



Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification

Version:

2.0-errata-v1.0

Editors:

Jeff Hodges, NeuStar, Inc.
Robert Aarts, Hewlett-Packard
Paul Madsen, NTT
Scott Cantor, Internet2 / The Ohio State University

Contributors:

Conor Cahill, America Online, Inc.
Darryl Champagne, IEEE-ISTO
Gary Ellison, Sun Microsystems, Inc.

Rob

Lockhart

, IEEE-ISTO
Greg Whitehead, Hewlett-Packard

Abstract:

Abstract

This specification defines an ID-WSF Authentication Protocol based on a profile of the Simple Authentication and Security Layer (SASL) framework mapped onto ID-* SOAP-bound messages. It also defines an ID-WSF Authentication Service which Identity Providers may offer. This service is based on the authentication protocol. The authentication service enables Web Services Consumers and/or Liberty-enabled User Agents or Devices to authenticate with Identity Providers, using various authentication mechanisms, and obtain ID-WSF security tokens. Next, it defines the ID-WSF Single Sign-On Service, which provides SAML authentication assertions to Web Service Consumers via profiles of the SAML 2.0 Authentication Request protocol, enabling Web Service Consumers and/or Liberty-enabled User Agents or Devices to interact with SAML-based services. Finally, it defines the ID-WSF Identity Mapping Service, which allows Web Service Consumers to obtain identity tokens for use in web service invocations and referencing principals while preserving privacy.

Filename: liberty-idwsf-authn-svc-2.0-diff-v1.0.pdf

1

Notice

2 This document has been prepared by Sponsors of the Liberty Alliance. Permission is hereby granted to use the document
3 solely for the purpose of implementing the Specification. No rights are granted to prepare derivative works of this
4 Specification. Entities seeking permission to reproduce portions of this document for other uses must contact the Liberty
5 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 rights,
7 including without limitation, patent rights. The Sponsors of and any other contributors to the Specification are not and
8 shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual
9 property rights. **This Specification is provided "AS IS", and no participant in the Liberty Alliance makes any
10 warranty of any kind, express or implied, including any implied warranties of merchantability, non-infringe-
11 ment of third party intellectual property rights, and fitness for a particular purpose.** Implementers of this
12 Specification are advised to review the Liberty Alliance Project's website (<http://www.projectliberty.org/>) for infor-
13 mation 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.; Amer-
16 ican Express Company; Amsoft Systems Pvt Ltd.; Avatier Corporation; BIPAC; BMC Software, Inc.; Axalto; Bank of
17 America Corporation; Beta Systems Software AG; BIPAC; 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 Systems
20 LLC; Fidelity Investments; Forum Systems, Inc.; France Télécom; French Government Agence pour le développement
21 de l'administration électronique (ADAE); Fugen Solutions, Inc; Fulvens Ltd.; GSA Office of Governmentwide Policy;
22 Gamefederation; Gemalto; General Motors; GeoFederation; Giesecke & Devrient GmbH; Hewlett-Packard GSA Office
23 Company; Hochhauser & Co., Policy; Hewlett-Packard LLC; IBM Corporation; Intel Corporation; Intuit Inc.; Kant-
24 ega; 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 Corpo-
27 ration; Novell, Inc.; NTT DoCoMo, Inc.; OpenNetwork; Oracle Corporation; Ping Identity Corporation; RSA Security
28 Inc.; Reactivity Inc.; Royal Mail Group plc; RSA Security Inc.; SAP AG; Senforce; Sharp Laboratories of America;
29 Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Supremacy Financial Corporation; Symlabs, Inc.;
30 Telecom Italia S.p.A.; Telefónica Móviles, S.A.; Telenor R&D; Thales e-Security; Trusted Network Technologies;
31 UNINETT AS; UTI; 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

38 Contents

39	1. Introduction.....	6
40	2. Notation and Conventions	7
41	2.1. Requirements Keywords.....	7
42	2.2. XML Namespaces.....	7
43	3. Terminology.....	9
44	4. Authentication Protocol	12
45	4.1. Conceptual Model.....	12
46	4.2. Schema Declarations	12
47	4.3. SOAP Header Blocks and SOAP Binding	12
48	4.3.1. SOAP Binding.....	12
49	4.4. SASL Profile Particulars	13
50	4.4.1. SASL "Service Name"	13
51	4.4.2. Composition of SASL Mechanism Names	13
52	4.5. Authentication Exchange Security.....	13
53	4.6. Protocol Messages	13
54	4.6.1. The <SASLRequest> Message	13
55	4.6.2. The <SASLResponse> Message	16
56	4.7. Sequencing of the Authentication Exchange	19
57	5. Authentication Service.....	23
58	5.1. Conceptual Model	23
59	5.1.1. Stipulating a Particular Authentication Context	23
60	5.2. URI Declarations	24
61	5.3. Rules for Authentication Service Providers	24
62	5.4. Rules for Authentication Service Consumers	25
63	5.5. Authentication Service Interaction Example	26
64	6. Single Sign-On Service.....	28
65	6.1. Conceptual Model.....	28
66	6.2. Single Sign-On Service URIs	29
67	6.3. ID-WSF Enhanced Client or Proxy SSO Profile.....	29
68	6.3.1. Profile Overview	29
69	6.3.2. Profile Description.....	29
70	6.4. ID-WSF SAML Token Service Profile.....	30
71	6.4.1. Profile Overview	31
72	6.4.2. Profile Description.....	31
73	6.4.3. Use of SAML 2.0 Authentication Request Protocol.....	32
74	6.5. Use of Metadata.....	33
75	6.6. Inclusion of ID-WSF Endpoint References.....	33
76	7. Identity Mapping Service	34
77	7.1. Conceptual Model	34
78	7.2. Schema Declarations	34
79	7.3. SOAP Binding.....	34
80	7.3.1. Identity Mapping Service URIs	35
81	7.4. Protocol Messages and Usage	35
82	7.4.1. Element <IdentityMappingRequest>	35
83	7.4.2. Element <IdentityMappingResponse>	36
84	7.5. SAML Identity Tokens.....	38
85	7.5.1. Assertions	38
86	7.5.2. Identifiers	39
87	7.6. Security and Privacy Considerations	39
88	7.7. Example Identity Mapping Exchange.....	40
89	8. Password Transformations: The PasswordTransforms Element	41

90	9. Acknowledgments	43
91	References.....	44
92	A. Listing of Simple Authentication and Security Layer (SASL) Mechanisms	47
93	B. Password Transformations	49
94	1. Truncation.....	49
95	2. Lowercase.....	49
96	3. Uppercase	49
97	4. Select	49
98	C. liberty-idwsf-authn-svc-v2.0.xsd Schema Listing	51
99	D. liberty-idwsf-idmapping-svc-v2.0.xsd Schema Listing	54
100	E. liberty-idwsf-utility-v2.0.xsd Schema Listing	56
101	F. liberty-idwsf-authn-svc-v2.0.wsdl WSDL Listing	58
102	G. liberty-idwsf-sso-svc-v2.0.wsdl WSDL Listing.....	60
103	H. liberty-idwsf-idmapping-svc-v2.0.wsdl WSDL Listing.....	62

104 1. Introduction

105 The Simple Object Access Protocol (SOAP) specifications, [SOAPv1.1] and [SOAPv1.2], define an XML-based
106 [XML] messaging paradigm, but do not specify any particular security mechanisms. They do not, in particular, describe
107 how one *SOAP node* may authenticate with another *SOAP node* via an exchange of SOAP messages. Thus it is left to
108 SOAP-based web services frameworks to provide their own notions of security, such as defining how authentication
109 is accomplished.

110 This specification defines how to perform *general identity authentication* [WooLam92], also known as *peer entity*
111 *authentication* [RFC2828], over SOAP, in the context of the Liberty Identity Web Services Framework (ID-WSF)
112 [LibertyIDWSFOverview]. Rather than specify the particulars of one or more *authentication mechanisms* directly in
113 this specification, we profile the Simple Authentication and Security Layer (SASL) framework [RFC4422].

114 SASL is an approach to modularizing protocol *design* such that the security design components, e.g., authentication
115 and security layer mechanisms, are reduced to a uniform abstract interface. This facilitates a protocol's use of an open-
116 ended set of security mechanisms, as well as a so-called "late binding" between implementations of the protocol and
117 the security mechanisms' implementations. This late binding can occur at implementation- and/or deployment-time.
118 The SASL specification also defines how one packages authentication and security layer mechanisms to fit into the
119 SASL framework, where they are known as *SASL mechanisms*, as well as register them with the Internet Assigned
120 Numbers Authority (IANA) [IANA] for reuse.

121 This specification is organized as follows. First, it defines the ID-WSF Authentication Protocol. Then, it defines an
122 ID-WSF Authentication Service Identity Providers may offer, which is based on the authentication protocol. This
123 authentication service enables Web Services Consumers and/or Liberty-enabled User Agents or Devices to authenticate
124 with Identity Providers using various authentication mechanisms and obtain ID-WSF security tokens. Next, it defines
125 the ID-WSF Single Sign-On Service, which provides SAML authentication assertions to Web Service Consumers via
126 profiles of the SAML 2.0 Authentication Request protocol, enabling Web Service Consumers and/or Liberty-enabled
127 User Agents or Devices to interact with SAML-based services. Finally, it defines the ID-WSF Identity Mapping Service,
128 which allows Web Service Consumers to obtain identity tokens for use in web service invocations and referencing
129 principals while preserving privacy.

130 2. Notation and Conventions

131 This specification uses schema documents conforming to W3C XML Schema [Schema1-2] and normative text to
132 describe the syntax and semantics of XML-encoded protocol messages.

133 2.1. Requirements Keywords

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

136 "They MUST only be used where it is actually required for interoperation or to limit behavior which
137 has potential for causing harm (e.g., limiting retransmissions)"

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

141 2.2. XML Namespaces

142 This specification uses the XML namespace prefixes listed in Table 1.

143

Table 1. XML Namespaces used in this specification

Prefix	Namespace
sa:	<p>Represents the ID-WSF Authentication Service namespace: urn:liberty:sa:2006-08</p> <p>Note</p> <p>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.</p>
disco:	Represents the namespace defined in [LibertyDisco].
sec:	Represents the namespace defined in [LibertySecMech].
md:	Represents the namespace defined in [SAMLMeta2].
pp:	Represents the namespace defined in [LibertyIDPP].
s:	Represents the SOAP namespace: http://www.w3.org/2001/12/soap-envelope , defined in [SOAPv1.1].
saml2:	Represents the SAML V2.0 Assertion namespace defined in [SAMLCore2]
samlp2:	Represents the SAML V2.0 Protocol namespace defined in [SAMLCore2]
sb:	Represents the Liberty namespace defined in [LibertySOAPBinding]
lu:	Represents the Liberty ID-WSF utility namespace (see Appendix E).
xs:	Represents the W3C XML schema namespace (http://www.w3.org/2001/XMLSchema) defined in [Schema1-2].

144 3. Terminology

145 This section defines key terminology used in this specification. Definitions for these, as well as other Liberty-specific
 146 terms, may also be found in [LibertyGlossary]. Note that the definition of some terms below differ slightly from the
 147 definition given in [LibertyGlossary]. For example see the definitions for *client* and *server*. This is because in such
 148 cases, the definition given in [LibertyGlossary] is a more general one, and the definition given here is a narrower one,
 149 specific to the context of this specification. See also [RFC2828] for overall definitions of security-related terms, in
 150 general. Other specific references are also cited below.

151 authentication 152	<i>Authentication</i> is the process of confirming a <i>system entity</i> 's asserted <i>identity</i> with a specified, or understood, level of confidence [TrustInCyberspace].
153 authentication assertion 154 155	A <i>SAML assertion</i> typically consisting of a single <AuthenticationStatement>. The assertion issuer is stating that the subject of the assertion authenticated with it at some point in time. Assertions are typically time-limited [SAMLCore2].
156 authentication exchange	See <i>authentication protocol exchange</i> .
157 authentication mechanism 158 159	An <i>authentication mechanism</i> is a particular, identifiable, process or technique that results in a confirmation of a <i>system entity</i> 's asserted identity with a specified, or understood, level of confidence.
160 authentication protocol exchange 161 162	<i>Authentication protocol exchange</i> is the term used in [RFC4422] to refer to the sequence of messages exchanged between the <i>client</i> and <i>server</i> as specified and governed by a particular <i>SASL mechanism</i> being employed to effect an act of <i>authentication</i> .
163 authentication server 164	The precise, specific <i>role</i> played by a <i>server</i> in the protocol message exchanges defined in this specification.
165 Authentication Service (AS)	Short form of "ID-WSF Authentication Service ." The AS is a discoverable ID-WSF service.
166 Authentication Service Consumer 167	A <i>Web Service Consumer</i> (WSC) implementing the <i>client</i> -side of the ID-WSF Authentication Protocol (which is defined in this specification).
168 Authentication Service Provider (AS Provider) 169	A <i>Web Service Provider</i> (WSP) implementing the <i>server</i> -side of the ID-WSF Authentication Service defined in this specification (Section 5: Authentication Service).
170 client 171 172	A <i>role</i> assumed by a <i>system entity</i> who either explicitly or implicitly initiates an authentication exchange [RFC2828]. <i>Client</i> is implicitly defined in [RFC4422]. Also known as a <i>SASL client</i> .
173 discoverable 174 175	A <i>discoverable</i> "in principle" service is one having a <i>service type URI</i> assigned (this is typically in done in the specification defining the service). A discoverable "in practice" service is one that is registered in some discovery service instance.
176 177	ID-WSF <i>services</i> are by definition discoverable "in principle" because such services are assigned a <i>service type URI</i> facilitating their registration in <i>Discovery Service</i> instances.
178 final SASL response 179	The final <SASLResponse> message sent from the <i>server</i> to the <i>client</i> in an <i>authentication exchange</i> .
180 ID-WSF EPR 181	An <i>ID-WSF Endpoint Reference</i> is a reference to a <i>service instance</i> . It contains the address, security context, and other metadata necessary for contacting the identified service instance.

182		The underlying structure of an ID-WSF EPR is based on the <i>wsa:EndpointReference</i> of
183		[WSAv1.0-SOAP] [WSAv1.0].
184	initial response	A [RFC4422] term referring to <i>authentication exchange data</i> sent by the <i>client</i> in the <i>initial</i>
185		<i>SASL request</i> . It is used by a subset of SASL mechanisms. See Section 5.1 of [RFC4422].
186	initial SASL request	The initial <SASLRequest> message sent from the <i>client</i> to the <i>server</i> in an <i>authentication</i>
187		<i>exchange</i> .
188	(LUAD-)WSC	A <i>Web Service Consumer</i> (WSC) that may or may not also be a <i>Liberty-enabled User Agent</i>
189		<i>or Device</i> .
190	mechanism	A process or technique for achieving a result [Merriam-Webster].
191	message thread	A <i>message thread</i> is a synchronous exchange of messages in a request-response <i>MEP</i> between
192		two <i>SOAP nodes</i> . All the messages of a given message thread are "linked" via each message's
193		<wsa:RelatesTo> header block value being set, by the sender, from the previous success-
194		fully received message's <wsa:MessageID> header block value.
195	requester	A <i>system entity</i> which sends a <i>service request</i> to a <i>provider</i> .
196	role	A function or part performed, especially in a particular operation or process [Merriam-Web-
197		ster].
198	SASL mechanism	A <i>SASL mechanism</i> is an <i>authentication mechanism</i> that has been profiled for use in the
199		context of the <i>SASL framework</i> [RFC4422]. See [RFC2444] for a particular example of
200		profiling an existing authentication mechanism—one-time passwords [RFC2289]—for use
201		in the SASL context. SASL mechanisms are "named"; Mechanism names are listed in the
202		column labeled as "MECHANISMS" in [SASLReg] (a copy of this registry document is
203		reproduced in Appendix A for informational convenience; implementors should always fetch
204		the most recent revision directly from [IANA]).
205	server	A <i>role</i> donned by a <i>system entity</i> which is intended to engage in defined exchanges with
206		<i>clients</i> . This term is implicitly defined in [RFC4422] and in this specification is always syn-
207		onymous with <i>authentication server</i> .
208	service instance	The physical instantiation of a service. A service instance is a web service at a distinct end-
209		point.
210	Service Provider (SP)	(1)A <i>role</i> donned by <i>system entities</i> . In the Liberty architecture, <i>Service Providers</i> interact
211		with other system entities primarily via vanilla HTTP.
212		(2) From a Principal's perspective, a Service Provider is typically a website providing services
213		and/or goods.
214	SOAP header block	A [SOAPv1.2] term meaning: An [element] used to delimit data that logically constitutes a
215		single computational unit within the SOAP header. In [SOAPv1.1] these are known as simply
216		<i>SOAP headers</i> , or simply <i>headers</i> . This specification borrows the SOAPv1.2 terminology.
217	SOAP node	A [SOAPv1.2] term describing <i>system entities</i> who are parties to SOAP-based message ex-
218		changes that are, for purposes of this specification, also the ultimate destination of the
219		exchanged messages, <i>i.e.</i> , <i>SOAP endpoints</i> . In [SOAPv1.1], SOAP nodes are referred to as
220		<i>SOAP endpoints</i> , or simply <i>endpoints</i> . This specification borrows the SOAPv1.2 terminology.

