



Liberty ID-WSF SOAP Binding Specification

Version: 2.0-errata-v1.0

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
Rob Lockhart, IEEE-ISTO
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-2.0-errata-v1.0.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 © 2007 2FA Technology; Adobe Systems; Agencia Catalana De Certificacio; America Online, Inc.;
16 American Express Company; Amssoft Systems Pvt Ltd.; Avatier Corporation; BIPAC; BMC Software, Inc.; Bank of
17 America Corporation; Beta Systems Software AG; British Telecommunications plc; Computer Associates
18 International, Inc.; Credentica; DataPower Technology, Inc.; Deutsche Telekom AG, T-Com; Diamelle Technologies,
19 Inc.; Diversinet Corp.; Drummond Group Inc.; Enosis Group LLC; Entrust, Inc.; Epok, Inc.; Ericsson; Falkin
20 Systems LLC; Fidelity Investments; Forum Systems, Inc.; France Télécom; French Government Agence pour le
21 développement de l'administration électronique (ADAE); Fugen Solutions, Inc; Fulvens Ltd.; GSA Office of
22 Governmentwide Policy; Gamefederation; Gemalto; General Motors; GeoFederation; Giesecke & Devrient GmbH;
23 Hewlett-Packard Company; Hochhauser & Co., LLC; IBM Corporation; Intel Corporation; Intuit Inc.; Kantega;
24 Kayak Interactive; Livo Technologies; Luminance Consulting Services; MasterCard International; MedCommons
25 Inc.; Mobile Telephone Networks (Pty) Ltd; NEC Corporation; NTT DoCoMo, Inc.; Netegrity, Inc.; Neustar, Inc.;
26 New Zealand Government State Services Commission; Nippon Telegraph and Telephone Corporation; Nokia
27 Corporation; Novell, Inc.; OpenNetwork; Oracle Corporation; Ping Identity Corporation; RSA Security Inc.;
28 Reactivity Inc.; Royal Mail Group plc; SAP AG; Senforce; Sharp Laboratories of America; Sigaba; SmartTrust; Sony
29 Corporation; Sun Microsystems, Inc.; Supremacy Financial Corporation; Symlabs, Inc.; Telecom Italia S.p.A.;
30 Telefónica Móviles, S.A.; Telenor R&D; Thales e-Security; Trusted Network Technologies; UNINETT AS; UTI;
31 VeriSign, Inc.; Vodafone Group Plc.; Wave Systems Corp. All rights reserved.

32 Liberty Alliance Project
33 Licensing Administrator
34 c/o IEEE-ISTO
35 445 Hoes Lane
36 Piscataway, NJ 08855-1331, USA
37 info@projectliberty.org

Contents

38		
39	1. Introduction	5
40	2. Notation and Conventions	7
41	2.1. XML Namespaces	7
42	2.2. Terminology	8
43	2.3. Treatment of Boolean Values	10
44	2.4. String and URI Values	10
45	2.5. Time Values	10
46	3. Schema Particulars	11
47	3.1. Schema Declarations	11
48	3.2. "ID" Attributes	11
49	3.3. Status Types	11
50	3.3.1. Status Codes	11
51	3.4. SOAP Fault Types	13
52	4. SOAP Binding	15
53	4.1. SOAP Version	15
54	4.2. The SOAPAction HTTP Header	15
55	4.3. Ordinary ID-* Messages	15
56	4.4. ID-* Fault Messages	15
57	4.5. SOAP-bound ID-* Messages	16
58	5. Messaging-specific Header Blocks	19
59	5.1. The <wsu:Timestamp> element in the <wsse:Security> Header Block	19
60	5.2. The <wsa:MessageID> Header Block	19
61	5.2.1. <wsa:MessageID> Value Requirements	19
62	5.3. The <wsa:RelatesTo> Header Block	19
63	5.4. The <wsa:To> Header Block	19
64	5.5. The <wsa:Action> Header Block	20
65	5.6. The <wsa:ReplyTo> Header Block	20
66	5.7. The <wsa:FaultTo> Header Block	20
67	5.8. The <wsa:Framework> Header Block	20
68	5.9. The <Sender> Header Block	21
69	5.10. The <TargetIdentity> Header Block	22
70	5.11. Messaging Processing Rules	23
71	5.11.1. Constructing and Sending a SOAP-bound ID-* Message	24
72	5.11.2. Receiving and Processing a SOAP-bound ID-* Message	25
73	5.12. Examples	29
74	6. Optional Header Blocks	32
75	6.1. The <ProcessingContext> Header Block	32
76	6.1.1. The <ProcessingContext> Type and Element	32
77	6.1.2. <ProcessingContext> Header Block Semantics and Processing Rules	33
78	6.2. The <Consent> Header Block	35
79	6.2.1. The <Consent> Type and Element	35
80	6.3. The <CredentialsContext> Header Block	36
81	6.3.1. Overview	36
82	6.3.2. CredentialsContext Type and Element	37
83	6.3.3. CredentialsContext Example	37
84	6.3.4. Processing Rules	38
85	6.4. The <EndpointUpdate> Header Block	38
86	6.4.1. Overview	38
87	6.4.2. EndpointUpdate Type and Element	39
88	6.4.3. EndpointUpdate Examples	39
89	6.4.4. Processing Rules for the EndpointUpdate header	42
90	6.4.5. Processing Rules for the EndpointUpdated SOAP Fault	43

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

1. Introduction

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

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

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

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

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

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

- Processing Context:

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

- Consent Claims:

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

- Credentials Context:

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

- Endpoint Update:

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

- Timeout:

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

167 • Usage Directives:

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

171 • Application EPR:

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

174 • User Interaction:

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

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

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

185 2. Notation and Conventions

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

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

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

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

195 2.1. XML Namespaces

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

197

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].

198

2.2. Terminology

199

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

200

201

affiliation

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

202

203

204

205

206	Affiliation ID	An <i>Affiliation ID</i> identifies an <i>affiliation</i> . It is schematically represented by the <code>affiliationID</code> attribute of the <code><AffiliationDescriptor></code> metadata element [LibertyMetadata].
207		
208		
209	client	A <i>role</i> assumed by a <i>system entity</i> which makes a request of another system entity, often termed a <i>server</i> [RFC2828], i.e., a client is also a <i>sender</i> .
210		
211	ID-*	A shorthand designator referring to the Liberty ID-WSF, ID-FF, and ID-SIS specification sets. For example, one might say that the former specification sets are all part of the Liberty ID-* specification suite.
212		
213		
214	ID-* header block	One of the header blocks defined in this specification, or defined in any of the other Liberty ID-* specification suite.
215		
216	ID-* message	Equivalent to <i>ordinary ID-* message</i> .
217	ID-* fault message	See Section 4.4 .
218	ID-SIS	Liberty Identity Service Interface specification set.
219	ID-WSF	Liberty Identity Web Services Framework specification set.
220	MEP	see Message Exchange Pattern.
221	Message Exchange Pattern	A [SOAPv1.2] term for the overall notion of various patterns of message exchange between SOAP nodes. For example, request-reply and one-way are two <i>MEPs</i> used in this specification.
222		
223		
224	message thread	A <i>message thread</i> is an exchange of messages in a request-response <i>MEP</i> between two <i>SOAP nodes</i> . All the messages of a given message thread are "linked" via each message's <code><wsa:RelatesTo></code> header block value being set, by the sender, from the previous successfully received message's <code><wsa:MessageID></code> header block value.
225		
226		
227		
228	Ordinary ID-* message	See Section 4.3 .
229	processing context	A <i>processing context</i> is the collection of specific circumstances under which a particular processing step or set of steps take place.
230		
231	processing context facet	A <i>processing context facet</i> is an identified aspect, inherent or additive, of a <i>processing context</i> .
232		
233	provider	A <i>provider</i> is a Liberty-enabled entity that performs one or more of the provider roles in the Liberty architecture, for example Service Provider or Identity Provider. See also <i>Liberty-enabled Provider</i> in [LibertyGlossary]. Providers are identified in Liberty protocol interactions by their <i>Provider IDs</i> or optionally their <i>Affiliation ID</i> if they are a member of an affiliation(s) and are acting in that capacity.
234		
235		
236		
237		
238	Provider ID	A <i>Provider ID</i> identifies an entity known as a <i>provider</i> . It is schematically represented by the <code>providerID</code> attribute of the <code><EntityDescriptor></code> metadata element [LibertyMetadata].
239		
240	receiver	A <i>role</i> taken by a <i>system entity</i> when it receives a message sent by another system entity. See also <i>SOAP receiver</i> in [SOAPv1.2].
241		
242	role	A function or part performed, especially in a particular operation or process [Merriam-Webster].
243		

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

267 2.3. Treatment of Boolean Values

268 For readability, when an XML Schema type is specified to be `xsd:boolean`, this document discusses the values as
269 TRUE and FALSE rather than "1" and "0", which will exist in a document instance conforming to the SOAP Envelope
270 1.1 schema [SOAPv1.1-Schema].

271 2.4. String and URI Values

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

277 Note

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

281 2.5. Time Values

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

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

3. Schema Particulars

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

3.1. Schema Declarations

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

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

3.2. "ID" Attributes

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

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

3.3. Status Types

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

```
<xs:complexType name="StatusType">
  <xs:annotation>
    <xs:documentation>
      A type that may be used for status codes.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="code" type="xs:string" use="required"/>
  <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
  <xs:attribute name="comment" type="xs:string" use="optional"/>
</xs:complexType>

<xs:element name="Status" type="StatusType">
  <xs:annotation>
    <xs:documentation>
      A standard Status type
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

335

Figure 1. status and statusType Schema

336

3.3.1. Status Codes

337

This section lists, in [Table 2](#), the values defined in this specification for the code attribute of the <Status> element.

338

Other specifications MAY define additional code attribute values.

339

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

3.4. SOAP Fault Types

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

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

Note

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

```
<xs:element name="Fault" type="tns:Fault"/>
<xs:complexType name="Fault" final="extension">
  <xs:annotation>
    <xs:documentation>
      Fault reporting structure
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="faultcode" type="xs:QName"/>
    <xs:element name="faultstring" type="xs:string"/>
    <xs:element name="faultactor" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="detail" type="tns:detail" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="detail">
  <xs:sequence>
    <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

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

4. SOAP Binding

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

4.1. SOAP Version

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

4.2. The SOAPAction HTTP Header

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

Note

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

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

4.3. Ordinary ID-* Messages

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

```
<idpp:Query>
:
<!-- various message-specific subelements may go here -->
:
</idpp:Query>
```

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

4.4. ID-* Fault Messages

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

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

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

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

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

427 **Note**

428 When reporting SOAP processing errors, the WS-Addressing action <http://www.w3.org/2005/08/addressing/soap/fault>
429 SHOULD be used. When reporting WS-Addressing processing errors, the WS-Addressing action
430 <http://www.w3.org/2005/08/addressing/fault> SHOULD be used. When reporting other processing errors, if no
431 specific WS-Addressing action is defined, then <http://www.w3.org/2005/08/addressing/soap/fault> SHOULD be used.

432 **4.5. SOAP-bound ID-* Messages**

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

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

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

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

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

```
447
448 <S:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
449   xmlns:sb="..."
450   xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
451
452   <S:Header>
453
454     ...
455
456     <wsse:Security>
457       <wsu:Timestamp>
458         <wsu:Created>2005-06-17T04:49:17Z</wsu:Created>
459       </wsu:Timestamp>
460     </wsse:Security>
461
462     <wsa:MessageId>...</wsa:MessageId>
463
464     <wsa:To>...</wsa:To>
465
466     <wsa:Action>...</wsa:Action>
467
468     <!-- reference params from target EndpointReference -->
469
470     <sbf:Framework version="2.0"/>
471
472     <sb:Sender providerID="..." affiliationID="..."/>
473
474     ...
475
476   </S:Header>
477
478   <S:Body>
479
480     <idpp:Query> <!-- This is an ID-PP "Query" message bound -->
481       :         <!-- into the <S:Body> of a SOAP message.  -->
482       :
483     </idpp:Query>
484
485   </S:Body>
486
487 </S:Envelope>
```

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

```
489
490 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
491   xmlns:sb="..."
492   xmlns:pp="urn:liberty:id-sis-pp:2003-08">
493
494   <S:Header>
495
496     ...
497
498     <wsse:Security>
499       <wsu:Timestamp>
500         <wsu:Created>2005-06-17T04:49:18Z</wsu:Created>
501       </wsu:Timestamp>
502     </wsse:Security>
503
504     <wsa:MessageId>...</wsa:MessageId>
505
506     <wsa:RelatesTo>...</wsa:RelatesTo>
507
508     <wsa:To>...</wsa:To>
509
510     <wsa:Action>...</wsa:Action>
511
512     <sbf:Framework version="2.0"/>
513
514     <sb:Sender providerID="..." />
515
516     ...
517
518   </S:Header>
519
520   <S:Body>
521
522     <S:Fault>
523       <faultcode>S:server</faultcode>
524       <faultstring>Server Error</faultstring>
525       <!-- <S:faultactor> should be absent -->
526
527       <detail>
528         <lu:Status code="SomeStatus"
529           ref="Foo"
530           comment="Bar" />
531       </detail>
532     </S:Fault>
533
534   </S:Body>
535
536 </S:Envelope>
537
```

538

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

539 5. Messaging-specific Header Blocks

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

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

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

545 Note

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

547 5.1. The <wsu:Timestamp> element in the <wsse:Security> Header 548 Block

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

552 Note

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

555 5.2. The <wsa:MessageID> Header Block

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

558 5.2.1. <wsa:MessageID> Value Requirements

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

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

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

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

567 5.3. The <wsa:RelatesTo> Header Block

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

571 **Note**

572 When the relationship is <http://www.w3.org/2005/03/addressing/reply>, the RelationshipType attribute may be
573 omitted.

