



Liberty ID-WSF SOAP Binding Specification

Version: 2.0

Editors:

Jeff Hodges, NeuStar, Inc.
John Kemp, Nokia Corporation
Robert Aarts, Hewlett-Packard
Greg Whitehead, Hewlett-Packard
Paul Madsen, NTT

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: liberty-idwsf-soap-binding-v2.0.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.** Implementers
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 © 2006 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; UTI; VeriSign, Inc.; Vodafone
27 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	8
39	2.3. Treatment of Boolean Values	10
40	2.4. String and URI Values	10
41	2.5. Time Values	10
42	3. Schema Particulars	11
43	3.1. Schema Declarations	11
44	3.2. "ID" Attributes	11
45	3.3. Status Types	11
46	3.3.1. Status Codes	11
47	3.4. SOAP Fault Types	13
48	4. SOAP Binding	15
49	4.1. SOAP Version	15
50	4.2. The SOAPAction HTTP Header	15
51	4.3. Ordinary ID-* Messages	15
52	4.4. ID-* Fault Messages	15
53	4.5. SOAP-bound ID-* Messages	16
54	5. Messaging-specific Header Blocks	19
55	5.1. The <wsu:Timestamp> element in the <wsse:Security> Header Block	19
56	5.2. The <wsa:MessageID> Header Block	19
57	5.2.1. <wsa:MessageID> Value Requirements	19
58	5.3. The <wsa:RelatesTo> Header Block	19
59	5.4. The <wsa:To> Header Block	19
60	5.5. The <wsa:Action> Header Block	20
61	5.6. The <wsa:ReplyTo> Header Block	20
62	5.7. The <wsa:FaultTo> Header Block	20
63	5.8. The <wsa:Framework> Header Block	20
64	5.9. The <Sender> Header Block	21
65	5.10. The <TargetIdentity> Header Block	22
66	5.11. Messaging Processing Rules	23
67	5.11.1. Constructing and Sending a SOAP-bound ID-* Message	24
68	5.11.2. Receiving and Processing a SOAP-bound ID-* Message	25
69	5.12. Examples	29
70	6. Optional Header Blocks	32
71	6.1. The <ProcessingContext> Header Block	32
72	6.1.1. The <ProcessingContext> Type and Element	32
73	6.1.2. <ProcessingContext> Header Block Semantics and Processing Rules	33
74	6.2. The <Consent> Header Block	35
75	6.2.1. The <Consent> Type and Element	35
76	6.3. The <CredentialsContext> Header Block	36
77	6.3.1. Overview	36
78	6.3.2. CredentialsContext Type and Element	37
79	6.3.3. CredentialsContext Example	37
80	6.3.4. Processing Rules	38
81	6.4. The <EndpointUpdate> Header Block	38
82	6.4.1. Overview	39
83	6.4.2. EndpointUpdate Type and Element	39
84	6.4.3. EndpointUpdate Examples	39
85	6.4.4. Processing Rules for the EndpointUpdate header	42
86	6.4.5. Processing Rules for the EndpointUpdated SOAP Fault	43

87	6.5. The <Timeout> Header Block	43
88	6.5.1. Overview	43
89	6.5.2. Timeout Type and Element	44
90	6.5.3. Timeout Example	44
91	6.5.4. Processing Rules	45
92	6.6. The <UsageDirective> Header Block	46
93	6.6.1. Overview	46
94	6.6.2. UsageDirective Type and Element	46
95	6.6.3. Usage Directive Examples	47
96	6.6.4. Processing Rules	48
97	6.7. The <ApplicationEPR> Header Block	48
98	6.8. The <UserInteraction> Header Block	49
99	6.8.1. Overview	49
100	6.8.2. UserInteraction Element	49
101	6.8.3. UserInteraction Examples	50
102	6.8.4. Processing Rules	51
103	6.8.5. Cross-principal interactions	52
104	7. The RedirectRequest Protocol	53
105	7.1. RedirectRequest Element	53
106	7.1.1. Processing Rules	53
107	7.2. RedirectRequest Protocol	54
108	7.2.1. Step 1: WSC Issues Normal ID-WSF Request	54
109	7.2.2. Step 2: WSP Responds with <RedirectRequest>	54
110	7.2.3. Step 3: WSC Instructs User Agent to Contact the WSP	54
111	7.2.4. Step 4: WSP Interacts with User Agent	55
112	7.2.5. Step 5: WSP Redirects User Agent Back to WSC	55
113	7.2.6. Step 6: User Agent Requests ReturnToURL from WSC	55
114	7.2.7. Step 7: WSC Resends Message	55
115	7.2.8. Steps 8: WSP sends response	55
116	7.2.9. Steps 9: WSC sends HTTP response to User Agent	56
117	8. Security Considerations	57
118	9. Acknowledgements	58
119	References	59
120	A. liberty-idwsf-soap-binding.xsd Schema Listing	61
121	B. liberty-idwsf-soap-binding-v2.0.xsd Schema Listing	62
122	C. liberty-idwsf-utility-v2.0.xsd Schema Listing	65
123	D. liberty-utility-v2.0.xsd Schema Listing	67
124	E. wss-util-1.0.xsd Schema Listing	69
125	F. ws-addr-1.0.xsd Schema Listing	72

126 1. Introduction

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

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

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

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

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

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

145 • Processing Context:

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

149 • Consent Claims:

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

153 • Credentials Context:

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

157 • Endpoint Update:

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

160 • Timeout:

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

163 • Usage Directives:

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

167 • Application EPR:

168 This specification defines the <ApplicationEPR> header block as a means for a sender to specify application
169 endpoints that may be referenced from the SOAP Body of the message.

170 • User Interaction:

171 A WSC that interacts with a user (typically through a web-site offered by the WSC) may need to indicate its
172 readiness to redirect the user agent of the user, or its readiness to pose questions to the user on behalf of other
173 parties (such as WSPs). This specification defines the <UserInteraction> header block for this purpose.

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

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

181 2. Notation and Conventions

182 This specification uses schema documents conforming to W3C XML Schema [[Schema1-2](#)] and normative text to
183 describe the syntax and semantics of XML-encoded protocol messages.

184 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT",
185 "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119](#)]:

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

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

191 2.1. XML Namespaces

192 This specification makes normative use of the XML namespace prefixes noted in [Table 1](#).

193

Table 1. XML Namespaces and Prefixes

Prefix	Namespace
sb:	Represents the Liberty SOAP Binding namespace (v2.0): <code>urn:liberty:sb:2006-08</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.
sbf:	Represents the Liberty SOAP Binding namespace (cross-version framework): <code>urn:liberty:sb</code> Note This is the point of definition of this namespace.
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].
samlp:	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 [Schema1-2].

194 2.2. Terminology

195 This section defines key terminology used in this specification. Definitions for other Liberty-specific terms can be
196 found in [LibertyGlossary]. See also [RFC2828] for overall definitions of security-related terms.

197 affiliation An *affiliation* is a set of one or more entities, described by *Provider IDs*, who may perform
198 Liberty interactions as a member of the set. An affiliation is referenced by exactly one
199 *Affiliation ID*, and is administered by exactly one entity identified by their Provider ID.
200 Members of an affiliation may invoke services either as a member of the affiliation—by virtue
201 of their Affiliation ID, or individually by virtue of their Provider ID [LibertyGlossary].

202	Affiliation ID	An <i>Affiliation ID</i> identifies an <i>affiliation</i> . It is schematically represented by the
203		<code>affiliationID</code> attribute of the <code><AffiliationDescriptor></code> metadata element
204		[LibertyMetadata].
205	client	A <i>role</i> assumed by a <i>system entity</i> which makes a request of another system entity, often
206		termed a <i>server</i> [RFC2828], i.e. a client is also a <i>sender</i> .
207	ID-*	A shorthand designator referring to the Liberty ID-WSF, ID-FF, and ID-SIS specification
208		sets. For example, one might say that the former specification sets are all part of the Liberty
209		ID-* specification suite.
210	ID-* header block	One of the header blocks defined in this specification, or defined in any of the other Liberty
211		ID-* specification suite.
212	ID-* message	Equivalent to <i>ordinary ID-* message</i> .
213	ID-* fault message	See Section 4.4 .
214	ID-SIS	Liberty Identity Service Interface specification set.
215	ID-WSF	Liberty Identity Web Services Framework specification set.
216	MEP	see Message Exchange Pattern.
217	Message Exchange Pattern	A [SOAPv1.2] term for the overall notion of various patterns of message exchange
218		between SOAP nodes. For example, request-reply and one-way are two <i>MEPs</i> used in this
219		specification.
220	message thread	A <i>message thread</i> is an exchange of messages in a request-response <i>MEP</i> between two
221		<i>SOAP nodes</i> . All the messages of a given message thread are "linked" via each message's
222		<code><wsa:RelatesTo></code> header block value being set, by the sender, from the previous success-
223		fully received message's <code><wsa:MessageID></code> header block value.
224	Ordinary ID-* message	See Section 4.3 .
225	processing context	A <i>processing context</i> is the collection of specific circumstances under which a particular
226		processing step or set of steps take place.
227	processing context facet	A <i>processing context facet</i> is an identified aspect, inherent or additive, of a <i>processing</i>
228		<i>context</i> .
229	provider	A <i>provider</i> is a Liberty-enabled entity that performs one or more of the provider roles
230		in the Liberty architecture, for example Service Provider or Identity Provider. See also
231		<i>Liberty-enabled Provider</i> in [LibertyGlossary]. Providers are identified in Liberty protocol
232		interactions by their <i>Provider IDs</i> or optionally their <i>Affiliation ID</i> if they are a member of an
233		affiliation(s) and are acting in that capacity.
234	Provider ID	A <i>Provider ID</i> identifies an entity known as a <i>provider</i> . It is schematically represented by the
235		<code>providerID</code> attribute of the <code><EntityDescriptor></code> metadata element [LibertyMetadata].
236	receiver	A <i>role</i> taken by a <i>system entity</i> when it receives a message sent by another system entity. See
237		also <i>SOAP receiver</i> in [SOAPv1.2].
238	role	A function or part performed, especially in a particular operation or process [Merriam-
239		Webster].

240	sender	A <i>role</i> donned by a <i>system entity</i> when it constructs and sends a message to another system
241		entity. See also <i>SOAP sender</i> in [SOAPv1.2].
242	server	A <i>role</i> performed by a <i>system entity</i> that provides a service in response to requests from other
243		system entities called <i>clients</i> [RFC2828]. Note that in order to provide a service to clients; a
244		server will often be both a <i>sender</i> and a <i>receiver</i> .
245	service request	A <i>service request</i> is another term for an <i>ordinary ID-* message</i> . Service request is also
246		loosely equivalent to a "SOAP-bound (ordinary) ID-* message".
247	SOAP-bound ID-* message	See Section 4.5.
248	SOAP header block	A [SOAPv1.2] term whose definition is: An [element] used to delimit data that logically
249		constitutes a single computational unit within the SOAP header. In [SOAPv1.1] these are
250		known as simply <i>SOAP headers</i> , or simply <i>headers</i> . This specification uses the SOAPv1.2
251		terminology.
252	SOAP message	In this specification, the term <i>SOAP message</i> refers to a message consisting of only a
253		<S:Envelope> element as defined in [SOAPv1.1]. It contains two top-level subelements:
254		<S:Header> and <S:Body>. This message is in turn mapped onto a lower-layer transport or
255		transfer protocol, typically HTTP [RFC2616].
256	SOAP node	A [SOAPv1.2] term describing <i>system entities</i> who are parties to SOAP-based message
257		exchanges that are, for purposes of this specification, also the ultimate destination of the
258		exchanged messages, i.e. <i>SOAP endpoints</i> . In [SOAPv1.1], SOAP nodes are referred to as
259		<i>SOAP endpoints</i> , or simply <i>endpoints</i> . This specification uses the SOAPv1.2 terminology.
260	system entity	An active element of a computer/network system. For example, an automated process or set
261		of processes, a subsystem, a person or group of persons that incorporates a distinct set of
262		functionality [SAMLGloss2].

263 2.3. Treatment of Boolean Values

264 For readability, when an XML Schema type is specified to be `xsd:boolean`, this document discusses the values as
265 `TRUE` and `FALSE` rather than "1" and "0", which will exist in a document instance conforming to the SOAP Envelope
266 1.1 schema [SOAPv1.1-Schema].

267 2.4. String and URI Values

268 All string and URI [RFC3986] values in this specification have the types `string` (as a base type in this case) and
269 `anyURI` respectively, which are built in to the W3C XML Schema Datatypes specification [Schema2-2]. All strings
270 in ID-WSF messages MUST consist of at least one non-whitespace character (whitespace is defined in the XML
271 Recommendation [XML] section 2.3). Empty and whitespace-only values are disallowed. Also, unless otherwise
272 indicated in this specification, all URI values MUST consist of at least one non-whitespace character.

273 Note

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

277 2.5. Time Values

278 All time values in this specification have the type `dateTime`, which is built in to the W3C XML Schema Datatypes
279 specification [Schema2-2] and MUST be expressed in UTC form.

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

282 3. Schema Particulars

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

285 3.1. Schema Declarations

286 This specification normatively defines and describes an XML schema which is constituted in the XML Schema
287 [Schema1-2] files ("Liberty ID-WSF SOAP Binding Schema v2.0", reproduced in [Appendix A](#)). In addition, the
288 Liberty ID-WSF SOAP Binding Schema file explicitly includes, in the XML Schema sense, the Liberty ID-WSF
289 utility schema file (reproduced in [Appendix C](#)).

290 Also, the Liberty ID-WSF SOAP Binding Schema files explicitly depend upon the SOAP Message Security Utility
291 1.0 schema [wss-sms] (reproduced in [Appendix E](#)) and Web Services Addressing 1.0 schema [WSAv1.0-Schema]
292 (reproduced in [Appendix F](#)).

293 3.2. "ID" Attributes

294 The XML Schema [Schema1-2] type `xs:ID` is used to declare *ID* attributes on elements, such as SOAP header blocks,
295 that must be referenceable, say by an XML Signature [[LibertySecMech](#)]. It should be noted that XML processors,
296 such as XML Signature verifiers, must be aware of the `xs:ID` type of these ID attributes in order resolve references
297 to the elements they identify.

298 In this specification, as in Web Services Security and Web Services Addressing specifications on which this specifi-
299 cation builds, `xs:anyAttribute` is used on all elements that must be capable of carrying an ID attribute. Interoper-
300 ability profiles such as the ID-WSF SCR [[LibertyIDWSF20SCR](#)] may require use of a particular ID attribute such as
301 `xml:id`. In the absence of such profile requirements `wsu:Id` [[wss-sms](#)] MUST be used.

302 3.3. Status Types

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

```
306
307 <xs:complexType name="StatusType">
308   <xs:annotation>
309     <xs:documentation>
310       A type that may be used for status codes.
311     </xs:documentation>
312   </xs:annotation>
313   <xs:sequence>
314     <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
315   </xs:sequence>
316   <xs:attribute name="code" type="xs:string" use="required"/>
317   <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
318   <xs:attribute name="comment" type="xs:string" use="optional"/>
319 </xs:complexType>
320
321 <xs:element name="Status" type="StatusType">
322   <xs:annotation>
323     <xs:documentation>
324       A standard Status type
325     </xs:documentation>
326   </xs:annotation>
327 </xs:element>
328
329
330
```

331

Figure 1. status and statusType Schema

332 **3.3.1. Status Codes**

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

335

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:Client
TargetIdentityNotValid	The receiver does not consider the target identity to be valid.	S:Client
FrameworkVersionMismatch	The framework version used in the conveyed ID-* message does not match what was expected by the receiver.	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
EndpointUpdated	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
InteractionRequired	the recipient has a need to start an interaction in order to satisfy the service request but the <code>interact</code> attribute value was set to <code>DoNotInteract</code> .	S:Server
InteractionRequiredForData	the service request could not be satisfied because the WSP would have to interact with the requesting principal in order to obtain (some of) the requested data but the <code>interact</code> attribute value was set to <code>DoNotInteractForData</code> .	S:Server
InteractionTimeNotSufficient	the recipient has a need to start an interaction but has reason to believe that more time is needed that allowed for by the value of the <code>maxInteractTime</code> attribute.	S:Server
InteractionTimeout	the recipient could not satisfy the service request due to an unfinished interaction.	S:Server

336 3.4. SOAP Fault Types

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

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

340 Note

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

```
343
344 <xs:element name="Fault" type="tns:Fault"/>
345
346 <xs:complexType name="Fault" final="extension">
347   <xs:annotation>
348     <xs:documentation>
349       Fault reporting structure
350     </xs:documentation>
351 </xs:annotation>
352   <xs:sequence>
353     <xs:element name="faultcode" type="xs:QName"/>
354     <xs:element name="faultstring" type="xs:string"/>
355     <xs:element name="faultactor" type="xs:anyURI" minOccurs="0"/>
356     <xs:element name="detail" type="tns:detail" minOccurs="0"/>
357   </xs:sequence>
358 </xs:complexType>
359
360 <xs:complexType name="detail">
361   <xs:sequence>
362     <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
363   </xs:sequence>
364   <xs:anyAttribute namespace="##any" processContents="lax"/>
365 </xs:complexType>
366
367
```

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

369 4. SOAP Binding

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

373 4.1. SOAP Version

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

376 4.2. The SOAPAction HTTP Header

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

379 Note

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

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

384 4.3. Ordinary ID-* Messages

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

```
390  
391     <idpp:Query>  
392     :  
393     <!-- various message-specific subelements may go here -->  
394     :  
395     </idpp:Query>  
396
```

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

398 4.4. ID-* Fault Messages

399 An *ID-* Fault Message* consists of a SOAP <S:Fault> element (see [Section 3.4: SOAP Fault Types](#)) constructed as
400 specified herein.