221	system entity	An active element of a computer/network system. For example, an automated process or set
222		of processes, a subsystem, a person or group of persons that incorporates a distinct set of
223		functionality [SAMLGloss2].
224	user identifier	AKA <i>user name</i> or <i>Principal</i> .
225	web service	Generically, a <i>service</i> defined in terms of an <i>XML</i> -based protocol, often transported over
226		<i>SOAP</i> , and/or a service whose instances, and possibly data objects managed therein, are con-
227		cisely addressable via <i>URIs</i> .
228		As specifically used in Liberty specifications, usually in terms of <i>WSCs</i> and <i>WSPs</i> , it means
229		a web service that's defined in terms of the <i>ID</i> -* " <i>stack</i> ," and thus utilizes [LibertySOAP-
230		Binding], [LibertySecMech], and is "discoverable" [LibertyDisco].
231	Web Service Consum- er	A <i>role</i> donned by a <i>system entity</i> when it makes a request to a <i>web service</i> .
232	Web Service Provider	A <i>role</i> donned by a <i>system entity</i> when it provides a <i>web service</i> .

233 4. Authentication Protocol

234 This section defines the ID-WSF Authentication Protocol. This protocol facilitates authentication between two ID-*
235 entities, and is a profile of SASL [RFC4422].

236 4.1. Conceptual Model

237 The conceptual model for the ID-WSF Authentication Protocol is as follows: an ID-WSF *system entity*, acting in a
238 *Web Services Consumer (WSC) role*, makes an authentication request to another ID-WSF system entity, acting in a
239 *Web Service Provider (WSP) role*, and if the WSP is willing and able, an authentication exchange will ensue.

240 The authentication exchange is comprised of SOAP-bound ID-* messages [LibertySOAPBinding], and can involve an
241 arbitrary number of round trips, dictated by the particular SASL mechanism employed [RFC4422]. The WSC may
242 have out-of-band knowledge of the server's supported SASL mechanisms, or it may send the server its own list of
243 supported SASL mechanisms and allow the server to choose one from among them.

244 At the end of this exchange of messages, the WSC will either be authenticated or not, the nature of the authentication
245 depending upon the SASL mechanism that was employed. Also depending on the SASL mechanism employed, the
246 WSP may be authenticated as well.

247 Other particulars, such as how the WSC knows which WSP to contact for authentication, are addressed below in
248 [Section 6: Single Sign-On Service](#).

249 Note

250 This document does not specify the use of SASL security layers.

251 4.2. Schema Declarations

252 The XML schema [Schemal-2] normatively defined in this section is constituted in the XML Schema file: liberty-
253 idwsf-authn-svc-v2.0.xsd, entitled " [Liberty ID-WSF Authentication Service XSD v2.0](#) " (see [Appendix C](#)).

254 Additionally, [Liberty ID-WSF Authentication Service XSD v2.0](#) imports items from liberty-idwsf-utility-
255 v2.0.xsd (see [Appendix E: Liberty ID-WSF Utility XSD v2.0](#)), and also from saml-schema-
256 protocol-2.0.xsd (see [SAMLCore2]).

257 4.3. SOAP Header Blocks and SOAP Binding

258 This specification does not define any SOAP header blocks. [Section 4.3.1](#), below, constitutes the SOAP binding state-
259 ment for this specification.

260 4.3.1. SOAP Binding

261 The messages defined below in [Section 4.6](#), e.g., <SASLRequest>, are *ordinary ID-* messages* as defined in [Liber-
262 tySOAPBinding]. They are intended to be bound to the [SOAPv1.1] protocol by mapping them directly into the <s:
263 Body> element of the <s:Envelope> element comprising a SOAP message. [LibertySOAPBinding] normatively
264 specifies this binding.

265 Note

266 Implementations of this specification MUST use the "Messaging-specific Header **Blocks**," as specified in
267 [LibertySOAPBinding], to establish a *message thread* and thus correlate their authentication exchanges. See
268 [Section 5.5: Authentication Service Interaction Example](#) for an example.

269 4.4. SASL Profile Particulars

270 The ID-WSF Authentication Protocol is based on SASL [RFC4422], and thus "profiles" SASL. Section 4 of
271 [RFC4422] specifies SASL's "profiling requirements." This section of this specification addresses some particulars of
272 profiling SASL that are not otherwise addressed in the sections defining the protocol messages (Section 4.6: Protocol
273 Messages), and their sequencing (Section 4.7: Sequencing of the Authentication Exchange).

274 4.4.1. SASL "Service Name"

275 The SASL "Service Name" specified herein is: **idwsf**

276 4.4.2. Composition of SASL Mechanism Names

277 The protocol messages defined below at times convey a SASL mechanism name, or a list of SASL mechanism names,
278 as values of message element attributes.

279 These mechanism names are typically taken from the column labeled as "MECHANISMS" in [SASLReg], but MAY
280 be site-specific.

281 These names, and lists of these names, MUST follow these rules:

- 282 • The character composition of a SASL mechanism name MUST be as defined in [IANA]'s SASL Mechanism
283 Registry [SASLReg].
- 284 • A list of SASL mechanism names MUST be composed of names as defined above, separated by ASCII space chars
285 (hex "20").

286 4.5. Authentication Exchange Security

287 This authentication protocol features the flexibility of having implementations being able to select at runtime the actual
288 authentication mechanism (aka SASL mechanism) to employ. This however may introduce various vulnerabilities
289 depending on the actual mechanism employed. Some mechanisms may be vulnerable to passive and/or active attacks.
290 Also, since the server selects the SASL mechanism from a list supplied by the client, a compromised server, or a man-
291 in-the-middle, can cause the weakest mechanism offered by the client to be employed.

292 Thus it is RECOMMENDED that the authentication protocol exchange defined herein (Section 4.7: Sequencing of
293 the Authentication Exchange) be employed over a TLS/SSL channel [RFC4346] as amended by [RFC4366]. This will
294 ensure the integrity and confidentiality of the authentication protocol messages. Additionally, clients SHOULD au-
295 thenticate the server via TLS/SSL validation procedures. This will help guard against man-in-the-middle attacks.

296 4.6. Protocol Messages

297 This section defines the protocol's messages, along with their message element attribute values, and their semantics.
298 The sequencing of protocol interactions, also known as the *authentication exchange*, is defined below in Section 4.7:
299 Sequencing of the Authentication Exchange .

300 4.6.1. The <SASLRequest> Message

301 Figure 1 shows the schema fragment from Liberty ID-WSF Authentication Service XSD v2.0 describing the
302 <SASLRequest> message. This message has the following attributes:

- 303 • **mechanism** [Required] — Used to convey a list of one-or-more client-supported SASL mechanism names to the
304 server, or to signal the server if the client wishes to abort the exchange. It is included on all <SASLRequest>
305 messages sent by the client.

- 306 • **authzID** [Optional] — The authzID, also known as *user identifier* or *username* or *Principal*, that the client wishes
307 to establish as the "authorization identity" per [RFC4422].

- 308 • **advisoryAuthnID** [Optional] — The advisoryAuthnID may be used to advise the server what authentication
309 identity will be asserted by the client via the selected SASL mechanism; *i.e.*, it is a "hint." The
310 advisoryAuthnID provides a means for server implementations to optimize their behavior on a per authentication
311 identity basis. E.g. if a client requests to execute a certain SASL mechanism on behalf of some given authentication
312 identity (represented by advisoryAuthnID) and authorization identity (represented by authzID) pair, the server
313 can decide whether to proceed without having to execute the SASL mechanism (execution of which might involve
314 more than a single round-trip). Server implementations that make use of the optional advisoryAuthnID attribute
315 SHOULD be capable of processing initial <SASLRequest> messages that do not include the
316 advisoryAuthnID attribute.

- 317 • **Any Attributes** [Optional] — Zero or more extension attributes qualified by an XML namespace other than the
318 Authentication Service namespace.

```

319
320 <xs:element name="SASLRequest">
321   <xs:complexType>
322     <xs:sequence>
323
324       <xs:element name="Data" minOccurs="0">
325         <xs:complexType>
326           <xs:simpleContent>
327             <xs:extension base="xs:base64Binary"/>
328           </xs:simpleContent>
329         </xs:complexType>
330       </xs:element>
331
332       <xs:element ref="samlp2:RequestedAuthnContext" minOccurs="0"/>
333
334       <xs:element name="Extensions" minOccurs="0">
335         <xs:complexType>
336           <xs:sequence>
337             <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
338           </xs:sequence>
339         </xs:complexType>
340       </xs:element>
341
342     </xs:sequence>
343
344     <xs:attribute name="mechanism"
345       type="xs:string"
346       use="required"/>
347
348     <xs:attribute name="authzID"
349       type="xs:string"
350       use="optional"/>
351
352     <xs:attribute name="advisoryAuthnID"
353       type="xs:string"
354       use="optional"/>
355
356     <xs:anyAttribute namespace="##other" processContents="lax"/>
357
358   </xs:complexType>
359 </xs:element>

```

Figure 1. <SASLRequest> Message Element — Schema Fragment

361 The <SASLRequest> message has the following sub-elements:

- 362 • **<Data>** — This element is used by the client to send SASL mechanism data to the server. In [RFC4422] parlance,
363 this data is termed a "client **response**." Its content model is base64-encoded data.

- 364 • **<samlp2:RequestedAuthnContext>** — This element is used by the client to convey to the server a desired
365 authentication context. It is used only on the initial SASL request (see Section 4.7: [Sequencing of the Authen-](#)
366 [tication Exchange](#)). If present, the server uses the information in the <samlp2:RequestedAuthnContext> in
367 combination with mechanism attribute when choosing the SASL mechanism to execute. The background use case
368 for <samlp2:RequestedAuthnContext> is presented in Section 5.1: [Authentication Service: Conceptual](#)
369 [Model](#) . See also: [LibertyAuthnContext] and [LibertyProtSchema].

- 370 • **<Extensions>** — This contains optional request extensions that are agreed upon between the client and server.
371 Extension elements **MUST** be namespace-qualified by a non-AS namespace.

```

372
373 <?xml version="1.0" encoding="UTF-8"?>
374
375 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
376           xmlns:sa="urn:liberty:sa:2006-08"
377           xmlns:sb="urn:liberty:wsf:soap-bind:1.0"
378           xmlns:pp="urn:liberty:id-sis-pp:2003-08">
379
380   <S:Header>
381
382     <!-- various header blocks, as defined in the
383          SOAP Binding spec, go here -->
384
385   </S:Header>
386
387   <S:Body>
388
389     <sa:SASLRequest sa:mechanism="foo">
390       <sa:Data>
391         qwyGHhSWpjQu5yq.....vUUlVONmOZtfzGfz
392       </sa:Data>
393     </sa:SASLRequest>
394
395   </S:Body>
396
397 </S:Envelope>
398

```

Example 1. A SASLRequest Bound into a SOAP Message

4.6.1.1. <SASLRequest> Usage

The <SASLRequest> message is used to initially convey to the server a:

- 402 • list of one or more client-supported SASL mechanism names,
- 403 ..in combination with optional:
- 404 • authzID attribute, and/or,
- 405 • advisoryAuthnID attribute, and/or,
- 406 • <samlp2:RequestedAuthnContext> element.

407 In the case where a single SASL mechanism name is conveyed, the <SASLRequest> message can contain a so-called
408 *initial response* (see Section 5 of [RFC4422]) in the <Data> element.

409 If the server's subsequent <SASLResponse> message signals that the authentication exchange should continue—and
410 thus contains a server "challenge"—the client will send another <SASLRequest> message, with the <Data> element

411 containing the client's "response" to the challenge. This sequence of server challenges and client responses continues
412 until the server signals a successful completion or aborts the exchange.

413 The `mechanism` attribute is used in these intermediate `<SASLRequest>` messages to signal the client's intentions to
414 the server. This is summarized in the next section.

415 [Section 4.7: Sequencing of the Authentication Exchange](#), in combination with the next section, normatively defines
416 the precise `<SASLRequest>` message format as a function of the sequencing of the authentication exchange.

417 **4.6.1.2. Values for `mechanism` attribute of `<SASLRequest>`**

418 The list below defines the allowable values for the `mechanism` attribute of the `<SASLRequest>` message element,
419 and the resulting message semantics.

420 **Note**

421 In items #2 and #1, the `mechanism` attribute contains one or more SASL mechanism names, respectively.
422 The rules noted in [Section 4.4.2: Composition of SASL Mechanism Names](#) MUST be adhered to in such
423 cases.

424 1. **Multiple SASL mechanism names** — See [Example 2](#). In this case, the `<SASLRequest>` message MUST NOT
425 contain any "initial response" data, and MUST be the initial SASL request. See [Section 4.6.2.1.2](#) for details on
426 the returned `<SASLResponse>` message in this case.

```
427  
428     <SASLRequest mechanism="GSSAPI OTP PLAIN" />  
429
```

430 **Example 2. `<SASLRequest>` Specifying Multiple Client-supported Mechanism Names**

431 2. **A single SASL mechanism name** — In this case, the `<SASLRequest>` message MAY contain *initial response*
432 data. See [Example 3](#).

```
433  
434     <SASLRequest mechanism="GSSAPI">  
435       <Data>  
436         Q29ub3IgcQ2FoaWxsIGNhc3VhbGx5IGlhbmdsZXMGcGFzc3dvcnRzCg==  
437       </Data>  
438     </SASLRequest>  
439
```

440 **Example 3. `<SASLRequest>` Specifying a Single Mechanism Name**

441 3. **A NULL string ("")** — This indicates to the authentication server that the client wishes to abort the authentication
442 exchange. See [Example 4](#).

```
443  
444     <SASLRequest mechanism="" />  
445
```

446 **Example 4. `<SASLRequest>` Message Aborting the SASL Authentication Exchange**

447 **4.6.2. The `<SASLResponse>` Message**

448 [Figure 2](#) shows the schema fragment from [Liberty ID-WSF Authentication Service XSD v2.0](#) describing the
449 `<SASLResponse>` message. This message has the following attributes:

450 • **`serverMechanism`** [Optional] — The server's choice of SASL mechanism from among the list sent by the client.

- 451 • **Any Attributes** [Optional] — Zero or more extension attributes qualified by an XML namespace other than the
 452 Authentication Service namespace.

```

453
454 <xs:element name="SASLResponse">
455   <xs:complexType>
456     <xs:sequence>
457
458       <xs:element ref="Status"/>
459
460       <xs:element ref="PasswordTransforms" minOccurs="0"/>
461
462       <xs:element name="Data" minOccurs="0">
463         <xs:complexType>
464           <xs:simpleContent>
465             <xs:extension base="xs:base64Binary"/>
466           </xs:simpleContent>
467         </xs:complexType>
468       </xs:element>
469
470       <!-- ID-WSF EPRs -->
471       <xs:element ref="wsa:EndpointReference"
472         minOccurs="0"
473         maxOccurs="unbounded"/>
474
475     </xs:sequence>
476
477     <xs:attribute name="serverMechanism"
478       type="xs:string"
479       use="optional"/>
480
481     <xs:anyAttribute namespace="##other" processContents="lax"/>
482
483   </xs:complexType>
484 </xs:element>
485

```

486 **Figure 2. <SASLResponse> Message Element - Schema Fragment**

487 The <SASLResponse> message has the following sub-elements:

- 488 • **<Status>** — This element is from [Liberty ID-WSF Utility XSD v2.0](#) and is used to convey status from the server
 489 to the client. See below.
- 490 • **<PasswordTransforms>** — This element is used to convey to the client any required password transformations.
 491 See [Section 8: Password Transformations: The PasswordTransforms Element](#).
- 492 • **<Data>** — This element is used to return SASL mechanism data to the client. Its content model is base64-encoded
 493 data.
- 494 • **<wsa:EndpointReference>** — This element is to convey to the client an ID-WSF EPR for the server, in its
 495 role as a WSP, upon a successful authentication exchange completion. Multiple instances of it may be used to also
 496 convey ID-WSF EPRs for additional instances of other services. Note that any credentials returned as a result of a
 497 successful authentication exchange are conveyed within any returned ID-WSF EPRs [[LibertyDisco](#)]. See [Section 5: Authentication Service](#).
 498

499 **4.6.2.1. <SASLResponse> Usage**

500 This message is sent by the server in response to a client <SASLRequest> message. It is used to convey "server
 501 **challenges**," in [[RFC4422](#)] parlance, to the client during an authentication exchange. So-called "client responses" are
 502 correspondingly conveyed to the server via the <SASLRequest> message, defined above. A given authentication

503 exchange may occur in one "round-trip," or it may involve several round-trips. This depends on the SASL mechanism
 504 being executed.

505 The first <SASLResponse> sent by the server within an authentication exchange (as determined by the particular
 506 authentication mechanism being used) is explicitly distinguished from subsequent <SASLResponse> messages in
 507 terms of child elements and attributes. The final <SASLResponse> sent by the server in an authentication exchange
 508 is similarly distinguished, although with its own particular characteristics. These details are specified below in [Section 4.7: Sequencing of the Authentication Exchange](#) .
 509

510 It is possible for different authentication mechanisms to be sequenced, the client authenticating to the server with one
 511 after another. For example, after a principal is authenticated with name and password (e.g., with PLAIN or CRAM-
 512 MD5), the service may (because of service or user policy) require additional authentication with SECUR-ID.
 513 Consequently, client implementations should be prepared for a message from the service with a "Continue" status code
 514 but a different "serviceMechanism" than that established in the previous authentication exchange. The message from
 515 the service that indicates such subsequent SASL mechanism may contain a <Data> element intended for processing
 516 by an implementation of the new mechanism. The client should process this message as specified in step 5 of [Section 4.7: Sequencing of the Authentication Exchange](#) .
 517

518 The <Status> element (see [Figure 3](#)) is used to convey the authentication server's assessment of the status of the
 519 authentication exchange to the client, via the code attribute (the <Status> element is declared in the [Liberty ID-WSF](#)
 520 [Utility XSD v2.0](#)).

```

521     <xs:complexType name="StatusType">
522       <xs:annotation>
523         <xs:documentation>
524           A type that may be used for status codes.
525         </xs:documentation>
526       </xs:annotation>
527       <xs:sequence>
528         <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
529       </xs:sequence>
530       <xs:attribute name="code" type="xs:string" use="required"/>
531       <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
532       <xs:attribute name="comment" type="xs:string" use="optional"/>
533     </xs:complexType>
534
535     <xs:element name="Status" type="StatusType">
536       <xs:annotation>
537         <xs:documentation>
538           A standard Status type
539         </xs:documentation>
540       </xs:annotation>
541     </xs:element>
542
```

543 **Figure 3. <Status> Element and Type - Schema Fragment (from liberty-idwsf-utility-v2.0.xsd)**

544 In the two sections below, first the values of the code attribute of the <Status> element are discussed, followed by
 545 discussion of the various forms of <SASLResponse> messages and their semantics.