574 **5.4. The <wsa:To> Header Block**

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

577 **Note**

578 In the typical case that a WS-Addressing endpoint reference is used to address a message, the value of this header
579 block is taken from the <wsa:Address> of the endpoint reference. If the <wsa:To> header block is not present,
580 the value defaults to <http://www.w3.org/2005/03/addressing/role/anonymous>; so, when constructing a message, the
581 header block can be omitted if this is the value that would be used. This typically allows the <wsa:To> header block
582 to be omitted during synchronous request-response message exchanges over HTTP. Please refer to [WSAv1.0] for
583 default processing rules in the absence of the <wsa:To> header block.

584 **5.5. The <wsa:Action> Header Block**

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

587 **5.6. The <wsa:ReplyTo> Header Block**

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

590 **Note**

591 If the <wsa:ReplyTo> header block is not present, the value defaults to <http://www.w3.org/2005/03/addressing/role/anonymous>;
 592 so, when constructing a message, the header block can be omitted if this is the value that would be used. This typically
 593 allows the <wsa:ReplyTo> header block to be omitted during synchronous request-response message exchanges
 594 over HTTP. Please refer to [WSAv1.0] for default processing rules in the absence of the <wsa:ReplyTo> header
 595 block.

596 **5.7. The <wsa:FaultTo> Header Block**

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

600 **5.8. The <sbfc:Framework> Header Block**

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

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

604 The schema fragment in Figure 3 defines the <sbfc:Framework> header block.

```
605
606
607 <!-- framework header block -->
608
609 <xs:complexType name="FrameworkType">
610   <xs:sequence>
611     <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
612   </xs:sequence>
613   <xs:attribute name="version" type="xs:string" use="required"/>
614   <xs:anyAttribute namespace="##other" processContents="lax"/>
615 </xs:complexType>
616
617 <xs:element name="Framework" type="FrameworkType"/>
618
619
620
```

621 **Figure 3. The <sbfc:Framework> Header Block Schema**

```
622
623 <sbfc:Framework version="2.0" S:mustUnderstand="1"
624   S:actor="http://schemas.../next"
625   wsu:Id="A72139...381"/>
626
```

627 **Example 4. An instantiated <sbfc:Framework> header block**

628 **5.9. The <Sender> Header Block**

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

Note

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

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

The <Sender> header block defines the following attributes:

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

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

```
643
644
645 <!-- sender header block -->
646
647 <xs:complexType name="SenderType">
648   <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
649   <xs:attribute name="affiliationID" type="xs:anyURI" use="optional"/>
650   <xs:anyAttribute namespace="##other" processContents="lax"/>
651 </xs:complexType>
652
653 <xs:element name="Sender" type="SenderType"/>
654
655
656
```

Figure 4. The <Sender> Header Block Schema

```
658
659 <sb:Sender S:mustUnderstand="1"
660   S:actor="http://schemas.../next "
661   wsu:Id="A72139...381"
662   providerID="http://example.com"
663   affiliationID="http://affiliation.com"/>
664
```

Example 5. An instantiated <Sender> header block

5.10. The <TargetIdentity> Header Block

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

Note

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

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

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

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

```
679
680
681 <!-- target identity header block -->
682
683 <xs:complexType name="TargetIdentityType">
684   <xs:sequence>
685     <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
686   </xs:sequence>
687   <xs:anyAttribute namespace="##other" processContents="lax"/>
688 </xs:complexType>
689
690 <xs:element name="TargetIdentity" type="TargetIdentityType"/>
691
692
693
```

Figure 5. The <TargetIdentity> Header Block Schema

```
695
696 <sb:TargetIdentity S:mustUnderstand="1"
697   S:actor="http://schemas.../next"
698   wsu:Id="A31739...293">
699   ...
700 </sb:TargetIdentity>
701
```

Example 6. An instantiated <TargetIdentity> header block

5.11. Messaging Processing Rules

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

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

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

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

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

720 **Note**

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

- 722 • <wsa:MessageID> [[WSAv1.0](#)]
- 723 • <wsa:RelatesTo> [[WSAv1.0](#)]
- 724 • <wsa:To> [[WSAv1.0](#)]
- 725 • <wsa:Action> [[WSAv1.0](#)]
- 726 • <wsa:ReplyTo> [[WSAv1.0](#)]
- 727 • <wsa:FaultTo> [[WSAv1.0](#)]
- 728 • <wsse:Security> [[LibertySecMech](#)]
- 729 • <sbef:Framework>
- 730 • <Sender>
- 731 • <TargetIdentity>
- 732 • <ProcessingContext>
- 733 • <Consent>
- 734 • <UsageDirective>
- 735 • <EndpointUpdate>
- 736 • <Timeout>
- 737 • <CredentialsContext>
- 738 • <ApplicationEPR>
- 739 • <UserInteraction>

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

743 **5.11.1. Constructing and Sending a SOAP-bound ID-* Message**

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

- 746 1. The outgoing message MUST satisfy the rules given in [Section 4: SOAP Binding](#).
- 747 2. The outgoing message MUST satisfy the rules given in [[WSAv1.0-SOAP](#)].

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

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

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

Note

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

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

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

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

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

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

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

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

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

4. Processing specific to the `<wsu:Timestamp>` element in the `<wsse:Security>` header block:

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

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

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

815 **Note**

816 Certain classes of client devices, such as consumer electronics, often do not have correctly set clocks. These
817 processing rules may be relaxed for messages received from such devices.

818 C. If the foregoing test (4.A) holds true, processing continues with step 5 .

819 D. Otherwise, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per Section 4.4)
820 with the <Status> element configured with:

821 • a code attribute with a value of:

822 • "IDStarMsgNotUnderstood" if the failed test is 4.A.

823 • "StaleMsg" if the failed test is 4.B,

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

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

826 The receiver MAY discard the incoming message. The receiver is finished processing this incoming message
827 at this point.

828 **Note**

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

832 5. Processing specific to the <wsa:MessageID> and <wsa:RelatesTo> header blocks:

833 A. A single <wsa:MessageID> header block MUST be present in the header of the message.

834 B. If the <wsa:RelatesTo> header block with RelationshipType equal to [http://www.w3.org/2005/03/ ad-](http://www.w3.org/2005/03/addressing/reply)
835 [dressing/reply](http://www.w3.org/2005/03/addressing/reply) is present, and if the receiver is participating in a request-response MEP with the send-
836 ing party, then the value of the <wsa:RelatesTo> header block SHOULD match the value of the
837 <wsa:MessageID> header block of a message previously sent by the receiver to the sender of the now
838 incoming message.

839 C. If the foregoing tests (5.A through 5.B) hold true, processing continues with step 6 .

840 D. Otherwise, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per
841 Section 4.4).

842 The receiver MAY discard the incoming message. The receiver is finished processing this incoming message
843 at this point.

844 **Note**

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

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

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

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

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

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

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

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

860 • "ProviderIDNotValid", or,

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

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

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

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

868 8. Processing specific to the <TargetIdentity> header block:

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

871 B. If the foregoing test (8.A) holds true, processing continues with step 9.

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

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

875 • "TargetIdentityNotValid"

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

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

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

880 9. All remaining ID-* header blocks SHOULD be processed at this point. See appropriate sections in this and other
881 specifications for the processing rules associated with these header blocks and the manner of reporting any issues
882 with this processing. If there are no issues with these header blocks, then processing continues with step 10
883 below, otherwise the receiver is finished processing this incoming message at this point.

884 **Note**

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

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

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

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

894 • "IDStarMsgNotUnderstood"

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

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

899 **Note**

900 Other specifications conforming to this binding that specify ordinary ID-* messages and their processing, such as
901 [\[LibertyIDPP\]](#) or [\[LibertyDisco\]](#), MAY define `<Status>` element code attribute values in addition to the ones defined
902 in [Section 3.3.1](#) of this document. These code attribute values SHOULD be used to signal to the sender any issues
903 with the incoming ordinary ID-* message found by the receiver. This specification does not define any such conditions
904 other than the one described above in [10](#), and they are not further discussed in this document.

905 **5.12. Examples**

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

```
908
909     <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
910       xmlns:sbf="urn:liberty:sb"
911       xmlns:sb="urn:liberty:sb:2006-08"
912       xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
913
914     <S:Header>
915
916       ...
917
918       <wsse:Security>
919         <wsu:Timestamp>
920           <wsu:Created>2005-06-17T04:49:17Z</wsu:Created>
921         </wsu:Timestamp>
922       </wsse:Security>
923
924       <wsa:MessageID>
925         http://spwsc.com/0123456789abcdef0123456789abcdef01234567
926       </wsa:MessageID>
927
928       <wsa:To>http://spwsp.com/idpp</wsa:To>
929
930       <wsa:Action>urn:liberty:id-sis-pp:2003-08:Modify</wsa:Action>
931
932       <!-- reference params from target EndpointReference -->
933
934       <sbf:Framework version="2.0"/>
935
936       <sb:Sender providerID="http://spwsc.com" affiliationID="http://affiliation.com"/>
937
938       ...
939
940     </S:Header>
941
942     <S:Body>
943
944       <idpp:Modify>
945         :      <!-- this is an ID-PP Modify message -->
946       </idpp:Modify>
947
948     </S:Body>
949
950   </S:Envelope>
951
```

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

952

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

```
956
957 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
958   xmlns:sbf="urn:liberty:sb"
959   xmlns:sb="urn:liberty:sb:2006-08"
960   xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
961
962   <S:Header>
963
964     ...
965
966     <wsse:Security>
967       <wsu:Timestamp>
968         <wsu:Created>2005-06-17T04:49:18Z</wsu:Created>
969       </wsu:Timestamp>
970     </wsse:Security>
971
972     <wsa:MessageID>
973       http://spwsp.com/ffeeddccbbaa99887766554433221100ffeebbcc
974     </wsa:MessageID>
975     <wsa:RelatesTo>
976       http://spwsc.com/0123456789abcdef0123456789abcdef01234567
977     </wsa:RelatesTo>
978
979     <wsa:Action>urn:liberty:id-sis-pp:2003-08:ModifyResponse</wsa:Action>
980
981     <sbf:Framework version="2.0" />
982
983     <sb:Sender providerID="http://spwsp.com" />
984
985     ...
986
987   </S:Header>
988
989   <S:Body>
990
991     <idpp:ModifyResponse>
992       : <!-- this is an ID-PP ModifyResponse message -->
993     </idpp:ModifyResponse>
994
995   </S:Body>
996
997 </S:Envelope>
998
```

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

6. Optional Header Blocks

The optional header blocks described in this specification are:

- <ProcessingContext>
- <Consent>
- <CredentialsContext>
- <EndpointUpdate>
- <Timeout>
- <UsageDirective>
- <ApplicationEPR>
- <UserInteraction>

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

Note

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

6.1. The <ProcessingContext> Header Block

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

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

6.1.1. The <ProcessingContext> Type and Element

The <ProcessingContext> content model is anyURI.

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

```
1027
1028
1029 <!-- processing context header block -->
1030
1031 <xs:complexType name="ProcessingContextType">
1032   <xs:simpleContent>
1033     <xs:extension base="xs:anyURI">
1034       <xs:anyAttribute namespace="##other" processContents="lax" />
1035     </xs:extension>
1036   </xs:simpleContent>
1037 </xs:complexType>
1038
1039 <xs:element name="ProcessingContext" type="ProcessingContextType" />
1040
1041
```

1042 **Figure 6. The <ProcessingContext> Header Block Schema**

```
1043
1044   <sb:ProcessingContext S:mustUnderstand="1">
1045     urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
1046   </sb:ProcessingContext>
1047
```

1048 **Example 9. An instantiated <ProcessingContext> header block**

1049 **6.1.2. <ProcessingContext> Header Block Semantics and Processing Rules**

1050 This section first describes the overall semantics of the <ProcessingContext> header block, then defines two
1051 *processing context facet URIs*, and concludes with defining specific processing rules.

1052 **6.1.2.1. <ProcessingContext> Header Block Semantics**

1053 The overall semantic of the <ProcessingContext> header block is:

1054 The <ProcessingContext> header block MAY be employed by a sender, who is acting in a web
1055 services client (WSC) role, to signal to a receiver, who is acting in a web services provider (WSP)
1056 role, that the latter should add a specific *processing context facet* to the overall *processing context*
1057 (see [Section 2.2: Terminology](#)) in which the *service request* is evaluated.

1058 The specific processing context facet being conveyed by the <ProcessingContext> header block is identified by
1059 the header block's URI element value.

1060 Such URIs are known as *processing context facet URIs*. An example of a processing context facet that may be signaled
1061 by such a URI is whether the principal should be considered to be online or not.

1062 An ID-WSF or ID-SIS WSP receiving a service request containing a <ProcessingContext> header block with
1063 one of the above processing context facet URIs SHOULD process the conveyed ID-* message **with the indicated**
1064 **processing context facet in force**. Thus the ID-* message's processing as well as any applicable access management
1065 policies are exercised within an overall processing context which includes the processing context facet. Finally, an
1066 indication of success or failure of the ID-* message processing is returned to the sender, in the same manner as would
1067 be done if the ID-* message had been sent without the attendant <ProcessingContext> header block.

1068 The above completely describes the semantic of this header block itself, and further description of particular effects
1069 on processing must be made in descriptions of processing context facet URIs. Such a description is given in the next
1070 section.