401 When reporting a SOAP processing error such as "S:VersionMismatch" or "S:MustUnderstand", the <S:Fault>
402 element SHOULD be constructed according to [\[SOAPv1.1\]](#).

403 When reporting a WS-Addressing processing error such as "wsa:InvalidAddress", the <S:Fault> element
404 SHOULD be constructed according to [\[WSAv1.0-SOAP\]](#).

405 For all other processing errors the <S:Fault> element's attributes and child elements MUST be constructed according
406 to these rules:

- 407 1. The `<S:Fault>` element:
- 408 A. SHOULD contain a `<faultcode>` element whose value SHOULD be one of "sbf:FrameworkVersionMismatch",
409 "S:server" or "S:client".
- 410 B. SHOULD contain a `<faultstring>` element. This string value MAY be localized.
- 411 C. SHOULD NOT contain a `<S:faultactor>` element.
- 412 2. The `<S:Fault>` element's `<detail>` child element SHOULD contain a `<Status>` element (see [Section 3.3:](#)
413 [Status Types](#)). The `<Status>` element:
- 414 A. MUST contain a `code` attribute set to the value as specified when the issuance of a ID-* Fault message
415 is indicated. Code attribute values defined in this specification are listed above in [Section 3.3.1](#). Other
416 specifications MAY define additional code attribute values.
- 417 B. MAY contain a `ref` attribute set to the value as specified in this specification when the issuance of a ID-*
418 Fault message is indicated.
- 419 C. MAY contain a `comment` attribute set to the value as specified in this specification when the issuance of a
420 ID-* Fault message is indicated. This string value MAY be localized.
- 421 3. Additionally, to aid in diagnostics, the header block or message body element referred to by the fault MAY be
422 included in the `<S:Fault>` element's `<detail>` element, after the `<Status>` element.

423 4.5. SOAP-bound ID-* Messages

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

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

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

- 431 • having all required ID-* header blocks in its `<S:Header>` element, and,
- 432 • perhaps having other optional ID-* header blocks in its `<S:Header>` element, and,
- 433 • containing either an ordinary ID-* message, or an ID-* fault message, in its `<S:Body>` element. The former is
434 known as an *ordinary SOAP-bound ID-* message* (see [Example 2](#)), and the latter is known as a *SOAP-bound ID-**
435 *fault message* (see [Example 3](#)).

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

```
438
439 <S:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
440   xmlns:sb="..."
441   xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
442
443   <S:Header>
444
445     ...
446
447     <wsse:Security>
448       <wsu:Timestamp>
449         <wsu:Created>2005-06-17T04:49:17Z</wsu:Created>
450       </wsu:Timestamp>
451     </wsse:Security>
452
453     <wsa:MessageId>...</wsa:MessageId>
454
455     <wsa:To>...</wsa:To>
456
457     <wsa:Action>...</wsa:Action>
458
459     <!-- reference params from target EndpointReference -->
460
461     <sbf:Framework version="2.0"/>
462
463     <sb:Sender providerID="..." affiliationID="..."/>
464
465     <wsa:ReplyTo>
466       <wsa:Address>...</wsa:Address>
467     </wsa:ReplyTo>
468
469     ...
470
471   </S:Header>
472
473   <S:Body>
474
475     <idpp:Query> <!-- This is an ID-PP "Query" message bound -->
476       :   <!-- into the <S:Body> of a SOAP message.   -->
477       :
478     </idpp:Query>
479
480   </S:Body>
481
482 </S:Envelope>
```

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

```

484
485 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
486   xmlns:sb="..."
487   xmlns:pp="urn:liberty:id-sis-pp:2003-08">
488
489   <S:Header>
490
491     ...
492
493     <wsse:Security>
494       <wsu:Timestamp>
495         <wsu:Created>2005-06-17T04:49:18Z</wsu:Created>
496       </wsu:Timestamp>
497     </wsse:Security>
498
499     <wsa:MessageId>...</wsa:MessageId>
500
501     <wsa:RelatesTo>...</wsa:RelatesTo>
502
503     <wsa:To>...</wsa:To>
504
505     <wsa:Action>...</wsa:Action>
506
507     <!-- reference params from FaultTo/ReplyTo EndpointReference -->
508
509     <sbf:Framework version="2.0"/>
510
511     <sb:Sender providerID="..." />
512
513     ...
514
515   </S:Header>
516
517   <S:Body>
518
519     <S:Fault>
520       <faultcode>S:server</faultcode>
521       <faultstring>Server Error</faultstring>
522       <!-- <S:faultactor> should be absent -->
523
524       <detail>
525         <lu:Status code="SomeStatus"
526           ref="Foo"
527           comment="Bar" />
528       </detail>
529     </S:Fault>
530
531   </S:Body>
532
533 </S:Envelope>
534

```

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

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

537 This section profiles the use of WS-Addressing SOAP Binding [WSAv1.0-SOAP] and WS-Security [wss-sms] header
538 blocks, as well as defining several new ID-* header blocks, to implement the ID-* message exchange model.

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

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

542 **Note**

543 Other ID-* specifications MAY define additional ID-* header blocks.

544 **5.1. The <wsu:Timestamp> element in the <wsse:Security> Header** 545 **Block**

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

549 **Note**

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

552 **5.2. The <wsa:MessageID> Header Block**

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

555 **5.2.1. <wsa:MessageID> Value Requirements**

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

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

560 The mechanism by which SOAP-based ID-* senders or receivers ensure that an identifier is unique is left to
561 implementations. In the case that a pseudorandom technique is employed, the above requirement MAY be met by
562 randomly choosing a value 160 bits in length.

563 Note that [WSAv1.0] requires that <wsa:MessageID> values be absolute IRIs.

564 **5.3. The <wsa:RelatesTo> Header Block**

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

568 **Note**

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

571 **5.4. The `<wsa:To>` Header Block**

572 The `<wsa:To>` header block is defined in [[WSAv1.0-SOAP](#)]. The value of this header block specifies the intended
573 destination of the message.

574 **Note**

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

580 **5.5. The `<wsa:Action>` Header Block**

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

583 **Note**

584 The value of this header block SHOULD the same value as the SOAPAction HTTP header (see [Section 4.2: The](#)
585 SOAPAction HTTP Header).

586 **5.6. The `<wsa:ReplyTo>` Header Block**

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

589 Note

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

593 5.7. The <wsa:FaultTo> Header Block

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

598 5.8. The <sbf:Framework> Header Block

599 This section defines the <sbf:Framework> header block. When included in a message, this header provides a means
600 for a sender to communicate the version of the ID-WSF framework used to construct the message.

601 Framework versions are defined in ID-WSF SCR documents, such as [LibertyIDWSF20SCR].

602 The schema fragment in [Figure 3](#) defines the <sbf:Framework> header block.

```
603  
604  
605 <!-- framework header block -->  
606  
607 <xs:complexType name="FrameworkType">  
608   <xs:sequence>  
609     <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>  
610   </xs:sequence>  
611   <xs:attribute name="version" type="xs:string" use="required"/>  
612   <xs:anyAttribute namespace="##other" processContents="lax"/>  
613 </xs:complexType>  
614  
615 <xs:element name="Framework" type="FrameworkType"/>  
616  
617  
618
```

619 **Figure 3. The <sbf:Framework> Header Block Schema**

```
620  
621 <sbf:Framework version="2.0" S:mustUnderstand="1"  
622   S:actor="http://schemas.../next"  
623   wsu:Id="A72139...381"/>  
624
```

625 **Example 4. An instantiated <sbf:Framework> header block**

626 5.9. The <Sender> Header Block

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

631 **Note**

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

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

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

- 638 • providerID [Required] – The Provider ID of the sender.
- 639 • affiliationID [Optional] – The Affiliation ID of the sender, if any.

640 The schema fragment in Figure 4 defines the <Sender> header block.

```
641  
642  
643 <!-- sender header block -->  
644  
645 <xs:complexType name="SenderType">  
646   <xs:attribute name="providerID" type="xs:anyURI" use="required"/>  
647   <xs:attribute name="affiliationID" type="xs:anyURI" use="optional"/>  
648   <xs:anyAttribute namespace="##other" processContents="lax"/>  
649 </xs:complexType>  
650  
651 <xs:element name="Sender" type="SenderType"/>  
652  
653  
654
```

655 **Figure 4. The <Sender> Header Block Schema**

```
656  
657 <sb:Sender S:mustUnderstand="1"  
658   S:actor="http://schemas.../next "  
659   wsu:Id="A72139...381"  
660   providerID="http://example.com"  
661   affiliationID="http://affiliation.com"/>  
662
```

663 **Example 5. An instantiated <Sender> header block**

664 5.10. The <TargetIdentity> Header Block

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

670 Note

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

673 The <TargetIdentity> header is typically only required in cross-principal scenarios such as when one user is
674 accessing the resources of another user.

675 The <TargetIdentity> header block has a content model of any.

676 The schema fragment in [Figure 5](#) defines the The <TargetIdentity> header block.

```
677
678
679 <!-- target identity header block -->
680
681 <xs:complexType name="TargetIdentityType">
682   <xs:sequence>
683     <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
684   </xs:sequence>
685   <xs:anyAttribute namespace="##other" processContents="lax"/>
686 </xs:complexType>
687
688 <xs:element name="TargetIdentity" type="TargetIdentityType"/>
689
690
691
```

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

```
693
694 <sb:TargetIdentity S:mustUnderstand="1"
695   S:actor="http://schemas.../next"
696   wsu:Id="A31739...293">
697   ...
698 </sb:TargetIdentity>
699
```

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

701 5.11. Messaging Processing Rules

702 Overall processing of SOAP-bound ID-* messages follows the rules of the SOAP processing model described in
703 [\[SOAPv1.1\]](#); specifically, the SOAP `mustUnderstand` and `actor` attributes MAY be used to mandate header block
704 processing and target header blocks, respectively. Where applicable, specific processing rules for these attributes are
705 given in the overall processing rules defined below.

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

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

712 The processing rules are described below in terms of [Constructing and Sending a SOAP-bound ID-* Message](#) and
713 [Receiving and Processing a SOAP-bound ID-* Message](#). A sender instigating a one-way message exchange will
714 perform only the steps outlined in the former section. A sender participating in a request-response message exchange

715 will perform the steps in the former section when sending a message, and the steps in the latter section when receiving
716 and processing the response. A receiver participating in a request-response exchange will do the reverse. Note that a
717 receiver of an asynchronous one-way message will perform the steps in the latter section.

718 **Note**

719 The label "ID-* header block(s)" is used to refer to at least one of, or all of, the following set of header blocks:

720 • <wsa:MessageID> [[WSAv1.0](#)]

721 • <wsa:RelatesTo> [[WSAv1.0](#)]

722 • <wsa:To> [[WSAv1.0](#)]

723 • <wsa:Action> [[WSAv1.0](#)]

724 • <wsa:ReplyTo> [[WSAv1.0](#)]

725 • <wsa:FaultTo> [[WSAv1.0](#)]

726 • <wsse:Security> [[LibertySecMech](#)]

727 • <sbf:Framework>

728 • <Sender>

729 • <TargetIdentity>

730 • <ProcessingContext>

731 • <Consent>

732 • <UsageDirective>

733 • <EndpointUpdate>

734 • <Timeout>

735 • <CredentialsContext>

736 • <ApplicationEPR>

737 • <UserInteraction>

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

741 **5.11.1. Constructing and Sending a SOAP-bound ID-* Message**

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

744 1. The outgoing message MUST satisfy the rules given in [Section 4: SOAP Binding](#).

745 2. The outgoing message MUST satisfy the rules given in [[WSAv1.0-SOAP](#)].

- 746 3. The outgoing message MUST include exactly one `<wsa:MessageID>` header block in the `<S:Header>` child
747 element of the `<S:Envelope>` element and its value SHOULD be set according to the rules presented in
748 [Section 5.2.1: `<wsa:MessageID>` Value Requirements](#) .
- 749 4. If the sender is participating in a request-response MEP and is
- 750 A. sending a request message, the outgoing message MUST include exactly one `<wsa:ReplyTo>` header block
751 and at most one `<wsa:FaultTo>` header block (if the `<wsa:FaultTo>` header block is not included, faults
752 will be delivered to the `<wsa:ReplyTo>` endpoint)
- 753 B. responding to a prior-received request message, the outgoing message MUST include exactly one
754 `<wsa:RelatesTo>` header block with `RelationshipType` equal to `http://www.w3.org/2005/03/`
755 `addressing/reply` in the `<S:Header>` child element of the `<S:Envelope>` element (note that this is
756 the default `RelationshipType` and so the attribute MAY be omitted). The value of this header block MUST
757 be set to the value of the `<wsa:MessageID>` header block from the prior-received message.
- 758 5. The outgoing message MUST include exactly one `<wsse:Security>` header block. The `<wsse:Security>`
759 header block MUST include a `<wsu:Timestamp>` element. The `<wsu:Timestamp>` element MUST include a
760 `<wsu:Created>` element, the value of which SHOULD be set to the time at which the message is prepared for
761 transmission. This value MUST conform to the rules presented in [Section 2.5: Time Values](#).
762 If no clock is available to the message sender then a time value of `1970-01-01T00:00:00Z` SHOULD be used.
- 763 6. The sender MUST include exactly one `<sbef:Framework>` header block in the `<S:Header>` child element of
764 the `<S:Envelope>` element. The `version` attribute of this `<sbef:Framework>` header element MUST be set to
765 ID-WSF version in use by the sender.
- 766 7. If the sender is acting in the role of a Liberty provider, the message MUST include exactly one `<Sender>` header
767 block in the `<S:Header>` child element of the `<S:Envelope>` element. The attributes of this `<Sender>` header
768 block MUST be set as follows:
- 769 A. `providerID` MUST be present and SHOULD be set to a value appropriate for the sender to claim
770 [\[LibertyMetadata\]](#).
- 771 B. `affiliationID` MAY be present. If so, it SHOULD be set to a value appropriate for the sender to claim
772 [\[LibertyMetadata\]](#).
- 773 8. The sender MAY include a `<TargetIdentity>` header block, as needed, to identify the target identity of the
774 message. The sender MUST NOT include more than one `<TargetIdentity>` header block.
- 775 9. The sender MAY include other ID-* header blocks in the message, in addition to those enumerated above,
776 as required by the overall messaging and processing context. For example, the sender may include a
777 `<wsse:Security>` header block [\[LibertySecMech\]](#).
- 778 10. The sender adds either:
- 779 A. an ordinary ID-* message (as described in [Section 4.3: Ordinary ID-* Messages](#); see [Example 2](#)), or,
780 B. an ID-* fault message (as **prescribed** in [Section 4.4: ID-* Fault Messages](#); see [Example 3](#)),
781 to the SOAP-bound ID-* message's `<S:Body>` element.
- 782 11. The sender also performs any needed additional preparation of the message, for example including other header
783 blocks, and signing some or all of the message elements, and then sends the message to the receiver. See
784 [Section 5.12: Examples](#) .

785 5.11.2. Receiving and Processing a SOAP-bound ID-* Message

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

788 **Note**

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

793 1. The incoming message MUST satisfy the rules given in [Section 4: SOAP Binding](#).

794 2. The incoming message MUST satisfy the rules given in [\[WSAv1.0-SOAP\]](#).

795 3. Processing specific to the `<sbf:Framework>` header block:

796 A. A single `<sbf:Framework>` header block MUST be present in the header of the message.

797 B. The value of the `version` attribute of the `<sbf:Framework>` header element MUST specify an ID-
798 WSF version supported by the receiver. Further processing MUST be according to the processing rules of
799 the specified version.