546 **4.6.2.1.1. Values for the code attribute of <Status>**

547 If the value of code is:

- 548 • "Continue" — the server expects the client to craft and send a new <SASLRequest> message containing data
 549 appropriate for whichever step the execution of the SASL mechanism is at.
- 550 • "OK" — the server considers the authentication exchange to have completed successfully.

551 The <SASLResponse> message will typically contain ID-WSF EPR(s) (i.e., <wsa:EndpointReference> ele-
 552 ment(s)) containing credentials, as described below in [Section 5.3: Rules for Authentication Service Providers](#) ,

553 enabling the client to interact further with this provider, for example to invoke another ID-WSF service such as the
554 Discovery Service.

555 Additionally, the <SASLResponse> message can convey ID-WSF EPRs for other providers.

556 See [Section 4.7: Sequencing of the Authentication Exchange](#) for the normative specification of the composition
557 of the <SASLResponse> message in this case. See also [Section 5.3: Rules for Authentication Service Providers](#).
558

559 • "Abort" — the server is aborting the authentication exchange. It will not send any more messages on this message
560 thread.

561 4.6.2.1.2. Returning the Server's Selected SASL Mechanism

562 The server will choose one SASL mechanism from among the intersection of the list sent by the client and the server's
563 set of supported and willing-to-execute SASL mechanisms. It will return the name of this selected SASL mechanism
564 as the value for the `serverMechanism` attribute on the initial <SASLResponse> message. See [Example 5](#).

```
565  
566 <SASLResponse serverMechanism="DIGEST-MD5">  
567   <Status code="Continue" />  
568   <Data>  
569     Q29ub3IqQ2FoaWxsIGNhcn3VhbGx5IG1hbmdsZXMgcGFzc3dvcnRzCg==  
570   </Data>  
571 </SASLResponse>  
572
```

573 **Example 5. <SASLResponse> Indicating Server's Chosen SASL Mechanism**

574 If there is no intersection between the client-supplied list of SASL mechanisms and the set of supported, and willing-
575 to-execute, server-side SASL mechanisms, then the server will return a <SASLResponse> message with a `code`
576 attribute whose value is "Abort." See [Example 6](#), and also item #3 in [Section 4.7: Sequencing of the Authentication Exchange](#).
577

```
578  
579 <SASLResponse>  
580   <Status code="Abort" />  
581 </SASLResponse>  
582
```

583 **Example 6. <SASLResponse> Indicating a Server-side Abort**

584 4.7. Sequencing of the Authentication Exchange

585 The authentication exchange is sequenced as follows:

586 1. The authentication exchange MUST begin by the client sending the server a <SASLRequest> message. This
587 message:

588 • MUST contain a `mechanism` attribute whose value is a string containing one or more SASL mechanisms the
589 client supports and is prepared to negotiate (see [Section 4.6.1.2: Values for mechanism attribute of <SASLRequest>](#)).
590

591 • MAY contain a <Data> element containing an initial response, specific to the cited SASL mechanism, if the
592 `mechanism` attribute contains only a single SASL mechanism. See section 5 of [\[RFC4422\]](#).

593 • MAY contain a `<samlp2:RequestedAuthnContext>` element.

- 594 • SHOULD contain an `authzID` attribute whose value is an identifier string for the Principal being authenti-
595 cated.
- 596 • MAY contain an `advisoryAuthnID` attribute whose value is an identifier asserted by the client to represent
597 the authentication identity being established by this authentication event.
- 598 2. If the server is prepared to execute, with this client, at least one of the SASL mechanism(s) cited by the client in
599 the previous step, then processing continues with step 4.
- 600 3. Otherwise, the server does not support, or is not prepared to negotiate, any of the SASL mechanisms cited by the
601 client. The server MUST respond to the client with a `<SASLResponse>` message containing:
- 602 • A `<Status>` element with a `code` attribute with a value of "**Abort.**"
- 603 • No `<PasswordTransforms>` element.
- 604 • No `<Data>` element.
- 605 • No `<wsa:EndpointReference>` element.
- 606 • No `serverMechanism` attribute.
- 607 After this message is sent to the client, processing continues with step 7.
- 608 4. The server sends to the client a `<SASLResponse>` message.
- 609 If this message is the first `<SASLResponse>` sent to the client in this authentication exchange (as determined by
610 a particular authentication mechanism, see substep "**A. Continue,**" below), this message:
- 611 • MUST contain a `serverMechanism` attribute whose value is a single SASL mechanism name, chosen by the
612 server from the list sent by the client.
- 613 • MAY contain a `<Data>` element containing a SASL mechanism-specific challenge.
- 614 • MAY contain a `<PasswordTransforms>` element. See [Section 8: Password Transformations: The](#)
615 [PasswordTransforms Element](#) for details on the client's subsequent obligations in this case.
- 616 • MUST contain a `<Status>` element with a `code` attribute whose value is given by either item **A**, or **B**, or
617 **C**:
- 618 A. "**Continue**" — either the execution of the SASL mechanism is not complete or the authentication ex-
619 change was successful but the server expects the client to authenticate again using a different authenti-
620 cation mechanism; the server expects the client to process this message and respond.
- 621 If the server is indicating that the client should continue by authenticating with a different mechanism,
622 the server MUST specify the desired mechanism as the value for "**serverMechanism.**" The authentication
623 mechanism specified MUST be taken from the list previously sent by the client in the prior authentication
624 exchange. The server MAY include a `<Data>` element (and `<PasswordTransforms>`) with content
625 appropriate for the new authentication mechanism.
- 626 If the reason for the server indicating that the client should continue is that the client presented invalid
627 credentials, the server SHOULD include a second level status `<Status`
628 `code="InvalidCredentials">`. The server MAY also return a `<Data>` element (e.g., with a new
629 challenge according to the mechanism already established) and the client can respond according to the
630 mechanism. Processing continues with step 5.

- 631 B. "OK" — the server declares the authentication exchange has completed successfully.
- 632 In this case, this *final SASL response* message can contain, in addition to the items listed above, <wsa:EndpointReference> element(s), containing requisite credentials. This is specified in [Section 5.3: Rules for Authentication Service Providers](#).
- 633
- 634
- 635 Processing continues with step 6.
- 636 C. "Abort" — the server declares the authentication exchange has completed unsuccessfully. For example,
- 637 the user may have supplied incorrect information, such as an incorrect password. See step 7, below, for
- 638 additional information.
- 639 In this case, this <SASLResponse> message MUST NOT contain any <wsa:EndpointReference>
- 640 element(s).
- 641 Processing continues with step 7.
- 642 Otherwise, this message:
- 643 • MUST NOT contain a `serverMechanism` attribute.
 - 644 • MAY contain a <Data> element containing a SASL mechanism-specific challenge.
 - 645 • MUST NOT contain a <PasswordTransforms> element.
 - 646 • MUST contain a <Status> element with a `code` attribute whose value is given by either item A, or B, or
 - 647 C:
- 648 A. "Continue" — the execution of the SASL mechanism is not complete; the server expects the client to
- 649 process this message and respond. Processing continues with step 5.
- 650 B. "OK" — the server declares the authentication exchange has completed successfully.
- 651 In this case, this "final response" <SASLResponse> message can contain, in addition to the items listed
- 652 above, <wsa:EndpointReference> element(s) with requisite credentials. This is specified in [Section 5.3: Rules for Authentication Service Providers](#).
- 653
- 654 Processing continues with step 6.
- 655 C. "Abort" — the server declares the authentication process has completed unsuccessfully. For example,
- 656 the user may have supplied incorrect information, such as an incorrect password.
- 657 If the reason for the server aborting is that the client presented invalid credentials, the server SHOULD
- 658 include a second level status <Status code="InvalidCredentials">.
- 659 In this case, this <SASLResponse> message MUST NOT contain any <wsa:EndpointReference>
- 660 element(s).
- 661 Processing continues with step 7.
- 662 5. The client sends the server a <SASLRequest> message. This message:
- 663 • SHOULD contain a `mechanism` attribute set to the same value as sent by the server, as the value of the
 - 664 `serverMechanism` attribute, in its first <SASLResponse> message (see [Section 4.6.2.1.2: Returning the Server's Selected SASL Mechanism](#)).
 - 665

666

Note

667

The client MAY, however, choose to abort the authentication exchange by setting the `mechanism` attribute to either a "null" string, or to a mechanism name different than the one returned by the server in its first `<SASLResponse>` message.

668

669

670

If the client chooses to abort, processing continues with step 8.

671

- SHOULD contain a `<Data>` element containing data specific to the cited SASL mechanism.

672

- MUST NOT contain a `<samlp2:RequestedAuthnContext>` element.

673

Processing continues with steps 4 and 5 until the server signals success, failure, or aborts — or the client aborts the exchange using the technique noted in the first bullet item, above, of this step.

674

675

6. The authentication exchange has completed successfully. The client is now authenticated in the server's view, and the server may be authenticated in the client's view, depending upon the SASL mechanism employed. [Section 5.1: Authentication Service: Conceptual Model](#) discusses what the next interaction steps between the client and server are in the ID-WSF authentication service case.

676

677

678

679

7. The authentication exchange has completed unsuccessfully due to an exception on the server side. The client SHOULD cease sending messages on this message thread.

680

681

The reasons for an authentication exchange failing are manifold. Often it is simply a case of the user having supplied incorrect information, such as a password or ~~pass phrase~~. Or, there may have been a problem on the server's part, such as an authentication database being unavailable or unreachable.

682

683

684

8. The client aborted the authentication exchange.

685 5. Authentication Service

686 The ID-WSF Authentication Service provides web service-based authentication facilities to Web Service Consumers
687 (WSCs). This service is built around the SASL-based ID-WSF Authentication Protocol as specified above in [Sec-
688 tion 4](#).

689 This section first outlines the Authentication Service's conceptual model and then defines the service itself.

690 5.1. Conceptual Model

691 ID-WSF-based Web Service Providers (WSPs) may require requesters, AKA Web Service Consumers (WSCs), to
692 present security tokens in order to successfully interact (security token specifics, are specified in [\[LibertySecMech\]](#)).

693 A Discovery Service [\[LibertyDisco\]](#), which itself is just a WSP, is able to create security tokens authorizing WSCs to
694 interact with other WSPs, on whose behalf a Discovery Service has been configured to speak. Also, Discovery Service
695 instances might themselves be configured to require WSCs to present security tokens when making requests of them.

696 The ID-WSF Authentication Service addresses the above conundrum by providing the means for WSCs to prove their
697 identities—to authenticate—and obtain security tokens enabling further interactions with other services, at the same
698 provider, on whose behalf the Authentication Service instance is authorized to speak. These offered services may be,
699 for example, a Discovery Service or Single Sign-On Service. WSCs may then use these latter services to discover and
700 become capable of interacting with yet other services.

701 Note that although an Authentication Service itself does not require requesters to present security tokens in order to
702 interact with it, an Authentication Service may, in some situations, be configured to understand presented security
703 tokens and use them when applying policy.

704 5.1.1. Stipulating a Particular Authentication Context

705 In some situations, a WSC may need to stipulate some of the properties for an authentication exchange. A scenario
706 illustrating a use case of this is:

707 Suppose a Principal is wielding a Liberty-enabled user agent or device (LUAD) that is acting as a
708 WSC (i.e., a LUAD-WSC). The Principal authenticates with her bank, say, and authenticates via the
709 ID-WSF authentication service using some authentication mechanism, such as PLAIN [\[SASLReg\]](#).
710 At some point, the Principal wants to transfer a large sum of money to the Fund for Poor Specification
711 Editors (using some (fictitious) ID-SIS-based web service), and the bank's system indicates to the
712 LUAD-WSC that the Principal's present authentication is "inappropriate." The bank's system also
713 includes a `<RequestedAuthnContext>`.

714 Now, the LUAD-WSC "knows" that it needs to help the Principal reauthenticate—as her present
715 credentials aren't being honored for the financial transaction she wishes to carry out. So the LUAD-
716 WSC prompts the Principal for permission to reauthenticate her, and (assuming the answer was "yes")
717 initiates the ID-WSF Authentication Protocol with the appropriate authentication service provider,
718 and includes the supplied-by-the-bank `<RequestedAuthnContext>`. The authentication service
719 provider factors the requested authentication context into its selection of SASL mechanism for the
720 ensuing authentication exchange. And upon successful authentication, the Principal is able to suc-
721 cessfully make the funds transfer.

722 When initiating an authentication exchange, a WSC can stipulate some properties for the ensuing authentication event,
723 and thus the subsequently issued (if successful) credentials. It does this by including a
724 `<RequestedAuthnContext>` in the initial `<SASLRequest>`.

725 **5.2. URI Declarations**

726 The URI declarations for the ID-WSF Authentication Service are given below in [Table 2](#).

727 **Table 2. Authentication Service URIs**

Use	URI
Service Type	<i>urn:liberty:sa:2006-08</i>
SASLRequest wsa: Action	<i>urn:liberty:sa:2006-08: SASLRequest</i>
SASLResponse wsa: Action	<i>urn:liberty:sa:2006-08: SASLResponse</i>

728 **5.3. Rules for Authentication Service Providers**

729 Providers offering ID-WSF Authentication Services MUST adhere to the following rules:

- 730 1. Authentication Service Providers (AS Providers) MUST implement the ID-WSF Authentication Protocol, as
 731 defined in [Section 4: Authentication Protocol](#) . The Authentication Service Provider MUST play the role of the
 732 *authentication server*.
- 733 2. Upon successful completion of an authentication exchange the **first** ID-WSF EPR, as materialized as an `<wsa: EndpointReference>`
 734 element instance and contained in the *final SASL response*, SHOULD refer to services
 735 at the Authentication Service **provider**—i.e., at the "same provider"—that said AS Provider can offer to the Au-
 736 thentication Service consumer.

737 For example, Identity Providers may often also include an ID-WSF EPR for the Discovery Service of the Principal
 738 just authenticated, as well as ID-WSF EPRs for other offered services, such as an SSO Service.

739 **Note**

740 If the Authentication Service is invoked via a message whose indicated "framework version" [[Liberty-
 741 SOAPBinding](#)] is "2.0," then if the AS is returning ID-WSF EPRs for other services as noted above, then
 742 the AS SHOULD return ID-WSF EPRs for ID-WSFv2.0 services, rather than other versions of ID-WSF
 743 services.

744 See [Section 4.7: Sequencing of the Authentication Exchange](#) , Step 4.

745 The Provider MAY also include additional ID-WSF EPRs referring to services offered by other **providers**—i.e.,
 746 providers other than the AS Provider.

- 747 3. Any included credentials SHOULD be useful for a reasonable time (note that credentials will be contained within
 748 the ID-WSF EPRs, as profiled in [[LibertyDisco](#)]). Even if the AS Consumer recently authenticated with the
 749 Authentication Service, **i.e.**, an earlier issued credential for consumption by the AS Provider is still valid, the AS
 750 Provider SHOULD issue credential(s) that have later expiration times than the earlier issued credential(s). The
 751 AS Provider MAY choose to re-authenticate, using any of the available SASL mechanisms, or issue new creden-
 752 tials without a engaging in an authentication exchange. This can be accomplished by responding to the AS
 753 Consumer's initial SASL request with a final SASL response containing an ID-WSF EPR, itself containing the
 754 requisite credentials.

755 **Note**

756 Credentials containing <saml2:AuthnStatement>(s) should have their <saml2:AuthnInstant>
757 (s) set to the time when the authentication event actually took place. See [SAMLCore2].

758 4. Additionally, if the first <SASLRequest> in an exchange contains a <samlp2:RequestedAuthnContext>
759 element, then upon successful authentication, the Authentication Service MUST either: return credentials (em-
760 bedded within returned ID-WSF EPR(s)) that satisfy the <samlp2:RequestedAuthnContext>, or, abort the
761 authentication exchange. See [SAMLCore2] for a detailed description of the processing rules governing evaluation
762 of the <samlp2:RequestedAuthnContext> element.

763 5. An Authentication Service instance SHOULD be deployed such that the security mechanism [LibertySec-
764 Mech]:

765 urn:liberty:security:2003-08:TLS:null

766 can be used by the WSC.

767 **Note**

768 In practice this means that the Authentication Service should be exposed on an endpoint for which the
769 URL should have https as the protocol field.

770 6. An Authentication Service implementation SHOULD support the following SASL mechanisms [SASLReg]:
771 PLAIN, CRAM-MD5.

772 **5.4. Rules for Authentication Service Consumers**

773 WSCs implementing the client-side of the ID-WSF Authentication Protocol, and thus also known as *Authentication*
774 *Service Consumers* (AS Consumers), MUST adhere to the following rules:

775 1. AS Consumers MUST implement the ID-WSF Authentication Protocol, as defined in Section 4: *Authentication*
776 *Protocol* in the role of the client.