1071 **Note**

1072 Whether or not a receiver honors a <ProcessingContext> header block is a matter of local policy at the
1073 receiver, as is whether or not a receiver honors any given request from any given sender. For example, the
1074 <ProcessingContext> header block functionality has security implications in the sense of possibly facilitating
1075 an adversary to probe a receiver's behavior given adversary-chosen inputs. For these reasons, whether or not the
1076 <ProcessingContext> header block functionality is enabled on the part of a receiver with respect to a particular
1077 sender should be a matter of business-level agreement between the receiver and the sender.

1078 **6.1.2.2. Processing Context Facet URIs: PrincipalOnline, PrincipalOffline, and Simulate**

1079 Three processing context facet URIs are defined below for use with the <ProcessingContext> header block:

1080 urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
1081 Conduct the processing of the ID-* message as if the Principal is offline.

1082 urn:liberty:sb:2003-08:ProcessingContext:PrincipalOnline
1083 Conduct the processing of the ID-* message as if the Principal is online.

1084 urn:liberty:sb:2003-08:ProcessingContext:Simulate
1085 *Simulate* the processing of the ID-* message.

1086 If the sender includes a <UserInteraction> header block in addition to the <ProcessingContext> header block
1087 in the SOAP-bound ID-* request message, the receiver and sender **MUST** appropriately initiate the indicated user
1088 interaction, and incorporate information supplied by the user as a part of the resultant user interaction, into the
1089 appropriate data and/or policy stores.

1090 **Note**

1091 Any processing context facet that was conveyed in the <ProcessingContext> header block **MUST NOT** be enforced
1092 during such a user interaction. Rather, it applies only to the processing of the ID-* message itself.

1093 In summary, the overall intended side-effect of using the above-defined processing context facets is for the receiver to
1094 evaluate applicable policy, and return a putative indication of success or failure to the sender. This provides WSCs the
1095 capability to make an ID-WSF or ID-SIS service request and ascertain whether it will be successful or not—without
1096 the service request actually being carried out. Additionally, it facilitates carrying out any user interaction that may be
1097 indicated by the current combination of service request context and applicable policy. This will, for example, facilitate
1098 some WSCs to fashion more "user friendly" experiences.

1099 **6.1.2.3. Defining New Processing Context Facet URIs**

1100 The rightmost portions of the processing context facet URIs after the "ProcessingContext:" component are referred
1101 to as *processing context facet identifiers*. For example, whether the Principal is online or not is a facet of a request
1102 context. New processing context facet identifiers **MAY** be defined in other specifications, for example in ID-SIS data
1103 service specifications. An ID-SIS data service may define as many levels of request context identifiers as necessary
1104 to address the application's needs.

1105 **6.1.2.4. Sender Processing Rules**

1106 A sender **MAY** include a <ProcessingContext> header block in a SOAP-bound ID-* message. The sender **MUST**
1107 include a processing context facet URI in the <ProcessingContext> header block. The sender then sends the ID-*
1108 SOAP-bound message to an ID-WSF or ID-SIS service-hosting node (AKA the receiver).

1109 A sender **MAY** indicate that it believes either that the Principal is currently "online" or "offline" when it sends a
1110 message by specifying one of the two processing context facet URIs:

- 1111 • `urn:liberty:sb:2003-08:ProcessingContext:PrincipalOnline`
- 1112 • `urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline`

1113 The sender will typically receive a response from the receiver indicating success or failure or will receive a SOAP
1114 fault indicating a processing error with the SOAP-bound ID-* message. Note that in the case of a "successful" request
1115 *simulation*, the service will not return any result data other than an indication of success or failure to the sender.

1116 **6.1.2.5. Receiver Processing Rules**

1117 The receiver of a request containing a `<ProcessingContext>` header block **MUST** examine the included processing
1118 context facet URI. If it is known to the data service, then the data service **MUST** attempt to process the data service
1119 request, represented by the ID-* message, in an overall processing context including the processing context facet as
1120 indicated by the conveyed processing context facet URI, and return an indication of success or failure to the sender.

1121 If the data service request is malformed or has some other issue that would normally cause the receiver to issue a
1122 SOAP fault, the receiver **SHOULD** do so.

1123 If the receiver is asked to simulate processing of the request (by the inclusion of the
1124 `urn:liberty:sb:2003-08:ProcessingContext:Simulate` facet URI), and they are both able and willing to
1125 honor that processing context, then the receiver **MUST** evaluate the conveyed ID-* message, but **MUST NOT** actually
1126 perform the operation. That is, the receiver **MUST NOT** make actual changes to underlying ID-* service datastore,
1127 and it **MUST NOT** return any data as a result of evaluating the ID-* message.

1128 If the sender includes a `<UserInteraction>` header block, in addition to the `<ProcessingContext>` header
1129 block, then both participants **MUST** initiate the indicated user interaction appropriately, and incorporate infor-
1130 mation supplied by the user as a part of the interaction into appropriate data and/or policy stores, even if the
1131 `urn:liberty:sb:2003-08:ProcessingContext:Simulate` URI is specified in a `<ProcessingContext>`
1132 header.

1133 In the event the receiver does not understand the included processing context facet URI, the receiver **MAY** respond
1134 with a SOAP-bound ID-* fault message (per [Section 4.4: ID-* Fault Messages](#)) with the `<Status>` element configured
1135 with:

- 1136 • a `code` attribute with a value of:
 - 1137 • `"ProcCtxURINotUnderstood"`
- 1138 • and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

1139 In the event the receiver is not willing to enforce a stipulated processing context, the receiver **MAY** respond with a
1140 SOAP-bound ID-* fault message (per [Section 4.4: ID-* Fault Messages](#)) with the `<Status>` element configured with:

- 1141 • a `code` attribute with a value of:
 - 1142 • `"ProcCtxUnwilling"`
- 1143 • and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

1144 **Note**

1145 The receiver MAY reference multiple <ProcessingContext> headers in the <detail> of the fault response (in
1146 accordance with the rules specified in [Section 4.4](#)).

1147 **6.2. The <Consent> Header Block**

1148 This section defines the <Consent> header block. This header block is used to explicitly claim that the Principal
1149 consented to the present interaction.

1150 **6.2.1. The <Consent> Type and Element**

1151 The <Consent> header block element MAY be employed by either a sender or a receiver. For example, the Principal
1152 may be using a Liberty-enabled client or proxy (common in the wireless world), and in that sort of environment the
1153 mobile operator may cause the Principal's terminal (AKA: cell phone) to prompt the principal for consent for some
1154 interaction.

1155 The <Consent> header block has the following attributes:

- 1156 • **uri** [Required] – A URI indicating that the Principal's consent was obtained.
1157 Optionally, the URI MAY identify a particular *Consent Agreement Statement* defining the specific nature of the
1158 consent obtained.

1159 This specification defines one well-known URI Liberty implementors and deployers MAY use to indicate positive
1160 Principal consent was obtained with respect to whatever ID-* interaction is underway or being initiated. This URI
1161 is known as the "Principal Consent Obtained" URI (PCO). The value of this URI is:

1162 urn:liberty:consent:obtained

1163 This URI does not correspond to any particular Consent Agreement Statement. Rather, it simply states that consent
1164 was obtained. The full meaning and implication of this will need to be derived from the execution context.

- 1165 • **timestamp** [Optional] – For denoting the time at which the sender obtained Principal consent with the POC.

1166 The schema fragment in [Figure 7](#) defines the Consent header block type.

```
1167  
1168  
1169 <!-- consent header block -->  
1170  
1171 <xs:complexType name="ConsentType">  
1172   <xs:attribute name="uri" type="xs:anyURI" use="required"/>  
1173   <xs:attribute name="timestamp" type="xs:dateTime" use="optional"/>  
1174   <xs:anyAttribute namespace="##other" processContents="lax"/>  
1175 </xs:complexType>  
1176  
1177 <xs:element name="Consent" type="ConsentType"/>  
1178  
1179
```

1180 **Figure 7. The <Consent> Header Block Schema**

```
1181  
1182 <sb:Consent  
1183   uri="urn:liberty:consent:obtained"  
1184   timestamp="2112-03-15T11:12:10Z"/>  
1185
```

1186 **Example 10. An instantiated <Consent> header block**

6.3. The <CredentialsContext> Header Block

6.3.1. Overview

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

6.3.2. CredentialsContext Type and Element

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

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

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

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

```
<!-- credentials context header block -->
<xs:complexType name="CredentialsContextType">
  <xs:sequence>
    <xs:element ref="samlp:RequestedAuthnContext" minOccurs="0" />
    <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>
<xs:element name="CredentialsContext" type="CredentialsContextType" />
```

Figure 8. The <CredentialsContext> Header Block Schema

6.3.3. CredentialsContext Example

```
1217 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1218         xmlns:sb="urn:liberty:sb:2006-08"
1219         xmlns:lib="urn:liberty:iff:2003-08" >
1220
1221   <S:Header>
1222     ...
1223     <!-- Says that the sender would like credentials that include RequestAuthnContext
1224           as specified -->
1225
1226     <sb:CredentialsContext mustUnderstand="1">
1227
1228       <samlp:RequestedAuthnContext>
1229         ...
1230       </samlp:RequestedAuthnContext>
1231
1232     </sb:CredentialsContext>
1233     ...
1234   </S:Header>
1235
1236   <S:Body>
1237
1238     <!-- a fault in the body indicates that the WSP's policy requires
1239           different (perhaps "stronger") credentials than were originally
1240           provided in the request -->
1241
1242     <S:Fault>
1243       <faultcode>S:Server</faultcode>
1244       <faultstring>
1245         Your request contained inappropriate credentials.
1246       </faultstring>
1247       <detail>
1248         <lu:Status code="InappropriateCredentials"/>
1249         <wsse:Security id="a6352...564"/>
1250       </detail>
1251     </S:Fault>
1252   </S:Body>
1253
1254 </S:Envelope>
```

1255 **Example 11. A CredentialsContext Header Offered in Response to a Request with Inappropriate Credentials.**

1256 6.3.4. Processing Rules

1257 6.3.4.1. Sender Processing Rules

1258 A sender including this header **MUST** specify at least one RequestAuthnContext or one SecurityMechID.

1259 The SecurityMechID elements **SHOULD** be listed in order of preference by the sender.

1260 6.3.4.2. Receiver Processing Rules

1261 The receiver of a <CredentialsContext> header containing one or more SecurityMechID elements **SHOULD**
1262 use the highest-listed (first) SecurityMechID that it supports in future requests to the sender of this header.

1263 The receiver of a <CredentialsContext> header containing a RequestAuthnContext element **SHOULD** use
1264 credentials that conform to the policies specified therein in any future requests to the sender of this header (where
1265 credentials are required).

1266 6.4. The <EndpointUpdate> Header Block

1267 6.4.1. Overview

1268 It may be necessary for an entity receiving an ID-* message to indicate that messages from the sender should be
1269 directed to a different endpoint, or that they wish a different credential to be used than was originally specified
1270 by the entity for access to the requested resource. The <EndpointUpdate> response header allows a message
1271 receiver to indicate that a new endpoint or new credentials should be employed by the sender of the message on any
1272 subsequent messages. This header block may be used in conjunction with the <sb:InappropriateCredentials>
1273 and <sb:EndpointUpdated> faults, to indicate that the current message processing failed for those reasons, and
1274 should be submitted with the changes noted in any accompanying <EndpointUpdate> header block.

1275 **Note**

1276 The use of this header block allows the sender of the message to convey updates to security tokens, essentially
1277 providing a token renewal mechanism. This is not discussed further in this specification.

1278 **6.4.2. EndpointUpdate Type and Element**

1279 Receivers of an ID-* message may add an <EndpointUpdate> element to the SOAP header of their response.

1280 This element is based upon the EndpointUpdateType which is an extension of wsa:EndpointReferenceType
1281 that adds the following attributes:

- 1282 • updateType [Optional] – A URI attribute indicating whether the update should be interpreted as completely
1283 superseding the endpoint reference used to send the current request (the default) or whether it should be interpreted
1284 as a partial updated.

1285 urn:liberty:sb:2006-08:EndpointUpdate:Complete
1286 A complete update.

1287 urn:liberty:sb:2006-08:EndpointUpdate:Partial
1288 A partial update. The complete updated endpoint reference is constructed according to the processing rules below.