800 C. If the foregoing test (3.A) holds true, processing continues with step 4 .

801 D. Otherwise, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per [Section 4.4](#))
802 with the `<faultcode>` of `sbf:FrameworkVersionMismatch`.

803 The receiver MAY discard the incoming message. The receiver is finished processing this incoming message
804 at this point.

805 4. Processing specific to the `<wsa:MessageID>` and `<wsa:RelatesTo>` header blocks and the
806 `<wsu:Timestamp>` element in the `<wsse:Security>` header block:

807 A. A single `<wsse:Security>` header block MUST be present in the header of the message.
808 The `<wsse:Security>` header block MUST include a `<wsu:Timestamp>` element. The
809 `<wsu:Timestamp>` element MUST include a `<wsu:Created>` element.

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

812 If the `<wsu:Timestamp>` element includes an `<wsu:Expires>` element, the time at the receiver MUST
813 be before that time.

- 814 **Note**
- 815 Certain classes of client devices, such as consumer electronics, often do not have correctly set clocks. These
816 processing rules may be relaxed for messages received from such devices.
- 817 C. A single `<wsa:MessageID>` header block **MUST** be present in the header of the message.
- 818 D. If the `<wsa:RelatesTo>` header block with `RelationshipType` equal to `http://www.w3.org/2005/03/`
819 `addressing/reply` is present, and if the receiver is participating in a request-response MEP with the
820 sending party, then the value of the `<wsa:RelatesTo>` header block **SHOULD** match the value of the
821 `<wsa:MessageID>` header block of a message previously sent by the receiver to the sender of the now
822 incoming message.
- 823 E. If the foregoing tests (4.A through 4.D) hold true, processing continues with step 5 .
- 824 F. Otherwise, the receiver **MAY** respond to the sender with a SOAP-bound ID-* Fault message (per [Section 4.4](#))
825 with the `<Status>` element configured with:
- 826
 - a `code` attribute with a value of:
- 827
 - "IDStarMsgNotUnderstood" if the failed test is [4.A](#) or [4.C](#).
- 828
 - "StaleMsg" if the failed test is [4.B](#),
- 829
 - "InvalidRefToMsgID" if the failed test is [4.D](#),
- 830
 - and a `ref` attribute with its value taken from the `messageID` value of the incoming message.
- 831 The `<S:Fault>` **SHOULD** contain a `<faultcode>` of `S:Client`.
- 832 The receiver **MAY** discard the incoming message. The receiver is finished processing this incoming message
833 at this point.

834 **Note**

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

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

840 6. Processing specific to the <Sender> header block:

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

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

846 B. If the foregoing test (6.A) holds true, processing continues with step 7.

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

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

850 • "ProviderIDNotValid", or,

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

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

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

856 The receiver MAY discard the incoming message. The receiver is finished processing this incoming message
857 at this point.

858 7. Processing specific to the <TargetIdentity> header block:

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

861 B. If the foregoing test (7.A) holds true, processing continues with step 8.

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

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

865 • "TargetIdentityNotValid"

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

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

868 The receiver MAY discard the incoming message. The receiver is finished processing this incoming message
869 at this point.

870 8. All remaining ID-* header blocks SHOULD be processed at this point. See appropriate sections in this and other
871 specifications for the processing rules associated with these header blocks and the manner of reporting any issues
872 with this processing. If there are no issues with these header blocks, then processing continues with step 9 below,
873 otherwise the receiver is finished processing this incoming message at this point.

874 **Note**

875 It should be noted that the receiver MAY return an `InappropriateCredentials` based on their processing
876 of the `<wsse:Security>` header block, under conditions specified below in [Section 6.4](#) and [Section 6.3](#), in
877 addition to other conditions such as an expired credential (see [\[LibertySecMech\]](#)).

878 9. If the incoming message's applicable header blocks have passed all specified and applicable tests, the incoming
879 message SHOULD be dispatched for further processing as appropriate.

880 If the message contained in the encompassing SOAP message's `<S:Body>` element is not dispatchable, the
881 receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per [Section 4.4](#)) with the `<Status>`
882 element configured with:

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

884 • "IDStarMsgNotUnderstood"

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

886 Receivers MUST be able to avoid ID-* fault message "loops". For example, if the incoming message is conveying
887 an ID-* fault message, and there is some issue with one or more of its ID-* header blocks, the receiver should not
888 issue a SOAP-bound ID-* Fault message in response.

889 **Note**

890 Other specifications conforming to this binding that specify ordinary ID-* messages and their processing, such as
891 [\[LibertyIDPP\]](#) or [\[LibertyDisco\]](#), MAY define `<Status>` element code attribute values in addition to the ones defined
892 in [Section 3.3.1](#) of this document. These code attribute values SHOULD be used to signal to the sender any issues
893 with the incoming ordinary ID-* message found by the receiver. This specification does not define any such conditions
894 other than the one described above in [9](#), and they are not further discussed in this document.

895 **5.12. Examples**

896 [Example 7](#) illustrates a SOAP-bound ID-* message conveying a Personal Profile (ID-PP) Modify request message
897 [\[LibertyIDPP\]](#).

```

898
899     <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
900       xmlns:sbf="urn:liberty:sb"
901       xmlns:sb="urn:liberty:sb:2006-08"
902       xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
903
904     <S:Header>
905
906       ...
907
908     <wsse:Security>
909       <wsu:Timestamp>
910         <wsu:Created>2005-06-17T04:49:17Z</wsu:Created>
911       </wsu:Timestamp>
912     </wsse:Security>
913
914     <wsa:MessageID>http://spwsc.com/0123456789abcdef0123456789abcdef01234567</wsa:Mes
915 sageID>
916
917     <wsa:To>http://spwsp.com/idpp</wsa:To>
918
919     <wsa:Action>urn:liberty:id-sis-pp:2003-08:Modify</wsa:Action>
920
921     <!-- reference params from target EndpointReference -->
922
923     <sbf:Framework version="2.0"/>
924
925     <sb:Sender providerID="http://spwsc.com" affiliationID="http://affiliation.com"/>
926
927     <wsa:ReplyTo>
928       <wsa:Address>http://www.w3.org/2005/03/addressing/role/anonymous</wsa:Address>
929     </wsa:ReplyTo>
930
931     ...
932
933   </S:Header>
934
935   <S:Body>
936
937     <idpp:Modify>
938       : <!-- this is an ID-PP Modify message -->
939     </idpp:Modify>
940
941   </S:Body>
942
943 </S:Envelope>
944

```

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

946 [Example 8](#) illustrates a SOAP-bound ID-* response to the message in the previous example, which conveyed an ID-PP
 947 Modify message. Note how the <wsa:RelatesTo> header value references the <wsa:MessageID> in the example
 948 above.

```
949
950 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
951   xmlns:sbf="urn:liberty:sb"
952   xmlns:sb="urn:liberty:sb:2006-08"
953   xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
954
955   <S:Header>
956
957     ...
958
959     <wsse:Security>
960       <wsu:Timestamp>
961         <wsu:Created>2005-06-17T04:49:18Z</wsu:Created>
962       </wsu:Timestamp>
963     </wsse:Security>
964
965     <wsa:MessageID>http://spwsp.com/ffeeddccbbaa99887766554433221100ffeebbc
966 c</wsa:MessageID>
967     <wsa:RelatesTo>http://spwsc.com/0123456789abcdef0123456789abcdef01234567</wsa:Relate
968 sTo>
969
970     <wsa:Action>urn:liberty:id-sis-pp:2003-08:ModifyResponse</wsa:Action>
971
972     <sbf:Framework version="2.0"/>
973
974     <sb:Sender providerID="http://spwsp.com" />
975
976     ...
977
978   </S:Header>
979
980   <S:Body>
981
982     <idpp:ModifyResponse>
983       : <!-- this is an ID-PP ModifyResponse message -->
984     </idpp:ModifyResponse>
985
986   </S:Body>
987
988 </S:Envelope>
989
```

990

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

991 **6. Optional Header Blocks**

992 The optional header blocks described in this specification are:

- 993 • <ProcessingContext>
- 994 • <Consent>
- 995 • <CredentialsContext>
- 996 • <EndpointUpdate>
- 997 • <Timeout>
- 998 • <UsageDirective>
- 999 • <ApplicationEPR>
- 1000 • <UserInteraction>

1001 The following sections describe these optional ID-* header blocks along with their specific processing rules.

1002 **Note**

1003 Whenever an optional header block appears in a SOAP-bound ID-* message, the processing rules specific to that
1004 header block (which are given in this section, below) **MUST** be used in combination with the messaging processing
1005 rules given above in [Section 5.11: Messaging Processing Rules](#). This applies whether the message is being constructed
1006 and sent, or being received and processed.

1007 **6.1. The <ProcessingContext> Header Block**

1008 This section defines the <ProcessingContext> header block. This header block may be employed by a sender to
1009 signal to a receiver that the latter should add a specific additional facet to the overall *processing context* in which
1010 any action(s) are invoked as a result of processing any ID-* message also conveyed in the overall SOAP-bound ID-*
1011 message. The full semantics of this header block are described below in [Section 6.1.2: <ProcessingContext>](#)
1012 Header Block Semantics and Processing Rules .

1013 Processing context facets are denoted by URIs. URIs are assigned to denote specific processing context facets. This
1014 specification defines several such URIs below in [Section 6.1.2.2](#).

1015 **6.1.1. The <ProcessingContext> Type and Element**

1016 The <ProcessingContext> content model is anyURI.

1017 The schema fragment in [Figure 6](#) defines the <ProcessingContext> header block.

```
1018
1019
1020 <!-- processing context header block -->
1021
1022 <xs:complexType name="ProcessingContextType">
1023   <xs:simpleContent>
1024     <xs:extension base="xs:anyURI">
1025       <xs:anyAttribute namespace="##other" processContents="lax"/>
1026     </xs:extension>
1027   </xs:simpleContent>
1028 </xs:complexType>
1029
1030 <xs:element name="ProcessingContext" type="ProcessingContextType"/>
1031
1032
```

1033 **Figure 6. The <ProcessingContext> Header Block Schema**

```
1034
1035   <sb:ProcessingContext S:mustUnderstand="1">
1036     urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
1037   </sb:ProcessingContext>
1038
```

1039 **Example 9. An instantiated <ProcessingContext> header block**

1040 6.1.2. <ProcessingContext> Header Block Semantics and Processing Rules

1041 This section first describes the overall semantics of the <ProcessingContext> header block, then defines two
1042 *processing context facet URIs*, and concludes with defining specific processing rules.

1043 6.1.2.1. <ProcessingContext> Header Block Semantics

1044 The overall semantic of the <ProcessingContext> header block is:

1045 The <ProcessingContext> header block MAY be employed by a sender, who is acting in a web
1046 services client (WSC) role, to signal to a receiver, who is acting in a web services provider (WSP)
1047 role, that the latter should add a specific *processing context facet* to the overall *processing context*
1048 (see [Section 2.2: Terminology](#)) in which the *service request* is evaluated.

1049 The specific processing context facet being conveyed by the <ProcessingContext> header block is identified by
1050 the header block's URI element value.

1051 Such URIs are known as *processing context facet URIs*. An example of a processing context facet that may be signaled
1052 by such a URI is whether the principal should be considered to be online or not.

1053 An ID-WSF or ID-SIS WSP receiving a service request containing a <ProcessingContext> header block with
1054 one of the above processing context facet URIs SHOULD process the conveyed ID-* message **with the indicated**
1055 **processing context facet in force**. Thus the ID-* message's processing as well as any applicable access management
1056 policies are exercised within an overall processing context which includes the processing context facet. Finally, an
1057 indication of success or failure of the ID-* message processing is returned to the sender, in the same manner as would
1058 be done if the ID-* message had been sent without the attendant <ProcessingContext> header block.

1059 The above completely describes the semantic of this header block itself, and further description of particular effects
1060 on processing must be made in descriptions of processing context facet URIs. Such a description is given in the next
1061 section.

1062 **Note**

1063 Whether or not a receiver honors a <ProcessingContext> header block is a matter of local policy at the
1064 receiver, as is whether or not a receiver honors any given request from any given sender. For example, the
1065 <ProcessingContext> header block functionality has security implications in the sense of possibly facilitating
1066 an adversary to probe a receiver's behavior given adversary-chosen inputs. For these reasons, whether or not the
1067 <ProcessingContext> header block functionality is enabled on the part of a receiver with respect to a particular
1068 sender should be a matter of business-level agreement between the receiver and the sender.

1069 **6.1.2.2. Processing Context Facet URIs: PrincipalOnline, PrincipalOffline, and Simulate**

1070 Three processing context facet URIs are defined below for use with the <ProcessingContext> header block:

1071 urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
1072 Conduct the processing of the ID-* message as if the Principal is offline.

1073 urn:liberty:sb:2003-08:ProcessingContext:PrincipalOnline
1074 Conduct the processing of the ID-* message as if the Principal is online.

1075 urn:liberty:sb:2003-08:ProcessingContext:Simulate
1076 *Simulate* the processing of the ID-* message.

1077 If the sender includes a <UserInteraction> header block in addition to the <ProcessingContext> header block
1078 in the SOAP-bound ID-* request message, the receiver and sender **MUST** appropriately initiate the indicated user
1079 interaction, and incorporate information supplied by the user as a part of the resultant user interaction, into the
1080 appropriate data and/or policy stores.

1081 **Note**

1082 Any processing context facet that was conveyed in the <ProcessingContext> header block **MUST NOT** be enforced
1083 during such a user interaction. Rather, it applies only to the processing of the ID-* message itself.

1084 In summary, the overall intended side-effect of using the above-defined processing context facets is for the receiver to
1085 evaluate applicable policy, and return a putative indication of success or failure to the sender. This provides WSCs the
1086 capability to make an ID-WSF or ID-SIS service request and ascertain whether it will be successful or not—without
1087 the service request actually being carried out. Additionally, it facilitates carrying out any user interaction that may be
1088 indicated by the current combination of service request context and applicable policy. This will, for example, facilitate
1089 some WSCs to fashion more "user friendly" experiences.

1090 **6.1.2.3. Defining New Processing Context Facet URIs**

1091 The rightmost portions of the processing context facet URIs after the "ProcessingContext:" component are referred
1092 to as *processing context facet identifiers*. For example, whether the Principal is online or not is a facet of a request
1093 context. New processing context facet identifiers **MAY** be defined in other specifications, for example in ID-SIS data
1094 service specifications. An ID-SIS data service may define as many levels of request context identifiers as necessary
1095 to address the application's needs.

1096 **6.1.2.4. Sender Processing Rules**

1097 A sender **MAY** include a <ProcessingContext> header block in a SOAP-bound ID-* message. The sender **MUST**
1098 include a processing context facet URI in the <ProcessingContext> header block. The sender then sends the ID-*
1099 SOAP-bound message to an ID-WSF or ID-SIS service-hosting node (AKA the receiver).

1100 A sender **MAY** indicate that it believes either that the Principal is currently "online" or "offline" when it sends a
1101 message by specifying one of the two processing context facet URIs:

1102 • `urn:liberty:sb:2003-08:ProcessingContext:PrincipalOnline`

1103 • `urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline`

1104 The sender will typically receive a response from the receiver indicating success or failure or will receive a SOAP
1105 fault indicating a processing error with the SOAP-bound ID-* message. Note that in the case of a "successful" request
1106 *simulation*, the service will not return any result data other than an indication of success or failure to the sender.

1107 **6.1.2.5. Receiver Processing Rules**

1108 The receiver of a request containing a `<ProcessingContext>` header block **MUST** examine the included processing
1109 context facet URI. If it is known to the data service, then the data service **MUST** attempt to process the data service
1110 request, represented by the ID-* message, in an overall processing context including the processing context facet as
1111 indicated by the conveyed processing context facet URI, and return an indication of success or failure to the sender.

1112 If the data service request is malformed or has some other issue that would normally cause the receiver to issue a
1113 SOAP fault, the receiver **SHOULD** do so.

1114 If the receiver is asked to simulate processing of the request (by the inclusion of the
1115 `urn:liberty:sb:2003-08:ProcessingContext:Simulate` facet URI), and they are both able and willing to
1116 honor that processing context, then the receiver **MUST** evaluate the conveyed ID-* message, but **MUST NOT** actually
1117 perform the operation. That is, the receiver **MUST NOT** make actual changes to underlying ID-* service datastore,
1118 and it **MUST NOT** return any data as a result of evaluating the ID-* message.