777 **Note**

778 The AS Consumer may include various SOAP header blocks, e.g., a <wsse:Security> element |
779 [LibertySecMech] which can house a security token(s) obtained earlier from an Authentication Service
780 or Discovery Service [LibertyDisco]. In such a case, the Authentication Service SHOULD evaluate the
781 presented security token(s) in combination with applicable policy, as a part of the overall authentication
782 event. This provides a means, for example, of "security token renewal." |

783 2. In case the AS Consumer has not been provisioned with the <disco:SecurityMechID> for the Authentication
784 Service instance that it uses, the AS Consumer SHOULD assume that the required security mechanism is this
785 one:

786 urn:liberty:security:2003-08:TLS:null

787 **Note**

788 <disco:SecurityMechID> elements are contained within the <disco:SecurityContext> element
789 (s), themselves occurring within ID-WSF EPRs (profiled <wsa:EndpointReference>s) [LibertyDis-
790 co].

791 Only when the endpoint URL of the Authentication Service is prescribed to have http as the protocol MAY the
792 WSC presume a security mechanism of:

793 urn:liberty:security:2003-08:null:null

794 3. It is RECOMMENDED that the WSC support the password transformations specified in [Appendix B](#).

795 5.5. Authentication Service Interaction Example

796 [Example 7](#) through [Example 10](#) illustrate an example exchange between a LUAD-WSC and an ID-WSF Authentication
797 Service (AS). The AS includes information about the Discovery Service (DS) in its final response. Here the DS is
798 offered by the same provider.

```
799
800 <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
801   <s:Header>
802     ...
803   </s:Header>
804   <s:Body>
805     <SASLRequest mechanism="CRAM-MD5"
806       advisoryAuthnID="358408021451"
807       xmlns="urn:liberty:sa:2004-04" />
808   </s:Body>
809 </s:Envelope>
810
```

811 **Example 7. The WSC sends a <SASLRequest> on behalf of a Principal, asserting that the authentication identity is**
812 **"358408021451" and indicates it desire to use the "CRAM-MD5" SASL mechanism.**

```
813
814 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
815   <S:Header>
816     ...
817   </S:Header>
818   <S:Body>
819     <SASLResponse serverMechanism="CRAM-MD5"
820       xmlns="urn:liberty:sa:2004-04">
821     <Status code="continue"/>
822     <Data>
823       ...a CRAM-MD5 challenge here...
824     </Data>
825   </SASLResponse>
826 </s:Body>
827 </s:Envelope>
828
```

829 **Example 8. The AS replies, agreeing to use CRAM-MD5, and issues a CRAM-MD5 challenge.**

```
830
831 <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
832   <s:Header>
833     ...
834   </s:Header>
835   <s:Body>
836     <SASLRequest mechanism="CRAM-MD5"
837       xmlns="urn:liberty:sa:2004-04">
838     <Data>
839       ...some CRAM-MD5 response here...
840     </Data>
841   </SASLRequest>
842 </s:Body>
843 </s:Envelope>
844
```

845 **Example 9. The WSC responds with a CRAM-MD5 response.**

```

846
847 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
848   <S:Header>
849     ...
850   </S:Header>
851   <S:Body id="msgBody">
852
853     <sa:SASLResponse xmlns:sa="urn:liberty:sa:2004-04"
854       xmlns:disco="urn:liberty:disco:2003-08">
855       <Status code="sa:OK"/>
856
857       <wsa:EndpointReference>
858
859         <wsa:Address>
860           http://tg2.example.com:8080/tfs-soap/IdPDiscoveryService
861         </wsa:Address>
862
863         <wsa:Metadata>
864
865           <disco:ServiceType>urn:liberty:disco:2003-08</disco:ServiceType>
866
867           <disco:ProviderID>http://tg2.example.com:8080/tfs</disco:ProviderID>
868
869           <ds:SecurityContext>
870
871             <disco:SecurityMechID>
872               urn:liberty:security:2005-02:null:Bearer
873             </disco:SecurityMechID>
874
875             <sec:Token>
876               <saml2:Assertion
877                 ID="ilb42508103cab657f34e5ef189f28ea10dd86926"
878                 Version="2.0"
879                 IssueInstant="2004-02-03T22:12:33Z">
880                 <Issuer>
881                   http://tg2.trustgenix.com:8080/tfs
882                 </Issuer>
883                 ....
884                 ....
885               </saml2:Assertion>
886             </sec:Token>
887
888           </ds:SecurityContext>
889
890         </wsa:Metadata>
891
892       </wsa:EndpointReference>
893
894     </sa:SASLResponse>
895
896   </S:Body>
897 </S:Envelope>
898

```

899 **Example 10.** The AS replies with its "final" <SASLResponse> message, which includes credentials with which the WSC
900 may subsequently use to invoke a DS.

901 6. Single Sign-On Service

902 The ID-WSF Single Sign-On Service (SSO Service, or SSOS) provides requesters with an ID-WSF-based means to
903 obtain *SAML 2.0 authentication assertions* enabling them to interact with *SAML 2.0 Service Providers* (SPs) [[SAML-
904 Core2](#)] as well as other services that accept SAML 2.0 assertions as security tokens, such as web services (including
905 ID-WSF WSPs). The SSOS is based on a pair of profiles of the SAML 2.0 Authentication Request protocol [[SAML-
906 Core2](#)], one of which is a refinement of the SAML 2.0 Enhanced Client or Proxy SSO profile [[SAMLProf2](#)].

907 This section first outlines the ID-WSF SSO Service's conceptual model and then defines the SSO Service in terms of
908 the SAML profiles it supports.

909 6.1. Conceptual Model

910 In the Liberty architecture, it is conceivable for any concrete *system entity* to don any architectural *role* that it is
911 physically capable of bearing. For example, a Liberty *Service Provider* (SP) is essentially just a SAML 2.0 SSO-enabled
912 ~~web site~~. Such Service Providers can also be simultaneously cast into WSC and WSP roles.

913 Similarly, *user agents* in the Liberty architecture range from standard web browsers, to modestly Liberty-enabled
914 browsers (ECPs), to arbitrarily complex SOAP-based clients. These latter user agents, termed *Liberty-enabled User
915 Agents or Devices* (LUADs) will conceivably be dynamically cast into the full range of Liberty architectural roles;
916 they will be called upon to be a browser one moment, and a WSC the next, and even a WSP at times.

917 As noted in [Section 5](#), a (LUAD-)WSC that needs to obtain security tokens in order to interact with a Discovery Service
918 (and subsequently other ID-WSF services) can utilize an ID-WSF Authentication Service to obtain the requisite security
919 tokens. However, not all useful services (SOAP-based or otherwise) that accept SAML security tokens will be registered
920 with a Discovery Service. Furthermore, SAML 2.0 SSO-enabled web sites often rely on the ability to issue requests
921 for authentication directly to less capable clients and expect them to relay the request and subsequent response. LUADs
922 thus need a way to participate in that exchange.

923 Another class of use cases involves calls by one principal (or a WSC acting on behalf of a principal) to invoke services
924 belonging to another principal. These so-called cross-principal invocations often require a WSC to utilize the invoking
925 principal's Single Sign-On Service to obtain security tokens for the target principal's Discovery Service or other ID-
926 WSF services.

927 The ID-WSF Single Sign-On Service addresses these use cases with profiles of the SAML 2.0 Authentication Request
928 protocol [[SAMLCore2](#)] Two distinct, but similar, profiles are defined in order to address differences that arise in the
929 content of security tokens, and protocol processing behavior that is specific to SAML 2.0 SSO SPs. The profile ad-
930 dressing these SPs is a refinement or specialization of the existing SAML 2.0 Enhanced Client/Proxy SSO Profile
931 [[SAMLProf2](#)]. A SAML 2.0 SP can treat a LUAD in the same way as any other enhanced client. A second, more
932 generic, profile permits a LUAD to obtain SAML assertions useful in accessing other kinds of services, including use
933 cases defined in the future.

934 In both profiles, requesters authenticate to the SSOS using ID-WSF security mechanisms, making the SSOS itself an
935 ID-WSF service. A LUAD wishing to interact with the SSOS can use the Authentication Service at an Identity Provider
936 (IdP) to obtain security tokens that enable it to invoke the SSOS at that IdP in order to obtain additional security tokens
937 to convey to SAML 2.0 SPs or other SAML-enabled services.

938 In fact, if a LUAD successfully authenticates with an IdP via the IdP's Authentication Service [Section 5](#), the IdP can
939 ensure that the LUAD will have in its possession an ID-WSF EPR (a profiled `<wsa:EndpointReference>`; [[Lib-
940 ertyDisco](#)]), containing any necessary credentials, for the ID-WSF Single Sign-On Service at the same IdP, simplifying
941 the process of invoking the SSOS. Additionally, the IdP can, at the same time, ensure that the LUAD possesses an ID-
942 WSF EPR containing any necessary credentials for the Discovery Service (DS) of the Principal wielding the LUAD,
943 thus enabling the LUAD to simultaneously utilize SAML and ID-WSF-based services on behalf of the Principal based
944 on one sign-on interaction, from the Principal's perspective.

945 6.2. Single Sign-On Service URIs

946 **Table 3. Single Sign-On Service URIs**

Use	URI
Service Type	<i>urn:liberty:ssos:2006-08</i>
AuthnRequest wsa: Action	<i>urn:liberty:ssos:2006-08:AuthnRequest</i>
Response wsa: Action	<i>urn:liberty:ssos:2006-08:Response</i>

947 6.3. ID-WSF Enhanced Client or Proxy SSO Profile

948 The SAML 2.0 Enhanced Client or Proxy SSO Profile [SAMLProf2] enables SSO to web sites by SAML-aware clients,
 949 but leaves the authentication of the client to the IdP out of scope. This profile is a refinement that adds the ID-WSF
 950 SOAP Binding [LibertySOAPBinding] and Security Mechanisms [LibertySecMech] specifications to the communi-
 951 cation with the IdP, enabling a LUAD to participate in SSO to SAML 2.0-enabled web sites.

952 6.3.1. Profile Overview

953 As introduced above, this profile is simply a constrained version of the interactions specified in [SAMLProf2]. Spe-
 954 cifically, it adds additional requirements to steps 4-6 of the ECP Profile, which involve the interactions between the
 955 IdP and the client. In all other respects, all processing rules defined by the base profile, and in turn the underlying
 956 SAML 2.0 Browser SSO Profile are observed. In particular, note that the content of the SAML protocol messages and
 957 assertion(s) used in this constrained version are entirely unchanged from [SAMLProf2].

958 6.3.2. Profile Description

959 The following sections provide detailed definitions of the individual steps. Except where noted, the steps and processing
 960 rules are as specified in the ECP Profile [SAMLProf2].

961 1. LUAD issues HTTP Request to Service Provider

962 Step 1 is identical to step 1 of the ECP Profile, but MAY be omitted in cases in which the LUAD wishes to
 963 construct the request to the IdP itself and possesses sufficient knowledge of the SP to do so.

964 2. Service Provider issues <samlp2:AuthnRequest> to Identity Provider via LUAD

965 Step 2 is identical to step 2 of the ECP Profile, but note that the <samlp2:AuthnRequest> message MAY be
 966 constructed independently by the LUAD rather than obtained from the SP. From the perspective of the SP, the
 967 eventual response in step 7 will be treated as unsolicited.

968 3. LUAD Determines Identity Provider

969 Step 3, out of scope in the ECP Profile, is similarly out of scope here.

970 4. LUAD forwards <samlp2:AuthnRequest> to Identity Provider

971 In step 4, the <samlp2:AuthnRequest> message is sent to the selected IdP's ID-WSF Single Sign-On Service
 972 endpoint using the Liberty SOAP binding [LibertySOAPBinding]. This message MUST be authenticated using
 973 a security mechanism defined by [LibertySecMech].

974 When communicating with the Identity Provider, the client MUST adhere to the Liberty SOAP binding as specified
975 in [LibertySOAPBinding]; in case of conflict with the SOAP binding as specified in [SAMLBind2] the Liberty
976 SOAP Binding shall take precedence.

977 **Note**

978 The client MAY (and generally will) include various other header blocks, e.g., a <wsse:Security>
979 header block [LibertySecMech] [wss-sms]. Such a header block could contain a security token obtained
980 from the ID-WSF Authentication Service.

981 Note that the <samlp2:AuthnRequest> message may also be signed by the SP (or the LUAD if constructed by
982 it). In this and other respects, the message rules specified in the SAML 2.0 Browser SSO profile in Section 4.1.4.1
983 of [SAMLProf2] MUST be observed.

984 5. Identity Provider Identifies Principal

985 In step 5, the ID-WSF peer-entity authentication mechanism used by the LUAD in step 4 MUST be used to identify
986 the Principal.

987 6. Identity Provider issues <samlp2:Response> message to Service Provider via LUAD

988 Step 6 is identical to step 6 of the ECP Profile, except for the use of the Liberty SOAP Binding during the exchange.
989 The SSOS SHOULD NOT respond in step 6 with any content other than SOAP. For example, the MIME type of
990 the HTTP response must be set according to [LibertySOAPBinding].

991 **Note**

992 This is different from the SAML 2.0 ECP profile [SAMLProf2] in which an IdP is permitted to respond
993 with any content acceptable to the requester during the authentication process.

994 The SSOS MAY take advantage of various optional header blocks defined in [LibertySOAPBinding]. For exam-
995 ple, instead of attempting to establish a local session via an HTTP cookie, the SSOS may include a <disco:
996 SecurityContext> element in an <sb:EndpointUpdate> header block. The requester must of course un-
997 derstand such header blocks.

998 7. LUAD forwards <samlp2:Response> to Service Provider

999 Step 7 is identical to step 7 of the ECP Profile. When the client receives the <samlp2:Response> message from
1000 the IdP, it MUST NOT forward it to any location other than that specified in the
1001 AssertionConsumerServiceURL attribute contained in the mandatory <ecp:Response> header block re-
1002 ceived from the IdP.

1003 Note however that in the case that the LUAD initiated the profile by constructing the <samlp2:
1004 AuthnRequest> message in step 2, then there is no explicit comparison to be made against the
1005 AssertionConsumerServiceURL attribute in the ECP Response header block.

1006 8. Service Provider Grants or Denies Access to Principal

1007 Step 8 is identical to step 8 of the ECP profile.

1008 6.4. ID-WSF SAML Token Service Profile

1009 The SAML Token Service Profile is an ID-WSF-based profile of the SAML 2.0 Authentication Request protocol
1010 [SAMLCore2] that permits a requester to obtain SAML assertions for use by one or more relying parties. Relying
1011 parties might be ID-WSF-based web services, generically defined web services, or other application services that
1012 support SAML. The profile is a direct exchange between the requester and the IdP offering the token service using the

1013 ID-WSF SOAP Binding [LibertySOAPBinding] and is authenticated using the ID-WSF Security Mechanisms speci-
1014 fication [LibertySecMech].

1015 6.4.1. Profile Overview

1016 As introduced above, this profile uses the SAML 2.0 Authentication Request protocol [SAMLCore2] to enable an IdP
1017 to offer a relatively unconstrained capability to issue SAML assertions containing authentication and other information
1018 as security tokens for use by the requester with specified relying parties. Unlike the SSO-oriented profiles defined in
1019 [SAMLProf2] and the previous section, this profile directly involves only two parties: the requester (e.g., a LUAD) |
1020 and the IdP. The client in this profile is the actual requester, rather than an intermediary between the IdP and the eventual
1021 relying party.

1022 Operationally, another difference between this profile and the SSO profiles relates to the content of the SAML assertions
1023 that can be issued. Other than a few basic assumptions, the IdP is generally expected to have sufficient knowledge of
1024 the relying parties identified by the requester so as to support the issuance of appropriately constructed assertions
1025 supporting those parties' requirements. The means by which it can obtain such knowledge are out of scope, and could
1026 include the out of band exchange of policy information, direct configuration, or it may rely on the requester to indicate
1027 to it what kinds of information to include.

1028 This profile is a combination of the SAML Authentication Request protocol and the ID-WSF SOAP Binding [Liber-
1029 tySOAPBinding] and Security Mechanisms [LibertySecMech] specifications, along with a few guidelines on the use
1030 of the <samlp2: AuthnRequest> message.

1031 6.4.2. Profile Description

1032 The following sections provide detailed definitions of the individual steps.

1033 1. Requester issues <samlp2: AuthnRequest> to Identity Provider

1034 In step 1, the requester sends its <samlp2: AuthnRequest> message to the selected IdP's ID-WSF Single Sign-
1035 On Service endpoint using the Liberty SOAP binding [LibertySOAPBinding]. This message MUST be authen-
1036 ticated using a security mechanism defined by [LibertySecMech].

1037 When communicating with the Identity Provider, the client MUST adhere to the Liberty SOAP binding as specified
1038 in [LibertySOAPBinding]; in case of conflict with the SOAP binding as specified in [SAMLBind2] the Liberty
1039 SOAP Binding shall take precedence.

1040 Note

1041 The client MAY (and generally will) include various other header blocks, e.g., a <wsse: Security> |
1042 header block [LibertySecMech] [wss-sms]. Such a header block could contain a security token obtained
1043 from the ID-WSF Authentication Service.

1044 2. Identity Provider Identifies Principal

1045 In step 2, the ID-WSF peer-entity authentication mechanism used by the requester in step 1 MUST be used to
1046 identify the requesting Principal.

