



# Liberty ID-WSF Authentication Service and Single Sign-On Service Specification

Version: v1.1

## Editors:

Jeff Hodges, Sun Microsystems, Inc.  
Robert Aarts, Nokia Corporation

## Contributors:

Conor Cahill, AOL Time Warner, Inc.  
Gary Ellison, Sun Microsystems, Inc.  
Greg Whitehead, Trustgenix, Inc.

## 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. Next, it 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. Finally, it defines an ID-WSF Single Sign-On Service. This service provides authentication assertions to Web Service Consumers via a profile of the ID-FF Single Sign-On Protocol, enabling Web Service Consumers to interact with ID-FF-based, or other, Service Providers.

**Filename:** liberty-idwsf-authn-svc-v1.1.pdf

1 **Notice**

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

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

15 Copyright © 2004-2005 ADAE; Adobe Systems; America Online, Inc.; American Express Company; Avatier  
16 Corporation; Axalto; Bank of America Corporation; BIPAC; Computer Associates International, Inc.; DataPower  
17 Technology, Inc.; Diversinet Corp.; Enosis Group LLC; Entrust, Inc.; Epok, Inc.; Ericsson; Fidelity Investments;  
18 Forum Systems, Inc. ; France Telecom; Gamefederation; Gemplus; General Motors; Giesecke & Devrient GmbH;  
19 Hewlett-Packard Company; IBM Corporation; Intel Corporation; Intuit Inc.; Kantega; Kayak Interactive; MasterCard  
20 International; Mobile Telephone Networks (Pty) Ltd; NEC Corporation; Netegrity, Inc.; NeuStar, Inc.; Nippon  
21 Telegraph and Telephone Corporation; Nokia Corporation; Novell, Inc.; NTT DoCoMo, Inc.; OpenNetwork; Oracle  
22 Corporation; Ping Identity Corporation; Royal Mail Group plc; RSA Security Inc.; SAP AG; Senforce; Sharp  
23 Laboratories of America; Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Telefonica Moviles, S.A.;  
24 Trusted Network Technologies.; Trustgenix; UTI; VeriSign, Inc.; Vodafone Group Plc. All rights reserved.

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

## 31 **Contents**

32	1. Introduction .....	4
33	2. Notation and Conventions .....	5
34	3. Terminology .....	6
35	4. Authentication Protocol .....	9
36	5. Authentication Service .....	20
37	6. Single Sign-On Service .....	26
38	7. Password Transformations: The PasswordTransforms Element .....	29
39	8. Acknowledgments .....	31
40	References .....	32
41	A. Listing of Simple Authentication and Security Layer (SASL) Mechanisms .....	35
42	B. Password Transformations .....	37
43	1. Truncation .....	37
44	2. Lowercase .....	37
45	3. Uppercase .....	37
46	4. Select .....	37
47	C. lib-arch-authn-svc.xsd Schema Listing .....	38
48	D. lib-arch-iwsf-utility.xsd Schema Listing .....	41

## 49 **1. Introduction**

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

55 This specification defines how to perform *general identity authentication* [\[WooLam92\]](#), also known as *peer entity*  
56 *authentication* [\[RFC2828\]](#), over SOAP, in the context of the Liberty Identity Web Services Framework (ID-WSF)  
57 [\[LibertyIDWSFOverview\]](#). Rather than specify the particulars of one or more *authentication mechanisms* directly in  
58 this specification, we profile the Simple Authentication and Security Layer (SASL) framework [\[RFC2222\]](#).

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

66 This specification is organized as follows. First, it defines the ID-WSF Authentication Protocol. Next, it defines  
67 an ID-WSF Authentication Service Identity Providers may offer, which is based on the authentication protocol.  
68 This authentication service enables Web Services Consumers and/or Liberty-enabled User Agents or Devices to  
69 authenticate with Identity Providers using various authentication mechanisms and obtain ID-WSF security tokens.  
70 Finally, it defines an ID-WSF Single Sign-On Service. This service provides authentication assertions to Web Services  
71 Consumers via a profile of the ID-FF Single Sign-On Protocol, enabling Web Services Consumers to interact with ID-  
72 FF-based Service Providers.



## 87 3. Terminology

88 This section defines key terminology used in this specification. Definitions for these, as well as other Liberty-specific  
89 terms, may also be found in [\[LibertyGlossary\]](#). Note that the definition of some terms below differ slightly from the  
90 definition given in [\[LibertyGlossary\]](#). For example see the definitions for *client* and *server*. This is because in such  
91 cases, the definition given in [\[LibertyGlossary\]](#) is a more general one, and the definition given here is a more narrow  
92 one, specific to the context of this specification. See also [\[RFC2828\]](#) for overall definitions of security-related terms,  
93 in general. Other specific references are also cited below.

94 Terminology

95 authentication *Authentication* is the process of confirming a *system entity*'s asserted *identity* with a  
96 specified, or understood, level of confidence [\[TrustInCyberspace\]](#).

97 authentication assertion A *SAML assertion* typically consisting of a single `<AuthenticationStatement>`.  
98 The assertion issuer is stating that the subject of the assertion authenticated with it at  
99 some point in time. Assertions are typically time-limited [\[SAMLCore11\]](#).

