUT ANY WARRANTY; without even the implied
2385     warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
2386
2387     [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
2388
2389     $Id: ws-addr-1.0.xsd,v 1.4 2005/09/23 18:20:29 dchampagne Exp $
2390 -->
2391 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2392     xmlns:tns="http://www.w3.org/2005/08/addressing"
2393     targetNamespace="http://www.w3.org/2005/08/addressing"
2394     blockDefault="#all"
2395     elementFormDefault="qualified"
2396     finalDefault=""
2397     attributeFormDefault="unqualified">
2398
2399     <!-- Constructs from the WS-Addressing Core -->
2400
2401     <xs:element name="EndpointReference" type="tns:EndpointReferenceType"/>
2402     <xs:complexType name="EndpointReferenceType" mixed="false">
2403         <xs:sequence>
2404             <xs:element name="Address" type="tns:AttributedURIType"/>
2405             <xs:element name="ReferenceParameters" type="tns:ReferenceParametersType" minOccurs="0"/>
2406             <xs:element ref="tns:Metadata" minOccurs="0"/>
2407             <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2408         </xs:sequence>
2409         <xs:anyAttribute namespace="##other" processContents="lax"/>
2410     </xs:complexType>
2411
2412     <xs:complexType name="ReferenceParametersType" mixed="false">
2413         <xs:sequence>
2414             <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2415         </xs:sequence>
2416         <xs:anyAttribute namespace="##other" processContents="lax"/>
2417     </xs:complexType>
2418
2419     <xs:element name="Metadata" type="tns:MetadataType"/>
2420     <xs:complexType name="MetadataType" mixed="false">
2421         <xs:sequence>
2422             <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2423         </xs:sequence>
2424         <xs:anyAttribute namespace="##other" processContents="lax"/>
2425     </xs:complexType>
2426
2427     <xs:element name="MessageID" type="tns:AttributedURIType"/>
2428     <xs:element name="RelatesTo" type="tns:RelatesToType"/>
2429     <xs:complexType name="RelatesToType" mixed="false">
2430         <xs:simpleContent>
2431             <xs:extension base="xs:anyURI">
2432                 <xs:attribute name="RelationshipType" type="tns:RelationshipTypeOpen
2433 Enum" use="optional"
2434                 default="http://www.w3.org/2005/08/addressing/reply"/>
2435             </xs:extension>
2436         </xs:simpleContent>
2437         <xs:anyAttribute namespace="##other" processContents="lax"/>
2438     </xs:complexType>
2439 </xs:schema>
```

```

2436         </xs:extension>
2437     </xs:simpleContent>
2438 </xs:complexType>
2439
2440 <xs:simpleType name="RelationshipTypeOpenEnum">
2441     <xs:union memberTypes="tns:RelationshipType xs:anyURI" />
2442 </xs:simpleType>
2443
2444 <xs:simpleType name="RelationshipType">
2445     <xs:restriction base="xs:anyURI">
2446         <xs:enumeration value="http://www.w3.org/2005/08/addressing/reply" />
2447     </xs:restriction>
2448 </xs:simpleType>
2449
2450 <xs:element name="ReplyTo" type="tns:EndpointReferenceType" />
2451 <xs:element name="From" type="tns:EndpointReferenceType" />
2452 <xs:element name="FaultTo" type="tns:EndpointReferenceType" />
2453 <xs:element name="To" type="tns:AttributedURIType" />
2454 <xs:element name="Action" type="tns:AttributedURIType" />
2455
2456 <xs:complexType name="AttributedURIType" mixed="false">
2457     <xs:simpleContent>
2458         <xs:extension base="xs:anyURI">
2459             <xs:anyAttribute namespace="##other" processContents="lax" />
2460         </xs:extension>
2461     </xs:simpleContent>
2462 </xs:complexType>
2463
2464 <!-- Constructs from the WS-Addressing SOAP binding -->
2465
2466 <xs:attribute name="IsReferenceParameter" type="xs:boolean" />
2467
2468 <xs:simpleType name="FaultCodesOpenEnumType">
2469     <xs:union memberTypes="tns:FaultCodesType xs:QName" />
2470 </xs:simpleType>
2471
2472 <xs:simpleType name="FaultCodesType">
2473     <xs:restriction base="xs:QName">
2474         <xs:enumeration value="tns:InvalidAddressingHeader" />
2475         <xs:enumeration value="tns:InvalidAddress" />
2476         <xs:enumeration value="tns:InvalidEPR" />
2477         <xs:enumeration value="tns:InvalidCardinality" />
2478         <xs:enumeration value="tns:MissingAddressInEPR" />
2479         <xs:enumeration value="tns:DuplicateMessageID" />
2480         <xs:enumeration value="tns:ActionMismatch" />
2481         <xs:enumeration value="tns:MessageAddressingHeaderRequired" />
2482         <xs:enumeration value="tns:DestinationUnreachable" />
2483         <xs:enumeration value="tns:ActionNotSupported" />
2484         <xs:enumeration value="tns:EndpointUnavailable" />
2485     </xs:restriction>
2486 </xs:simpleType>
2487
2488 <xs:element name="RetryAfter" type="tns:AttributedUnsignedLongType" />
2489 <xs:complexType name="AttributedUnsignedLongType" mixed="false">
2490     <xs:simpleContent>
2491         <xs:extension base="xs:unsignedLong">
2492             <xs:anyAttribute namespace="##other" processContents="lax" />
2493         </xs:extension>
2494     </xs:simpleContent>
2495 </xs:complexType>
2496
2497 <xs:element name="ProblemHeaderQName" type="tns:AttributedQNameType" />
2498 <xs:complexType name="AttributedQNameType" mixed="false">
2499     <xs:simpleContent>
2500         <xs:extension base="xs:QName">
2501             <xs:anyAttribute namespace="##other" processContents="lax" />
2502         </xs:extension>
    