1047 3. Identity Provider issues <samlp2: Response> message to Requester

1048 In step 3, the IdP returns a <samlp2: Response> message to the requester containing the status and any SAML
1049 assertion(s) issued as a result of the request. The SSOS SHOULD NOT respond with any content other than SOAP.
1050 For example, the MIME type of the HTTP response must be set according to [LibertySOAPBinding].

1051 The SSOS MAY take advantage of various optional header blocks defined in [LibertySOAPBinding]. For exam-
1052 ple, instead of attempting to establish a local session via an HTTP cookie, the SSOS may include a <disco:

1053 SecurityContext> element in an <sb:EndpointUpdate> header block. The requester must of course un-
1054 derstand such header blocks.

1055 **6.4.3. Use of SAML 2.0 Authentication Request Protocol**

1056 This profile is based on the SAML 2.0 Authentication Request protocol defined in [SAMLCore2]. In the nomenclature
1057 of actors enumerated in Section 3.4 of that document, the requester is the SAML requester, presenter, and the attesting
1058 entity, and is generally the requested subject. Relying parties may be identified in the request but are not a party to the
1059 profile. There may be additional attesting entities and relying parties at the discretion of the identity provider (see
1060 below).

1061 **6.4.3.1. <samlp2:AuthnRequest> Usage**

1062 Except as described below, a requester MAY include any message content described in [SAMLCore2], Section 3.4.1.
1063 All processing rules are as defined in [SAMLCore2].

1064 The <saml2:Issuer> element MUST NOT be present. This avoids duplication or conflict with the mandatory in-
1065 formation already available from the ID-WSF SOAP Binding.

1066 If the IdP cannot or will not satisfy the request, it MUST respond with a <samlp2:Response> message containing
1067 an appropriate error status code or codes.

1068 The ProtocolBinding attribute MUST be included and MUST be set to http://www.w3.org/2005/08/
1069 addressing/anonymous to indicate the response is to be returned directly to the requester. This value clearly dis-
1070 tinguishes this profile's use of the protocol from that of the SAML 2.0 SSO profiles. The request MUST NOT contain
1071 the AssertionConsumerServiceURL or AssertionConsumerServiceIndex attributes.

1072 If no <saml2:Subject> element is included, the invocation identity associated with the request is implied to be the
1073 requested subject. The requester MAY explicitly include a <saml2:Subject> element in the request that names the
1074 actual Principal about which it wishes to receive an assertion. If the IdP does not recognize the requester as that Principal
1075 (or an entity allowed to attest to it), then it MUST respond with a <samlp2:Response> message containing an error
1076 status and no assertions.

1077 In most cases, the identifier to return for the requested subject can be determined based on the identity of the relying
1078 party or parties. If this is not sufficient (for example if the relying party is to be treated as a member of an affiliation
1079 of providers), then the <samlp2:NameIDPolicy> element can be used to indicate this using the
1080 SPNameQualifier attribute. (Note that this precludes the identification of multiple relying parties in a single request.)

1081 If the requester wishes to permit the IdP to establish a new identifier for the Principal if none exists, it MUST include
1082 a <samlp2:NameIDPolicy> element with the AllowCreate attribute set to "true". Otherwise, only a principal for
1083 whom the IdP has previously established an identifier usable by the relying party or parties can be authenticated
1084 successfully.

1085 The requester MAY include one or more <saml2:SubjectConfirmation> elements in the request to specify at-
1086 testation mechanisms to be attached to the resulting assertion(s). Usually this is done to translate ID-WSF security
1087 mechanism requirements into the corresponding SAML confirmation methods that will be needed to satisfy the relying
1088 party's security policy. Refer to [LibertySecMech] for a detailed mapping of security mechanisms to SAML confir-
1089 mation methods.

1090 The requester MAY include a <saml2:Conditions> element in the request. The IdP is NOT obligated to honor the
1091 requested conditions, but SHOULD return an error if it cannot do so.

1092 The requester MAY include an <saml2:AudienceRestriction> element in the request to enumerate one or more
1093 relying parties by means of a <saml2:Audience> element containing a unique identifier for the relying party. The
1094 IdP can utilize whatever knowledge is at its disposal to determine the appropriate content to place in the resulting
1095 assertion(s), as well as how many assertions to issue, and whether the use of XML encryption is required.

1096 The request MUST be signed or otherwise integrity protected by the binding or transport layer.

1097 **6.4.3.2. <samlp2:Response><samlp2:Response> Usage**

1098 If the IdP wishes to return an error, it MUST NOT include any assertions in the <samlp2:Response> message.

1099 Otherwise the <samlp2:Response> element MUST conform to the following:

- 1100 • The <saml2:Issuer> element MAY be omitted, but if present MUST contain the unique identifier of the issuing
1101 IdP; the Format attribute MUST be omitted or have a value of urn:oasis:names:tc:SAML:2.0:nameid-
1102 format:entity
- 1103 • It MUST contain at least one <saml2:Assertion>. Each assertion's <saml2:Issuer> element MUST contain
1104 the unique identifier of the issuing IdP; the Format attribute MUST be omitted or have a value of urn:oasis:
1105 names:tc:SAML:2.0:nameid-format:entity
- 1106 • Each assertion returned MUST be signed.
- 1107 • The set of one or more assertions MUST contain at least one <saml2:AuthnStatement> that reflects the au-
1108 thentication of the subject to the IdP.
- 1109 • Confirmation methods and additional statements MAY be included in the assertion(s) at the discretion of the IdP.
1110 In particular, <saml2:AttributeStatement> elements MAY be included.
- 1111 • The assertions SHOULD contain an <saml2:AudienceRestriction> condition element containing the unique
1112 identifier of the intended relying party or parties within the included <saml2:Audience> elements.
- 1113 • Other conditions (and other <saml2:Audience> elements) MAY be included as requested or at the discretion of
1114 the IdP. Of course, all such conditions MUST be understood and accepted by the relying party in order for the
1115 assertion to be considered valid.

1116 **6.5. Use of Metadata**

1117 An IdP that offers an ID-WSF Single Sign-On Service supporting either of the profiles above SHOULD advertise
1118 support for this capability in its metadata [SAMLMeta2]. To accomplish this, it MUST include a <md:
1119 SingleSignOnService> element in its metadata with a Binding attribute of urn:liberty:sb:2006-08.

1120 The <md:IDPSSODescriptor> element's WantAuthnRequestsSigned attribute MAY be used by an IdP to docu-
1121 ment a requirement that requests be signed (as opposed to protected only at the transport layer).

1122 **6.6. Inclusion of ID-WSF Endpoint References**

1123 Both SSOS profiles support the inclusion of arbitrary content in the assertions returned. Specifically, the SSOS MAY
1124 include ID-WSF EPRs, encoded as SAML attributes, for the associated Principal's Discovery Service or other ID-WSF-
1125 based services. These EPRs MAY contain additional security tokens or MAY refer to the containing assertion if
1126 appropriate.

1127 7. Identity Mapping Service

1128 The ID-WSF Identity Mapping Service enables a requester to obtain one or more "identity tokens." As defined in
1129 [LibertySecMech], identity tokens can be used to refer to principals in a privacy-preserving manner. The Identity
1130 Mapping Service is an ID-WSF web service that translates references to a principal into alternative formats or identifier
1131 namespaces. It is a generalization of the Name Identifier Mapping protocol defined in [SAMLCore2].

1132 This section first outlines the service's conceptual model and then defines the protocol and message elements.

1133 7.1. Conceptual Model

1134 The ID-WSF Identity Mapping Service allows a requester in possession of one or more identity tokens to translate,
1135 update, or refresh them using the protocol defined here. An example employer of this service is the ID-WSF People
1136 Service [LibertyPeopleService]. A principal may also act on behalf of itself (or in conjunction with a WSC) to obtain
1137 an identity token representing that principal for use in subsequent web service invocations.

1138 An identity token can take a variety of forms, including many SAML-based representations such as SAML assertions
1139 or SAML identifier fragments (encrypted or plaintext) such as may appear in the subject of SAML assertions (e.g.,
1140 elements such as <saml2:NameID> or <saml2:EncryptedID>). A security token, such as a SAML authentication
1141 assertion can also serve as an identity token. Note that when evaluating the validity of an identity token in SAML
1142 assertion form, constraints may be imposed on its use by the issuer of the assertion.

1143 SAML assertions used as identity tokens can also be used to communicate ID-WSF EPR and credential information,
1144 such as for the associated Principal's Discovery Service or other ID-WSF-based services.

1145 Conceptually, the mapping protocol is a translation or exchange of one or more inputs for corresponding outputs. Each
1146 input consists of an identity token and a policy specifying the identity token to return. The security token of the invoking
1147 identity can also be referenced as the input token. The output is the requested identity token, the exact form of which
1148 may be up to the mapping service to establish.

1149 7.2. Schema Declarations

1150 The XML schema [Schema1-2] normatively defined in this section is constituted in the XML Schema file: liberty-
1151 idwsf-idmapping-svc-v2.0.xsd, entitled " Liberty ID-WSF Identity Mapping Service XSD v2.0 " (see Appen-
1152 dix D).

1153 Additionally, Liberty ID-WSF Identity Mapping Service XSD v2.0 imports items from liberty-idwsf-utility-
1154 v2.0.xsd (see Appendix E: Liberty ID-WSF Utility XSD v2.0), and also from liberty-idwsf-security-
1155 mechanisms-v2.0.xsd (see [LibertySecMech]).

1156 7.3. SOAP Binding

1157 The Identity Mapping Service is an ID-WSF service; as such, the messages defined in Section 7.4 constitute *ordinary*
1158 *ID-* messages* as defined in [LibertySOAPBinding]. They are intended to be bound to the [SOAPv1.1] protocol by
1159 mapping them directly into the <s:Body> element of the <s:Envelope> element comprising a SOAP message.

1160 [LibertySOAPBinding] normatively specifies this binding, as well as various required and optional SOAP header blocks
1161 usable with this protocol.

1162 7.3.1. Identity Mapping Service URIs

1163 **Table 4. Identity Mapping Service URIs**

Use	URI
Service Type	<i>urn:liberty:ims:2006-08</i>
IdentityMappingRequest wsa: Action	<i>urn:liberty:ims:2006-08:IdentityMappingRequest</i>
IdentityMappingResponse wsa: Action	<i>urn:liberty:ims:2006-08:IdentityMappingResponse</i>

1164 7.4. Protocol Messages and Usage

1165 The following request and response messages make up the Identity Mapping Service protocol:

1166 7.4.1. Element <IdentityMappingRequest>

1167 The <IdentityMappingRequest> element is of complex type **IdentityMappingRequestType**, and is defined in
 1168 [Figure 4](#) and in [Liberty ID-WSF Identity Mapping Service XSD v2.0](#) . This type contains the following attributes and
 1169 elements:

- 1170 • **Any Attributes** [Optional]

1171 Zero or more extension attributes qualified by an XML namespace other than the Identity Mapping Service name-
 1172 space.

- 1173 • **<MappingInput>** [One or More]

1174 One or more elements specifying the principals to return identity tokens for, and the policies describing the contents
 1175 of those tokens.

```

1176
1177 <xs:element name="IdentityMappingRequest" type="IdentityMappingRequestType" />
1178 <xs:complexType name="IdentityMappingRequestType">
1179   <xs:sequence>
1180     <xs:element ref="MappingInput" maxOccurs="unbounded" />
1181   </xs:sequence>
1182   <xs:anyAttribute namespace="##other" processContents="lax" />
1183 </xs:complexType>
    
```

1184 **Figure 4. Element <IdentityMappingRequest> Schema Fragment**

1185 7.4.1.1. Element <MappingInput>

1186 The <MappingInput> element is of complex type **MappingInputType**, and is defined in [Figure 5](#) and in [Liberty](#)
 1187 [ID-WSF Identity Mapping Service XSD v2.0](#) . This type contains the following attributes and elements:

- 1188 • **reqID** [Optional]

1189 Uniquely identifies a <MappingInput> element within a request. Used by the responder to correlate the
 1190 <MappingOutput> elements that it returns to their corresponding inputs.

- 1191 • **<sec:TokenPolicy>** [Optional]

1192 A container for information specifying the characteristics of the identity token the requester wants returned to it.

1193 • **<sec:Token>** [Required]

1194 A container for the identity token (or token reference) that specifies the principal for whom to return a new identity
1195 token.

1196 **Note**

1197 Notwithstanding the schema definition below that declares the `<sec:Token>` element to be optional, a
1198 `<sec:Token>` element **MUST** be present in the `<MappingInput>` element. The looser schema definition
1199 is designed to enable extension of the type by other specifications for which a mandatory `<sec:`
1200 `Token>` element may not be appropriate.

```
1201  
1202 <xs:element name="MappingInput" type="MappingInputType" />  
1203 <xs:complexType name="MappingInputType">  
1204   <xs:sequence>  
1205     <xs:element ref="sec:TokenPolicy" minOccurs="0" />  
1206     <xs:element ref="sec:Token" minOccurs="0" />  
1207   </xs:sequence>  
1208   <xs:attribute name="reqID" type="lu:IDType" use="optional" />  
1209 </xs:complexType>
```

1210 **Figure 5. Element <MappingInput> Schema Fragment**

1211 7.4.1.2. Request Usage

1212 An `<IdentityMappingRequest>` consists of one or more `<MappingInput>` elements. If multiple
1213 `<MappingInput>` elements are included in a request, then each element **MUST** contain a `reqID` attribute so that the
1214 response contents can be correlated to them.

1215 Each input consists of an identity (in the form of a `<sec:Token>`) and an optional `<sec:TokenPolicy>`.

1216 The input identity token describes the principal for whom the requester desires a new identity token. This **MAY** be a
1217 reference to a token elsewhere in the message or in some other location.

1218 The input token policy describes the nature of the identity token to be returned, generally focusing on the nature of the
1219 identifier. Often a principal will possess many alternate identifiers of different formats or scoped to different usage
1220 contexts, such as a particular SP or affiliation. The policy allows the requester to translate from the input token to some
1221 other form.

1222 If no token policy is supplied, then the resulting output token **SHOULD** have the same general characteristics as the
1223 input token, save perhaps for associated information such as its lifetime. This might be used to renew a token, for
1224 example.

1225 As identity tokens come in a variety of forms, so too the form of the input policy can vary. In the specific case of a
1226 SAML-based identity token, a `<samlp2:NameIDPolicy>` **SHOULD** be used, as defined in [SAMLCore2].

1227 The token policy **SHOULD** identify the entity for whom the identity token is being created, if other than the requester.
1228 If this cannot be otherwise inferred from the policy, the `issueTo` attribute **SHOULD** be used to identify this entity.
1229 When the `<samlp2:NameIDPolicy>` is used, the `SPNameQualifier` attribute will often supply this information, at
1230 least for persistent identifiers in typical use cases, making use of the `issueTo` attribute redundant.

1231 7.4.2. Element <IdentityMappingResponse>

1232 The `<IdentityMappingResponse>` element is of complex type **IdentityMappingResponseType**, and is defined in
1233 [Figure 6](#) and in [Liberty ID-WSF Identity Mapping Service XSD v2.0](#). This type contains the following attributes and
1234 elements:

- 1235 • **Any Attributes** [Optional]
- 1236 Zero or more extension attributes qualified by an XML namespace other than the Identity Mapping Service name-
- 1237 space.
- 1238 • **<lu: Status>** [Required]
- 1239 The status of the request. This element is defined in [Liberty ID-WSF Utility XSD v2.0](#) .
- 1240 • **<MappingOutput>** [Zero or More]
- 1241 Zero or more elements containing the identity tokens returned.
- 1242
- 1243

```
<xs:element name="IdentityMappingResponse" type="IdentityMappingResponseType"/>
```
- 1244

```
<xs:complexType name="IdentityMappingResponseType">
```
- 1245

```
  <xs:sequence>
```
- 1246

```
    <xs:element ref="lu:Status"/>
```
- 1247

```
    <xs:element ref="MappingOutput" minOccurs="0" maxOccurs="unbounded"/>
```
- 1248

```
  </xs:sequence>
```
- 1249

```
  <xs:anyAttribute namespace="##other" processContents="lax"/>
```
- 1250

```
</xs:complexType>
```

Figure 6. Element <IdentityMappingResponse> Schema Fragment

1252 **7.4.2.1. Element <MappingOutput>**

1253 The <MappingOutput> element is of complex type **MappingOutputType**, and is defined in [Figure 7](#) and in [Liberty](#)
 1254 [ID-WSF Identity Mapping Service XSD v2.0](#) . This type contains the following attributes and elements:

- 1255 • **reqRef** [Optional]
- 1256 Uniquely identifies a <MappingInput> element within the corresponding request. Used to correlate
- 1257 <MappingOutput> elements to their corresponding inputs.
- 1258 • **<sec: Token>** [Required]
- 1259 A container for the identity token (or token reference) returned by the responder.
- 1260
- 1261

```
<xs:element name="MappingOutput" type="MappingOutputType"/>
```
- 1262

```
<xs:complexType name="MappingOutputType">
```
- 1263

```
  <xs:sequence>
```
- 1264

```
    <xs:element ref="sec:Token"/>
```
- 1265

```
  </xs:sequence>
```
- 1266

```
  <xs:attribute name="reqRef" type="lu:IDReferenceType" use="optional"/>
```
- 1267

```
</xs:complexType>
```

Figure 7. Element <MappingOutput> Schema Fragment

1269 **7.4.2.2. Response Usage**

1270 An <IdentityMappingResponse> consists of a status element and zero or more <MappingOutput> elements, one
 1271 for each successfully processed token request. Unsuccessfully processed <MappingInput> elements do not result in a
 1272 corresponding <MappingOutput> element. ~~elements~~–If multiple <MappingInput> elements were included in a re-
 1273 quest, then each output element MUST contain a reqRef attribute matching it to the corresponding input element.