1119 If the sender includes a `<UserInteraction>` header block, in addition to the `<ProcessingContext>` header
1120 block, then both participants **MUST** initiate the indicated user interaction appropriately, and incorporate infor-
1121 mation supplied by the user as a part of the interaction into appropriate data and/or policy stores, even if the
1122 `urn:liberty:sb:2003-08:ProcessingContext:Simulate` URI is specified in a `<ProcessingContext>`
1123 header.

1124 In the event the receiver does not understand the included processing context facet URI, the receiver **MAY** respond
1125 with a SOAP-bound ID-* fault message (per [Section 4.4: ID-* Fault Messages](#)) with the `<Status>` element configured
1126 with:

1127 • a `code` attribute with a value of:

1128 • `"ProcCtxURINotUnderstood"`

1129 • and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

1130 In the event the receiver is not willing to enforce a stipulated processing context, the receiver **MAY** respond with a
1131 SOAP-bound ID-* fault message (per [Section 4.4: ID-* Fault Messages](#)) with the `<Status>` element configured with:

1132 • a `code` attribute with a value of:

1133 • `"ProcCtxUnwilling"`

1134 • and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

1135 **Note**

1136 The receiver MAY reference multiple <ProcessingContext> headers in the <detail> of the fault response (in
 1137 accordance with the rules specified in [Section 4.4](#)).

1138 **6.2. The <Consent> Header Block**

1139 This section defines the <Consent> header block. This header block is used to explicitly claim that the Principal
 1140 consented to the present interaction.

1141 **6.2.1. The <Consent> Type and Element**

1142 The <Consent> header block element MAY be employed by either a sender or a receiver. For example, the Principal
 1143 may be using a Liberty-enabled client or proxy (common in the wireless world), and in that sort of environment the
 1144 mobile operator may cause the Principal's terminal (AKA: cell phone) to prompt the principal for consent for some
 1145 interaction.

1146 The <Consent> header block has the following attributes:

- 1147 • `uri` [Required] – A URI indicating that the Principal's consent was obtained.
 1148 Optionally, the URI MAY identify a particular *Consent Agreement Statement* defining the specific nature of the
 1149 consent obtained.
 1150 This specification defines one well-known URI Liberty implementors and deployers MAY use to indicate positive
 1151 Principal consent was obtained with respect to whatever ID-* interaction is underway or being initiated. This URI
 1152 is known as the "Principal Consent Obtained" URI (PCO). The value of this URI is:
 1153 `urn:liberty:consent:obtained`
 1154 This URI does not correspond to any particular Consent Agreement Statement. Rather, it simply states that consent
 1155 was obtained. The full meaning and implication of this will need to be derived from the execution context.
- 1156 • `timestamp` [Optional] – For denoting the time at which the sender obtained Principal consent with the POC.

1157 The schema fragment in [Figure 7](#) defines the Consent header block type.

```

1158
1159
1160 <!-- consent header block -->
1161
1162 <xs:complexType name="ConsentType">
1163   <xs:attribute name="uri" type="xs:anyURI" use="required"/>
1164   <xs:attribute name="timestamp" type="xs:dateTime" use="optional"/>
1165   <xs:anyAttribute namespace="##other" processContents="lax"/>
1166 </xs:complexType>
1167
1168 <xs:element name="Consent" type="ConsentType"/>
1169
1170
  
```

1171 **Figure 7. The <Consent> Header Block Schema**

```

1172
1173   <sb:Consent
1174     uri="urn:liberty:consent:obtained"
1175     timestamp="2112-03-15T11:12:10Z"/>
1176
  
```

1177 **Example 10. An instantiated <Consent> header block**

1178 **6.3. The <CredentialsContext> Header Block**

1179 **6.3.1. Overview**

1180 It may be necessary for an entity receiving an ID-* message to indicate the type of credentials that should be used by
1181 the sender in submitting a message.

1182 **6.3.2. CredentialsContext Type and Element**

1183 Receivers of an ID-* message MAY add <CredentialsContext> elements to the SOAP header of their response.

1184 The element is based upon the CredentialsContextType which is defined as:

- 1185 • `samlp:RequestedAuthnContext` [Optional] – a container that allows the expression of a requested authentication context (see [SAMLCore2]).
- 1186
- 1187 • `SecurityMechID` [Optional] – A set of elements that specify ID-WSF security mechanism URIs (see [Liberty-
1188 SecMech]).

1189 The following schema fragment describes the <CredentialsContext> header block.

```
1190  
1191  
1192 <!-- credentials context header block -->  
1193  
1194 <xs:complexType name="CredentialsContextType">  
1195   <xs:sequence>  
1196     <xs:element ref="samlp:RequestedAuthnContext" minOccurs="0" />  
1197     <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded" />  
1198   </xs:sequence>  
1199   <xs:anyAttribute namespace="##other" processContents="lax" />  
1200 </xs:complexType>  
1201  
1202 <xs:element name="CredentialsContext" type="CredentialsContextType" />  
1203  
1204  
1205
```

1206 **Figure 8. The <CredentialsContext> Header Block Schema**

1207 **6.3.3. CredentialsContext Example**

```
1208
1209 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1210         xmlns:sb="urn:liberty:sb:2006-08"
1211         xmlns:lib="urn:liberty:iff:2003-08">
1212
1213   <S:Header>
1214     ...
1215
1216     <!-- Says that the sender would like credentials that include RequestAuthnContext
1217           as specified -->
1218
1219     <sb:CredentialsContext mustUnderstand="1">
1220       <sampl:RequestedAuthnContext>
1221         ...
1222       </sampl:RequestedAuthnContext>
1223
1224     </sb:CredentialsContext>
1225
1226     ...
1227
1228   </S:Header>
1229
1230   <S:Body>
1231     <!-- a fault in the body indicates that the WSP's policy requires
1232           different (perhaps "stronger") credentials than were originally
1233           provided in the request -->
1234
1235     <S:Fault>
1236       <faultcode>S:Server</faultcode>
1237       <faultstring>
1238         Your request contained inappropriate credentials.
1239       </faultstring>
1240       <detail>
1241         <lu:Status code="InappropriateCredentials"/>
1242         <wsse:Security id="a6352...564"/>
1243       </detail>
1244     </S:Fault>
1245   </S:Body>
1246 </S:Envelope>
```

1252 **Example 11. A CredentialsContext Header Offered in Response to a Request with Inappropriate Credentials.**

1253 6.3.4. Processing Rules

1254 6.3.4.1. Sender Processing Rules

1255 A sender including this header MUST specify at least one RequestAuthnContext or one SecurityMechID.

1256 The SecurityMechID elements SHOULD be listed in order of preference by the sender.

1257 6.3.4.2. Receiver Processing Rules

1258 The receiver of a <CredentialsContext> header containing one or more SecurityMechID elements SHOULD
1259 use the highest-listed (first) SecurityMechID that it supports in future requests to the sender of this header.

1260 The receiver of a <CredentialsContext> header containing a RequestAuthnContext element SHOULD use
1261 credentials that conform to the policies specified therein in any future requests to the sender of this header (where
1262 credentials are required).

1263 **6.4. The <EndpointUpdate> Header Block**

1264 **6.4.1. Overview**

1265 It may be necessary for an entity receiving an ID-* message to indicate that messages from the sender should be
1266 directed to a different endpoint, or that they wish a different credential to be used than was originally specified
1267 by the entity for access to the requested resource. The <EndpointUpdate> response header allows a message
1268 receiver to indicate that a new endpoint or new credentials should be employed by the sender of the message on any
1269 subsequent messages. This header block may be used in conjunction with the <sb:InappropriateCredentials>
1270 and <sb:EndpointUpdated> faults, to indicate that the current message processing failed for those reasons, and
1271 should be submitted with the changes noted in any accompanying <EndpointUpdate> header block.

1272 **Note**

1273 The use of this header block allows the sender of the message to convey updates to security tokens, essentially
1274 providing a token renewal mechanism. This is not discussed further in this specification.

1275 **6.4.2. EndpointUpdate Type and Element**

1276 Receivers of an ID-* message may add an <EndpointUpdate> element to the SOAP header of their response.

1277 This element is based upon the EndpointUpdateType which is an extension of wsa:EndpointReferenceType
1278 that adds the following attributes:

1279 • updateType [Optional] – A URI attribute indicating whether the update should be interpreted as completely
1280 superseding the endpoint reference used to send the current request (the default) or whether it should be interpreted
1281 as a partial update.

1282 urn:liberty:sb:2006-08:EndpointUpdate:Complete
1283 A complete update.

1284 urn:liberty:sb:2006-08:EndpointUpdate:Partial
1285 A partial update. The complete updated endpoint reference is constructed according to the processing rules below.

1286 The following schema fragment describes the <EndpointUpdate> header block.

```
1287  
1288  
1289 <!-- epr update header block -->  
1290  
1291 <xs:complexType name="EndpointUpdateType">  
1292   <xs:complexContent>  
1293     <xs:extension base="wsa:EndpointReferenceType">  
1294       <xs:attribute name="updateType" type="xs:anyURI" use="optional"/>  
1295     </xs:extension>  
1296   </xs:complexContent>  
1297 </xs:complexType>  
1298  
1299 <xs:element name="EndpointUpdate" type="EndpointUpdateType"/>  
1300  
1301  
1302
```

1303 **Figure 9. The <EndpointUpdate> Header Block Schema**

1304 **6.4.3. EndpointUpdate Examples**

```
1305
1306
1307     1. Service responds to a request, indicating a new security mechanism and credential
1308
1309 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1310     xmlns:sb="urn:liberty:sb:2006-08"
1311     xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
1312
1313   <S:Header>
1314
1315     ...
1316
1317     <sb:EndpointUpdate mustUnderstand="1" updateType="urn:liberty:sb:2004-04:Partial">
1318       <wsa:Address>urn:liberty:sb:2006-08:EndpointUpdate:NoChange</wsa:Address>
1319       <wsa:Metadata>
1320         <ds:SecurityContext>
1321           <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</ds:SecurityMechID>
1322           <wsse:SecurityTokenReference>
1323             <wsse:Embedded>
1324               <wsse:BinarySecurityToken xmlns:wsse="..." wsu:Id="..."
1325                 ValueType="anyNSprefix:ServiceSessionContext">
1326                 ZjgzOWZlNzgyZTk1ZWU3OWEyMTRlODVmNGZkYzE4MmQ2ZDNhMzc3Nwo=
1327               </wsse:BinarySecurityToken>
1328             </wsse:Embedded>
1329           </wsse:SecurityTokenReference>
1330         </ds:SecurityContext>
1331       </wsa:Metadata>
1332     </sb:EndpointUpdate>
1333
1334     ...
1335   </S:Header>
1336
1337   <S:Body>
1338     <idpp:QueryResponse>
1339       ...
1340     </idpp:QueryResponse>
1341
1342   </S:Body>
1343
1344 </S:Envelope>
```

```
1347
1348
1349     2. The client sends a new request, using the contents of the EndpointUpdate
1350
1351 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1352     xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
1353
1354     <S:Header>
1355
1356         ...
1357
1358     <wsse:Security xmlns:wsse="...">
1359
1360         <wsse:BinarySecurityToken xmlns:wsse="..." wsu:Id="bst"
1361             ValueType="anyNSprefix:ServiceSessionContext">
1362             ZjgzOWZlNzgyZTk1ZWU3OWEyMTRlODVmNGZkYzE4MmQ2ZDZhMzc3Nwo=
1363         </wsse:BinarySecurityToken>
1364
1365     </wsse:Security>
1366
1367         ...
1368
1369     </S:Header>
1370
1371     <S:Body>
1372
1373         <idpp:Query>
1374             ...
1375         </idpp:Query>
1376
1377     </S:Body>
1378
1379 </S:Envelope>
1380
```

1381 **Example 12. An EndpointUpdate Specifying a ServiceSessionContext Token, and the TLS Bearer Security Mechanism.**

```

1382
1383 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1384         xmlns:sb="urn:liberty:sb:2006-08">
1385
1386   <S:Header>
1387
1388     ...
1389
1390     <sb:EndpointUpdate mustUnderstand="1" updateType="urn:liberty:sb:2004-04:Partial">
1391       <wsa:Address>http://example.com:443/soap</wsa:Address>
1392     </sb:EndpointUpdate>
1393
1394     ...
1395
1396   </S:Header>
1397
1398   <S:Body>
1399
1400     <!-- a fault in the body indicates that the request corresponding
1401          to this response should be re-submitted to the endpoint -->
1402
1403     <S:Fault>
1404
1405       <faultcode>S:Server</faultcode>
1406
1407       <faultstring>
1408         You must resubmit this request to the new endpoint.
1409       </faultstring>
1410
1411       <detail>
1412         <lu:Status code="EndpointUpdated"/>
1413       </detail>
1414
1415     </S:Fault>
1416
1417   </S:Body>
1418
1419 </S:Envelope>
1420

```

1421 **Example 13. An EndpointUpdate Specifying an Updated Address.**

1422 **6.4.4. Processing Rules for the EndpointUpdate header**

1423 **6.4.4.1. Sender Processing Rules**

1424 The receiver of an ID-* message MAY add an <EndpointUpdate> header block to their response.

1425 If updateType is not present or has the value urn:liberty:sb:2006-08:EndpointUpdate:Complete, the
 1426 <wsa:EndpointUpdate> MUST be a completely specified endpoint reference.

1427 If updateType has the value urn:liberty:sb:2006-08:EndpointUpdate:Partial, the <wsa:EndpointUpdate>
 1428 MAY omit any direct children of <wsa:ReferenceParameters> or <wsa:Metadata> that have not changed from
 1429 the original endpoint reference used to send the current request. Similarly, any extension elements that have not
 1430 changed MAY be omitted. If the address has not changed, then the URI
 1431 urn:liberty:sb:2006-08:EndpointUpdate:NoChange

1432 MAY be used in the <wsa:Address> value to indicate that the original address should continue to be used.

1433 **Note**

1434 The expressiveness of partial updates is limited. In particular, updates to `<wsa:ReferenceParameters>` and
1435 `<wsa:Metadata>` are done based on the qualified names of the direct children of those containers. If any child
1436 with a matching name is provided in the update, then all children with that name in the original are replaced. It is also
1437 impossible, with a partial update, to remove an element; elements may only be added or replaced.

1438 **6.4.4.2. Receiver Processing Rules**

1439 The receiver of an `<EndpointUpdate>` header SHOULD use the specified endpoint reference values to address any
1440 future requests to the sender of the header (where the endpoint reference used to address the request that resulted in
1441 the response containing the header would have been used), until newer information is obtained through this or some
1442 other mechanism or the updated information expires. If the updated information has a shorter lifetime than the current
1443 information (that it updates), then the current information SHOULD be retained as a fallback for when the updated
1444 information expires.

1445 If `updateType` is not present or has the value `urn:liberty:sb:2006-08:EndpointUpdate:Complete`, the
1446 `<wsa:EndpointUpdate>` is a completely specified endpoint reference.

1447 If `updateType` has the value `urn:liberty:sb:2006-08:EndpointUpdate:Partial`, the `<wsa:EndpointUpdate>`
1448 is a partially specified endpoint reference. The following steps are used to construct a complete endpoint reference
1449 from the endpoint reference that was used to address the request that resulted in the response containing this header:

- 1450 1. Take the `<wsa:Address>` from the `<wsa:EndpointUpdate>`. If the value is `urn:liberty:sb:2006-08:`
1451 `EndpointUpdate:NoChange`, then take the `<wsa:Address>` from the original endpoint reference.
- 1452 2. Take the `<wsa:ReferenceParameters>` from the `<wsa:EndpointUpdate>`, if present. Then, if
1453 `<wsa:ReferenceParameters>` is present in the original endpoint reference, take each direct child from
1454 that element that does not match an element already taken from the update (comparing the namespace qualified
1455 names of the elements).
- 1456 3. Take the `<wsa:Metadata>` from the `<wsa:EndpointUpdate>`, if present. Then, if `<wsa:Metadata>` is
1457 present in the original endpoint reference, take each direct child from that element that does not match an element
1458 already taken from the update (comparing the namespace qualified names of the elements).
- 1459 4. Take any extension elements from the `<wsa:EndpointUpdate>`, if present. Then, if any extension elements are
1460 present in the original endpoint reference, take each one that does not match an element already taken from the
1461 update (comparing the namespace qualified names of the elements).

1462 **6.4.5. Processing Rules for the EndpointUpdated SOAP Fault**

1463 **6.4.5.1. Sender Processing Rules**

1464 The receiver of an ID-* message MAY issue a SOAP Fault indicating that the endpoint to which this message was
1465 submitted has permanently changed.