```

```
2503     </xs:simpleContent>
2504 </xs:complexType>
2505
2506 <xs:element name="ProblemHeader" type="tns:AttributedAnyType"/>
2507 <xs:complexType name="AttributedAnyType" mixed="false">
2508   <xs:sequence>
2509     <xs:any namespace="##any" processContents="lax" minOccurs="1" maxOccurs="1"/>
2510   </xs:sequence>
2511   <xs:anyAttribute namespace="##other" processContents="lax"/>
2512 </xs:complexType>
2513
2514 <xs:element name="ProblemIRI" type="tns:AttributedURIType"/>
2515
2516 <xs:element name="ProblemAction" type="tns:ProblemActionType"/>
2517 <xs:complexType name="ProblemActionType" mixed="false">
2518   <xs:sequence>
2519     <xs:element ref="tns:Action" minOccurs="0"/>
2520     <xs:element name="SoapAction" minOccurs="0" type="xs:anyURI"/>
2521   </xs:sequence>
2522   <xs:anyAttribute namespace="##other" processContents="lax"/>
2523 </xs:complexType>
2524
2525 </xs:schema>
2526
2527
2528
```

2529 F. wss-util-1.0.xsd Schema Listing

```
2530
2531     <?xml version="1.0" encoding="UTF-8"?>
2532 <!--
2533 OASIS takes no position regarding the validity or scope of any intellectual property or other
2534 rights that might be claimed to pertain to the implementation or use of the technology described
2535 in this document or the extent to which any license under such rights might or might not be
2536 available; neither does it represent that it has made any effort to identify any such rights.
2537 Information on OASIS's procedures with respect to rights in OASIS specifications can be found
2538 at the OASIS website. Copies of claims of rights made available for publication and any
2539 assurances of licenses to be made available, or the result of an attempt made to obtain a
2540 general license or permission for the use of such proprietary rights by implementors or users
2541 of this specification, can be obtained from the OASIS Executive Director.
2542 OASIS invites any interested party to bring to its attention any copyrights, patents or
2543 patent applications, or other proprietary rights which may cover technology that may be
2544 required to implement this specification. Please address the information to the OASIS Executive Director.
2545
2546 Copyright © OASIS Open 2002-2004. All Rights Reserved.
2547 This document and translations of it may be copied and furnished to others, and derivative
2548 works that comment on or otherwise explain it or assist in its implementation may be prepared,
2549 copied, published and distributed, in whole or in part, without restriction of any kind,
2550 provided that the above copyright notice and this paragraph are included on all such copies
2551 and derivative works. However, this document itself does not be modified in any way, such
2552 as by removing the copyright notice or references to OASIS, except as needed for the purpose
2553 of developing OASIS specifications, in which case the procedures for copyrights defined
2554 in the OASIS Intellectual Property Rights document must be followed, or as required to
2555 translate it into languages other than English.
2556 The limited permissions granted above are perpetual and will not be revoked by OASIS
2557 or its successors or assigns.
2558 This document and the information contained herein is provided on an "AS IS" basis
2559 and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
2560 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
2561 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
2562 -->
2563 <xsd:schema targetNamespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-u
2564 tility-1.0.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2565
2566
2567
2568 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-uti
2569 lity-1.0.xsd" xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-u
2570 tility-1.0.xsd"
2571 elementFormDefault="qualified" attributeFormDefault="unqualified" version="0.1">
2572 <!-- // Fault Codes ////////////////////////////////////// -->
2573 <xsd:simpleType name="tTimestampFault">
2574 <xsd:annotation>
2575 <xsd:documentation>
2576 This type defines the fault code value for Timestamp message expiration.
2577 </xsd:documentation>
2578 </xsd:annotation>
2579 <xsd:restriction base="xsd:QName">
2580 <xsd:enumeration value="wsu:MessageExpired"/>
2581 </xsd:restriction>
2582 </xsd:simpleType>
2583 <!-- // Global attributes ////////////////////////////////////// -->
2584 <xsd:attribute name="Id" type="xsd:ID">
2585 <xsd:annotation>
2586 <xsd:documentation>
2587 This global attribute supports annotating arbitrary elements with an ID.
2588 </xsd:documentation>
2589 </xsd:annotation>
2590 </xsd:attribute>
2591 <xsd:attributeGroup name="commonAtts">
2592 <xsd:annotation>
2593 <xsd:documentation>
2594 Convenience attribute group used to simplify this schema.
```