1289 The following schema fragment describes the <EndpointUpdate> header block.

```
1290  
1291  
1292 <!-- epr update header block -->  
1293  
1294 <xs:complexType name="EndpointUpdateType">  
1295   <xs:complexContent>  
1296     <xs:extension base="wsa:EndpointReferenceType">  
1297       <xs:attribute name="updateType" type="xs:anyURI" use="optional"/>  
1298     </xs:extension>  
1299   </xs:complexContent>  
1300 </xs:complexType>  
1301  
1302 <xs:element name="EndpointUpdate" type="EndpointUpdateType"/>  
1303  
1304  
1305
```

1306 **Figure 9. The <EndpointUpdate> Header Block Schema**

1307 **6.4.3. EndpointUpdate Examples**

```
1308
1309
1310     1. Service responds to a request, indicating a new security mechanism and credential
1311
1312 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1313     xmlns:sb="urn:liberty:sb:2006-08"
1314     xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
1315
1316     <S:Header>
1317
1318         ...
1319
1320     <sb:EndpointUpdate mustUnderstand="1" updateType="urn:liberty:sb:2004-04:Partial">
1321     <wsa:Address>urn:liberty:sb:2006-08:EndpointUpdate:NoChange</wsa:Address>
1322     <wsa:Metadata>
1323     <ds:SecurityContext>
1324     <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</sb:SecurityMechID>
1325     <wsse:SecurityTokenReference>
1326     <wsse:Embedded>
1327     <wsse:BinarySecurityToken xmlns:wsse="..." wsu:Id="..."
1328     ValueType="anyNSprefix:ServiceSessionContext">
1329     ZjgzOWZlNzgyZTk1ZWU3OWEyMTRlODVmNGZkYzE4MmQ2ZDNhMzc3Nwo=
1330     </wsse:BinarySecurityToken>
1331     </wsse:Embedded>
1332     </wsse:SecurityTokenReference>
1333     </ds:SecurityContext>
1334     </wsa:Metadata>
1335     </sb:EndpointUpdate>
1336
1337     ...
1338
1339     </S:Header>
1340
1341     <S:Body>
1342
1343     <idpp:QueryResponse>
1344     ...
1345     </idpp:QueryResponse>
1346
1347     </S:Body>
1348
1349 </S:Envelope>
```

```
1350
1351
1352     2. The client sends a new request, using the contents of the EndpointUpdate
1353
1354 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1355     xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
1356
1357   <S:Header>
1358
1359     ...
1360
1361   <wsse:Security xmlns:wsse="...">
1362
1363     <wsse:BinarySecurityToken xmlns:wsse="..." wsu:Id="bst "
1364       ValueType="anyNSprefix:ServiceSessionContext">
1365       ZjgzOWZlNzgyZTk1ZWU3OWEyMTRlODVmNGZkYzE4MmQ2ZDZhMzc3Nwo=
1366     </wsse:BinarySecurityToken>
1367
1368   </wsse:Security>
1369
1370     ...
1371
1372   </S:Header>
1373
1374   <S:Body>
1375
1376     <idpp:Query>
1377       ...
1378     </idpp:Query>
1379
1380   </S:Body>
1381
1382 </S:Envelope>
1383
```

1384 **Example 12. An EndpointUpdate Specifying a ServiceSessionContext Token, and the TLS Bearer Security Mechanism.**

```
1385
1386 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1387           xmlns:sb="urn:liberty:sb:2006-08">
1388
1389   <S:Header>
1390
1391     ...
1392
1393     <sb:EndpointUpdate mustUnderstand="1" updateType="urn:liberty:sb:2004-04:Partial">
1394       <wsa:Address>http://example.com:443/soap</wsa:Address>
1395     </sb:EndpointUpdate>
1396
1397     ...
1398
1399   </S:Header>
1400
1401   <S:Body>
1402
1403     <!-- a fault in the body indicates that the request corresponding
1404           to this response should be re-submitted to the endpoint -->
1405
1406     <S:Fault>
1407
1408       <faultcode>S:Server</faultcode>
1409
1410       <faultstring>
1411         You must resubmit this request to the new endpoint.
1412       </faultstring>
1413
1414       <detail>
1415         <lu:Status code="EndpointUpdated"/>
1416       </detail>
1417
1418     </S:Fault>
1419
1420   </S:Body>
1421
1422 </S:Envelope>
1423
```

1424 **Example 13. An EndpointUpdate Specifying an Updated Address.**

1425 **6.4.4. Processing Rules for the EndpointUpdate header**

1426 **6.4.4.1. Sender Processing Rules**

1427 The receiver of an ID-* message MAY add an <EndpointUpdate> header block to their response.

1428 If updateType is not present or has the value urn:liberty:sb:2006-08:EndpointUpdate:Complete, the
1429 <wsa:EndpointUpdate> MUST be a completely specified endpoint reference.

1430 If updateType has the value urn:liberty:sb:2006-08:EndpointUpdate:Partial, the <wsa:EndpointUpdate>
1431 MAY omit any direct children of <wsa:ReferenceParameters> or <wsa:Metadata> that have not changed from
1432 the original endpoint reference used to send the current request. Similarly, any extension elements that have not
1433 changed MAY be omitted. If the address has not changed, then the URI

1434 urn:liberty:sb:2006-08:EndpointUpdate:NoChange

1435 MAY be used in the <wsa:Address> value to indicate that the original address should continue to be used.

1436 **Note**

1437 The expressiveness of partial updates is limited. In particular, updates to `<wsa:ReferenceParameters>` and
1438 `<wsa:Metadata>` are done based on the qualified names of the direct children of those containers. If any child
1439 with a matching name is provided in the update, then all children with that name in the original are replaced. It is also
1440 impossible, with a partial update, to remove an element; elements may only be added or replaced.

1441 **6.4.4.2. Receiver Processing Rules**

1442 The receiver of an `<EndpointUpdate>` header SHOULD use the specified endpoint reference values to address any
1443 future requests to the sender of the header (where the endpoint reference used to address the request that resulted in
1444 the response containing the header would have been used), until newer information is obtained through this or some
1445 other mechanism or the updated information expires. If the updated information has a shorter lifetime than the current
1446 information (that it updates), then the current information SHOULD be retained as a fallback for when the updated
1447 information expires.

1448 If `updateType` is not present or has the value `urn:liberty:sb:2006-08:EndpointUpdate:Complete`, the
1449 `<wsa:EndpointUpdate>` is a completely specified endpoint reference.

1450 If `updateType` has the value `urn:liberty:sb:2006-08:EndpointUpdate:Partial`, the `<wsa:EndpointUpdate>`
1451 is a partially specified endpoint reference. The following steps are used to construct a complete endpoint reference
1452 from the endpoint reference that was used to address the request that resulted in the response containing this header:

- 1453 1. Take the `<wsa:Address>` from the `<wsa:EndpointUpdate>`. If the value is `urn:liberty:sb:2006-08:`
1454 `EndpointUpdate:NoChange`, then take the `<wsa:Address>` from the original endpoint reference.
- 1455 2. Take the `<wsa:ReferenceParameters>` from the `<wsa:EndpointUpdate>`, if present. Then, if
1456 `<wsa:ReferenceParameters>` is present in the original endpoint reference, take each direct child from
1457 that element that does not match an element already taken from the update (comparing the namespace qualified
1458 names of the elements).
- 1459 3. Take the `<wsa:Metadata>` from the `<wsa:EndpointUpdate>`, if present. Then, if `<wsa:Metadata>` is
1460 present in the original endpoint reference, take each direct child from that element that does not match an element
1461 already taken from the update (comparing the namespace qualified names of the elements).
- 1462 4. Take any extension elements from the `<wsa:EndpointUpdate>`, if present. Then, if any extension elements are
1463 present in the original endpoint reference, take each one that does not match an element already taken from the
1464 update (comparing the namespace qualified names of the elements).

1465 **6.4.5. Processing Rules for the EndpointUpdated SOAP Fault**

1466 **6.4.5.1. Sender Processing Rules**

1467 The receiver of an ID-* message MAY issue a SOAP Fault indicating that the endpoint to which this message was
1468 submitted has permanently changed.

1469 Once the receiver has sent this fault response, no further processing of the message should take place.

1470 If the receiver chooses to send the fault response, then it SHOULD also include an `<EndpointUpdate>` header,
1471 indicating the new endpoint which should be used to re-submit this message, and any further messages directed to the
1472 responding service.

1473 **6.4.5.2. Receiver Processing Rules**

1474 If the receiver of this fault response also received an `<EndpointUpdate>` header in the response, it MAY re-submit
1475 the failed request to any endpoint specified in that header, but it SHOULD provide a different `<wsa:MessageID>`
1476 header block value in the request.

1477 **6.5. The `<Timeout>` Header Block**

1478 **6.5.1. Overview**

1479 A requesting entity may wish to indicate that they would like a request to be processed within some specified amount
1480 of time. Such an entity would indicate their wish via the `<Timeout>` header block.

1481 **6.5.2. Timeout Type and Element**

1482 Senders of ID-* messages MAY add a `<Timeout>` element to the SOAP header of their request.

1483 This element is based upon the `TimeoutType` which is defined as:

- 1484 • `maxProcessingTime` [Required] – an integer specifying (in seconds) the maximum amount of time the sender
1485 wishes the receiver to spend in processing their request

1486 The following schema fragment describes the `<Timeout>` header block.

```
1487  
1488  
1489 <!-- timeout header block -->  
1490  
1491 <xs:complexType name="TimeoutType">  
1492   <xs:attribute name="maxProcessingTime" type="xs:integer" use="required"/>  
1493   <xs:anyAttribute namespace="##other" processContents="lax"/>  
1494 </xs:complexType>  
1495  
1496 <xs:element name="Timeout" type="TimeoutType"/>  
1497  
1498  
1499
```

1500 **Figure 10. The `<Timeout>` Header Block Schema**

1501 **6.5.3. Timeout Example**

```

1502
1503     <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1504               xmlns:sb="urn:liberty:sb:2006-08"
1505               xmlns:idpp="urn:liberty:id-sis-pp:2003-08">
1506
1507     <S:Header>
1508
1509         ...
1510
1511     <sb:Timeout mustUnderstand="1" wsu:Id="timeout.123"
1512               maxProcessingTime="7"/>
1513
1514         ...
1515
1516     </S:Header>
1517
1518     <S:Body>
1519
1520     <idpp:Query>
1521         ...
1522     </idpp:Query>
1523
1524     </S:Body>
1525
1526 </S:Envelope>
1527

```

Example 14. Example of a Request with Timeout Specified

```

1529
1530 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1531           xmlns:sb="urn:liberty:sb:2006-08">
1532
1533     <S:Header>
1534
1535         ...
1536
1537     </S:Header>
1538
1539     <S:Body>
1540
1541     <S:Fault>
1542
1543     <faultcode>
1544         S:Server
1545     </faultcode>
1546
1547     <detail>
1548
1549     <lu:Status code="ProcessingTimeout"/>
1550
1551     <!-- Reference the specified Timeout header, if it was supplied
1552           by the sender -->
1553
1554     <sb:Timeout wsu:Id="timeout.123"/>
1555
1556     </detail>
1557     </S:Fault>
1558
1559     </S:Body>
1560
1561 </S:Envelope>
1562

```

Example 15. Example of a Timed-out Response

1564 6.5.4. Processing Rules

1565 6.5.4.1. Receiver Processing Rules

1566 The receiver of a `<Timeout>` header SHOULD NOT begin processing of a message (beyond processing of the
1567 SOAP headers as noted in this specification) if it expects that such processing would exceed the value specified in
1568 the `maxProcessingTime` attribute.

1569 The receiver MUST respond to the message within the number of seconds specified in the `maxProcessingTime`
1570 attribute.

1571 If the receiver is unable to complete processing within the number of seconds specified in the `maxProcessingTime` at-
1572 tribute of the `<Timeout>` header, then they MUST respond with a SOAP Fault with a `code` of `ProcessingTimeout`.

1573 Note

1574 If the sender of a message does not include a `<Timeout>` header, but the receiver wishes to indicate to the sender
1575 that server processing failed due to a timeout, then the receiver MAY respond with a SOAP Fault with a `code` of
1576 `ProcessingTimeout`.

1577 6.6. The `<UsageDirective>` Header Block

1578 This section defines the ID-* usage directive facilities.

1579 6.6.1. Overview

1580 Participants in the ID-WSF framework may need to indicate the privacy policy associated with a message. To facilitate
1581 this, senders, acting as either a client or a server, may add one or more `<UsageDirective>` header blocks to the
1582 SOAP Header of the message being sent. A `<UsageDirective>` appearing in a SOAP-based ID-* request message
1583 expresses *intended usage*. A `<UsageDirective>` appearing in a response expresses *how* the receiver of the response
1584 is to use the response data. A `<UsageDirective>` in a response message containing no ID-WSF response message
1585 data, a fault response for example, may be used to express policies acceptable to the responder.

1586 6.6.2. `UsageDirective` Type and Element

1587 Senders MAY add a `<UsageDirective>` element to the SOAP header. This element is based upon the
1588 `UsageDirectiveType` which is defined as:

- 1589 • `ref` [Required] – An attribute referring to an element of the SOAP-based ID-* message to which the usage directive
1590 applies.
- 1591 • `<element>(s)` [Optional] – Elements, comprising an instance of some policy expression language, whose
1592 purpose is to express the actual policy the usage directive is conveying. The `ref` attribute above points at the
1593 element in the overall SOAP-based ID-* message to which the usage directive applies.