1274 Each output element consists of an identity token (in the form of a <sec: Token>).

1275 Any tokens returned MUST be constructed in accordance with the policy supplied in the input. A specific exception
1276 to this requirement is that any `validUntil` attribute specified by the requester MAY be ignored. If no policy was
1277 specified, the identity token to return is presumed to be of the same nature as the identity token used as input.

1278 The responder MUST take appropriate steps to ensure the privacy of the principal by encrypting the resulting identity
1279 information such that only the principal and parties known to be privy to the information can read it.

1280 If each input element is satisfied in the resulting response, then the responder MUST return a top-level `<lu:
1281 Status>` code of "OK."

1282 If at least one, but not all, of the resulting inputs cannot be satisfied, then the responder MUST return a top-level
1283 `<lu:Status>` code of "Partial." It MAY return nested `<lu:Status>` elements reflecting the specific result of the
1284 failed inputs.

1285 If none of the resulting inputs can be satisfied, then the responder MUST return a top-level `<lu:Status>` code of
1286 "Failed." It MAY return nested `<lu:Status>` elements reflecting the specific result of the failed inputs.

1287 When multiple inputs are present, any nested `<lu:Status>` elements MUST contain a `ref` attribute equal to the
1288 associated `<MappingInput>`'s `reqID` attribute.

1289 7.4.2.3. Second-Level Status Codes

1290 The following second-level codes are defined to represent common error conditions that may arise. Others may be
1291 defined by implementations as required.

- 1292 • "UnknownPrincipal" — the input token did not match a principal known to the service
- 1293 • "BadInput" — the input token or policy was malformed or not understood
- 1294 • "Denied" — the requested token translation was a violation of user or system policy

1295 7.5. SAML Identity Tokens

1296 As described in [LibertySecMech], identity tokens can be expressed in many ways. Even in the specific case of SAML,
1297 many different formulations are possible, depending on the requirements. This section outlines a few common ways
1298 of expressing identification using SAML with different security and privacy characteristics. The identity mapping
1299 service is responsible for selecting an appropriate choice based on the requester, input policy, and the expected purpose.
1300 It is outside the scope of this specification how such purposes are to be understood.

1301 7.5.1. Assertions

1302 SAML assertions can be used as identity tokens that reference the subject of the assertion. Such assertions are generally
1303 signed for integrity, and often contain no statements, only a `<saml2:Subject>` element, possibly with conditions
1304 that limit its use.

1305 When privacy is required, a `<saml2:EncryptedID>` element SHOULD be used in the subject. The decrypted element
1306 SHOULD NOT itself be an assertion (as it would be redundant).

1307 If it is unnecessary to reveal the content of the enclosing assertion to the requester, then a `<saml2:
1308 EncryptedAssertion>` element SHOULD be used, as it is simpler for the requester to handle. Alternatively, a
1309 `<saml2:EncryptedID>` element could be returned directly (see the following section).

1310 SAML assertions used as identity tokens MAY contain ID-WSF EPR attributes and credentials, such as for the asso-
1311 ciated Principal's Discovery Service or other ID-WSF-based services.

1312 7.5.2. Identifiers

1313 SAML identifier elements (<saml2:BaseID>, <saml2:NameID>, or <saml2:EncryptedID>) can also be used as
1314 identity tokens. Encrypted identifiers, in particular, can contain actual signed assertions, making them potential carriers
1315 of the assertion forms described in the previous section.

1316 However, in cases where privacy is not a consideration and limits on the use of the identifier are not relevant, a plaintext
1317 form may also be useful, particularly as an input token to a mapping request. For example, an SP that shares an identifier
1318 for a principal with an IdP offering an Identity Mapping service might use a <saml2:NameID> by itself as an input
1319 token.

1320 7.6. Security and Privacy Considerations

1321 Privacy is a critical consideration in the operation of the Identity Mapping Service, because its primary purpose is to
1322 enable entities to invoke services on behalf of principals without requiring the use of globally unique identifiers. Because
1323 identifiers in ID-WSF are typically scoped to particular providers, care must be exercised when allowing providers to
1324 map between them.

1325 In particular, it is usually the case that the identifiers returned by the IMS SHOULD be encrypted when returned to
1326 parties other than those with prior knowledge of them. The use of XML encryption generally results in unique ciphertext
1327 each time a particular value is encrypted, preventing correlation by parties without access to the underlying plaintext.

1328 It is also important for the IMS to take into consideration the relationship between the input and output tokens. Under
1329 most circumstances, it should not be possible for a requester to map to its own namespace from another, as this would
1330 permit the requester to correlate the original identifier to one that it knows. Rather, the IMS is more generally used to
1331 map from a known identifier into another entity's namespace, returning the result in an encrypted form so that it can
1332 be decrypted by that entity.

1333 7.7. Example Identity Mapping Exchange

1334 The following example shows a request for a SAML identity token. The policy and input token indicate a request to
 1335 map from an identifier scoped to one SP into an identifier scoped to another. In this case, the input token is a bare
 1336 identifier (probably extracted from another SAML token).

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

1350

1351

```

<sa:IdentityMappingRequest>
  <sa:MappingInput>
    <sec:TokenPolicy type="urn:liberty:security:2006-08:IdentityTokenType:SAML20Assertion"><sec:TokenPolicy>
    <-----<samlp2:NameIDPolicy Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
    SPNameQualifier="https://spb.example.com"/>
    </sec:TokenPolicy>
    <sec:Token>
      <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
      NameQualifier="https://idp.example.com" SPNameQualifier="https://spa.example.com">
      DBC63923-C718-4249-83CE-1E53D80D8A4A
      </saml2:NameID>
    </sec:Token>
  </sa:MappingInput>
</sa:IdentityMappingRequest>
  
```

1352 The following is a possible response to the request above. The returned token is a signed SAML assertion **with** within
 1353 an encrypted **name** identifier. The requester can establish the expiration from the response, giving it guidance as to
 1354 when the token might need renewal.

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

```

<sa:IdentityMappingResponse>
  <sa:Status code="OK" />
  <sa:MappingOutput>
    <sec:Token>
      <saml2:Assertion Version="2.0" IssueInstant="2006-03-19T07:35:00Z"
      ID="e9ab6ff0-4ee0-4ce2-868f-18873bdc87de">
      <saml2:Issuer>https://idp.example.com</saml2:Issuer>
      <ds:Signature>...</ds:Signature><saml2:EncryptedID>
      <-----<saml2:Subject>
      <saml2:EncryptedID>
      <xenc:EncryptedData>U2XTCNvRX7B11NK182nmY00TEk==</xenc:EncryptedData>
      </saml2:EncryptedID>
      </saml2:Subject>
      <saml2:Conditions<xenc:EncryptedKey-NotOnOrAfter="2006-03-19T08:35:00Z">
      <saml2:AudienceRestriction>
      <saml2:Audience>https://spb.example.com</saml2:Audience>
      </saml2:AudienceRestriction>
      </saml2:Conditions>Recipient="https://spb.example.com">...</xenc:EncryptedKey>
      </saml2:Assertion>
    </sec:Token>
  </sa:MappingOutput>
</sa:IdentityMappingResponse>
  
```

1378

Example 11.

1379 8. Password Transformations: The PasswordTransforms Element

1380 This section defines the <PasswordTransforms> element. Authentication servers MAY use this element to convey
1381 password pre-processing obligations to clients.

1382 For example, an authentication server may have been configured such that it presumes that the strings users enter as
1383 their passwords have been pre-processed in some fashion before being further processed and/or stored. For example
1384 the passwords may be truncated to a given length, and all upper case characters may be folded to lower case, and
1385 whitespace may be eliminated. The authentication server can communicate these requirements dynamically to clients
1386 using the <PasswordTransforms> element in an initial <SASLResponse>. See [Figure 8](#).

```

1387
1388 <xs:element name="PasswordTransforms">
1389
1390   <xs:annotation>
1391     <xs:documentation>
1392       Contains ordered list of sequential password transformations
1393     </xs:documentation>
1394   </xs:annotation>
1395
1396   <xs:complexType>
1397     <xs:sequence>
1398
1399       <xs:element name="Transform" maxOccurs="unbounded">
1400         <xs:complexType>
1401           <xs:sequence>
1402
1403             <xs:element name="Parameter"
1404               minOccurs="0"
1405               maxOccurs="unbounded">
1406               <xs:complexType>
1407                 <xs:simpleContent>
1408                   <xs:extension base="xs:string">
1409                     <xs:attribute name="name"
1410                       type="xs:string"
1411                       use="required" />
1412                   </xs:extension>
1413                 </xs:simpleContent>
1414               </xs:complexType>
1415             </xs:element>
1416           </xs:sequence>
1417
1418           <xs:attribute name="name"
1419             type="xs:anyURI"
1420             use="required" />
1421
1422           <xs:anyAttribute namespace="##other" processContents="lax" />
1423
1424         </xs:complexType>
1425       </xs:element>
1426     </xs:sequence>
1427   </xs:complexType>
1428 </xs:element>
1429
1430

```

1431 **Figure 8. The PasswordTransforms element**

```
1432
1433 <PasswordTransforms>
1434   <Transform name="urn:liberty:sa:pm:truncate">
1435     <Parameter name="length">8</Parameter>
1436   </Transform>
1437   <Transform name="urn:liberty:sa:pm:lowercase" />
1438 </PasswordTransforms>
1439
```

1440 **Figure 9. Example of a PasswordTransforms**

1441 Servers MAY include a <PasswordTransforms> element along with their **initial** <SASLResponse> to a client. A
1442 <PasswordTransforms> element contains one or more <Transform> elements. Each <Transform> is identified
1443 by the value of the name attribute which must be a URI [RFC3986]. This URI MUST specify a particular transformation
1444 on the password. Transforms are specified elsewhere, for example in configuration data at implementation- and/or
1445 deployment-time. A basic set is specified in [Appendix B: Password Transformations](#).

1446 A client receiving an **initial** <SASLResponse> message containing a <PasswordTransforms> element MUST apply
1447 the specified transformations to any password that is used as input for the SASL mechanism indicated in the
1448 <SASLResponse>.

1449 The client MUST apply the transformations in the order given in the <PasswordTransforms> element, and MUST
1450 apply each transform to the result of the preceding transform. Of course, the first transform MUST be applied to the
1451 raw password.

1452 Unless the specification of a <Transform> states otherwise, it is specified in terms of [[Unicode](#)] *abstract charac-*
1453 *ters*. An abstract character is a character as rendered to a user. Since an abstract character may require more than one
1454 octet to represent, there is not necessarily a one-to-one mapping between an abstract character, or sequence of abstract
1455 characters, and its corresponding *coded character representation*.

1456 For example, if a truncation transform indicates, "truncate after the first eight **characters**," the characters after the eighth
1457 abstract character should be removed; in some languages and character encodings this could mean that more than 8
1458 octets remain.

1459 See also [Appendix B](#).

1460 **9. Acknowledgments**

1461 This spec leverages techniques and ideas from draft-nystrom-http-sasl-xx (an IETF Internet-Draft), RFC3080,
1462 RFC2251, RFC2829, RFC2830, et al (all are various IETF Requests For Comments). The authors of those specs are
1463 gratefully acknowledged. Thanks also to Alexy Melnikov, Paul Madsen, and RL "Bob" Morgan for their feedback and
1464 insights. The docbook source code for this specification was hand set to the tunes of Brad, Bob Mould, Weather Report,
1465 Miles Davis, John Coltrane, Liz Phair, The Wallflowers, Alan Holdsworth, Chick Corea, Jennifer Trynin, Elisa Ko-
1466 renne, The Cowboy Junkies, Fugazi, Blues Traveler, Blink-182, CSN, Pearl Jam, and various others. Thanks also to
1467 whatever deities are responsible for the existence of coffee, dark chocolate, and fermented cereals.

1468 References

1469 Normative

- 1470 [LibertyAuthnContext] Madsen, Paul, eds. "Liberty ID-FF Authentication Context Specification," Version 2.0-01,
1471 Liberty Alliance Project (21 November 2004). <http://www.projectliberty.org/specs>
- 1472 [LibertyClientProfiles] Aarts, Robert, Kainulainen, Jukka, Kemp, John, eds. "Liberty ID-WSF Profiles for Liberty-
1473 Enabled User Agents and Devices," Version 2.0-errata-v1.0, Liberty Alliance Project (22 January, 2007).
1474 <http://www.projectliberty.org/specs>
- 1475 [LibertyDisco] Cahill, Conor, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-
1476 errata-v1.0, Liberty Alliance Project (29 November, 2006). <http://www.projectliberty.org/specs>
- 1477 [LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-
1478 errata-v1.0, Liberty Alliance Project (21 April, 2007). <http://www.projectliberty.org/specs>
- 1479 [LibertyPAOS] Aarts, Robert, Kemp, John, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version
1480 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1481 [LibertyPeopleService] Koga, Yuzo, Madsen, Paul, eds. "Liberty ID-WSF People Service Specification," Version 1.0-
1482 errata-v1.0, Liberty Alliance Project (06 March, 2007). <http://www.projectliberty.org/specs>
- 1483 [LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version
1484 1.2-errata-v3.0, Liberty Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 1485 [LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version 2.0-errata-v1.0,
1486 Liberty Alliance Project (21 April, 2007). <http://www.projectliberty.org/specs>
- 1487 [LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0, Liberty Alliance Project (30 July,
1488 2006). <http://www.projectliberty.org/specs>
- 1489 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Liberty ID-
1490 WSF SOAP Binding Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007). [http://](http://www.projectliberty.org/specs)
1491 www.projectliberty.org/specs
- 1492 [LibertyIDWSFv20Errata] Champagne, Darryl, Lockhart, Rob, Tiffany, Eric, eds. "Liberty ID-WSF 2.0 Errata," Ver-
1493 sion 1.0, Liberty Alliance Project (13 April, 2007). <http://www.projectliberty.org/specs>
- 1494 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet Engi-
1495 neering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>
- 1496 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., eds. (June 1999).
1497 "Hypertext Transfer Protocol -- HTTP/1.1," RFC 2616, The Internet Engineering Task Force [http://](http://www.ietf.org/rfc/rfc2616.txt)
1498 www.ietf.org/rfc/rfc2616.txt
- 1499 [RFC3066] Alvestrand, H., eds. (January 2001). "Tags for the Identification of Languages," RFC 3066., Internet En-
1500 gineering Task Force <http://www.ietf.org/rfc/rfc3066.txt>
- 1501 [RFC3986] Berners-Lee, T., Fielding, R., Masinter, L., eds. (January 2005). "Uniform Resource Identifier (URI):
1502 Generic Syntax," RFC 3986 (Obsoletes RFC2732, RFC2396, RFC1808) (Updates RFC1738) (Also STD0066)
1503 (Status: STANDARD), The Internet Engineering Task Force <http://www.ietf.org/rfc/rfc3986.txt>
- 1504 [RFC4346] Dierks, T., Rescorla, E., eds. (April 2006). "The Transport Layer Security (TLS) Protocol," Version 1.1
1505 RFC 4346, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc4346.txt>

- 1506 [RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., Wright, T., eds. (April 2006). "Transport
1507 Layer Security (TLS) Extensions," RFC 4366, The Internet Engineering Task Force [http://www.ietf.org/rfc/
1508 rfc4366.txt](http://www.ietf.org/rfc/rfc4366.txt)
- 1509 [RFC4422] "Simple Authentication and Security Layer (SASL)," Melnikov, A., Zeilenga, K., eds. (June 2006). RFC
1510 4422, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc4422.txt>
- 1511 [SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (2 September 2003). "Assertions and Protocol for the
1512 OASIS Security Assertion Markup Language (SAML) V1.1," SAML v1.1, OASIS Standard, Organization
1513 for the Advancement of Structured Information Standards [http://www.oasis-open.org/committees/down-
1514 load.php/3406/oasis-sstc-saml-core-1.1.pdf](http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf)
- 1515 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions and Protocol
1516 for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organ-
1517 ization for the Advancement of Structured Information Standards [http://docs.oasis-open.org/security/saml/
1518 v2.0/saml-core-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf)
- 1519 [SAMLGloss2] Hodges, Jeff, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Glossary for the OASIS Security
1520 Assertion Markup Language (SAML) V2.0," SAML 2.0, OASIS Standard, Organization for the Advancement
1521 of Structured Information Standards <http://docs.oasis-open.org/security/saml/v2.0/saml-glossary-2.0-os.pdf>
- 1522 [SAMLMeta2] Cantor, Scott, Moreh, Jahan, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Metadata for the OASIS
1523 Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for the
1524 Advancement of Structured Information Standards [http://docs.oasis-open.org/security/saml/v2.0/saml-met-
1525 adata-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf)
- 1526 [SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler,
1527 Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0,"
1528 SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards [http://
1529 docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf)
- 1530 [SASLReg] "Simple Authentication and Security Layer (SASL) Mechanisms," Internet Assigned Numbers Authority
1531 (IANA) <http://www.iana.org/assignments/sasl-mechanisms>
- 1532 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October 2004).
1533 "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium [http://
1534 www.w3.org/TR/xmlschema-1/](http://www.w3.org/TR/xmlschema-1/)
- 1535 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David, Kakivaya, Gopal, Layman,
1536 Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C
1537 Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 1538 [Unicode] The Unicode Consortium (2003). "The Unicode Standard, version 4.0," Addison-Wesley [Unicode 4.0.0
1539 \[http://www.unicode.org\]](http://www.unicode.org)
- 1540 [WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, Rogers, Tony, eds.
1541 World Wide Web Consortium W3C Recommendation (9 May 2006). [http://www.w3.org/TR/2006/REC-ws-
1542 addr-core-20060509/](http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/)
- 1543 [WSAv1.0-SOAP] "WS-Addressing 1.0 SOAP Binding," Gudgin, Martin, Hadley, Marc, eds. World Wide Web Con-
1544 sultium W3C Recommendation (9 May 2006). <http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/>
- 1545 [WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Meredith,
1546 Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001). [http://
1547 www.w3.org/TR/2001/NOTE-wsdl-20010315](http://www.w3.org/TR/2001/NOTE-wsdl-20010315)