```
2595         </xsd:documentation>
2596     </xsd:annotation>
2597     <xsd:attribute ref="wsu:Id" use="optional" />
2598     <xsd:anyAttribute namespace="##other" processContents="lax" />
2599 </xsd:attributeGroup>
2600 <!-- // Utility types ////////////////////////////////////// -->
2601 <xsd:complexType name="AttributedDateTime">
2602     <xsd:annotation>
2603         <xsd:documentation>
2604 This type is for elements whose [children] is a psuedo-dateTime and can have arbitrary attributes.
2605         </xsd:documentation>
2606     </xsd:annotation>
2607     <xsd:simpleContent>
2608         <xsd:extension base="xsd:string">
2609             <xsd:attributeGroup ref="wsu:commonAtts" />
2610         </xsd:extension>
2611     </xsd:simpleContent>
2612 </xsd:complexType>
2613 <xsd:complexType name="AttributedURI">
2614     <xsd:annotation>
2615         <xsd:documentation>
2616 This type is for elements whose [children] is an anyURI and can have arbitrary attributes.
2617         </xsd:documentation>
2618     </xsd:annotation>
2619     <xsd:simpleContent>
2620         <xsd:extension base="xsd:anyURI">
2621             <xsd:attributeGroup ref="wsu:commonAtts" />
2622         </xsd:extension>
2623     </xsd:simpleContent>
2624 </xsd:complexType>
2625 <!-- // Timestamp header components ////////////////////////////////////// -->
2626 <xsd:complexType name="TimestampType">
2627     <xsd:annotation>
2628         <xsd:documentation>
2629 This complex type ties together the timestamp related elements into a composite type.
2630         </xsd:documentation>
2631     </xsd:annotation>
2632     <xsd:sequence>
2633         <xsd:element ref="wsu:Created" minOccurs="0" />
2634         <xsd:element ref="wsu:Expires" minOccurs="0" />
2635         <xsd:choice minOccurs="0" maxOccurs="unbounded">
2636             <xsd:any namespace="##other" processContents="lax" />
2637         </xsd:choice>
2638     </xsd:sequence>
2639     <xsd:attributeGroup ref="wsu:commonAtts" />
2640 </xsd:complexType>
2641 <xsd:element name="Timestamp" type="wsu:TimestampType">
2642     <xsd:annotation>
2643         <xsd:documentation>
2644 This element allows Timestamps to be applied anywhere element wildcards are present ,
2645 including as a SOAP header.
2646         </xsd:documentation>
2647     </xsd:annotation>
2648 </xsd:element>
2649 <!-- global element decls to allow individual elements to appear anywhere -->
2650 <xsd:element name="Expires" type="wsu:AttributedDateTime">
2651     <xsd:annotation>
2652         <xsd:documentation>
2653 This element allows an expiration time to be applied anywhere element wildcards are present.
2654         </xsd:documentation>
2655     </xsd:annotation>
2656 </xsd:element>
2657 <xsd:element name="Created" type="wsu:AttributedDateTime">
2658     <xsd:annotation>
2659         <xsd:documentation>
2660 This element allows a creation time to be applied anywhere element wildcards are present.
2661         </xsd:documentation>
```

```
2662         </xsd:annotation>
2663     </xsd:element>
2664 </xsd:schema>
2665
2666
```