1594 The schema fragment in [Figure 11](#) defines the <UsageDirective> header type and element.

```
1595
1596
1597 <!-- usage directive header block -->
1598
1599 <xs:complexType name="UsageDirectiveType">
1600   <xs:sequence>
1601     <xs:any namespace="##other" processContents="lax"
1602       maxOccurs="unbounded" />
1603   </xs:sequence>
1604   <xs:attribute name="ref" type="xs:IDREF" use="required" />
1605   <xs:anyAttribute namespace="##other" processContents="lax" />
1606 </xs:complexType>
1607
1608 <xs:element name="UsageDirective" type="UsageDirectiveType" />
1609
1610
```

1611 **Figure 11. The <UsageDirective> Header Block Schema**

1612 6.6.3. Usage Directive Examples

1613 [Example 16](#) illustrates a SOAP-based ID-* message, containing a <UsageDirective> header block, and
1614 conveying a Personal Profile (ID-PP) Modify message [[LibertyIDPP](#)]. The <UsageDirective> header block
1615 contains a usage directive expressed in a policy language identified by the cot: namespace and the URI
1616 <http://cot.example.com/policies/eu-compliant>, and applying to the ID-PP Query message identified by the id of
1617 datarequest001.

```
1618
1619 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1620   xmlns:sb="urn:liberty:sb:2006-08"
1621   xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1622
1623   <S:Header>
1624
1625     ...
1626
1627     <sb:UsageDirective S:mustUnderstand="1"
1628       ref="#datarequest001">
1629
1630       <cot:PrivacyPolicyReference
1631         xmlns:cot="http://cot.example.com/isf">
1632         http://cot.example.com/policies/eu-compliant
1633       </cot:PrivacyPolicyReference>
1634
1635     </sb:UsageDirective>
1636
1637     ...
1638
1639   </S:Header>
1640
1641   <S:Body>
1642
1643     <pp:Query id="datarequest001" xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1644       <pp:ResourceID>data:d8ddw6dd7m28v628</pp:ResourceID>
1645       <pp:QueryItem>
1646         <pp:Select>/pp:IDPP/pp:IDPPAddressCard</pp:Select>
1647       </pp:QueryItem>
1648     </pp:Query>
1649
1650   </S:Body>
1651
1652 </S:Envelope>
1653
```

1654 **Example 16. A Usage Directive on a Request for the Address of a Principal.**

1655 **6.6.4. Processing Rules**

1656 **6.6.4.1. Sender Processing Rules**

1657 The sender of a SOAP-based ID-* message with a <UsageDirective> header block MUST ensure that the value
1658 of the ref attribute is set to the value of the id of the appropriate element in the message. The sender SHOULD
1659 ensure that the <UsageDirective> is integrity-protected. The protection mechanism, if utilized, SHOULD be in
1660 accordance with those defined in [LibertySecMech].

1661 **6.6.4.2. Receiver Processing Rules**

1662 A receiver of a SOAP-based ID-* message with an attached <UsageDirective> header block MUST check the
1663 actor attribute and determine if it, the receiver, is the actor the header block is targeted at. If so, the receiver MUST
1664 check the mustUnderstand attribute. If set to TRUE the receiver MUST process the contents. If the attribute is absent
1665 or set to FALSE the receiver SHOULD attempt to process the content of the <UsageDirective> header block.

1666 A receiver that processes the contents of a <UsageDirective> header block SHOULD verify the integrity of the
1667 header block – that is, it should verify any digital signatures that list the header block in its manifest [XMLDsig]. The
1668 receiver MUST verify that the ref attribute refers to an element in the message. That receiver MUST further process
1669 the message according to the policy expressed by the children elements of the <UsageDirective> header block.
1670 Those children elements will be imported from a foreign namespace, and MUST be parsed and interpreted according
1671 to the applicable schema and processing rules of that foreign namespace.

1672 A receiver that cannot process a <UsageDirective> with mustUnderstand set to TRUE MUST respond with a
1673 <S:Fault>. The <s:Fault> MUST contain a <detail> element which in turn MUST contain a <Status> element
1674 with its code attribute set to CannotHonourUsageDirective. The <Status> element SHOULD possess a ref
1675 attribute with its value set to the value of the id attribute of the offending <UsageDirective> header block in the
1676 request message.

1677 A receiver that cannot honor a non-mandatory (without mustUnderstand set to TRUE) <UsageDirective> must
1678 respond according to the contained policy. In addition, in this case the receiver MAY respond with a SOAP-based
1679 ID-* message that includes a <Status> element with its code attribute set to CannotHonourUsageDirective.
1680 This <Status> element instance SHOULD include a ref attribute with its value set to the value of the id attribute of
1681 the <UsageDirective> header block in the request message that could not be honored.

1682 In this case, the receiver MAY include one or more new <UsageDirective> header blocks in its response message,
1683 each expressing a policy that the receiver would have been able to honor. The ref attribute of these headers SHOULD
1684 be set to the value of the <wsa:MessageID> header block in the request.

1685 **6.7. The <ApplicationEPR> Header Block**

1686 This section defines the <ApplicationEPR> header block. This header may be included in a message zero or more
1687 times and provides a means for a sender to specify application endpoints that may be referenced from the SOAP Body
1688 of the message.

1689 The <ApplicationEPR> header block is an extension of <wsa:EndpointReferenceType>.

1690 The schema fragment in Figure 12 defines the The <ApplicationEPR> header block.

```

1691
1692
1693 <!-- application epr header block -->
1694
1695 <xs:element name="ApplicationEPR" type="wsa:EndpointReferenceType" />
1696
1697

```

Figure 12. The <ApplicationEPR> Header Block Schema

```

1699
1700 <sb:ApplicationEPR S:mustUnderstand="1"
1701   S:actor="http://schemas.../next"
1702   wsu:Id="NotifyTo-001">
1703   <wsa:Address>...</wsa:Address>
1704 </sb:ApplicationEPR>
1705

```

Example 17. An instantiated <ApplicationEPR> header block

6.8. The <UserInteraction> Header Block

6.8.1. Overview

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

6.8.2. UserInteraction Element

The <UserInteraction> element contains:

InteractionService [Optional]

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

interact [Optional]

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

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

1734 **Note:**

1735 Implementors may choose to define additional values to indicate finer grained control over the user interactions.

1736 language [Optional]

1737 This attribute indicates languages that the user is likely able to process. The value of this attribute is a space
1738 separated list of language identification tags ([RFC3066]). The WSC can obtain this information from the
1739 HTTP ([RFC2616]) *Accept-Language* header, or by other means, for example from a *personal profile service*.

1740 redirect [Optional]

1741 An optional attribute to indicate that the sender supports the `<RedirectRequest>` element that a WSP may
1742 include in a message to the WSC. The value is *true* or *false*. When absent the default behavior will be as if
1743 *false*.

1744 maxInteractTime [Optional]

1745 This is used to indicate the maximum time in seconds that the sender regards as reasonable for any possible
1746 interaction. The receiver is not expected to start any interaction if it has reason to assume that such an
1747 interaction is likely to take more time. In case an interaction is started and does seem to take longer the
1748 receiver is expected to respond with a message that contains a *InteractionTimeout* status code to the sender.

1749 The schema fragment in [Figure 13](#) defines the The `<UserInteraction>` header block.

```

1750
1751 <!-- user interaction header block -->
1752
1753 <xs:complexType name="UserInteractionHeaderType">
1754   <xs:sequence>
1755     <xs:element name="InteractionService" type="wsa:EndpointReferenceType"
1756       minOccurs="0" maxOccurs="unbounded" />
1757   </xs:sequence>
1758   <xs:attribute name="interact" type="xs:string" use="optional" default="InteractIfNeeded" />
1759   <xs:attribute name="language" type="xs:NMTOKENS" use="optional" />
1760   <xs:attribute name="redirect" type="xs:boolean" use="optional" default="0" />
1761   <xs:attribute name="maxInteractTime" type="xs:integer" use="optional" />
1762   <xs:anyAttribute namespace="##other" processContents="lax" />
1763 </xs:complexType>
1764
1765 <xs:element name="UserInteraction" type="UserInteractionHeaderType" />
1766
1767
1768

```

1769 **Figure 13. The `<UserInteraction>` Header Block Schema**

1770 **6.8.3. UserInteraction Examples**

1771 Below is an example for a WSC that is prepared to redirect the user to a WSP, and also is ready to accept an
1772 `<is:InteractionRequest>`. The WSC wishes that the WSP will not attempt to prompt the resource owner for
1773 missing data; but accepts interactions for consent, as long as questioning the user will not take more than 60 seconds.
1774 The WSC expects the user to understand US English and Finnish.

```

1775
1776 <sb:UserInteraction interact="DoNotInteractForData" language="en-US fi"
1777   maxInteractTime="60" redirect="true">
1778
1779   <sb:InteractionService xmlns:disco="urn:liberty:disco:2006-08">
1780     <wsa:Address>endpoint for interaction requests</wsa:Address>
1781     <wsa:Metadata>
1782       <disco:ServiceType>urn:liberty:is:2006-08</disco:ServiceType>
1783       <disco:Provider>http://someWSC</disco:Provider>

```

```

1784     <disco:Description>
1785         <disco:Endpoint>http://IS.com/soap</disco:Endpoint>
1786     </disco:Description>
1787 </wsa:Metadata>
1788     </sb:InteractionService>
1789 </sb:UserInteraction>

```

1790 The following is an example for a WSC that wants to ensure that the WSP will not attempt to contact the requesting principal, even if this may hinder serving the actual request; the WSC would rather receive an error, or a less optimal response (e.g., fewer profile attributes).

```

1793     <sb:UserInteraction interact="DoNotInteract" />
1794
1795

```

1796 6.8.4. Processing Rules

1797 If the sender includes an `InteractionService` element, it MUST set the value of `<disco:ServiceType>` within to `urn:liberty:is:2006-08`.

1799 If the sender sets `interact="DoNotInteract"` it MUST omit the `InteractionService` element, as well as the `language`, `redirect` and `maxInteractTime` attributes.

1801 The recipient of a message with a `UserInteraction` element MUST NOT respond with a `<RedirectRequest>` if the `redirect` is `false` or if `redirect` is absent.

1803 The recipient MUST NOT start a requesting principal interaction if the `interact` attribute has a value of `"DoNotInteract"`.

1805 The recipient MUST NOT interact with the requesting principal to obtain data that is to be included in a successful service response if the `interact` attribute has a value of `"DoNotInteractForData"`. In this case the recipient MAY start an interaction if a policy concerning available data so requires; for example if a policy requires that the Principal must be prompted for consent.

1809 The recipient SHOULD NOT start a requesting principal interaction if it expects that the time to complete the interaction will exceed the value of the `maxInteractTime`.

1811 The recipient MUST respond to the message after at most the number of seconds given as the value of the `maxInteractTime` attribute.

1813 The sender must ensure that the `UserInteraction` element is integrity protected; i.e., if message level authentication (see [LibertySecMech]) is used the sender MUST sign the `UserInteraction` element. Likewise the receiver must ensure that the integrity of the `UserInteraction` element is not compromised, according to the processing rules in [LibertySecMech].

1817 6.8.4.1. UserInteraction Faults

1818 A processor of a `UserInteraction` that must indicate an error situation related to this header SHOULD respond to the sender with an ID-WSF message that contains a `Status` element in the `detail` element of a `S:Fault`, or in a service specific `S:Body` component, or inside a higher level `Status` element. The `code` attribute of the included `Status` element can be set to one of the following values:

- 1822 • *InteractionRequired*, as indication that the recipient has a need to start an interaction in order to satisfy the service request but the `interact` attribute value was set to `DoNotInteract`.

- 1824 • *InteractionRequiredForData*. This indicates that the service request could not be satisfied because the WSP would
1825 have to interact with the requesting principal in order to obtain (some of) the requested data but the *interact*
1826 attribute value was set to *DoNotInteractForData*.
- 1827 • *InteractionTimeNotSufficient*, as indication that the recipient has a need to start an interaction but has reason to
1828 believe that more time is needed than allowed for by the value of the *maxInteractTime* attribute.
- 1829 • *InteractionTimeout*, as indication that the recipient could not satisfy the service request due to an unfinished
1830 interaction.

1831 **6.8.5. Cross-principal interactions**

1832 A 'cross-principal' interaction is defined by a WSC making a request on behalf of a principal who is different
1833 than the principal who 'owns' the resource in question. In such a case, the identity of the requesting principal
1834 will be identified by the security context of the message. The identity of the resource owner is expressed by the
1835 <sb:TargetIdentity> header.

1836 Any *sb:UserInteraction* header in such a message always refers to the requesting principal. Consequently, if
1837 the WSP desires to interact with the requesting principal, it may use the interaction options as indicated by the
1838 *UserInteraction* (if present) or discover the requesting principal's permanent IS.

1839 If the WSP desires to interact with the resource owner (as indicated by the *TargetIdentity* header), it will
1840 necessarily need to discover that principal's permanent IS as the alternative interaction mechanisms are not an option.

7. The RedirectRequest Protocol

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

7.1. RedirectRequest Element

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

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

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

The schema fragment for the element is:

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

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

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/02/addressing">...</wsa:MessageID>
    <wsa:RelatesTo>...</wsa:RelatesTo>
  </S:Header>
  <S:Body>
    <S:Fault>
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring>Server Error</faultstring>
      <detail>
        <RedirectRequest redirectURL="https://someWSP/getConsent?transID=de67hj89jk65nk34">
          Redirecting to AP to obtain consent
        </RedirectRequest>
      </detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

7.1.1. Processing Rules

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

1887 The recipient MUST attempt to direct the user agent to issue an HTTP request ([RFC2616]) for the URL in the
1888 `redirectURL` attribute of the `<RedirectRequest>`. That user agent MUST be associated with the ID-WSF request
1889 that caused the `<RedirectRequest>`. The recipient MUST add a `ReturnToURL` parameter to the `redirectURL`
1890 with its value the URL-encoded URL which the recipient wants the user agent directed back to. It is recommended
1891 that this `ReturnToURL` includes an identifier that associates the URL to the originating ID-WSF message to the WSP.
1892 The recipient MAY add an `IDP` parameter to the `redirectURL` with its value the `providerID` of an identity provider
1893 that was used to authenticate the user to the WSC.

1894 The recipient may instruct the user agent to submit either an HTTP GET or an HTTP POST request to the URL; in
1895 this way the WSC can avoid problems with user agents that can handle only short URLs. If the user agent is instructed
1896 to submit a HTTP POST, *all* URL parameters should be form-encoded, and the HTTP content-type header of the
1897 request MUST be `application/x-www-form-urlencoded`. Note that this implies that the WSP SHOULD accept
1898 both an HTTP GET as well as an HTTP POST request for the `redirectURL`, but in either case retrieval of URL
1899 parameters can be done using well-known techniques; most HTTP server environments effectively encapsulate the
1900 different methods for submission of parameters.