1548 Informational

- 1549 [IANA] "The Internet Assigned Numbers Authority," <http://www.iana.org/>
- 1550 [LibertyIDPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification," Ver-
1551 sion 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 1552 [LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework Over-
1553 view," Version 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1554 [Merriam-Webster] "Merriam-Webster Dictionary," <http://www.merriam-webster.com/>
- 1555 [RFC2289] "A One-Time Password System," N. Haller C. Metz P. Nessner M. Straw (February 1998). RFC 2289,
1556 Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2289.txt>
- 1557 [RFC2444] Newman, C., eds. (October 1998). "The One-Time-Password SASL Mechanism," RFC 2444, The Internet
1558 Engineering Task Force <http://www.ietf.org/rfc/rfc2444.txt>
- 1559 [RFC2828] Shirey, R., eds. (May 2000). "Internet Security Glossary," RFC 2828., Internet Engineering Task Force
1560 <http://www.ietf.org/rfc/rfc2828.txt>
- 1561 [RFC3163] Zuccherato, R., Nystrom, M., eds. (August 2001). "ISO/IEC 9798-3 Authentication SASL Mechanism,"
1562 RFC 3163, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc3163.txt>
- 1563 [SAMLBind2] Cantor, Scott, Hirsch, Frederick, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Bind-
1564 ings for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard,
1565 Organization for the Advancement of Structured Information Standards [http://docs.oasis-open.org/security/
1566 saml/v2.0/saml-bindings-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf)
- 1567 [SOAPv1.2] "SOAP Version 1.2 Part 1: Messaging Framework," Gudgin, Martin, Hadley, Marc, Mendelsohn, Noah,
1568 Moreau, Jean-Jacques, Nielsen, Henrik Frystyk, eds. World Wide Web Consortium W3C Recommendation
1569 (07 May 2003). <http://www.w3.org/TR/2003/PR-soap12-part1-20030507/>
- 1570 [TrustInCyberspace] Schneider, Fred B., eds. "Trust in Cyberspace," National Research Council (1999). [http://
1571 www.nap.edu/readingroom/books/trust/](http://www.nap.edu/readingroom/books/trust/)
- 1572 [WooLam92] Thomas Y. C. Woo Simon S. Lam "Authentication for Distributed Systems," (January, 1992). IEEE
1573 Computer Society IEEE Computer (Vol. 25, No. 1), pp. 39-52 [http://doi.ieeecomputersociety.org/
1574 10.1109/2.108052](http://doi.ieeecomputersociety.org/10.1109/2.108052)
- 1575 [wss-sms] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (January, 2004). "Web
1576 Services Security: SOAP Message Security," OASIS Standard V1.0 [OASIS 200401], Organization for the
1577 Advancement of Structured Information Standards [http://docs.oasis-open.org/wss/2004/01/oasis-200401-
1578 wss-soap-message-security-1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)
- 1579 [wss-saml11] Monzillo, Ronald, Kaler, Chris, Nadalin, Anthony, Hallam-Baker, Phillip, eds. (June 28, 2005). Organ-
1580 ization for the Advancement of Structured Information Standards [http://www.oasis-open.org/committees/
1581 download.php/13405/wss-v1.1-spec-pr-SAMLTokenProfile-01.pdf](http://www.oasis-open.org/committees/download.php/13405/wss-v1.1-spec-pr-SAMLTokenProfile-01.pdf) "Web Services Security: SAML Token
1582 Profile 1.1," OASIS Public Review Draft 01,
- 1583 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).
1584 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consorti-
1585 um <http://www.w3.org/TR/2004/REC-xml-20040204>

1586 A. Listing of Simple Authentication and Security Layer (SASL) Mecha- 1587 nisms

1588 Ref: [SASLReg]

1589 Note

1590 The file listed below IS SUBJECT TO CHANGE! It is presented here as non-normative background infor-
1591 mation only. Implementers and deployers should always retrieve a fresh copy of this file from [IANA].

1592

1593 SIMPLE AUTHENTICATION AND SECURITY LAYER (SASL) MECHANISMS

1594 -----

1595

1596 (last updated 15 May 2006)

1597

1598 The Simple Authentication and Security Layer (SASL) [RFC-ietf-sasl-rfc2222bis-15.txt] is a
1599 method for adding authentication support to connection-based
1600 protocols. To use this specification, a protocol includes a command
1601 for identifying and authenticating a user to a server and for
1602 optionally negotiating a security layer for subsequent protocol
1603 interactions. The command has a required argument identifying a SASL
1604 mechanism.

1605

1606 SASL mechanisms are named by strings, from 1 to 20 characters in
1607 length, consisting of upper-case letters, digits, hyphens, and/or
1608 underscores. SASL mechanism names must be registered with the IANA.

1609 Procedures for registering new SASL mechanisms are described in
1610 RFC-ietf-sasl-rfc2222bis-15.txt.

1611

1612 Registration Procedures:

1613 First Come First Serve for Mechanisms

1614 Expert Review with Mailing List for Family Name Registrations

1615

1616 MECHANISMS	1616 USAGE	1616 REFERENCE	1616 OWNER
1617 -----	1617 -----	1617 -----	1617 -----
1618 KERBEROS_V4	1618 OBSOLETE	1618 [RFC2222]	1618 IESG <iesg@ietf.org>
1620 GSSAPI	1620 COMMON	1620 [RFC2222]	1620 IESG <iesg@ietf.org>
1622 SKEY	1622 OBSOLETE	1622 [RFC2444]	1622 IESG <iesg@ietf.org>
1624 EXTERNAL	1624 COMMON	1624 [RFC-ietf-sasl-rfc2222bis-15.txt]	1624 IESG <iesg@ietf.org>
1626 CRAM-MD5	1626 LIMITED	1626 [RFC2195]	1626 IESG <iesg@ietf.org>
1628 ANONYMOUS	1628 COMMON	1628 [RFC-ietf-sasl-anon-05.txt]	1628 IESG <iesg@ietf.org>
1630 OTP	1630 COMMON	1630 [RFC2444]	1630 IESG <iesg@ietf.org>
1632 GSS-SPNEGO	1632 LIMITED	1632 [Leach]	1632 Paul Leach <paulle@microsoft.com>
1634 PLAIN	1634 COMMON	1634 [RFC2595]	1634 IESG <iesg@ietf.org>
1636 SECURID	1636 COMMON	1636 [RFC2808]	1636 Magnus Nystrom <magnus@rsasecurity.com>
1638 NTLM	1638 LIMITED	1638 [Leach]	1638 Paul Leach <paulle@microsoft.com>
1640 NMAS_LOGIN	1640 LIMITED	1640 [Gayman]	1640 Mark G. Gayman <mgayman@novell.com>
1642 NMAS_AUTHEN	1642 LIMITED	1642 [Gayman]	1642 Mark G. Gayman <mgayman@novell.com>
1644 DIGEST-MD5	1644 COMMON	1644 [RFC2831]	1644 IESG <iesg@ietf.org>
1646 9798-U-RSA-SHA1-ENC	1646 COMMON	1646 [RFC3163]	1646 robert.zuccherato@entrust.com

1647
1648 9798-M-RSA-SHA1-ENC COMMON [RFC3163] robert.zuccherato@entrust.com
1649
1650 9798-U-DSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com
1651
1652 9798-M-DSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com
1653
1654 9798-U-ECDSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com
1655
1656 9798-M-ECDSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com
1657
1658 KERBEROS_V5 COMMON [Josefsson] Simon Josefsson <simon@josefsson.org>
1659
1660 NMAS-SAMBA-AUTH LIMITED [Brimhall] Vince Brimhall <vbrimhall@novell.com>
1661

1662

1663 References

1664 -----

1665 [RFC2195] Klensin, J., Catoe, R., Krumviede, P. "IMAP/POP AUTHorize
1666 Extension for Simple Challenge/Response," RFC 2195, MCI,
1667 September 1997. |
1668
1669 [RFC2222] J. Myers, "Simple Authentication and Security Layer (SASL)," |
1670 RFC 2222, October 1997. |
1671
1672 [RFC2444] Newman, C., "The One-Time-Password SASL Mechanism," RFC |
1673 2444, October 1998. |
1674
1675 [RFC2595] Newman, C., "Using TLS with IMAP, POP3 and ACAP," RFC 2595, |
1676 Innosoft, June 1999. |
1677
1678 [RFC2808] Nystrom, M., "The SecurID(r) SASL Mechanism," RFC 2808, |
1679 April 2000. |
1680
1681 [RFC2831] Leach, P. and C. Newman, "Using Digest Authentication as a |
1682 SASL Mechanism," RFC 2831, May 2000. |
1683
1684 [RFC3163] R. Zuccherato and M. Nystrom, "ISO/IEC 9798-3 Authentication |
1685 SASL Mechanism," RFC 3163, August 2001. |
1686
1687 [RFC-ietf-sasl-anon-05.txt]
1688 K. Zeilenga, Ed., "The Anonymous SASL Mechanism," RFC XXXX, |
1689 Month Year. |
1690
1691 [RFC-ietf-sasl-rfc2222bis-15.txt]
1692 A. Melnikov and K. Zeilenga, "Simple Authentication and Security |
1693 Layer (SASL)," RFC XXXX, Month Year. |
1694

1695 People

1696 -----

1697
1698 [Brimhall] Vince Brimhall, <vbrimhall@novell.com>, April 2004.
1699
1700 [Gayman] Mark G. Gayman, <mgayman@novell.com>, September 2000.
1701
1702 [Josefsson] Simon Josefsson, <simon@josefsson.org>, January 2004.
1703
1704 [Leach] Paul Leach, <paulle@microsoft.com>, December 1998, June 2000.
1705
1706 []

1707 B. Password Transformations

1708 This section defines a number of password transformations.

1709 1. Truncation

1710 The `urn:liberty:sa:pw:truncate` transformation instructs processors to remove all (Unicode abstract) subse-
1711 quent characters after a given number of characters have been obtained (from the user). Subsequent processing MUST
1712 take only the given number of characters as input. The number of characters that shall remain is given in a
1713 `<Parameter>` element with name "length".

```
1714  
1715 <Transform name="urn:liberty:sa:pw:truncate">  
1716     <Parameter name="length">8</Parameter>  
1717 </Transform>  
1718
```

1719 **Figure B.1. Example of truncation transformation**

1720 2. Lowercase

1721 The `urn:liberty:sa:pw:lowercase` transformation instructs processors to replace all uppercase characters with
1722 lowercase characters. Characters that do not have case must remain unchanged. This transformation has no parameters.
1723 Note that the "case" of the abstract Unicode character is decisive, *i.e.*, only characters that have the `UPPERCASE` property
1724 should be replaced with equivalent characters with the `LOWERCASE` property. This mapping from `UPPERCASE` to
1725 `lowercase` should confirm to the relevant sections (*e.g.*, 4.2) of [Unicode].

```
1726  
1727 <Transform name="urn:liberty:sa:pw:lowercase" />  
1728
```

1729 **Figure B.2. Example of lowercase transformation**

1730 3. Uppercase

1731 The `urn:liberty:sa:pw:uppercase` transformation instructs processors to replace all lowercase characters with
1732 uppercase characters. Characters that do not have case must remain unchanged. This transformation has no parameters.
1733 Note that the "case" of the abstract Unicode character is decisive, *i.e.*, only characters that have the `LOWERCASE` property
1734 should be replaced with equivalent characters with the `UPPERCASE` property. This mapping from `lowercase` to
1735 `UPPERCASE` should confirm to the relevant sections (*e.g.*, 4.2) of [Unicode].

```
1736  
1737 <Transform name="urn:liberty:sa:pw:uppercase" />  
1738
```

1739 **Figure B.3. Example of uppercase transformation**

1740 4. Select

1741 The `urn:liberty:sa:pw:select` transformation instructs processors to remove all characters except those speci-
1742 fied in the "allowed" parameter. Note that the allowed characters refer to abstract Unicode characters. In the message
1743 that contains the `<Transform>` element these characters are encoded with the same encoding as used for the xml
1744 document that contains the message (usually UTF-8).

```
1745
1746 <Transform name="urn:liberty:sa:pw:select">
1747   <Parameter name="allowed">0123456789abcdefghijklmnopqrstvwxyz</Parameter>
1748 </Transform>
1749
```

1750

Figure B.4. Example of select transformation

1751 C. liberty-idwsf-authn-svc-v2.0.xsd Schema Listing

```

1752 <?xml version="1.0" encoding="UTF-8"?>
1753
1754 <xs:schema
1755     targetNamespace="urn:liberty:sa:2006-08"
1756     xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1757     xmlns:sa="urn:liberty:sa:2006-08"
1758     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1759     xmlns:samlp2="urn:oasis:names:tc:SAML:2.0:protocol"
1760     xmlns:wsa="http://www.w3.org/2005/08/addressing"
1761     xmlns:lu="urn:liberty:util:2006-08"
1762     xmlns="urn:liberty:sa:2006-08"
1763     elementFormDefault="qualified"
1764     attributeFormDefault="unqualified"
1765     version="09"
1766     >
1767
1768 <!-- Filename: lib_arch_authn_svc.xsd -->
1769 <!-- $Id: lib_arch_authn_svc.xsd 3793 2006-07-28 02:44:20Z dchampagne $ -->
1770 <!-- Author: Jeff Hodges -->
1771 <!-- Last editor: $Author: dchampagne $ -->
1772 <!-- $Date: 2006-07-27 22:44:20 0400 (Thu, 27 Jul 2006) $ -->
1773 <!-- $Revision: 3793 $ -->
1774
1775 <xs:import
1776     namespace="http://www.w3.org/2005/08/addressing"
1777     schemaLocation="ws-addr-1.0.xsd"/>
1778
1779 <xs:import
1780     namespace="urn:oasis:names:tc:SAML:2.0:protocol"
1781     schemaLocation="saml-schema-protocol-2.0.xsd"/>
1782
1783 <xs:import namespace="urn:liberty:util:2006-08"
1784     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
1785
1786
1787
1788 <xs:annotation>
1789 <xs:documentation>
1790     Liberty ID-WSF Authentication Service XSD
1791 </xs:documentation>
1792 <xs:documentation>
1793     The source code in this XSD file was excerpted verbatim from:
1794
1795     Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification
1796     Version 2.0
1797     30 July, 2006
1798
1799     Copyright (c) 2006 Liberty Alliance participants,
1800     see http://www.projectliberty.org/specs/idwsf_2_0_copyrights.php
1801 </xs:documentation>
1802 </xs:annotation>
1803
1804
1805 <!-- SASLRequest and SASLResponse ID-* messages -->
1806
1807 <xs:element name="SASLRequest">
1808     <xs:complexType>
1809         <xs:sequence>
1810
1811             <xs:element name="Data" minOccurs="0">
1812                 <xs:complexType>
1813                     <xs:simpleContent>
1814                         <xs:extension base="xs:base64Binary"/>
1815                     </xs:simpleContent>
1816                 </xs:complexType>

```

```

1817         </xs:element>
1818
1819         <xs:element ref="samlp2:RequestedAuthnContext" minOccurs="0"/>
1820
1821         <xs:element name="Extensions" minOccurs="0">
1822             <xs:complexType>
1823                 <xs:sequence>
1824                     <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
1825                 </xs:sequence>
1826             </xs:complexType>
1827         </xs:element>
1828
1829     </xs:sequence>
1830
1831     <xs:attribute name="mechanism"
1832                 type="xs:string"
1833                 use="required"/>
1834
1835     <xs:attribute name="authzID"
1836                 type="xs:string"
1837                 use="optional"/>
1838
1839     <xs:attribute name="advisoryAuthnID"
1840                 type="xs:string"
1841                 use="optional"/>
1842
1843     <xs:anyAttribute namespace="##other" processContents="lax"/>
1844
1845 </xs:complexType>
1846 </xs:element>
1847
1848 <xs:element name="SASLResponse">
1849     <xs:complexType>
1850         <xs:sequence>
1851
1852             <xs:element ref="lu:Status"/>
1853
1854             <xs:element ref="PasswordTransforms" minOccurs="0"/>
1855
1856             <xs:element name="Data" minOccurs="0">
1857                 <xs:complexType>
1858                     <xs:simpleContent>
1859                         <xs:extension base="xs:base64Binary"/>
1860                     </xs:simpleContent>
1861                 </xs:complexType>
1862             </xs:element>
1863
1864             <!-- ID-WSF EPRs -->
1865             <xs:element ref="wsa:EndpointReference"
1866                         minOccurs="0"
1867                         maxOccurs="unbounded"/>
1868
1869         </xs:sequence>
1870
1871         <xs:attribute name="serverMechanism"
1872                     type="xs:string"
1873                     use="optional"/>
1874
1875         <xs:anyAttribute namespace="##other" processContents="lax"/>
1876     </xs:complexType>
1877 </xs:element>
1878
1879 <!-- Password Transformations -->
1880
1881 <xs:element name="PasswordTransforms">