1466 Once the receiver has sent this fault response, no further processing of the message should take place.

1467 If the receiver chooses to send the fault response, then it SHOULD also include an `<EndpointUpdate>` header,
1468 indicating the new endpoint which should be used to re-submit this message, and any further messages directed to the
1469 responding service.

1470 **6.4.5.2. Receiver Processing Rules**

1471 If the receiver of this fault response also received an `<EndpointUpdate>` header in the response, it MAY re-submit
1472 the failed request to any endpoint specified in that header, but it SHOULD provide a different `<wsa:MessageID>`
1473 header block value in the request.

1474 **6.5. The `<Timeout>` Header Block**

1475 **6.5.1. Overview**

1476 A requesting entity may wish to indicate that they would like a request to be processed within some specified amount
1477 of time. Such an entity would indicate their wish via the `<Timeout>` header block.

1478 **6.5.2. Timeout Type and Element**

1479 Senders of ID-* messages MAY add a `<Timeout>` element to the SOAP header of their request.

1480 This element is based upon the `TimeoutType` which is defined as:

- 1481 • `maxProcessingTime` [Required] – an integer specifying (in seconds) the maximum amount of time the sender
1482 wishes the receiver to spend in processing their request

1483 The following schema fragment describes the `<Timeout>` header block.

```
1484  
1485  
1486 <!-- timeout header block -->  
1487  
1488 <xs:complexType name="TimeoutType">  
1489   <xs:attribute name="maxProcessingTime" type="xs:integer" use="required"/>  
1490   <xs:anyAttribute namespace="##other" processContents="lax"/>  
1491 </xs:complexType>  
1492  
1493 <xs:element name="Timeout" type="TimeoutType"/>  
1494  
1495  
1496
```

1497 **Figure 10. The `<Timeout>` Header Block Schema**

1498 **6.5.3. Timeout Example**

```

1499
1500 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1501         xmlns:sb="urn:liberty:sb:2006-08"
1502         xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
1503
1504 <S:Header>
1505
1506     ...
1507
1508     <sb:Timeout mustUnderstand="1" wsu:Id="timeout.123"
1509         maxProcessingTime="7"/>
1510
1511     ...
1512
1513 </S:Header>
1514
1515 <S:Body>
1516
1517     <idpp:Query>
1518         ...
1519     </idpp:Query>
1520
1521 </S:Body>
1522
1523 </S:Envelope>
1524
  
```

1525 **Example 14. Example of a Request with Timeout Specified**

```

1526
1527 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1528         xmlns:sb="urn:liberty:sb:2006-08">
1529
1530 <S:Header>
1531
1532     ...
1533
1534 </S:Header>
1535
1536 <S:Body>
1537
1538     <S:Fault>
1539
1540         <faultcode>
1541             S:Server
1542         </faultcode>
1543
1544         <detail>
1545
1546             <lu:Status code="ProcessingTimeout"/>
1547
1548             <!-- Reference the specified Timeout header, if it was supplied
1549                 by the sender -->
1550
1551             <sb:Timeout wsu:Id="timeout.123"/>
1552
1553         </detail>
1554     </S:Fault>
1555
1556 </S:Body>
1557
1558 </S:Envelope>
1559
  
```

1560 **Example 15. Example of a Timed-out Response**

1561 **6.5.4. Processing Rules**

1562 **6.5.4.1. Receiver Processing Rules**

1563 The receiver of a `<Timeout>` header SHOULD NOT begin processing of a message (beyond processing of the
1564 SOAP headers as noted in this specification) if it expects that such processing would exceed the value specified in
1565 the `maxProcessingTime` attribute.

1566 The receiver MUST respond to the message within the number of seconds specified in the `maxProcessingTime`
1567 attribute.

1568 If the receiver is unable to complete processing within the number of seconds specified in the `maxProcessingTime` at-
1569 tribute of the `<Timeout>` header, then they MUST respond with a SOAP Fault with a `code` of `ProcessingTimeout`.

1570 **Note**

1571 If the sender of a message does not include a `<Timeout>` header, but the receiver wishes to indicate to the sender
1572 that server processing failed due to a timeout, then the receiver MAY respond with a SOAP Fault with a `code` of
1573 `ProcessingTimeout`.

1574 **6.6. The `<UsageDirective>` Header Block**

1575 This section defines the ID-* usage directive facilities.

1576 **6.6.1. Overview**

1577 Participants in the ID-WSF framework may need to indicate the privacy policy associated with a message. To facilitate
1578 this, senders, acting as either a client or a server, may add one or more `<UsageDirective>` header blocks to the
1579 SOAP Header of the message being sent. A `<UsageDirective>` appearing in a SOAP-based ID-* request message
1580 expresses *intended usage*. A `<UsageDirective>` appearing in a response expresses *how* the receiver of the response
1581 is to use the response data. A `<UsageDirective>` in a response message containing no ID-WSF response message
1582 data, a fault response for example, may be used to express policies acceptable to the responder.

1583 **6.6.2. UsageDirective Type and Element**

1584 Senders MAY add a `<UsageDirective>` element to the SOAP header. This element is based upon the
1585 `UsageDirectiveType` which is defined as:

- 1586 • `ref` [Required] – An attribute referring to an element of the SOAP-based ID-* message to which the usage directive
1587 applies.
- 1588 • `<element>(s)` [Optional] – Elements, comprising an instance of some policy expression language, whose
1589 purpose is to express the actual policy the usage directive is conveying. The `ref` attribute above points at the
1590 element in the overall SOAP-based ID-* message to which the usage directive applies.

1591 The schema fragment in [Figure 11](#) defines the <UsageDirective> header type and element.

```
1592
1593
1594 <!-- usage directive header block -->
1595
1596 <xs:complexType name="UsageDirectiveType">
1597   <xs:sequence>
1598     <xs:any namespace="##other" processContents="lax"
1599       maxOccurs="unbounded"/>
1600   </xs:sequence>
1601   <xs:attribute name="ref" type="xs:IDREF" use="required"/>
1602   <xs:anyAttribute namespace="##other" processContents="lax"/>
1603 </xs:complexType>
1604
1605 <xs:element name="UsageDirective" type="UsageDirectiveType"/>
1606
1607
```

1608 **Figure 11. The <UsageDirective> Header Block Schema**

1609 6.6.3. Usage Directive Examples

1610 [Example 16](#) illustrates a SOAP-based ID-* message, containing a <UsageDirective> header block, and
1611 conveying a Personal Profile (ID-PP) Modify message [[LibertyIDPP](#)]. The <UsageDirective> header block
1612 contains a usage directive expressed in a policy language identified by the cot: namespace and the URI
1613 <http://cot.example.com/policies/eu-compliant> , and applying to the ID-PP Query message identified by
1614 the id of datarequest001.

```
1615
1616 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1617   xmlns:sb="urn:liberty:sb:2006-08"
1618   xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1619
1620   <S:Header>
1621
1622     ...
1623
1624     <sb:UsageDirective S:mustUnderstand="1"
1625       ref="#datarequest001">
1626
1627       <cot:PrivacyPolicyReference
1628         xmlns:cot="http://cot.example.com/isf">
1629         http://cot.example.com/policies/eu-compliant
1630       </cot:PrivacyPolicyReference>
1631
1632     </sb:UsageDirective>
1633
1634     ...
1635
1636   </S:Header>
1637
1638   <S:Body>
1639
1640     <pp:Query id="datarequest001" xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1641       <pp:ResourceID>data:d8ddw6dd7m28v628</pp:ResourceID>
1642       <pp:QueryItem>
1643         <pp:Select>/pp:IDPP/pp:IDPPAddressCard</pp:Select>
1644       </pp:QueryItem>
1645     </pp:Query>
1646
1647   </S:Body>
1648
1649 </S:Envelope>
1650
```

1651 **Example 16. A Usage Directive on a Request for the Address of a Principal.**

1652 **6.6.4. Processing Rules**

1653 **6.6.4.1. Sender Processing Rules**

1654 The sender of a SOAP-based ID-* message with a <UsageDirective> header block **MUST** ensure that the value
1655 of the `ref` attribute is set to the value of the `id` of the appropriate element in the message. The sender **SHOULD**
1656 ensure that the <UsageDirective> is integrity-protected. The protection mechanism, if utilized, **SHOULD** be in
1657 accordance with those defined in [[LibertySecMech](#)].

1658 **6.6.4.2. Receiver Processing Rules**

1659 A receiver of a SOAP-based ID-* message with an attached <UsageDirective> header block **MUST** check the
1660 `actor` attribute and determine if it, the receiver, is the actor the header block is targeted at. If so, the receiver **MUST**
1661 check the `mustUnderstand` attribute. If set to `TRUE` the receiver **MUST** process the contents. If the attribute is absent
1662 or set to `FALSE` the receiver **SHOULD** attempt to process the content of the <UsageDirective> header block.

1663 A receiver that processes the contents of a <UsageDirective> header block **SHOULD** verify the integrity of the
1664 header block – that is, it should verify any digital signatures that list the header block in its manifest [[XMLDsig](#)]. The
1665 receiver **MUST** verify that the `ref` attribute refers to an element in the message. That receiver **MUST** further process
1666 the message according to the policy expressed by the children elements of the <UsageDirective> header block.
1667 Those children elements will be imported from a foreign namespace, and **MUST** be parsed and interpreted according
1668 to the applicable schema and processing rules of that foreign namespace.

1669 A receiver that cannot process a <UsageDirective> with `mustUnderstand` set to `TRUE` **MUST** respond with a
1670 <S:Fault>. The <S:Fault> **MUST** contain a <detail> element which in turn **MUST** contain a <Status> element
1671 with its `code` attribute set to `CannotHonourUsageDirective`. The <Status> element **SHOULD** possess a `ref`
1672 attribute with its value set to the value of the `id` attribute of the offending <UsageDirective> header block in the
1673 request message.

1674 A receiver that cannot honor a non-mandatory (without `mustUnderstand` set to `TRUE`) <UsageDirective> **must**
1675 respond according to the contained policy. In addition, in this case the receiver **MAY** respond with a SOAP-based
1676 ID-* message that includes a <Status> element with its `code` attribute set to `CannotHonourUsageDirective`.
1677 This <Status> element instance **SHOULD** include a `ref` attribute with its value set to the value of the `id` attribute of
1678 the <UsageDirective> header block in the request message that could not be honored.

1679 In this case, the receiver **MAY** include one or more new <UsageDirective> header blocks in its response message,
1680 each expressing a policy that the receiver would have been able to honor. The `ref` attribute of these headers **SHOULD**
1681 be set to the value of the <wsa:MessageID> header block in the request.

1682 **6.7. The <ApplicationEPR> Header Block**

1683 This section defines the <ApplicationEPR> header block. This header may be included in a message zero or more
1684 times and provides a means for a sender to specify application endpoints that may be referenced from the SOAP Body
1685 of the message.

1686 The <ApplicationEPR> header block is an extension of <wsa:EndpointReferenceType>.

1687 The schema fragment in [Figure 12](#) defines the The <ApplicationEPR> header block.

```

1688
1689
1690 <!-- application epr header block -->
1691
1692 <xs:element name="ApplicationEPR" type="wsa:EndpointReferenceType"/>
1693
1694
  
```

1695 **Figure 12. The <ApplicationEPR> Header Block Schema**

```

1696
1697 <sb:ApplicationEPR S:mustUnderstand="1"
1698   S:actor="http://schemas.../next"
1699   wsu:Id="NotifyTo-001">
1700   <wsa:Address>...</wsa:Address>
1701 </sb:ApplicationEPR>
1702
  
```

1703 **Example 17. An instantiated <ApplicationEPR> header block**

1704 6.8. The <UserInteraction> Header Block

1705 6.8.1. Overview

1706 A WSC that interacts with a user (typically through a web-site offered by the WSC) may need to indicate its readiness
 1707 to redirect the user agent of the user, or its readiness to pose questions to the user on behalf of other parties (such
 1708 as WSPs). The <UserInteraction> header block provides a means by which a WSC can indicate its preferences
 1709 and capabilities for interactions with requesting principals and, additionally, a SOAP fault message and HTTP redirect
 1710 profile that enables the WSC and WSP to cooperate in redirecting the requesting principal to the WSP and, after
 1711 browser interaction, back to the WSC.

1712 6.8.2. UserInteraction Element

1713 The <UserInteraction> element contains:

1714 *InteractionService* [Optional]

1715 If present, this element MUST describe an interaction service hosted by the sender. This indicates that the
 1716 sender can process messages defined for the interaction service [[LibertyInteract](#)], posing questions from the
 1717 recipient of the message to the Principal.

1718 *interact* [Optional]

1719 Indicates any preference that the sender has about interactions between the receiver and the requesting
 1720 principal. The value is a string, for which we define the following values:

- 1721 • *InteractIfNeeded* to indicate to the recipient that it should interact with the requesting principal if needed to satisfy
 1722 the ID-WSF request. This is the default.
- 1723 • *DoNotInteract* to indicate to the recipient that it MUST NOT interact with the requesting principal, either directly
 1724 or indirectly. The sender prefers to receive an error response over the situation where the requesting principal
 1725 would be distracted by an interaction.
- 1726 • *DoNotInteractForData* to indicate to the recipient that it MAY interact with the requesting principal *only* if an
 1727 explicit policy for the offered service so requires. The sender prefers to receive an error response over the situation
 1728 where the WSP would obtain service response data (e.g. Personal Profile data) from the resource owner, but the
 1729 sender does prefer to obtain a positive service response even if that requires policy-related interaction for e.g.
 1730 obtaining consent.

1731 **Note:**

1732 Implementors may choose to define additional values to indicate finer grained control over the user interactions.

1733 language [Optional]

1734 This attribute indicates languages that the user is likely able to process. The value of this attribute is a space
 1735 separated list of language identification tags ([RFC3066]). The WSC can obtain this information from the
 1736 HTTP ([RFC2616]) *Accept-Language* header, or by other means, for example from a *personal profile service*.

1737 redirect [Optional]

1738 An optional attribute to indicate that the sender supports the `<RedirectRequest>` element that a WSP may
 1739 include in a message to the WSC. The value is *true* or *false*. When absent the default behavior will be as if
 1740 *false*.

1741 maxInteractTime [Optional]

1742 This is used to indicate the maximum time in seconds that the sender regards as reasonable for any possible
 1743 interaction. The receiver is not expected to start any interaction if it has reason to assume that such an
 1744 interaction is likely to take more time. In case an interaction is started and does seem to take longer the
 1745 receiver is expected to respond with a message that contains a *InteractionTimeout* status code to the sender.

1746 The schema fragment in [Figure 13](#) defines the The `<UserInteraction>` header block.

1747
 1748
 1749
 1750
 1751
 1752
 1753
 1754
 1755
 1756
 1757
 1758
 1759
 1760
 1761
 1762
 1763
 1764

```
<!-- user interaction header block -->
<xs:complexType name="UserInteractionHeaderType">
  <xs:sequence>
    <xs:element name="InteractionService" type="wsa:EndpointReferenceType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="interact" type="xs:string" use="optional" default="interactIfNeeded"/>
  <xs:attribute name="language" type="xs:NMTOKENS" use="optional"/>
  <xs:attribute name="redirect" type="xs:boolean" use="optional" default="0"/>
  <xs:attribute name="maxInteractTime" type="xs:integer" use="optional"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
<xs:element name="UserInteraction" type="UserInteractionHeaderType"/>
```

1765 **Figure 13. The `<UserInteraction>` Header Block Schema**

1766 **6.8.3. UserInteraction Examples**

1767 Below is an example for a WSC that is prepared to redirect the user to a WSP, and also is ready to accept an
 1768 `<is:InteractionRequest>`. The WSC wishes that the WSP will not attempt to prompt the resource owner for
 1769 missing data; but accepts interactions for consent, as long as questioning the user will not take more than 60 seconds.
 1770 The WSC expects the user to understand US English and Finnish.

1771
 1772
 1773
 1774
 1775
 1776
 1777
 1778
 1779
 1780

```
<sb:UserInteraction interact="DoNotInteractForData" language="en-US fi"
  maxInteractTime="60" redirect="true">
  <sb:InteractionService xmlns:disco="urn:liberty:disco:2006-08">
    <wsa:Address>endpoint for interaction requests</wsa:Address>
    <wsa:Metadata>
      <disco:ServiceType>urn:liberty:is:2006-08</disco:ServiceType>
      <disco:Provider>http://someWSC</disco:Provider>
      <disco:Description>
```

```
1781         <disco:Endpoint>http://IS.com/soap</disco:Endpoint>
1782     </disco:Description>
1783 </wsa:Metadata>
1784 </sb:InteractionService>
1785 </sb:UserInteraction>
```

1786 The following is an example for a WSC that wants to ensure that the WSP will not attempt to contact the requesting
1787 principal, even if this may hinder serving the actual request; the WSC would rather receive an error, or a less optimal
1788 response (e.g. fewer profile attributes).

```
1789         <sb:UserInteraction interact="DoNotInteract" />
1790
1791
```

1792 6.8.4. Processing Rules

1793 If the sender includes an `InteractionService` element, it **MUST** set the value of `<disco:ServiceType>` within
1794 to `urn:liberty:is:2006-08`.