1901 As an example, assume that a Principal visits a service provider. As a result the service provider (acting as WSC)
1902 could have made a request to a WSP, and that WSP would have responded with a SOAP Fault similar to that of the
1903 example above. The WSC would now send a HTTP response to the user agent that would look like:

```
1904 HTTP 302
1905 Location:
1906 https://someWSP/getConsent?transID=de67hj89jk65nk34&ReturnToURL=
1907 https%3a%2f%2fsomeWSC%2fisReturn3bjsession%3d9A6F2E3A&IDP=1A2B3C4D5E1A2B3C4D5E
1908 ... other HTTP headers ...
1909
1910 <html>
1911   <head>
1912     <title>Redirecting...</title>
1913   </head>
1914   <body>
1915     <p>Redirecting to AP to obtain consent</p>
1916   </body>
1917 </html>
```

1918 The WSC appends its own `ReturnToURL` as a parameter to the value of the `redirectURL` element that the WSC
1919 specified in its `RedirectRequest`.

1920 7.2. RedirectRequest Protocol

1921 The `<RedirectRequest>` protocol consists of the following steps, each with normative rules:

1922 7.2.1. Step 1: WSC Issues Normal ID-WSF Request

1923 For the `<RedirectRequest>` protocol to be initiated the originating ID-WSF message MUST contain a
1924 `UserInteraction` element with its `redirect` attribute set to *true*.

1925 The ID-WSF message SHOULD contain a `wsa:FaultTo` element to which the WSC desires fault messages be sent.

1926 7.2.2. Step 2: WSP Responds with `<RedirectRequest>`

1927 If, and only if, an ID-WSF message contains a `<UserInteraction>` element with its `redirect` attribute set to *true*
1928 MAY the recipient of the ID-WSF message respond with a `<RedirectRequest>` message in a SOAP Fault.

1929 **Note:**

1930 The `redirectURL` attribute **MUST** be constructed as to include the necessary information for mapping the upcoming
1931 HTTP request to the originating ID-WSF message; for example by inclusion of the value of the `wsa:MessageID`
1932 Header from that message.

1933 **7.2.3. Step 3: WSC Instructs User Agent to Contact the WSP**

1934 When the WSC receives a `<RedirectRequest>` it **MUST** attempt to direct the user agent to issue an HTTP request
1935 for the URL in the `redirectURL` attribute of the `RedirectRequest`. The user agent **MUST** be associated with the
1936 ID-WSF message that caused the `<RedirectRequest>`. The WSC **MUST** append a `ReturnToURL` parameter to the
1937 `redirectURL` with its value the URL-encoded URL to which the WSC wants the user agent directed back.

1938 **Note:**

1939 How this step is performed will depend on the user agent. In most cases it is accomplished by a simple HTTP 302
1940 response with a `Location` header set to the `redirectURL`. Different user agents may be better served by other
1941 approaches, for example a WML browser may be able to handle a redirect deck better than a potentially long URL.
1942 See the [processing rules for the `<RedirectRequest>`](#).

1943 **7.2.4. Step 4: WSP Interacts with User Agent**

1944 In step 4 the user agent issues the HTTP request for the `redirectURL`, with the `ReturnToURL` parameter appended,
1945 with any `IDP` parameter also appended. The WSP **MUST** verify that the `ReturnToURL` points to the WSC, i.e., the
1946 host in the URL should be the same as the host to which the WSP sent the `<RedirectRequest>`. If this is not the
1947 case the WSP **MUST** ignore the `ReturnToURL`, abort the protocol, and construct a meaningful error message for the
1948 user. If verification succeeds, however, the service (WSP) **MAY** now proceed with a HTTP response that contains
1949 an inquiry directed at the user. The WSP **SHOULD** verify that the identity of the user is that of the owner of the
1950 resource that was targeted in the originating ID-WSF request, for example by means of a `<saml:AuthnRequest>`
1951 (see [SAMLCore2]). This step may be followed by any number of interactions between the user and the WSP, but the
1952 WSP should attempt to execute step 5 within a reasonable time.

1953 **7.2.5. Step 5: WSP Redirects User Agent Back to WSC**

1954 In step 5 the WSP that issued the `<RedirectRequest>` **MUST** attempt to instruct the user agent to issue an HTTP
1955 request for the `ReturnToURL` that was included as parameter on the URL of the HTTP request made in step 4. The
1956 WSP **SHOULD** append a `ResendMessage` parameter to the `ReturnToURL`. This parameter serves as a hint to the
1957 WSC about the next step. A value of `0` or `false` indicates that the WSC should not try to re-issue the originating
1958 ID-WSF request, presumably because the resource owner did not approve completion of the transaction. If the value
1959 of `ResendMessage` is `true`, `1`, or any string other than `0` or `false`, it is an indication that the WSP recommends that the
1960 WSC re-issue the originating request. It is **RECOMMENDED** that in this situation, the value of this parameter be set
1961 to the value of the `wsa:MessageID` element of the originating ID-WSF message.

1962 **7.2.6. Step 6: User Agent Requests `ReturnToURL` from WSC**

1963 In step 6 the user agent requests the `ReturnToURL` from the WSC. The WSC **SHOULD** check the value of the
1964 `ResendMessage` parameter; if the value is `0` or `false` the WSC **SHOULD NOT** send an ID-WSF message with a
1965 request for the same resource and/or action (as in step 1). If the value of the `ResendMessage` parameter is anything
1966 else, then the WSC **MAY** resend the message (Step 7).

1967 After receiving the response from the WSP, the WSC should send a HTTP response to the user agent.

1968 **7.2.7. Step 7: WSC Resends Message**

1969 If the WSC resends its request it **MUST** set the value of the `wsa:RelatesTo` SOAP Header to the same value of the
1970 `wsa:MessageID` SOAP Header of the SOAP Fault that carried the `<RedirectRequest>` element (in step 2). .

1971 **7.2.8. Steps 8: WSP sends response**

1972 The WSP responds to the WSC's second request. The WSP MUST set the value of the `wsa:RelatesTo` SOAP
1973 Header to the same value of the `wsa:MessageID` SOAP Header of the WSC's resent request.

1974 **7.2.9. Steps 9: WSC sends HTTP response to User Agent**

1975 Finally, the WSC returns an HTTP response to the user agent.

1976 8. Security Considerations

- 1977 • The header blocks specified in this document should be integrity-protected using the mechanisms detailed in
1978 [[LibertySecMech](#)].
- 1979 • Header blocks should be signed in accordance with [[LibertySecMech](#)]. The receiver of a message containing a
1980 signature that covers specific header blocks should verify the signature as part of verifying the integrity of the
1981 header block.
- 1982 • Metadata [[LibertyMetadata](#)] should be used to the greatest extent possible to verify message sender identity claims.
- 1983 • Message senders and receivers should be authenticated to one another via the mechanisms discussed in [[Liberty-
1984 SecMech](#)].
- 1985 • To prevent message replay, receivers should maintain a message cache, and check received `messageID` values
1986 against the cache.

1987 **9. Acknowledgements**

1988 The members of the Liberty Technical Expert group, especially Greg Whitehead, Jonathan Sergent, Xavier Serret,
1989 and Conor Cahill, provided valuable input to this specification. The docbook source code for this specification was
1990 hand set to the tunes of The Sugarcubes, King Crimson, Juliana Hatfield, Smashing Pumpkins, Evanescence, Mad at
1991 Gravity, Elisa Korenne, The Breeders, fIREHOSE, Polly Jean Harvey, Jimi Hendrix, and various others.

References

Normative

1992

1993

1994 [LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-
1995 errata-v1.0, Liberty Alliance Project (21 April, 2007). <http://www.projectliberty.org/specs>

1996 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 2.0-02,
1997 Liberty Alliance Project (25 November 2004). <http://www.projectliberty.org/specs>

1998 [LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version 2.0-errata-v1.0,
1999 Liberty Alliance Project (21 April, 2007). <http://www.projectliberty.org/specs>

2000 [RFC2045] Freed, N., Borenstein, N., eds. (November 1996). "Multipurpose Internet Mail Extensions
2001 (MIME) Part One: Format of Internet Message Bodies ," RFC 2045., Internet Engineering Task
2002 Force <http://www.ietf.org/rfc/rfc2045.txt>

2003 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
2004 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>

2005 [RFC2828] Shirey, R., eds. (May 2000). "Internet Security Glossary," RFC 2828., Internet Engineering Task Force
2006 <http://www.ietf.org/rfc/rfc2828.txt>

2007 [RFC3986] Berners-Lee, T., Fielding, R., Masinter, L., eds. (January 2005). "Uniform Resource Identifier
2008 (URI): Generic Syntax," RFC 3986 (Obsoletes RFC2732, RFC2396, RFC1808) (Updates RFC1738) (Also
2009 STD0066) (Status: STANDARD), The Internet Engineering Task Force <http://www.ietf.org/rfc/rfc3986.txt>

2010 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions
2011 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
2012 SIS Standard, Organization for the Advancement of Structured Information Standards [http://docs.oasis-
2013 open.org/security/saml/v2.0/saml-core-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf)

2014 [SAMLGloss2] Hodges, Jeff, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Glossary for the OASIS Security As-
2015 sertion Markup Language (SAML) V2.0," SAML 2.0, OASIS Standard, Organization for the Advancement
2016 of Structured Information Standards <http://docs.oasis-open.org/security/saml/v2.0/saml-glossary-2.0-os.pdf>

2017 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October
2018 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium
2019 <http://www.w3.org/TR/xmlschema-1/>

2020 [Schema2-2] Biron, Paul V., Malhotra, Ashok, eds. (28 October 2004). "XML Schema Part 2: Datatypes Second
2021 Edition," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlschema-2/>

2022 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David , Kakivaya, Gopal, Layman,
2023 Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C
2024 Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>

2025 [SOAPv1.1-Schema] "SOAP 1.1 Envelope schema," W3C W3C Note <http://schemas.xmlsoap.org/soap/envelope/>

2026 [WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Mered-
2027 ith, Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001).
2028 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

2029
2030
2031

2032
2033

2034
2035
2036

2037
2038

2039
2040

2041
2042
2043
2044

2045

2046
2047

2048
2049

2050
2051

2052
2053

2054
2055

2056
2057

2058

2059
2060
2061

2062
2063

2064
2065

2066
2067
2068

[XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004). "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/2004/REC-xml-20040204>

[XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlldsig-core>

[WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, Rogers, Tony, eds. World Wide Web Consortium W3C Recommendation (9 May 2006). <http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/>

[WSAv1.0-SOAP] "WS-Addressing 1.0 SOAP Binding," Gudgin, Martin, Hadley, Marc, eds. World Wide Web Consortium W3C Recommendation (9 May 2006). <http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/>

[WSAv1.0-Schema] "WS-Addressing 1.0 Schema," W3C, W3C Working Draft <http://dev.w3.org/cvsweb/~checkout~/2004/ws/addressing-addr.xsd>

[wss-sms] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (January, 2004). "Web Services Security: SOAP Message Security," OASIS Standard V1.0 [OASIS 200401], Organization for the Advancement of Structured Information Standards <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>

Informational

[LibertyDisco] Cahill, Conor, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (29 November, 2006). <http://www.projectliberty.org/specs>

[LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>

[LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework Overview," Version 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>

[LibertyIDWSF20SCR] Whitehead, Greg, eds. Version 1.0 errata v1.0, Liberty Alliance Project (21 April, 2007). <http://www.projectliberty.org/specs>

[LibertyIDPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification," Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>

[LibertyIDWSFv20Errata] Champagne, Darryl, Lockhart, Rob, Tiffany, Eric, eds. "Liberty ID-WSF 2.0 Errata," Version 1.0, Liberty Alliance Project (13 April, 2007). <http://www.projectliberty.org/specs>

[Merriam-Webster] "Merriam-Webster Dictionary," <http://www.merriam-webster.com/>

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., eds. (June 1999). "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, The Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2616.txt>

[RFC3066] Alvestrand, H., eds. (January 2001). "Tags for the Identification of Languages," RFC 3066., Internet Engineering Task Force <http://www.ietf.org/rfc/rfc3066.txt>

[RFC4086] Eastlake, D., Schiller, J., Crocker, S., eds. (June 2005). "Randomness Recommendations for Security," RFC 4086, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc4086.txt>

[SOAPv1.2] "SOAP Version 1.2 Part 1: Messaging Framework," Gudgin, Martin, Hadley, Marc, Mendelsohn, Noah, Moreau, Jean-Jacques, Nielsen, Henrik Frystyk, eds. World Wide Web Consortium W3C Recommendation (07 May 2003). <http://www.w3.org/TR/2003/PR-soap12-part1-20030507/>

2069 A. liberty-idwsf-soap-binding.xsd Schema Listing

```
2070 <?xml version="1.0" encoding="UTF-8"?>
2071 <xs:schema targetNamespace="urn:liberty:sb"
2072   xmlns:xs="http://www.w3.org/2001/XMLSchema"
2073   xmlns="urn:liberty:sb"
2074   elementFormDefault="qualified"
2075   attributeFormDefault="unqualified">
2076
2077   <!-- framework header block -->
2078
2079   <xs:complexType name="FrameworkType">
2080     <xs:sequence>
2081       <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2082     </xs:sequence>
2083     <xs:attribute name="version" type="xs:string" use="required"/>
2084     <xs:anyAttribute namespace="##other" processContents="lax"/>
2085   </xs:complexType>
2086
2087   <xs:element name="Framework" type="FrameworkType"/>
2088
2089 </xs:schema>
2090
```