```

```
1884
1885     <xs:annotation>
1886       <xs:documentation>
1887         Contains ordered list of sequential password transformations
1888       </xs:documentation>
1889     </xs:annotation>
1890
1891     <xs:complexType>
1892       <xs:sequence>
1893
1894         <xs:element name="Transform" maxOccurs="unbounded">
1895           <xs:complexType>
1896             <xs:sequence>
1897
1898               <xs:element name="Parameter"
1899                 minOccurs="0"
1900                 maxOccurs="unbounded">
1901                 <xs:complexType>
1902                   <xs:simpleContent>
1903                     <xs:extension base="xs:string">
1904                       <xs:attribute name="name"
1905                         type="xs:string"
1906                         use="required"/>
1907                     </xs:extension>
1908                   </xs:simpleContent>
1909                 </xs:complexType>
1910               </xs:element>
1911             </xs:sequence>
1912           </xs:complexType>
1913           <xs:attribute name="name"
1914             type="xs:anyURI"
1915             use="required"/>
1916           <xs:anyAttribute namespace="##other" processContents="lax"/>
1917         </xs:element>
1918       </xs:sequence>
1919     </xs:complexType>
1920   </xs:element>
1921 </xs:sequence>
1922 </xs:complexType>
1923 </xs:element>
1924 </xs:element>
1925
1926 </xs:schema>
```

1927 D. liberty-idwsf-idmapping-svc-v2.0.xsd Schema Listing

```

1928 <?xml version="1.0" encoding="UTF-8"?>
1929
1930 <xs:schema
1931     targetNamespace="urn:liberty:ims:2006-08"
1932     xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1933     xmlns:ims="urn:liberty:ims:2006-08"
1934     xmlns:sec="urn:liberty:security:2006-08"
1935     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1936     xmlns:lu="urn:liberty:util:2006-08"
1937     xmlns="urn:liberty:ims:2006-08"
1938     elementFormDefault="qualified"
1939     attributeFormDefault="unqualified"
1940     >
1941
1942 <!-- Filename: liberty_idwsf_idmapping_svc_v2.0.xsd -->
1943 <!-- $Id: lib_arch_idmapping_svc.xsd 3793 2006-07-28 02:44:20Z dchampagne $ -->
1944 <!-- Author: Scott Cantor -->
1945 <!-- Last editor: $Author: dchampagne $ -->
1946 <!-- $Date: 2006-07-27 22:44:20 0400 (Thu, 27 Jul 2006) $ -->
1947 <!-- $Revision: 3793 $ -->
1948
1949 <xs:import
1950     namespace="urn:liberty:security:2006-08"
1951     schemaLocation="liberty-idwsf-security-mechanisms-v2.0.xsd"/>
1952
1953 <xs:import namespace="urn:liberty:util:2006-08"
1954     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
1955
1956 <xs:annotation>
1957 <xs:documentation>
1958     Liberty ID-WSF Identity Mapping Service XSD
1959 </xs:documentation>
1960 <xs:documentation>
1961     The source code in this XSD file was excerpted verbatim from:
1962
1963     Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification
1964     Version 2.0
1965     30 July, 2006
1966
1967     Copyright (c) 2006 Liberty Alliance participants,
1968     see http://www.projectliberty.org/specs/idwsf_2_0_copyrights.php
1969 </xs:documentation>
1970 </xs:annotation>
1971
1972 <xs:element name="MappingInput" type="MappingInputType"/>
1973 <xs:complexType name="MappingInputType">
1974     <xs:sequence>
1975         <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
1976         <xs:element ref="sec:Token" minOccurs="0"/>
1977     </xs:sequence>
1978     <xs:attribute name="reqID" type="lu:IDType" use="optional"/>
1979 </xs:complexType>
1980
1981 <xs:element name="MappingOutput" type="MappingOutputType"/>
1982 <xs:complexType name="MappingOutputType">
1983     <xs:sequence>
1984         <xs:element ref="sec:Token"/>
1985     </xs:sequence>
1986     <xs:attribute name="reqRef" type="lu:IDReferenceType" use="optional"/>
1987 </xs:complexType>
1988
1989 <xs:element name="IdentityMappingRequest" type="IdentityMappingRequestType"/>
1990 <xs:complexType name="IdentityMappingRequestType">
1991     <xs:sequence>
1992         <xs:element ref="MappingInput" maxOccurs="unbounded"/>

```

```
1993         </xs:sequence>
1994         <xs:anyAttribute namespace="##other" processContents="lax"/>
1995     </xs:complexType>
1996
1997     <xs:element name="IdentityMappingResponse" type="IdentityMappingResponseType"/>
1998     <xs:complexType name="IdentityMappingResponseType">
1999         <xs:sequence>
2000             <xs:element ref="lu:Status"/>
2001             <xs:element ref="MappingOutput" minOccurs="0" maxOccurs="unbounded"/>
2002         </xs:sequence>
2003         <xs:anyAttribute namespace="##other" processContents="lax"/>
2004     </xs:complexType>
2005
2006 </xs:schema>
```

2007 E. liberty-idwsf-utility-v2.0.xsd Schema Listing

```

2008 <?xml version="1.0" encoding="UTF-8"?>
2009 <xs:schema targetNamespace="urn:liberty:util:2006-08"
2010   xmlns:xs="http://www.w3.org/2001/XMLSchema"
2011   xmlns="urn:liberty:util:2006-08"
2012   elementFormDefault="qualified"
2013   attributeFormDefault="unqualified"
2014   version="2.0-03">
2015
2016   <xs:annotation>
2017     <xs:documentation>
2018       Liberty Alliance Project utility schema. A collection of common
2019       Identity Web Services Framework (ID-WSF) elements and types.
2020       This schema is intended for use in ID-WSF schemas.
2021
2022       This version: 2006-08
2023
2024       Copyright (c) 2006 Liberty Alliance participants, see
2025       http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2026     </xs:documentation>
2027   </xs:annotation>
2028   <xs:simpleType name="IDType">
2029     <xs:annotation>
2030       <xs:documentation>
2031         This type should be used to provide IDs to components
2032         that have IDs that may not be scoped within the local
2033         xml instance document.
2034       </xs:documentation>
2035     </xs:annotation>
2036     <xs:restriction base="xs:string"/>
2037   </xs:simpleType>
2038   <xs:simpleType name="IDReferenceType">
2039     <xs:annotation>
2040       <xs:documentation>
2041         This type can be used when referring to elements that are
2042         identified using an IDType.
2043       </xs:documentation>
2044     </xs:annotation>
2045     <xs:restriction base="xs:string"/>
2046   </xs:simpleType>
2047   <xs:attribute name="itemID" type="IDType"/>
2048   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2049   <xs:complexType name="StatusType">
2050     <xs:annotation>
2051       <xs:documentation>
2052         A type that may be used for status codes.
2053       </xs:documentation>
2054     </xs:annotation>
2055     <xs:sequence>
2056       <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
2057     </xs:sequence>
2058     <xs:attribute name="code" type="xs:string" use="required"/>
2059     <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
2060     <xs:attribute name="comment" type="xs:string" use="optional"/>
2061   </xs:complexType>
2062
2063   <xs:element name="Status" type="StatusType">
2064     <xs:annotation>
2065       <xs:documentation>
2066         A standard Status type
2067       </xs:documentation>
2068     </xs:annotation>
2069   </xs:element>
2070
2071   <xs:complexType name="ResponseType">
2072     <xs:sequence>

```

```
2073         <xs:element ref="Status"          minOccurs="1" maxOccurs="1"/>
2074         <xs:element ref="Extension"       minOccurs="0" maxOccurs="unbounded"/>
2075     </xs:sequence>
2076     <xs:attribute ref="itemIDRef" use="optional"/>
2077     <xs:anyAttribute namespace="##other" processContents="lax"/>
2078 </xs:complexType>
2079 <xs:element name="TestResult" type="TestResultType"/>
2080 <xs:complexType name="TestResultType">
2081     <xs:simpleContent>
2082         <xs:extension base="xs:boolean">
2083             <xs:attribute ref="itemIDRef" use="required"/>
2084         </xs:extension>
2085     </xs:simpleContent>
2086 </xs:complexType>
2087 <xs:complexType name="EmptyType">
2088     <xs:annotation>
2089         <xs:documentation> This type may be used to create an empty element </xs:documentation>
2090     </xs:annotation>
2091     <xs:complexContent>
2092         <xs:restriction base="xs:anyType"/>
2093     </xs:complexContent>
2094 </xs:complexType>
2095 <xs:element name="Extension" type="extensionType">
2096     <xs:annotation>
2097         <xs:documentation>
2098             An element that contains arbitrary content extensions
2099             from other namespaces
2100         </xs:documentation>
2101     </xs:annotation>
2102 </xs:element>
2103 <xs:complexType name="extensionType">
2104     <xs:annotation>
2105         <xs:documentation>
2106             A type for arbitrary content extensions from other namespaces
2107         </xs:documentation>
2108     </xs:annotation>
2109     <xs:sequence>
2110         <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
2111     </xs:sequence>
2112 </xs:complexType>
2113 </xs:schema>
```

2114 F. liberty-idwsf-authn-svc-v2.0.wsdl WSDL Listing

```

2115 <?xml version="1.0"?>
2116 <definitions name="AuthenticationService"
2117     targetNamespace="urn:liberty:sa:2006-08"
2118     xmlns:tns="urn:liberty:sa:2006-08"
2119     xmlns:xs="http://www.w3.org/2001/XMLSchema"
2120     xmlns:S="http://schemas.xmlsoap.org/wsdl/soap/"
2121     xmlns="http://schemas.xmlsoap.org/wsdl/"
2122     xmlns:sa="urn:liberty:sa:2006-08"
2123     xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
2124     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2125     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2126     xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
2127         http://schemas.xmlsoap.org/wsdl/
2128         http://www.w3.org/2006/02/addressing/wsdl
2129         http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
2130
2131
2132
2133     <xsd:documentation>
2134
2135         XML Authentication Schema from Liberty ID-WSF Authentication,
2136         Single Sign On, and Identity Mapping Services Specification
2137
2138         ### NOTICE ###
2139
2140         Copyright (c) 2006 Liberty Alliance participants, see
2141         http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2142
2143     </xsd:documentation>
2144
2145     <types>
2146         <xs:schema
2147             <xs:import namespace="urn:liberty:sa:2006-08"
2148                 schemaLocation="liberty-idwsf-authn-svc-v2.0.xsd"/>
2149         </xs:schema>
2150     </types>
2151
2152     <message name="AuthenticationSoapRequest">
2153         <part name="parameters" element="sa:SASLRequest"/>
2154     </message>
2155
2156     <message name="AuthenticationSoapResponse">
2157         <part name="parameters" element="sa:SASLResponse"/>
2158     </message>
2159
2160     <portType name="AuthServicePortType">
2161         <operation name="Authenticate">
2162             <input message="sa:AuthenticationSoapRequest"
2163                 wsaw:Action="urn:liberty:sa:2006-08:SASLRequest"/>
2164             <output message="sa:AuthenticationSoapResponse"
2165                 wsaw:Action="urn:liberty:sa:2006-08:SASLResponse"/>
2166         </operation>
2167     </portType>
2168
2169     <binding name="AuthenticationSoapBinding" type="sa:AuthServicePortType">
2170         <S:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2171         <operation name="Authenticate">
2172             <S:operation soapAction="urn:liberty:sa:2006-08:Authenticate" style="document"/>
2173             <input>
2174                 <S:body use="literal"/>
2175             </input>
2176             <output>
2177                 <S:body use="literal"/>
2178             </output>
2179         </operation>
2180     </binding>
2181
2182     <service name="AuthenticationService">
2183         <port name="AuthServicePortType" binding="sa:AuthenticationSoapBinding">

```

```
2180         <S:address location="http://example.com/authentication"/>
2181         </port>
2182     </service>
2183 </definitions>
```

2184 G. liberty-idwsf-ssosvc-v2.0.wsdl WSDL Listing

```

2185 <?xml version="1.0"?>
2186 <definitions name="AuthenticationService"
2187     targetNamespace="urn:liberty:ssos:2006-08"
2188     xmlns:tns="urn:liberty:ssos:2006-08"
2189     xmlns:xs="http://www.w3.org/2001/XMLSchema"
2190     xmlns:S="http://schemas.xmlsoap.org/wsdl/soap/"
2191     xmlns="http://schemas.xmlsoap.org/wsdl/"
2192     xmlns:ssos="urn:liberty:ssos:2006-08"
2193     xmlns:samlp2="urn:oasis:names:tc:SAML:2.0:protocol"
2194     xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
2195     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2196     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2197     xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
2198         http://schemas.xmlsoap.org/wsdl/
2199         http://www.w3.org/2006/02/addressing/wsdl
2200         http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
2201
2202
2203
2204     <xsd:documentation>
2205
2206         XML SSO Schema from Liberty ID-WSF Authentication, Single
2207         Sign-On, and Identity Mapping Services Specification
2208
2209         ### NOTICE ###
2210
2211         Copyright (c) 2006 Liberty Alliance participants, see
2212         http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2213
2214     </xsd:documentation>
2215
2216     <types>
2217         <xs:schema
2218             <xs:import namespace="urn:oasis:names:tc:SAML:2.0:protocol"
2219                 schemaLocation="saml-schema-protocol-2.0.xsd" />
2220         </xs:schema>
2221
2222     </types>
2223
2224     <message name="SSOSoapRequest">
2225         <part name="parameters" element="samlp2:AuthnRequest" />
2226     </message>
2227
2228     <message name="SSOSoapResponse">
2229         <part name="parameters" element="samlp2:Response" />
2230     </message>
2231
2232     <portType name="SSOSPortType">
2233         <operation name="SingleSignOn">
2234             <input message="ssos:SSOSoapRequest"
2235                 wsaw:Action="urn:liberty:ssos:2006-08:AuthnRequest" />
2236             <output message="ssos:SSOSoapResponse"
2237                 wsaw:Action="urn:liberty:ssos:2006-08:Response" />
2238         </operation>
2239     </portType>
2240
2241     <binding name="SSOSSoapBinding" type="ssos:SSOSPortType">
2242         <S:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2243         <operation name="SingleSignOn">
2244             <S:operation soapAction="urn:liberty:ssos:2006-08:SingleSignOn" style="document"/>
2245             <input>
2246                 <S:body use="literal"/>
2247             </input>
2248             <output>
2249                 <S:body use="literal"/>
2250             </output>
2251         </operation>
2252     </binding>
2253
2254     <service name="SSOService">

```

```
2250         <port name="SSOSPortType" binding="ssos:SSOSSoapBinding">
2251             <S:address location="http://example.com/idmapping"/>
2252         </port>
2253     </service>
2254 </definitions>
```

2255 H. liberty-idwsf-idmapping-svc-v2.0.wsdl WSDL Listing

```

2256 <?xml version="1.0"?>
2257 <definitions name="AuthenticationService"
2258     targetNamespace="urn:liberty:ims:2006-08"
2259     xmlns:tns="urn:liberty:ims:2006-08"
2260     xmlns:xs="http://www.w3.org/2001/XMLSchema"
2261     xmlns:S="http://schemas.xmlsoap.org/wsdl/soap/"
2262     xmlns="http://schemas.xmlsoap.org/wsdl/"
2263     xmlns:ims="urn:liberty:ims:2006-08"
2264     xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
2265     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2266     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2267     xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
2268         http://schemas.xmlsoap.org/wsdl/
2269         http://www.w3.org/2006/02/addressing/wsdl
2270         http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
2271
2272
2273
2274     <xsd:documentation>
2275
2276         XML ID Mapping Schema from Liberty ID-WSF Authentication, Single
2277         Sign-On, and Identity Mapping Services Specification
2278
2279         ### NOTICE ###
2280
2281         Copyright (c) 2006 Liberty Alliance participants, see
2282         http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2283
2284     </xsd:documentation>
2285
2286     <types>
2287         <xs:schema
2288             <xs:import namespace="urn:liberty:ims:2006-08"
2289                 schemaLocation="liberty-idwsf-idmapping-svc-v2.0.xsd"/>schemaLocation="liberty-idwsf-idmapping
2290         </xs:schema>
2291     </types>
2292
2293     <message name="IdentityMappingSoapRequest">
2294         <part name="parameters" element="ims:IdentityMappingRequest"/>element="ims:IdentityMappingRequest"/>
2295     </message>
2296
2297     <message name="IdentityMappingSoapResponse">
2298         <part name="parameters" element="ims:IdentityMappingResponse"/>element="ims:IdentityMappingResponse"/>
2299     </message>
2300
2301     <portType name="IdMappingPortType">
2302         <operation name="IdentityMapping">
2303             <input message="ims:IdentityMappingSoapRequest"
2304                 wsaw:Action="urn:liberty:ims:2006-08:IdentityMappingRequest"/>
2305             <output message="ims:IdentityMappingSoapResponse"
2306                 wsaw:Action="urn:liberty:ims:2006-08:IdentityMappingResponse"/>
2307         </operation>
2308     </portType>
2309
2310     <binding name="IdMappingSoapBinding" type="ims:IdMappingPortType">
2311         <S:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2312         <operation name="IdentityMapping">
2313             <S:operation soapAction="urn:liberty:ims:2006-08:IdentityMapping" style="document"/>
2314             <input>
2315                 <S:body use="literal"/>use="literal"/>
2316             </input>
2317             <output>
2318                 <S:body use="literal"/>use="literal"/>
2319             </output>
2320         </operation>
2321     </binding>
2322
2323     <service name="IdMappingService">
2324         <port name="IdMappingPortType" binding="ims:IdMappingSoapBinding">

```

```
2321         <S:address location="http://example.com/idmapping"/>
2322         </port>
2323     </service>
2324 </definitions>
```