1795 If the sender sets `interact="DoNotInteract"` it **MUST** omit the `InteractionService` element, as well as the
1796 `language`, `redirect` and `maxInteractTime` attributes.

1797 The recipient of a message with a `UserInteraction` element **MUST NOT** respond with a `<RedirectRequest>` if
1798 the `redirect` is `false` or if `redirect` is absent.

1799 The recipient **MUST NOT** start a requesting principal interaction if the `interact` attribute has a value of `"DoNotIn-`
1800 `teract"`.

1801 The recipient **MUST NOT** interact with the requesting principal to obtain data that is to be included in a successful
1802 service response if the `interact` attribute has a value of `"DoNotInteractForData"`. In this case the recipient **MAY**
1803 start an interaction if a policy concerning available data so requires; for example if a policy requires that the Principal
1804 must be prompted for consent.

1805 The recipient **SHOULD NOT** start a requesting principal interaction if it expects that the time to complete the
1806 interaction will exceed the value of the `maxInteractTime`.

1807 The recipient **MUST** respond to the message after at most the number of seconds given as the value of the
1808 `maxInteractTime` attribute.

1809 The sender must ensure that the `UserInteraction` element is integrity protected; i.e. if message level authentication
1810 (see [\[LibertySecMech\]](#)) is used the sender **MUST** sign the `UserInteraction` element. Likewise the receiver must
1811 ensure that the integrity of the `UserInteraction` element is not compromised, according to the processing rules in
1812 [\[LibertySecMech\]](#).

1813 6.8.4.1. UserInteraction Faults

1814 A processor of a `UserInteraction` that must indicate an error situation related to this header **SHOULD** respond
1815 to the sender with an ID-WSF message that contains a `Status` element in the `detail` element of a `S:Fault`, or in
1816 a service specific `S:Body` component, or inside a higher level `Status` element. The `code` attribute of the included
1817 `Status` element can be set to one of the following values:

1818 • *InteractionRequired*, as indication that the recipient has a need to start an interaction in order to satisfy the service
1819 request but the `interact` attribute value was set to `DoNotInteract`.

- 1820 • *InteractionRequiredForData*. This indicates that the service request could not be satisfied because the WSP would
1821 have to interact with the requesting principal in order to obtain (some of) the requested data but the *interact*
1822 attribute value was set to *DoNotInteractForData*.
- 1823 • *InteractionTimeNotSufficient*, as indication that the recipient has a need to start an interaction but has reason to
1824 believe that more time is needed than allowed for by the value of the *maxInteractTime* attribute.
- 1825 • *InteractionTimeout*, as indication that the recipient could not satisfy the service request due to an unfinished
1826 interaction.

1827 **6.8.5. Cross-principal interactions**

1828 A 'cross-principal' interaction is defined by a WSC making a request on behalf of a principal who is different
1829 than the principal who 'owns' the resource in question. In such a case, the identity of the requesting principal
1830 will be identified by the security context of the message. The identity of the resource owner is expressed by the
1831 `<sb:TargetIdentity>` header.

1832 Any `sb:UserInteraction` header in such a message always refers to the requesting principal. Consequently, if
1833 the WSP desires to interact with the requesting principal, it may use the interaction options as indicated by the
1834 `UserInteraction` (if present) or discover the requesting principal's permanent IS.

1835 If the WSP desires to interact with the resource owner (as indicated by the `TargetIdentity` header), it will
1836 necessarily need to discover that principal's permanent IS as the alternative interaction mechanisms are not an option.

1837 7. The RedirectRequest Protocol

1838 In the `RedirectRequest` protocol the WSP requests the WSC to redirect the user agent of the Principal to a resource
1839 (URL) at the WSP. Once the user agent issues the HTTP request to fetch the URL the WSP has the opportunity to
1840 present one or more pages with questions and other information to the Principal. When the WSP has obtained the
1841 information that it required to serve the WSC, it redirects the user agent back to the WSC. The WSC can now re-issue
1842 its original request to the WSP. See [\[LibertyInteract\]](#) for an overview of various user interaction flows, including this
1843 redirect-based protocol.

1844 7.1. RedirectRequest Element

1845 The `RedirectRequest` element instructs the WSC to redirect the user to the WSP. It is an indication of the WSP that
1846 it cannot service a request made by the WSC before it obtains some more information from the user. The element is
1847 typically present in the `detail` element within a `<S:Fault>`. The `<RedirectRequest>` has one attribute:

1848 `redirectURL` [Required] The URL to which the WSC should redirect the user agent. This URL MUST NOT
1849 contain parameters named `ReturnToURL` or `IDP` as these are reserved for the recipient of the
1850 `<RedirectRequest>` (see the [RedirectRequest protocol](#)). The URL SHOULD start
1851 with `https:` to ensure the establishment of a secure connection between the user agent and
1852 the WSP.

1853 The optional text content of the element can be used to indicate the reason for the need for redirection of the requesting
1854 principal.

1855 The schema fragment for the element is:

```
1856 <xs:element name="RedirectRequest" type="RedirectRequestType"/>
1857 <xs:complexType name="RedirectRequestType">
1858   <xs:attribute name="redirectURL" type="xs:anyURI" use="required"/>
1859 </xs:complexType>
```

1861 An example of a `<RedirectRequest>` in a SOAP Fault could look like:

```
1862 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
1863   <S:Header>
1864     <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/02/addressing">...</wsa:MessageID>
1865     <wsa:RelatesTo>...</wsa:RelatesTo>
1866   </S:Header>
1867   <S:Body>
1868     <S:Fault>
1869       <faultcode>SOAP-ENV:Server</faultcode>
1870       <faultstring>Server Error</faultstring>
1871       <detail>
1872         <RedirectRequest redirectURL="https://someWSP/getConsent?transID=de67hj89jk65nk34">
1873           Redirecting to AP to obtain consent
1874         </RedirectRequest>
1875       </detail>
1876     </S:Fault>
1877   </S:Body>
1878 </S:Envelope>
```

1879 7.1.1. Processing Rules

1880 The recipient of a `<RedirectRequest>` MUST verify that the `redirectURL` points to the WSP, i.e. the host in the
1881 URL should be the same as the host to which the WSC sent its service request. If this is not the case the recipient
1882 MUST ignore the `<RedirectRequest>`.

1883 The recipient **MUST** attempt to direct the user agent to issue an HTTP request ([RFC2616]) for the URL in the
1884 `redirectURL` attribute of the `<RedirectRequest>`. That user agent **MUST** be associated with the ID-WSF request
1885 that caused the `<RedirectRequest>`. The recipient **MUST** add a `ReturnToURL` parameter to the `redirectURL`
1886 with its value the URL-encoded URL which the recipient wants the user agent directed back to. It is recommended
1887 that this `ReturnToURL` includes an identifier that associates the URL to the originating ID-WSF message to the WSP.
1888 The recipient **MAY** add an `IDP` parameter to the `redirectURL` with its value the `providerID` of an identity provider
1889 that was used to authenticate the user to the WSC.

1890 The recipient may instruct the user agent to submit either an HTTP GET or an HTTP POST request to the URL; in
1891 this way the WSC can avoid problems with user agents that can handle only short URLs. If the user agent is instructed
1892 to submit a HTTP POST, *all* URL parameters should be form-encoded, and the HTTP content-type header of the
1893 request **MUST** be `application/x-www-form-urlencoded`. Note that this implies that the WSP **SHOULD** accept
1894 both an HTTP GET as well as an HTTP POST request for the `redirectURL`, but in either case retrieval of URL
1895 parameters can be done using well-known techniques; most HTTP server environments effectively encapsulate the
1896 different methods for submission of parameters.

1897 As an example, assume that a Principal visits a service provider. As a result the service provider (acting as WSC)
1898 could have made a request to a WSP, and that WSP would have responded with a SOAP Fault similar to that of the
1899 example above. The WSC would now send a HTTP response to the user agent that would look like:

```
1900 HTTP 302
1901 Location: https://someWSP/getConsent?transID=de67hj89jk65nk34&ReturnToU
1902 RL=https%3a%2f%2fsomeWSC%2fisReturn3bjsession%3d9A6F2E3A&IDP=1A2B3C4D5E1A2B3C4D5E
1903 ... other HTTP headers ...
1904
1905 <html>
1906   <head>
1907     <title>Redirecting...</title>
1908   </head>
1909   <body>
1910     <p>Redirecting to AP to obtain consent</p>
1911   </body>
1912 </html>
```

1913 The WSC appends its own `ReturnToURL` as a parameter to the value of the `redirectURL` element that the WSC
1914 specified in its `RedirectRequest`.

1915 **7.2. RedirectRequest Protocol**

1916 The `<RedirectRequest>` protocol consists of the following steps, each with normative rules:

1917 **7.2.1. Step 1: WSC Issues Normal ID-WSF Request**

1918 For the `<RedirectRequest>` protocol to be initiated the originating ID-WSF message **MUST** contain a
1919 `UserInteraction` element with its `redirect` attribute set to *true*.

1920 The ID-WSF message **SHOULD** contain a `wsa:FaultTo` element to which the WSC desires fault messages be sent.

1921 **7.2.2. Step 2: WSP Responds with <RedirectRequest>**

1922 If, and only if, an ID-WSF message contains a `<UserInteraction>` element with its `redirect` attribute set to *true*
1923 **MAY** the recipient of the ID-WSF message respond with a `<RedirectRequest>` message in a SOAP Fault.

1924 **Note:**

1925 The `redirectURL` attribute **MUST** be constructed as to include the necessary information for mapping the upcoming
1926 HTTP request to the originating ID-WSF message; for example by inclusion of the value of the `wsa:MessageID`
1927 Header from that message.

1928 **7.2.3. Step 3: WSC Instructs User Agent to Contact the WSP**

1929 When the WSC receives a `<RedirectRequest>` it **MUST** attempt to direct the user agent to issue an HTTP request
1930 for the URL in the `redirectURL` attribute of the `RedirectRequest`. The user agent **MUST** be associated with the
1931 ID-WSF message that caused the `<RedirectRequest>`. The WSC **MUST** append a `ReturnToURL` parameter to the
1932 `redirectURL` with its value the URL-encoded URL to which the WSC wants the user agent directed back.

1933 **Note:**

1934 How this step is performed will depend on the user agent. In most cases it is accomplished by a simple HTTP 302
1935 response with a `Location` header set to the `redirectURL`. Different user agents may be better served by other
1936 approaches, for example a WML browser may be able to handle a redirect deck better than a potentially long URL.
1937 See the [processing rules for the `<RedirectRequest>`](#).

1938 **7.2.4. Step 4: WSP Interacts with User Agent**

1939 In step 4 the user agent issues the HTTP request for the `redirectURL`, with the `ReturnToURL` parameter appended,
1940 with any `IDP` parameter also appended. The WSP **MUST** verify that the `ReturnToURL` points to the WSC, i.e. the
1941 host in the URL should be the same as the host to which the WSP sent the `<RedirectRequest>`. If this is not the
1942 case the WSP **MUST** ignore the `ReturnToURL`, abort the protocol, and construct a meaningful error message for the
1943 user. If verification succeeds, however, the service (WSP) **MAY** now proceed with a HTTP response that contains
1944 an inquiry directed at the user. The WSP **SHOULD** verify that the identity of the user is that of the owner of the
1945 resource that was targeted in the originating ID-WSF request, for example by means of a `<saml:AuthnRequest>`
1946 (see [SAMLCore2]). This step may be followed by any number of interactions between the user and the WSP, but the
1947 WSP should attempt to execute step 5 within a reasonable time.

1948 **7.2.5. Step 5: WSP Redirects User Agent Back to WSC**

1949 In step 5 the WSP that issued the `<RedirectRequest>` **MUST** attempt to instruct the user agent to issue an HTTP
1950 request for the `ReturnToURL` that was included as parameter on the URL of the HTTP request made in step 4. The
1951 WSP **SHOULD** append a `ResendMessage` parameter to the `ReturnToURL`. This parameter serves as a hint to the
1952 WSC about the next step. A value of `0` or `false` indicates that the WSC should not try to re-issue the originating
1953 ID-WSF request, presumably because the resource owner did not approve completion of the transaction. If the value
1954 of `ResendMessage` is `true`, `1`, or any string other than `0` or `false`, it is an indication that the WSP recommends that the
1955 WSC re-issue the originating request. It is **RECOMMENDED** that in this situation, the value of this parameter be set
1956 to the value of the `wsa:MessageID` element of the originating ID-WSF message.

1957 **7.2.6. Step 6: User Agent Requests `ReturnToURL` from WSC**

1958 In step 6 the user agent requests the `ReturnToURL` from the WSC. The WSC **SHOULD** check the value of the
1959 `ResendMessage` parameter; if the value is `0` or `false` the WSC **SHOULD NOT** send an ID-WSF message with a
1960 request for the same resource and/or action (as in step 1). If the value of the `ResendMessage` parameter is anything
1961 else, then the WSC **MAY** resend the message (Step 7).

1962 After receiving the response from the WSP, the WSC should send a HTTP response to the user agent.

1963 **7.2.7. Step 7: WSC Resends Message**

1964 If the WSC resends its request it **MUST** set the value of the `wsa:RelatesTo` SOAP Header to the same value of the
1965 `wsa:MessageID` SOAP Header of the SOAP Fault that carried the `<RedirectRequest>` element (in step 2). .

1966 **7.2.8. Steps 8: WSP sends response**

1967 The WSP responds to the WSC's second request. The WSP MUST set the value of the `wsa:RelatesTo` SOAP
1968 Header to the same value of the `wsa:MessageID` SOAP Header of the WSC's resent request.

1969 **7.2.9. Steps 9: WSC sends HTTP response to User Agent**

1970 Finally, the WSC returns an HTTP response to the user agent.

1971 8. Security Considerations

- 1972 • The header blocks specified in this document should be integrity-protected using the mechanisms detailed in
1973 [[LibertySecMech](#)].
- 1974 • Header blocks should be signed in accordance with [[LibertySecMech](#)]. The receiver of a message containing a
1975 signature that covers specific header blocks should verify the signature as part of verifying the integrity of the
1976 header block.
- 1977 • Metadata [[LibertyMetadata](#)] should be used to the greatest extent possible to verify message sender identity claims.
- 1978 • Message senders and receivers should be authenticated to one another via the mechanisms discussed in [[Liberty-](#)
1979 [SecMech](#)].
- 1980 • To prevent message replay, receivers should maintain a message cache, and check received `messageID` values
1981 against the cache.

1982 **9. Acknowledgements**

1983 The members of the Liberty Technical Expert group, especially Greg Whitehead, Jonathan Sergent, Xavier Serret,
1984 and Conor Cahill, provided valuable input to this specification. The docbook source code for this specification was
1985 hand set to the tunes of The Sugarcubes, King Crimson, Juliana Hatfield, Smashing Pumpkins, Evanescence, Mad at
1986 Gravity, Elisa Korenne, The Breeders, fIREHOSE, Polly Jean Harvey, Jimi Hendrix, and various others.