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

2091

```
2092 <?xml version="1.0" encoding="UTF-8"?>
2093 <xs:schema targetNamespace="urn:liberty:sb:2006-08"
2094   xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
2095   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
2096   xmlns:wsa="http://www.w3.org/2005/08/addressing"
2097   xmlns:xs="http://www.w3.org/2001/XMLSchema"
2098   xmlns:lu="urn:liberty:util:2006-08"
2099   xmlns="urn:liberty:sb:2006-08"
2100   elementFormDefault="qualified"
2101   attributeFormDefault="unqualified">
2102
2103   <xs:import
2104     namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
2105     schemaLocation="wss-util-1.0.xsd"/>
2106
2107   <xs:import
2108     namespace="urn:oasis:names:tc:SAML:2.0:protocol"
2109     schemaLocation="saml-schema-protocol-2.0.xsd"/>
2110
2111   <xs:import
2112     namespace="http://www.w3.org/2005/08/addressing"
2113     schemaLocation="ws-addr-1.0.xsd"/>
2114
2115   <xs:import
2116     namespace="urn:liberty:util:2006-08"
2117     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
2118
2119   <!-- sender header block -->
2120
2121   <xs:complexType name="SenderType">
2122     <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
2123     <xs:attribute name="affiliationID" type="xs:anyURI" use="optional"/>
2124     <xs:anyAttribute namespace="##other" processContents="lax"/>
2125   </xs:complexType>
2126
2127   <xs:element name="Sender" type="SenderType"/>
2128
2129
2130   <!-- target identity header block -->
2131
2132   <xs:complexType name="TargetIdentityType">
2133     <xs:sequence>
2134       <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2135     </xs:sequence>
2136     <xs:anyAttribute namespace="##other" processContents="lax"/>
2137   </xs:complexType>
2138
2139   <xs:element name="TargetIdentity" type="TargetIdentityType"/>
2140
2141
2142   <!-- credentials context header block -->
2143
2144   <xs:complexType name="CredentialsContextType">
2145     <xs:sequence>
2146       <xs:element ref="samlp:RequestedAuthnContext" minOccurs="0"/>
2147       <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
2148     </xs:sequence>
2149     <xs:anyAttribute namespace="##other" processContents="lax"/>
2150   </xs:complexType>
2151
2152   <xs:element name="CredentialsContext" type="CredentialsContextType"/>
2153
2154
2155   <!-- epr update header block -->
```

```

2157
2158 <xs:complexType name="EndpointUpdateType">
2159   <xs:complexContent>
2160     <xs:extension base="wsa:EndpointReferenceType">
2161       <xs:attribute name="updateType" type="xs:anyURI" use="optional"/>
2162     </xs:extension>
2163   </xs:complexContent>
2164 </xs:complexType>
2165
2166 <xs:element name="EndpointUpdate" type="EndpointUpdateType"/>
2167
2168
2169 <!-- timeout header block -->
2170
2171 <xs:complexType name="TimeoutType">
2172   <xs:attribute name="maxProcessingTime" type="xs:integer" use="required"/>
2173   <xs:anyAttribute namespace="##other" processContents="lax"/>
2174 </xs:complexType>
2175
2176 <xs:element name="Timeout" type="TimeoutType"/>
2177
2178
2179 <!-- processing context header block -->
2180
2181 <xs:complexType name="ProcessingContextType">
2182   <xs:simpleContent>
2183     <xs:extension base="xs:anyURI">
2184       <xs:anyAttribute namespace="##other" processContents="lax"/>
2185     </xs:extension>
2186   </xs:simpleContent>
2187 </xs:complexType>
2188
2189 <xs:element name="ProcessingContext" type="ProcessingContextType"/>
2190
2191 <!-- consent header block -->
2192
2193 <xs:complexType name="ConsentType">
2194   <xs:attribute name="uri" type="xs:anyURI" use="required"/>
2195   <xs:attribute name="timestamp" type="xs:dateTime" use="optional"/>
2196   <xs:anyAttribute namespace="##other" processContents="lax"/>
2197 </xs:complexType>
2198
2199 <xs:element name="Consent" type="ConsentType"/>
2200
2201 <!-- usage directive header block -->
2202
2203 <xs:complexType name="UsageDirectiveType">
2204   <xs:sequence>
2205     <xs:any namespace="##other" processContents="lax"
2206       maxOccurs="unbounded"/>
2207   </xs:sequence>
2208   <xs:attribute name="ref" type="xs:IDREF" use="required"/>
2209   <xs:anyAttribute namespace="##other" processContents="lax"/>
2210 </xs:complexType>
2211
2212 <xs:element name="UsageDirective" type="UsageDirectiveType"/>
2213
2214 <!-- application epr header block -->
2215
2216 <xs:element name="ApplicationEPR" type="wsa:EndpointReferenceType"/>
2217
2218 <!-- user interaction header block -->
2219
2220 <xs:complexType name="UserInteractionHeaderType">
2221   <xs:sequence>
2222     <xs:element name="InteractionService" type="wsa:EndpointReferenceType"
2223       minOccurs="0" maxOccurs="unbounded"/>

```

```
2224     </xs:sequence>
2225     <xs:attribute name="interact" type="xs:string" use="optional" default="InteractIfNeeded"/>
2226     <xs:attribute name="language" type="xs:NMTOKENS" use="optional"/>
2227     <xs:attribute name="redirect" type="xs:boolean" use="optional" default="0"/>
2228     <xs:attribute name="maxInteractTime" type="xs:integer" use="optional"/>
2229     <xs:anyAttribute namespace="##other" processContents="lax"/>
2230 </xs:complexType>
2231
2232 <xs:element name="UserInteraction" type="UserInteractionHeaderType"/>
2233 <xs:element name="RedirectRequest" type="RedirectRequestType"/>
2234 <xs:complexType name="RedirectRequestType">
2235   <xs:attribute name="redirectURL" type="xs:anyURI" use="required"/>
2236 </xs:complexType>
2237 </xs:schema>
2238
```

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

2239

```
2240 <?xml version="1.0" encoding="UTF-8"?>
2241 <xs:schema targetNamespace="urn:liberty:util:2006-08"
2242   xmlns:xs="http://www.w3.org/2001/XMLSchema"
2243   xmlns="urn:liberty:util:2006-08"
2244   elementFormDefault="qualified"
2245   attributeFormDefault="unqualified"
2246   version="2.0-03">
2247
2248   <xs:annotation>
2249     <xs:documentation>
2250       Liberty Alliance Project utility schema. A collection of common
2251       IDentity Web Services Framework (ID-WSF) elements and types.
2252       This schema is intended for use in ID-WSF schemas.
2253
2254       This version: 2006-08
2255
2256       Copyright (c) 2006 Liberty Alliance participants, see
2257       http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2258     </xs:documentation>
2259   </xs:annotation>
2260   <xs:simpleType name="IDType">
2261     <xs:annotation>
2262       <xs:documentation>
2263         This type should be used to provide IDs to components
2264         that have IDs that may not be scoped within the local
2265         xml instance document.
2266       </xs:documentation>
2267     </xs:annotation>
2268     <xs:restriction base="xs:string"/>
2269   </xs:simpleType>
2270   <xs:simpleType name="IDReferenceType">
2271     <xs:annotation>
2272       <xs:documentation>
2273         This type can be used when referring to elements that are
2274         identified using an IDType.
2275       </xs:documentation>
2276     </xs:annotation>
2277     <xs:restriction base="xs:string"/>
2278   </xs:simpleType>
2279   <xs:attribute name="itemID" type="IDType"/>
2280   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2281   <xs:complexType name="StatusType">
2282     <xs:annotation>
2283       <xs:documentation>
2284         A type that may be used for status codes.
2285       </xs:documentation>
2286     </xs:annotation>
2287     <xs:sequence>
2288       <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
2289     </xs:sequence>
2290     <xs:attribute name="code" type="xs:string" use="required"/>
2291     <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
2292     <xs:attribute name="comment" type="xs:string" use="optional"/>
2293   </xs:complexType>
2294
2295   <xs:element name="Status" type="StatusType">
2296     <xs:annotation>
2297       <xs:documentation>
2298         A standard Status type
2299       </xs:documentation>
2300     </xs:annotation>
2301   </xs:element>
2302
2303   <xs:complexType name="ResponseType">
2304     <xs:sequence>
```

```
2305     <xs:element ref="Status"      minOccurs="1" maxOccurs="1"/>
2306     <xs:element ref="Extension"   minOccurs="0" maxOccurs="unbounded"/>
2307   </xs:sequence>
2308   <xs:attribute ref="itemIDRef" use="optional"/>
2309   <xs:anyAttribute namespace="##other" processContents="lax"/>
2310 </xs:complexType>
2311 <xs:element name="TestResult" type="TestResultType"/>
2312 <xs:complexType name="TestResultType">
2313   <xs:simpleContent>
2314     <xs:extension base="xs:boolean">
2315       <xs:attribute ref="itemIDRef" use="required"/>
2316     </xs:extension>
2317   </xs:simpleContent>
2318 </xs:complexType>
2319 <xs:complexType name="EmptyType">
2320   <xs:annotation>
2321     <xs:documentation> This type may be used to create an empty element </xs:documentation>
2322   </xs:annotation>
2323   <xs:complexContent>
2324     <xs:restriction base="xs:anyType"/>
2325   </xs:complexContent>
2326 </xs:complexType>
2327 <xs:element name="Extension" type="extensionType">
2328   <xs:annotation>
2329     <xs:documentation>
2330       An element that contains arbitrary content extensions
2331       from other namespaces
2332     </xs:documentation>
2333   </xs:annotation>
2334 </xs:element>
2335 <xs:complexType name="extensionType">
2336   <xs:annotation>
2337     <xs:documentation>
2338       A type for arbitrary content extensions from other namespaces
2339     </xs:documentation>
2340   </xs:annotation>
2341   <xs:sequence>
2342     <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
2343   </xs:sequence>
2344 </xs:complexType>
2345 </xs:schema>
2346
```

2347 D. liberty-utility-v2.0.xsd Schema Listing

```
2348 <?xml version="1.0" encoding="UTF-8"?>
2349 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2350   elementFormDefault="qualified"
2351   attributeFormDefault="unqualified"
2352   version="2.0-01">
2353   <xs:annotation>
2354     <xs:documentation>
2355       Liberty Alliance Project utility schema. A collection of common
2356       elements and types for use with independent Liberty XML Schema documents.
2357
2358       This file intended for inclusion, rather than importation, into other schemas.
2359       This version: 2004-12
2360
2361       Copyright (c) 2004 Liberty Alliance participants, see
2362       http://www.projectliberty.org/specs/idff_copyrights.html
2363     </xs:documentation>
2364   </xs:annotation>
2365   <xs:simpleType name="IDType">
2366     <xs:annotation>
2367       <xs:documentation>
2368         This type should be used to provide IDs to components
2369         that have IDs that may not be scoped within the local
2370         xml instance document.
2371       </xs:documentation>
2372     </xs:annotation>
2373     <xs:restriction base="xs:string"/>
2374   </xs:simpleType>
2375   <xs:simpleType name="IDReferenceType">
2376     <xs:annotation>
2377       <xs:documentation>
2378         This type can be used when referring to elements that are
2379         identified using an IDType.
2380       </xs:documentation>
2381     </xs:annotation>
2382     <xs:restriction base="xs:string"/>
2383   </xs:simpleType>
2384   <xs:complexType name="StatusType">
2385     <xs:annotation>
2386       <xs:documentation>
2387         A type that may be used for status codes.
2388       </xs:documentation>
2389     </xs:annotation>
2390     <xs:sequence>
2391       <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
2392     </xs:sequence>
2393     <xs:attribute name="code" type="xs:string" use="required"/>
2394     <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
2395     <xs:attribute name="comment" type="xs:string" use="optional"/>
2396   </xs:complexType>
2397
2398   <xs:element name="Status" type="StatusType">
2399     <xs:annotation>
2400       <xs:documentation>
2401         A standard Status type
2402       </xs:documentation>
2403     </xs:annotation>
2404   </xs:element>
2405
2406   <xs:complexType name="EmptyType">
2407     <xs:annotation>
2408       <xs:documentation> This type may be used to create an empty element </xs:documentation>
2409     </xs:annotation>
2410     <xs:complexContent>
2411       <xs:restriction base="xs:anyType"/>
2412     </xs:complexContent>
```

```
2413 </xs:complexType>
2414 <xs:element name="Extension" type="extensionType">
2415   <xs:annotation>
2416     <xs:documentation>
2417       An element that contains arbitrary content extensions
2418       from other namespaces
2419     </xs:documentation>
2420   </xs:annotation>
2421 </xs:element>
2422 <xs:complexType name="extensionType">
2423   <xs:annotation>
2424     <xs:documentation>
2425       A type for arbitrary content extensions from other namespaces
2426     </xs:documentation>
2427   </xs:annotation>
2428   <xs:sequence>
2429     <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
2430   </xs:sequence>
2431 </xs:complexType>
2432 </xs:schema>
2433
```