1987 **References**

1988 **Normative**

- 1989 [LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0,
1990 Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1991 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 2.0-02,
1992 Liberty Alliance Project (25 November 2004). <http://www.projectliberty.org/specs>
- 1993 [LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version v2.0, Liberty
1994 Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1995 [RFC2045] Freed, N., Borenstein, N., eds. (November 1996). "Multipurpose Internet Mail Extensions
1996 (MIME) Part One: Format of Internet Message Bodies ," RFC 2045., Internet Engineering Task
1997 Force <http://www.ietf.org/rfc/rfc2045.txt>
- 1998 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
1999 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>
- 2000 [RFC2828] Shirey, R., eds. (May 2000). "Internet Security Glossary," RFC 2828., Internet Engineering Task Force
2001 <http://www.ietf.org/rfc/rfc2828.txt>
- 2002 [RFC3986] Berners-Lee, T., Fielding, R., Masinter, L., eds. (January 2005). "Uniform Resource Identifier
2003 (URI): Generic Syntax," RFC 3986 (Obsoletes RFC2732, RFC2396, RFC1808) (Updates RFC1738) (Also
2004 STD0066) (Status: STANDARD), The Internet Engineering Task Force <http://www.ietf.org/rfc/rfc3986.txt>
- 2005 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions
2006 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
2007 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-
2008 open.org/security/saml/v2.0/saml-core-2.0-os.pdf)
- 2009 [SAMLGloss2] Hodges, Jeff, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Glossary for the OASIS Security As-
2010 ssertion Markup Language (SAML) V2.0," SAML 2.0, OASIS Standard, Organization for the Advancement
2011 of Structured Information Standards <http://docs.oasis-open.org/security/saml/v2.0/saml-glossary-2.0-os.pdf>
- 2012 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October
2013 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium
2014 <http://www.w3.org/TR/xmlschema-1/>
- 2015 [Schema2-2] Biron, Paul V., Malhotra, Ashok, eds. (28 October 2004). "XML Schema Part 2: Datatypes Second
2016 Edition," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlschema-2/>
- 2017 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David , Kakivaya, Gopal, Layman,
2018 Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C
2019 Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 2020 [SOAPv1.1-Schema] "SOAP 1.1 Envelope schema," W3C W3C Note <http://schemas.xmlsoap.org/soap/envelope/>
- 2021 [WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Mered-
2022 ith, Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001).
2023 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

- 2024 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).
2025 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium
2026 <http://www.w3.org/TR/2004/REC-xml-20040204>
- 2027 [XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and
2028 Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlldsig-core>
- 2029 [WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, Rogers, Tony, eds.
2030 World Wide Web Consortium W3C Recommendation (9 May 2006). <http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/>
2031
- 2032 [WSAv1.0-SOAP] "WS-Addressing 1.0 SOAP Binding," Gudgin, Martin, Hadley, Marc, eds. World Wide Web Con-
2033 sortium W3C Recommendation (9 May 2006). <http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/>
- 2034 [WSAv1.0-Schema] "WS-Addressing 1.0 Schema," W3C, W3C Working Draft <http://dev.w3.org/cvsweb/~checkout~/2004/ws/addressing-1.0-addr.xsd>
2035
- 2036 [wss-sms] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (January, 2004). "Web
2037 Services Security: SOAP Message Security," OASIS Standard V1.0 [OASIS 200401], Organization for
2038 the Advancement of Structured Information Standards <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>
2039
- ## 2040 Informational
- 2041 [LibertyDisco] Hodges, Jeff, Cahill, Conor, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0,
2042 Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 2043 [LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0, Liberty Alliance Project (30 July,
2044 2006). <http://www.projectliberty.org/specs>
- 2045 [LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework
2046 Overview," Version 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 2047 [LibertyIDWSF20SCR] Whitehead, Greg, eds. Version 1.0, Liberty Alliance Project (30 July, 2006).
2048 <http://www.projectliberty.org/specs>
- 2049 [LibertyIDPPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification,"
2050 Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 2051 [Merriam-Webster] "Merriam-Webster Dictionary," <http://www.merriam-webster.com/>
- 2052 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., eds. (June
2053 1999). "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, The Internet Engineering Task Force
2054 <http://www.ietf.org/rfc/rfc2616.txt>
- 2055 [RFC3066] Alvestrand, H., eds. (January 2001). "Tags for the Identification of Languages," RFC 3066., Internet
2056 Engineering Task Force <http://www.ietf.org/rfc/rfc3066.txt>
- 2057 [RFC4086] Eastlake, D., Schiller, J., Crocker, S., eds. (June 2005). "Randomness Recommendations for Security,"
2058 RFC 4086, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc4086.txt>
- 2059 [SOAPv1.2] "SOAP Version 1.2 Part 1: Messaging Framework," Gudgin, Martin, Hadley, Marc, Mendelsohn, Noah,
2060 Moreau, Jean-Jacques, Nielsen, Henrik Frystyk, eds. World Wide Web Consortium W3C Recommendation
2061 (07 May 2003). <http://www.w3.org/TR/2003/PR-soap12-part1-20030507/>

2062 A. liberty-idwsf-soap-binding.xsd Schema Listing

```
2063
2064     <?xml version="1.0" encoding="UTF-8"?>
2065 <xs:schema targetNamespace="urn:liberty:sb"
2066     xmlns:xs="http://www.w3.org/2001/XMLSchema"
2067     xmlns="urn:liberty:sb"
2068     elementFormDefault="qualified"
2069     attributeFormDefault="unqualified">
2070
2071     <!-- Author: John Kemp -->
2072     <!-- Last editor: $Author: dchampagne $ -->
2073     <!-- $Date: 2006/07/16 04:21:57 $ -->
2074     <!-- $Revision: 1.7.2.1 $ -->
2075
2076     <xs:annotation>
2077         <xs:documentation>
2078             Liberty ID-WSF SOAP Binding Specification XSD
2079         </xs:documentation>
2080         <xs:documentation>
2081             The source code in this XSD file was excerpted verbatim from:
2082
2083             Liberty ID-WSF SOAP Binding Specification
2084             Version 2.0
2085             30 July, 2006
2086
2087             Copyright (c) 2006 Liberty Alliance participants, see
2088             http://www.projectliberty.org/specs/idwsf\_2\_0\_final\_copyrights.php
2089         </xs:documentation>
2090     </xs:annotation>
2091
2092     <!-- framework header block -->
2093
2094     <xs:complexType name="FrameworkType">
2095         <xs:sequence>
2096             <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2097         </xs:sequence>
2098         <xs:attribute name="version" type="xs:string" use="required"/>
2099         <xs:anyAttribute namespace="##other" processContents="lax"/>
2100     </xs:complexType>
2101
2102     <xs:element name="Framework" type="FrameworkType"/>
2103
2104 </xs:schema>
2105
2106
2107
```

2108 B. liberty-idwsf-soap-binding-v2.0.xsd Schema Listing

```
2109
2110     <?xml version="1.0" encoding="UTF-8"?>
2111 <xs:schema targetNamespace="urn:liberty:sb:2006-08"
2112     xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
2113     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
2114     xmlns:wsa="http://www.w3.org/2005/08/addressing"
2115     xmlns:xs="http://www.w3.org/2001/XMLSchema"
2116     xmlns:lu="urn:liberty:util:2006-08"
2117     xmlns="urn:liberty:sb:2006-08"
2118     elementFormDefault="qualified"
2119     attributeFormDefault="unqualified">
2120
2121     <!-- Author: John Kemp -->
2122     <!-- Last editor: $Author: dchampagne $ -->
2123     <!-- $Date: 2006/07/16 04:21:57 $ -->
2124     <!-- $Revision: 1.3.2.1 $ -->
2125
2126     <xs:import
2127         namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-ut
2128     ility-1.0.xsd"
2129         schemaLocation="wss-util-1.0.xsd"/>
2130
2131     <xs:import
2132         namespace="urn:oasis:names:tc:SAML:2.0:protocol"
2133         schemaLocation="saml-schema-protocol-2.0.xsd"/>
2134
2135     <xs:import
2136         namespace="http://www.w3.org/2005/08/addressing"
2137         schemaLocation="ws-addr-1.0.xsd"/>
2138
2139     <xs:import
2140         namespace="urn:liberty:util:2006-08"
2141         schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
2142
2143     <xs:annotation>
2144         <xs:documentation>
2145             Liberty ID-WSF SOAP Binding Specification 2.0 XSD
2146         </xs:documentation>
2147         <xs:documentation>
2148             The source code in this XSD file was excerpted verbatim from:
2149
2150             Liberty ID-WSF SOAP Binding Specification
2151             Version 2.0
2152             30 July, 2006
2153
2154             Copyright (c) 2006 Liberty Alliance participants, see
2155             http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2156         </xs:documentation>
2157     </xs:annotation>
2158
2159
2160     <!-- sender header block -->
2161
2162     <xs:complexType name="SenderType">
2163         <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
2164         <xs:attribute name="affiliationID" type="xs:anyURI" use="optional"/>
2165         <xs:anyAttribute namespace="##other" processContents="lax"/>
2166     </xs:complexType>
2167
2168     <xs:element name="Sender" type="SenderType"/>
2169
2170
2171     <!-- target identity header block -->
2172
2173     <xs:complexType name="TargetIdentityType">
```

```

2174     <xs:sequence>
2175         <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
2176     </xs:sequence>
2177     <xs:anyAttribute namespace="##other" processContents="lax" />
2178 </xs:complexType>
2179
2180 <xs:element name="TargetIdentity" type="TargetIdentityType" />
2181
2182
2183 <!-- credentials context header block -->
2184
2185 <xs:complexType name="CredentialsContextType">
2186     <xs:sequence>
2187         <xs:element ref="samlp:RequestedAuthnContext" minOccurs="0" />
2188         <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded" />
2189     </xs:sequence>
2190     <xs:anyAttribute namespace="##other" processContents="lax" />
2191 </xs:complexType>
2192
2193 <xs:element name="CredentialsContext" type="CredentialsContextType" />
2194
2195
2196 <!-- epr update header block -->
2197
2198 <xs:complexType name="EndpointUpdateType">
2199     <xs:complexContent>
2200         <xs:extension base="wsa:EndpointReferenceType">
2201             <xs:attribute name="updateType" type="xs:anyURI" use="optional" />
2202         </xs:extension>
2203     </xs:complexContent>
2204 </xs:complexType>
2205
2206 <xs:element name="EndpointUpdate" type="EndpointUpdateType" />
2207
2208
2209 <!-- timeout header block -->
2210
2211 <xs:complexType name="TimeoutType">
2212     <xs:attribute name="maxProcessingTime" type="xs:integer" use="required" />
2213     <xs:anyAttribute namespace="##other" processContents="lax" />
2214 </xs:complexType>
2215
2216 <xs:element name="Timeout" type="TimeoutType" />
2217
2218
2219 <!-- processing context header block -->
2220
2221 <xs:complexType name="ProcessingContextType">
2222     <xs:simpleContent>
2223         <xs:extension base="xs:anyURI">
2224             <xs:anyAttribute namespace="##other" processContents="lax" />
2225         </xs:extension>
2226     </xs:simpleContent>
2227 </xs:complexType>
2228
2229 <xs:element name="ProcessingContext" type="ProcessingContextType" />
2230
2231 <!-- consent header block -->
2232
2233 <xs:complexType name="ConsentType">
2234     <xs:attribute name="uri" type="xs:anyURI" use="required" />
2235     <xs:attribute name="timestamp" type="xs:dateTime" use="optional" />
2236     <xs:anyAttribute namespace="##other" processContents="lax" />
2237 </xs:complexType>
2238
2239 <xs:element name="Consent" type="ConsentType" />
2240

```

```
2241 <!-- usage directive header block -->
2242
2243 <xs:complexType name="UsageDirectiveType">
2244   <xs:sequence>
2245     <xs:any namespace="##other" processContents="lax"
2246       maxOccurs="unbounded"/>
2247   </xs:sequence>
2248   <xs:attribute name="ref" type="xs:IDREF" use="required"/>
2249   <xs:anyAttribute namespace="##other" processContents="lax"/>
2250 </xs:complexType>
2251
2252 <xs:element name="UsageDirective" type="UsageDirectiveType"/>
2253
2254 <!-- application epr header block -->
2255
2256 <xs:element name="ApplicationEPR" type="wsa:EndpointReferenceType"/>
2257
2258 <!-- user interaction header block -->
2259
2260 <xs:complexType name="UserInteractionHeaderType">
2261   <xs:sequence>
2262     <xs:element name="InteractionService" type="wsa:EndpointReferenceType"
2263 minOccurs="0" maxOccurs="unbounded"/>
2264   </xs:sequence>
2265   <xs:attribute name="interact" type="xs:string" use="optional" default="interactIfNeeded"/>
2266   <xs:attribute name="language" type="xs:NMTOKENS" use="optional"/>
2267   <xs:attribute name="redirect" type="xs:boolean" use="optional" default="0"/>
2268   <xs:attribute name="maxInteractTime" type="xs:integer" use="optional"/>
2269   <xs:anyAttribute namespace="##other" processContents="lax"/>
2270 </xs:complexType>
2271
2272 <xs:element name="UserInteraction" type="UserInteractionHeaderType"/> <xs:element name="RedirectRequest" type="
2273
2274   <xs:complexType name="RedirectRequestType">
2275     <xs:attribute name="redirectURL" type="xs:anyURI" use="required"/>
2276   </xs:complexType>
2277 </xs:schema>
2278
2279
```

2280 C. liberty-idwsf-utility-v2.0.xsd Schema Listing

```
2281
2282     <?xml version="1.0" encoding="UTF-8"?>
2283 <xs:schema targetNamespace="urn:liberty:util:2006-08"
2284 xmlns:xs="http://www.w3.org/2001/XMLSchema"
2285 xmlns="urn:liberty:util:2006-08"
2286 elementFormDefault="qualified"
2287 attributeFormDefault="unqualified"
2288 version="2.0-03">
2289
2290 <xs:annotation>
2291   <xs:documentation>
2292     Liberty Alliance Project utility schema. A collection of common
2293     Identity Web Services Framework (ID-WSF) elements and types.
2294     This schema is intended for use in ID-WSF schemas.
2295
2296     This version: 2006-08
2297
2298     Copyright (c) 2006 Liberty Alliance participants, see
2299     http://www.projectliberty.org/specs/idwsf_2_0_final_c_copyrights.php
2300
2301   </xs:documentation>
2302 </xs:annotation>
2303 <xs:simpleType name="IDType">
2304   <xs:annotation>
2305     <xs:documentation>
2306       This type should be used to provide IDs to components
2307       that have IDs that may not be scoped within the local
2308       xml instance document.
2309     </xs:documentation>
2310   </xs:annotation>
2311   <xs:restriction base="xs:string"/>
2312 </xs:simpleType>
2313 <xs:simpleType name="IDReferenceType">
2314   <xs:annotation>
2315     <xs:documentation>
2316       This type can be used when referring to elements that are
2317       identified using an IDType.
2318     </xs:documentation>
2319   </xs:annotation>
2320   <xs:restriction base="xs:string"/>
2321 </xs:simpleType>
2322 <xs:attribute name="itemID" type="IDType"/>
2323 <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2324 <xs:complexType name="StatusType">
2325   <xs:annotation>
2326     <xs:documentation>
2327       A type that may be used for status codes.
2328     </xs:documentation>
2329   </xs:annotation>
2330   <xs:sequence>
2331     <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
2332   </xs:sequence>
2333   <xs:attribute name="code" type="xs:string" use="required"/>
2334   <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
2335   <xs:attribute name="comment" type="xs:string" use="optional"/>
2336 </xs:complexType>
2337
2338 <xs:element name="Status" type="StatusType">
2339   <xs:annotation>
2340     <xs:documentation>
2341       A standard Status type
2342     </xs:documentation>
2343   </xs:annotation>
2344 </xs:element>
2345
```

```
2346 <xs:complexType name="ResponseType">
2347   <xs:sequence>
2348     <xs:element ref="Status" minOccurs="1" maxOccurs="1"/>
2349     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2350   </xs:sequence>
2351   <xs:attribute ref="itemIDRef" use="optional"/>
2352   <xs:anyAttribute namespace="##other" processContents="lax"/>
2353 </xs:complexType>
2354 <xs:element name="TestResult" type="TestResultType"/>
2355 <xs:complexType name="TestResultType">
2356   <xs:simpleContent>
2357     <xs:extension base="xs:boolean">
2358       <xs:attribute ref="itemIDRef" use="required"/>
2359     </xs:extension>
2360   </xs:simpleContent>
2361 </xs:complexType>
2362 <xs:complexType name="EmptyType">
2363   <xs:annotation>
2364     <xs:documentation> This type may be used to create an empty element </xs:documentation>
2365   </xs:annotation>
2366   <xs:complexContent>
2367     <xs:restriction base="xs:anyType"/>
2368   </xs:complexContent>
2369 </xs:complexType>
2370 <xs:element name="Extension" type="extensionType">
2371   <xs:annotation>
2372     <xs:documentation>
2373       An element that contains arbitrary content extensions
2374       from other namespaces
2375     </xs:documentation>
2376   </xs:annotation>
2377 </xs:element>
2378 <xs:complexType name="extensionType">
2379   <xs:annotation>
2380     <xs:documentation>
2381       A type for arbitrary content extensions from other namespaces
2382     </xs:documentation>
2383   </xs:annotation>
2384   <xs:sequence>
2385     <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
2386   </xs:sequence>
2387 </xs:complexType>
2388 </xs:schema>
2389
2390
```

2391 D. liberty-utility-v2.0.xsd Schema Listing

```
2392
2393     <?xml version="1.0" encoding="UTF-8"?>
2394 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2395     elementFormDefault="qualified"
2396     attributeFormDefault="unqualified"
2397     version="2.0-01">
2398     <xs:annotation>
2399         <xs:documentation>
2400 Liberty Alliance Project utility schema. A collection of common
2401 elements and types for use with independent Liberty XML Schema documents.
2402
2403 This file intended for inclusion, rather than importation, into other schemas.
2404 This version: 2004-12
2405
2406         Copyright (c) 2004 Liberty Alliance participants, see
2407         http://www.projectliberty.org/specs/idff\_copyrights.html
2408
2409     </xs:documentation>
2410 </xs:annotation>
2411 <xs:simpleType name="IDType">
2412     <xs:annotation>
2413         <xs:documentation>
2414             This type should be used to provide IDs to components
2415             that have IDs that may not be scoped within the local
2416             xml instance document.
2417         </xs:documentation>
2418     </xs:annotation>
2419     <xs:restriction base="xs:string"/>
2420 </xs:simpleType>
2421 <xs:simpleType name="IDReferenceType">
2422     <xs:annotation>
2423         <xs:documentation>
2424             This type can be used when referring to elements that are
2425             identified using an IDType.
2426         </xs:documentation>
2427     </xs:annotation>
2428     <xs:restriction base="xs:string"/>
2429 </xs:simpleType>
2430 <xs:complexType name="StatusType">
2431     <xs:annotation>
2432         <xs:documentation>
2433             A type that may be used for status codes.
2434         </xs:documentation>
2435     </xs:annotation>
2436     <xs:sequence>
2437         <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
2438     </xs:sequence>
2439     <xs:attribute name="code" type="xs:string" use="required"/>
2440     <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
2441     <xs:attribute name="comment" type="xs:string" use="optional"/>
2442 </xs:complexType>
2443
2444 <xs:element name="Status" type="StatusType">
2445     <xs:annotation>
2446         <xs:documentation>
2447             A standard Status type
2448         </xs:documentation>
2449     </xs:annotation>
2450 </xs:element>
2451
2452 <xs:complexType name="EmptyType">
2453     <xs:annotation>
2454         <xs:documentation> This type may be used to create an empty element </xs:documentation>
2455     </xs:annotation>
2456     <xs:complexContent>
```

```
2457         <xs:restriction base="xs:anyType" />
2458     </xs:complexContent>
2459 </xs:complexType>
2460 <xs:element name="Extension" type="extensionType">
2461     <xs:annotation>
2462         <xs:documentation>
2463             An element that contains arbitrary content extensions
2464             from other namespaces
2465         </xs:documentation>
2466     </xs:annotation>
2467 </xs:element>
2468 <xs:complexType name="extensionType">
2469     <xs:annotation>
2470         <xs:documentation>
2471             A type for arbitrary content extensions from other namespaces
2472         </xs:documentation>
2473     </xs:annotation>
2474     <xs:sequence>
2475         <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded" />
2476     </xs:sequence>
2477 </xs:complexType>
2478 </xs:schema>
2479
2480
```

2481 E. wss-util-1.0.xsd Schema Listing

```
2482
2483     <?xml version="1.0" encoding="UTF-8"?>
2484 <!--
2485 OASIS takes no position regarding the validity or scope of any intellectual property or other
2486 rights that might be claimed to pertain to the implementation or use of the technology described
2487 in this document or the extent to which any license under such rights might or might not be
2488 available; neither does it represent that it has made any effort to identify any such rights.
2489 Information on OASIS's procedures with respect to rights in OASIS specifications can be found
2490 at the OASIS website. Copies of claims of rights made available for publication and any
2491 assurances of licenses to be made available, or the result of an attempt made to obtain a
2492 general license or permission for the use of such proprietary rights by implementors or users
2493 of this specification, can be obtained from the OASIS Executive Director.
2494 OASIS invites any interested party to bring to its attention any copyrights, patents or
2495 patent applications, or other proprietary rights which may cover technology that may be
2496 required to implement this specification. Please address the information to the OASIS Executive Director.
2497
2498 Copyright © OASIS Open 2002-2004. All Rights Reserved.
2499 This document and translations of it may be copied and furnished to others, and derivative
2500 works that comment on or otherwise explain it or assist in its implementation may be prepared,
2501 copied, published and distributed, in whole or in part, without restriction of any kind,
2502 provided that the above copyright notice and this paragraph are included on all such copies
2503 and derivative works. However, this document itself does not be modified in any way, such
2504 as by removing the copyright notice or references to OASIS, except as needed for the purpose
2505 of developing OASIS specifications, in which case the procedures for copyrights defined
2506 in the OASIS Intellectual Property Rights document must be followed, or as required to
2507 translate it into languages other than English.
2508 The limited permissions granted above are perpetual and will not be revoked by OASIS
2509 or its successors or assigns.
2510 This document and the information contained herein is provided on an "AS IS" basis
2511 and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
2512 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
2513 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
2514 -->
2515 <xsd:schema targetNamespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-u
2516 tility-1.0.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2517
2518
2519 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-uti
2520 lity-1.0.xsd" xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-u
2521 tility-1.0.xsd"
2522 elementFormDefault="qualified" attributeFormDefault="unqualified" version="0.1">
2523 <!-- // Fault Codes ////////////////////////////////////// -->
2524 <xsd:simpleType name="tTimestampFault">
2525 <xsd:annotation>
2526 <xsd:documentation>
2527 This type defines the fault code value for Timestamp message expiration.
2528 </xsd:documentation>
2529 </xsd:annotation>
2530 <xsd:restriction base="xsd:QName">
2531 <xsd:enumeration value="wsu:MessageExpired"/>
2532 </xsd:restriction>
2533 </xsd:simpleType>
2534 <!-- // Global attributes ////////////////////////////////////// -->
2535 <xsd:attribute name="Id" type="xsd:ID">
2536 <xsd:annotation>
2537 <xsd:documentation>
2538 This global attribute supports annotating arbitrary elements with an ID.
2539 </xsd:documentation>
2540 </xsd:annotation>
2541 </xsd:attribute>
2542 <xsd:attributeGroup name="commonAtts">
2543 <xsd:annotation>
2544 <xsd:documentation>
2545 Convenience attribute group used to simplify this schema.
2546
```

```
2547     </xsd:documentation>
2548   </xsd:annotation>
2549   <xsd:attribute ref="wsu:Id" use="optional" />
2550   <xsd:anyAttribute namespace="##other" processContents="lax" />
2551 </xsd:attributeGroup>
2552 <!-- // Utility types ////////////////////////////////////// -->
2553 <xsd:complexType name="AttributedDateTime">
2554   <xsd:annotation>
2555     <xsd:documentation>
2556 This type is for elements whose [children] is a psuedo-dateTime and can have arbitrary attributes.
2557   </xsd:documentation>
2558   </xsd:annotation>
2559   <xsd:simpleContent>
2560     <xsd:extension base="xsd:string">
2561       <xsd:attributeGroup ref="wsu:commonAtts" />
2562     </xsd:extension>
2563   </xsd:simpleContent>
2564 </xsd:complexType>
2565 <xsd:complexType name="AttributedURI">
2566   <xsd:annotation>
2567     <xsd:documentation>
2568 This type is for elements whose [children] is an anyURI and can have arbitrary attributes.
2569   </xsd:documentation>
2570   </xsd:annotation>
2571   <xsd:simpleContent>
2572     <xsd:extension base="xsd:anyURI">
2573       <xsd:attributeGroup ref="wsu:commonAtts" />
2574     </xsd:extension>
2575   </xsd:simpleContent>
2576 </xsd:complexType>
2577 <!-- // Timestamp header components ////////////////////////////////////// -->
2578 <xsd:complexType name="TimestampType">
2579   <xsd:annotation>
2580     <xsd:documentation>
2581 This complex type ties together the timestamp related elements into a composite type.
2582   </xsd:documentation>
2583   </xsd:annotation>
2584   <xsd:sequence>
2585     <xsd:element ref="wsu:Created" minOccurs="0" />
2586     <xsd:element ref="wsu:Expires" minOccurs="0" />
2587     <xsd:choice minOccurs="0" maxOccurs="unbounded">
2588       <xsd:any namespace="##other" processContents="lax" />
2589     </xsd:choice>
2590   </xsd:sequence>
2591   <xsd:attributeGroup ref="wsu:commonAtts" />
2592 </xsd:complexType>
2593 <xsd:element name="Timestamp" type="wsu:TimestampType">
2594   <xsd:annotation>
2595     <xsd:documentation>
2596 This element allows Timestamps to be applied anywhere element wildcards are present,
2597 including as a SOAP header.
2598   </xsd:documentation>
2599   </xsd:annotation>
2600 </xsd:element>
2601 <!-- global element decls to allow individual elements to appear anywhere -->
2602 <xsd:element name="Expires" type="wsu:AttributedDateTime">
2603   <xsd:annotation>
2604     <xsd:documentation>
2605 This element allows an expiration time to be applied anywhere element wildcards are present.
2606   </xsd:documentation>
2607   </xsd:annotation>
2608 </xsd:element>
2609 <xsd:element name="Created" type="wsu:AttributedDateTime">
2610   <xsd:annotation>
2611     <xsd:documentation>
2612 This element allows a creation time to be applied anywhere element wildcards are present.
2613   </xsd:documentation>
```

```
2614         </xsd:annotation>
2615     </xsd:element>
2616 </xsd:schema>
2617
2618
```

2619 F. ws-addr-1.0.xsd Schema Listing

```
2620
2621     <?xml version="1.0" encoding="utf-8"?>
2622 <!DOCTYPE xs:schema PUBLIC "-//W3C//DTD XMLSCHEMA 200102//EN" "http://www.w3.org/2001/XMLSchema.dtd">
2623
2624 <!--
2625     W3C XML Schema defined in the Web Services Addressing 1.0 specification
2626     http://www.w3.org/TR/ws-addr-core
2627
2628     Copyright © 2005 World Wide Web Consortium,
2629
2630     (Massachusetts Institute of Technology, European Research Consortium for
2631     Informatics and Mathematics, Keio University). All Rights Reserved. This
2632     work is distributed under the W3C® Software License [1] in the hope that
2633     it will be useful, but WITHOUT ANY WARRANTY; without even the implied
2634     warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
2635
2636     [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
2637
2638     $Id: ws-addr-1.0.xsd,v 1.4 2005/09/23 18:20:29 dchampagne Exp $
2639 -->
2640 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2641     xmlns:tns="http://www.w3.org/2005/08/addressing"
2642     targetNamespace="http://www.w3.org/2005/08/addressing"
2643     blockDefault="#all"
2644     elementFormDefault="qualified"
2645     finalDefault=""
2646     attributeFormDefault="unqualified">
2647
2648     <!-- Constructs from the WS-Addressing Core -->
2649
2650     <xs:element name="EndpointReference" type="tns:EndpointReferenceType"/>
2651     <xs:complexType name="EndpointReferenceType" mixed="false">
2652         <xs:sequence>
2653             <xs:element name="Address" type="tns:AttributedURIType"/>
2654             <xs:element name="ReferenceParameters" type="tns:ReferenceParametersType" minOccurs="0"/>
2655             <xs:element ref="tns:Metadata" minOccurs="0"/>
2656             <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2657         </xs:sequence>
2658         <xs:anyAttribute namespace="##other" processContents="lax"/>
2659     </xs:complexType>
2660
2661     <xs:complexType name="ReferenceParametersType" mixed="false">
2662         <xs:sequence>
2663             <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2664         </xs:sequence>
2665         <xs:anyAttribute namespace="##other" processContents="lax"/>
2666     </xs:complexType>
2667
2668     <xs:element name="Metadata" type="tns:MetadataType"/>
2669     <xs:complexType name="MetadataType" mixed="false">
2670         <xs:sequence>
2671             <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2672         </xs:sequence>
2673         <xs:anyAttribute namespace="##other" processContents="lax"/>
2674     </xs:complexType>
2675
2676     <xs:element name="MessageID" type="tns:AttributedURIType"/>
2677     <xs:element name="RelatesTo" type="tns:RelatesToType"/>
2678     <xs:complexType name="RelatesToType" mixed="false">
2679         <xs:simpleContent>
2680             <xs:extension base="xs:anyURI">
2681                 <xs:attribute name="RelationshipType" type="tns:RelationshipTypeOpen
2682 Enum" use="optional"
2683                 default="http://www.w3.org/2005/08/addressing/reply"/>
2684             <xs:anyAttribute namespace="##other" processContents="lax"/>

```

```

2685         </xs:extension>
2686     </xs:simpleContent>
2687 </xs:complexType>
2688
2689 <xs:simpleType name="RelationshipTypeOpenEnum">
2690     <xs:union memberTypes="tns:RelationshipType xs:anyURI" />
2691 </xs:simpleType>
2692
2693 <xs:simpleType name="RelationshipType">
2694     <xs:restriction base="xs:anyURI">
2695         <xs:enumeration value="http://www.w3.org/2005/08/addressing/reply" />
2696     </xs:restriction>
2697 </xs:simpleType>
2698
2699 <xs:element name="ReplyTo" type="tns:EndpointReferenceType" />
2700 <xs:element name="From" type="tns:EndpointReferenceType" />
2701 <xs:element name="FaultTo" type="tns:EndpointReferenceType" />
2702 <xs:element name="To" type="tns:AttributedURIType" />
2703 <xs:element name="Action" type="tns:AttributedURIType" />
2704
2705 <xs:complexType name="AttributedURIType" mixed="false">
2706     <xs:simpleContent>
2707         <xs:extension base="xs:anyURI">
2708             <xs:anyAttribute namespace="##other" processContents="lax" />
2709         </xs:extension>
2710     </xs:simpleContent>
2711 </xs:complexType>
2712
2713 <!-- Constructs from the WS-Addressing SOAP binding -->
2714
2715 <xs:attribute name="IsReferenceParameter" type="xs:boolean" />
2716
2717 <xs:simpleType name="FaultCodesOpenEnumType">
2718     <xs:union memberTypes="tns:FaultCodesType xs:QName" />
2719 </xs:simpleType>
2720
2721 <xs:simpleType name="FaultCodesType">
2722     <xs:restriction base="xs:QName">
2723         <xs:enumeration value="tns:InvalidAddressingHeader" />
2724         <xs:enumeration value="tns:InvalidAddress" />
2725         <xs:enumeration value="tns:InvalidEPR" />
2726         <xs:enumeration value="tns:InvalidCardinality" />
2727         <xs:enumeration value="tns:MissingAddressInEPR" />
2728         <xs:enumeration value="tns:DuplicateMessageID" />
2729         <xs:enumeration value="tns:ActionMismatch" />
2730         <xs:enumeration value="tns:MessageAddressingHeaderRequired" />
2731         <xs:enumeration value="tns:DestinationUnreachable" />
2732         <xs:enumeration value="tns:ActionNotSupported" />
2733         <xs:enumeration value="tns:EndpointUnavailable" />
2734     </xs:restriction>
2735 </xs:simpleType>
2736
2737 <xs:element name="RetryAfter" type="tns:AttributedUnsignedLongType" />
2738 <xs:complexType name="AttributedUnsignedLongType" mixed="false">
2739     <xs:simpleContent>
2740         <xs:extension base="xs:unsignedLong">
2741             <xs:anyAttribute namespace="##other" processContents="lax" />
2742         </xs:extension>
2743     </xs:simpleContent>
2744 </xs:complexType>
2745
2746 <xs:element name="ProblemHeaderQName" type="tns:AttributedQNameType" />
2747 <xs:complexType name="AttributedQNameType" mixed="false">
2748     <xs:simpleContent>
2749         <xs:extension base="xs:QName">
2750             <xs:anyAttribute namespace="##other" processContents="lax" />
2751         </xs:extension>

```

```
2752     </xs:simpleContent>
2753 </xs:complexType>
2754
2755 <xs:element name="ProblemHeader" type="tns:AttributedAnyType"/>
2756 <xs:complexType name="AttributedAnyType" mixed="false">
2757   <xs:sequence>
2758     <xs:any namespace="##any" processContents="lax" minOccurs="1" maxOccurs="1"/>
2759   </xs:sequence>
2760   <xs:anyAttribute namespace="##other" processContents="lax"/>
2761 </xs:complexType>
2762
2763 <xs:element name="ProblemIRI" type="tns:AttributedURIType"/>
2764
2765 <xs:element name="ProblemAction" type="tns:ProblemActionType"/>
2766 <xs:complexType name="ProblemActionType" mixed="false">
2767   <xs:sequence>
2768     <xs:element ref="tns:Action" minOccurs="0"/>
2769     <xs:element name="SoapAction" minOccurs="0" type="xs:anyURI"/>
2770   </xs:sequence>
2771   <xs:anyAttribute namespace="##other" processContents="lax"/>
2772 </xs:complexType>
2773
2774 </xs:schema>
2775
2776
2777
```