E. wss-util-1.0.xsd Schema Listing

2434

```
2435 <?xml version="1.0" encoding="UTF-8"?>
2436 <!--
2437 OASIS takes no position regarding the validity or scope of any intellectual property or other
2438 rights that might be claimed to pertain to the implementation or use of the technology described
2439 in this document or the extent to which any license under such rights might or might not be
2440 available; neither does it represent that it has made any effort to identify any such rights.
2441 Information on OASIS's procedures with respect to rights in OASIS specifications can be found
2442 at the OASIS website. Copies of claims of rights made available for publication and any
2443 assurances of licenses to be made available, or the result of an attempt made to obtain a
2444 general license or permission for the use of such proprietary rights by implementors or users
2445 of this specification, can be obtained from the OASIS Executive Director.
2446 OASIS invites any interested party to bring to its attention any copyrights, patents or
2447 patent applications, or other proprietary rights which may cover technology that may be
2448 required to implement this specification. Please address the information to the OASIS Executive Director.
2449
2450 Copyright © OASIS Open 2002-2004. All Rights Reserved.
2451 This document and translations of it may be copied and furnished to others, and derivative
2452 works that comment on or otherwise explain it or assist in its implementation may be prepared,
2453 copied, published and distributed, in whole or in part, without restriction of any kind,
2454 provided that the above copyright notice and this paragraph are included on all such copies
2455 and derivative works. However, this document itself does not be modified in any way, such
2456 as by removing the copyright notice or references to OASIS, except as needed for the purpose
2457 of developing OASIS specifications, in which case the procedures for copyrights defined
2458 in the OASIS Intellectual Property Rights document must be followed, or as required to
2459 translate it into languages other than English.
2460 The limited permissions granted above are perpetual and will not be revoked by OASIS
2461 or its successors or assigns.
2462 This document and the information contained herein is provided on an "AS IS" basis
2463 and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
2464 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
2465 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
2466 -->
2467 <xsd:schema targetNamespace="http://docs.oasis-open.org/wss/2004/01/oasis-
2468 200401-wss-wssecurity-utility-1.0.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2469
2470
2471
2472 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.
2473 0.xsd" xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
2474 elementFormDefault="qualified" attributeFormDefault="unqualified" version="0.1">
2475   <!-- // Fault Codes //////////////////////////////////////// -->
2476   <xsd:simpleType name="tTimestampFault">
2477     <xsd:annotation>
2478       <xsd:documentation>
2479         This type defines the fault code value for Timestamp message expiration.
2480       </xsd:documentation>
2481     </xsd:annotation>
2482     <xsd:restriction base="xsd:QName">
2483       <xsd:enumeration value="wsu:MessageExpired"/>
2484     </xsd:restriction>
2485   </xsd:simpleType>
2486   <!-- // Global attributes //////////////////////////////////////// -->
2487   <xsd:attribute name="Id" type="xsd:ID">
2488     <xsd:annotation>
2489       <xsd:documentation>
2490         This global attribute supports annotating arbitrary elements with an ID.
2491       </xsd:documentation>
2492     </xsd:annotation>
2493   </xsd:attribute>
2494   <xsd:attributeGroup name="commonAtts">
2495     <xsd:annotation>
2496       <xsd:documentation>
2497         Convenience attribute group used to simplify this schema.
2498       </xsd:documentation>
2499     </xsd:annotation>
```

```
2500     <xsd:attribute ref="wsu:Id" use="optional" />
2501     <xsd:anyAttribute namespace="##other" processContents="lax" />
2502 </xsd:attributeGroup>
2503 <!-- // Utility types ////////////////////////////////////// -->
2504 <xsd:complexType name="AttributedDateTime">
2505     <xsd:annotation>
2506         <xsd:documentation>
2507 This type is for elements whose [children] is a psuedo-dateTime and can have arbitrary attributes.
2508 </xsd:documentation>
2509     </xsd:annotation>
2510     <xsd:simpleContent>
2511         <xsd:extension base="xsd:string">
2512             <xsd:attributeGroup ref="wsu:commonAtts" />
2513         </xsd:extension>
2514     </xsd:simpleContent>
2515 </xsd:complexType>
2516 <xsd:complexType name="AttributedURI">
2517     <xsd:annotation>
2518         <xsd:documentation>
2519 This type is for elements whose [children] is an anyURI and can have arbitrary attributes.
2520 </xsd:documentation>
2521     </xsd:annotation>
2522     <xsd:simpleContent>
2523         <xsd:extension base="xsd:anyURI">
2524             <xsd:attributeGroup ref="wsu:commonAtts" />
2525         </xsd:extension>
2526     </xsd:simpleContent>
2527 </xsd:complexType>
2528 <!-- // Timestamp header components ////////////////////////////////////// -->
2529 <xsd:complexType name="TimestampType">
2530     <xsd:annotation>
2531         <xsd:documentation>
2532 This complex type ties together the timestamp related elements into a composite type.
2533 </xsd:documentation>
2534     </xsd:annotation>
2535     <xsd:sequence>
2536         <xsd:element ref="wsu:Created" minOccurs="0" />
2537         <xsd:element ref="wsu:Expires" minOccurs="0" />
2538         <xsd:choice minOccurs="0" maxOccurs="unbounded">
2539             <xsd:any namespace="##other" processContents="lax" />
2540         </xsd:choice>
2541     </xsd:sequence>
2542     <xsd:attributeGroup ref="wsu:commonAtts" />
2543 </xsd:complexType>
2544 <xsd:element name="Timestamp" type="wsu:TimestampType">
2545     <xsd:annotation>
2546         <xsd:documentation>
2547 This element allows Timestamps to be applied anywhere element wildcards are present,
2548 including as a SOAP header.
2549     </xsd:documentation>
2550     </xsd:annotation>
2551 </xsd:element>
2552 <!-- global element decls to allow individual elements to appear anywhere -->
2553 <xsd:element name="Expires" type="wsu:AttributedDateTime">
2554     <xsd:annotation>
2555         <xsd:documentation>
2556 This element allows an expiration time to be applied anywhere element wildcards are present.
2557 </xsd:documentation>
2558     </xsd:annotation>
2559 </xsd:element>
2560 <xsd:element name="Created" type="wsu:AttributedDateTime">
2561     <xsd:annotation>
2562         <xsd:documentation>
2563 This element allows a creation time to be applied anywhere element wildcards are present.
2564     </xsd:documentation>
2565     </xsd:annotation>
2566 </xsd:element>
```

2567 </xsd:schema>
2568

F. ws-addr-1.0.xsd Schema Listing

2569

```
2570 <?xml version="1.0" encoding="utf-8"?>
2571 <!DOCTYPE xs:schema PUBLIC "-//W3C//DTD XMLSCHEMA 200102//EN" "http://www.w3.org/2001/XMLSchema.dtd">
2572
2573 <!--
2574     W3C XML Schema defined in the Web Services Addressing 1.0 specification
2575     http://www.w3.org/TR/ws-addr-core
2576
2577     Copyright © 2005 World Wide Web Consortium,
2578
2579     (Massachusetts Institute of Technology, European Research Consortium for
2580     Informatics and Mathematics, Keio University). All Rights Reserved. This
2581     work is distributed under the W3C® Software License [1] in the hope that
2582     it will be useful, but WITHOUT ANY WARRANTY; without even the implied
2583     warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
2584
2585     [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
2586
2587     $Id: ws-addr-1.0.xsd 3060 2005-09-23 18:20:29Z dchampagne $
2588 -->
2589 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2590           xmlns:tns="http://www.w3.org/2005/08/addressing"
2591           targetNamespace="http://www.w3.org/2005/08/addressing"
2592           blockDefault="#all"
2593           elementFormDefault="qualified"
2594           finalDefault=""
2595           attributeFormDefault="unqualified">
2596
2597   <!-- Constructs from the WS-Addressing Core -->
2598
2599   <xs:element name="EndpointReference" type="tns:EndpointReferenceType"/>
2600   <xs:complexType name="EndpointReferenceType" mixed="false">
2601     <xs:sequence>
2602       <xs:element name="Address" type="tns:AttributedURIType"/>
2603       <xs:element name="ReferenceParameters" type="tns:ReferenceParametersType" minOccurs="0"/>
2604       <xs:element ref="tns:Metadata" minOccurs="0"/>
2605       <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2606     </xs:sequence>
2607     <xs:anyAttribute namespace="##other" processContents="lax"/>
2608   </xs:complexType>
2609
2610   <xs:complexType name="ReferenceParametersType" mixed="false">
2611     <xs:sequence>
2612       <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2613     </xs:sequence>
2614     <xs:anyAttribute namespace="##other" processContents="lax"/>
2615   </xs:complexType>
2616
2617   <xs:element name="Metadata" type="tns:MetadataType"/>
2618   <xs:complexType name="MetadataType" mixed="false">
2619     <xs:sequence>
2620       <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2621     </xs:sequence>
2622     <xs:anyAttribute namespace="##other" processContents="lax"/>
2623   </xs:complexType>
2624
2625   <xs:element name="MessageID" type="tns:AttributedURIType"/>
2626   <xs:element name="RelatesTo" type="tns:RelatesToType"/>
2627   <xs:complexType name="RelatesToType" mixed="false">
2628     <xs:simpleContent>
2629       <xs:extension base="xs:anyURI">
2630         <xs:attribute name="RelationshipType" type="tns:RelationshipTypeOpenEnum" use="optional"
2631
2632           default="http://www.w3.org/2005/08/addressing/reply"/>
2633         <xs:anyAttribute namespace="##other" processContents="lax"/>
2634       </xs:extension>

```

```

2635     </xs:simpleContent>
2636 </xs:complexType>
2637
2638 <xs:simpleType name="RelationshipTypeOpenEnum">
2639   <xs:union memberTypes="tns:RelationshipType xs:anyURI" />
2640 </xs:simpleType>
2641
2642 <xs:simpleType name="RelationshipType">
2643   <xs:restriction base="xs:anyURI">
2644     <xs:enumeration value="http://www.w3.org/2005/08/addressing/reply" />
2645   </xs:restriction>
2646 </xs:simpleType>
2647
2648 <xs:element name="ReplyTo" type="tns:EndpointReferenceType" />
2649 <xs:element name="From" type="tns:EndpointReferenceType" />
2650 <xs:element name="FaultTo" type="tns:EndpointReferenceType" />
2651 <xs:element name="To" type="tns:AttributedURIType" />
2652 <xs:element name="Action" type="tns:AttributedURIType" />
2653
2654 <xs:complexType name="AttributedURIType" mixed="false">
2655   <xs:simpleContent>
2656     <xs:extension base="xs:anyURI">
2657       <xs:anyAttribute namespace="##other" processContents="lax" />
2658     </xs:extension>
2659   </xs:simpleContent>
2660 </xs:complexType>
2661
2662 <!-- Constructs from the WS-Addressing SOAP binding -->
2663
2664 <xs:attribute name="IsReferenceParameter" type="xs:boolean" />
2665
2666 <xs:simpleType name="FaultCodesOpenEnumType">
2667   <xs:union memberTypes="tns:FaultCodesType xs:QName" />
2668 </xs:simpleType>
2669
2670 <xs:simpleType name="FaultCodesType">
2671   <xs:restriction base="xs:QName">
2672     <xs:enumeration value="tns:InvalidAddressingHeader" />
2673     <xs:enumeration value="tns:InvalidAddress" />
2674     <xs:enumeration value="tns:InvalidEPR" />
2675     <xs:enumeration value="tns:InvalidCardinality" />
2676     <xs:enumeration value="tns:MissingAddressInEPR" />
2677     <xs:enumeration value="tns:DuplicateMessageID" />
2678     <xs:enumeration value="tns:ActionMismatch" />
2679     <xs:enumeration value="tns:MessageAddressingHeaderRequired" />
2680     <xs:enumeration value="tns:DestinationUnreachable" />
2681     <xs:enumeration value="tns:ActionNotSupported" />
2682     <xs:enumeration value="tns:EndpointUnavailable" />
2683   </xs:restriction>
2684 </xs:simpleType>
2685
2686 <xs:element name="RetryAfter" type="tns:AttributedUnsignedLongType" />
2687 <xs:complexType name="AttributedUnsignedLongType" mixed="false">
2688   <xs:simpleContent>
2689     <xs:extension base="xs:unsignedLong">
2690       <xs:anyAttribute namespace="##other" processContents="lax" />
2691     </xs:extension>
2692   </xs:simpleContent>
2693 </xs:complexType>
2694
2695 <xs:element name="ProblemHeaderQName" type="tns:AttributedQNameType" />
2696 <xs:complexType name="AttributedQNameType" mixed="false">
2697   <xs:simpleContent>
2698     <xs:extension base="xs:QName">
2699       <xs:anyAttribute namespace="##other" processContents="lax" />
2700     </xs:extension>
2701   </xs:simpleContent>

```

```
2702     </xs:complexType>
2703
2704     <xs:element name="ProblemHeader" type="tns:AttributedAnyType"/>
2705     <xs:complexType name="AttributedAnyType" mixed="false">
2706         <xs:sequence>
2707             <xs:any namespace="##any" processContents="lax" minOccurs="1" maxOccurs="1"/>
2708         </xs:sequence>
2709         <xs:anyAttribute namespace="##other" processContents="lax"/>
2710     </xs:complexType>
2711
2712     <xs:element name="ProblemIRI" type="tns:AttributedURIType"/>
2713
2714     <xs:element name="ProblemAction" type="tns:ProblemActionType"/>
2715     <xs:complexType name="ProblemActionType" mixed="false">
2716         <xs:sequence>
2717             <xs:element ref="tns:Action" minOccurs="0"/>
2718             <xs:element name="SoapAction" minOccurs="0" type="xs:anyURI"/>
2719         </xs:sequence>
2720         <xs:anyAttribute namespace="##other" processContents="lax"/>
2721     </xs:complexType>
2722
2723 </xs:schema>
2724
2725
```