100 authentication exchange See *authentication protocol exchange*.

101 authentication mechanism An *authentication mechanism* is a particular, identifiable, process or technique that  
102 results in a confirmation of a *system entity*'s asserted identity with a specified, or  
103 understood, level of confidence.

104 authentication protocol exchange *Authentication protocol exchange* is the term used in [\[RFC2222\]](#) to refer to the  
105 sequence of messages exchanged between the *client* and *server* as specified and gov-  
106 erned by the particular *SASL mechanism* being employed to effect an act of *authentic-*  
107 *ation*.

108 authentication server The precise, specific *role* played by a *server* in the protocol message exchanges defined  
109 in this specification.

110 Authentication Service (AS) Short form of "ID-WSF Authentication Service". The AS is a discoverable ID-WSF  
111 service.

112 Authentication Service Consumer A *Web Service Consumer* (WSC) implementing the *client*-side of the ID-WSF  
113 Authentication Protocol (which is defined in this specification).

114 Authentication Service Provider (AS Provider) A *Web Service Provider* (WSP) implementing the *server*-side of  
115 the ID-WSF Authentication Service defined in this specification ([Section 5: Authentica-](#)  
116 [tion Service](#)).

117 client A *role* assumed by a *system entity* who either explicitly or implicitly initiates an  
118 authentication exchange [\[RFC2828\]](#). *Client* is implicitly defined in [\[RFC2222\]](#). Also  
119 known as a *SASL client*.

120 discoverable A *discoverable* "in principle" service is one having an *service type URI* assigned (this is  
121 typically in done in the specification defining the service). A discoverable "in practice"  
122 service is one that is registered in some discovery service instance.  
123 ID-WSF *services* are by definition discoverable "in principle" because such services are  
124 assigned a *service type URI* facilitating their registration in *Discovery Service* instances.

125 final SASL response The final `<SASLResponse>` message sent from the *server* to the *client* in an *authenti-*  
126 *cation exchange*.

---

127	initial response	A <a href="#">[RFC2222]</a> term referring to <i>authentication exchange data</i> sent by the <i>client</i> in the
128		<i>initial SASL request</i> . It is used by a subset of SASL mechanisms. See Section 5.1 of
129		<a href="#">[RFC2222]</a> .
130	initial SASL request	The initial <SASLRequest> message sent from the <i>client</i> to the <i>server</i> in an <i>authentication exchange</i> .
131		
132	(LUAD)-WSC	A <i>Web Service Consumer</i> (WSC), that may or may not also be a <i>Liberty-enabled User Agent or Device</i> .
133		
134	mechanism	A process or technique for achieving a result <a href="#">[Merriam-Webster]</a> .
135	message thread	A <i>message thread</i> is a synchronous exchange of messages in a request-response <i>MEP</i>
136		between two <i>SOAP nodes</i> . All the messages of a given message thread are "linked" via
137		each message's <Correlation> header block <code>refToMessageID</code> attribute value being
138		set, by the sender, from the previous successfully received message's <Correlation>
139		header block <code>messageID</code> attribute value.
140	requester	A <i>system entity</i> which sends a <i>service request</i> to a <i>provider</i> .
141	role	A function or part performed, especially in a particular operation or process <a href="#">[Merriam-</a>
142		<a href="#">Webster]</a> .
143	SASL mechanism	A <i>SASL mechanism</i> is an <i>authentication mechanism</i> that has been profiled for use in the
144		context of the <i>SASL framework</i> <a href="#">[RFC2222]</a> . See <a href="#">[RFC2444]</a> for a particular example of
145		profiling an existing authentication mechanism—one-time passwords <a href="#">[RFC2289]</a> —for
146		use in the SASL context. SASL mechanisms are "named"; Mechanism names are
147		listed in the column labeled as "MECHANISMS" in <a href="#">[SASLReg]</a> (a copy of this registry
148		document is reproduced in <a href="#">Appendix A</a> for informational convenience; implementors
149		should always fetch the most recent revision directly from <a href="#">[IANA]</a> ).
150	server	A <i>role</i> donned by a <i>system entity</i> who is intended to engage in defined exchanges with
151		<i>clients</i> . This term is implicitly defined in <a href="#">[RFC2222]</a> and in this specification is always
152		synonymous with <i>authentication server</i> .
153	Service Provider (SP)	(1)A <i>role</i> donned by <i>system entities</i> . In the Liberty architecture, <i>Service Providers</i>
154		interact with other system entities primarily via vanilla HTTP.
155		(2) From a Principal's perspective, a Service Provider is typically a website providing
156		services and/or goods.
157	SOAP header block	A <a href="#">[SOAPv1.2]</a> term meaning: An [element] used to delimit data that logically con-
158		stitutes a single computational unit within the SOAP header. In <a href="#">[SOAPv1.1]</a> these
159		are known as simply <i>SOAP headers</i> , or simply <i>headers</i> . This specification uses the
160		SOAPv1.2 terminology.
161	SOAP node	A <a href="#">[SOAPv1.2]</a> term describing <i>system entities</i> who are parties to SOAP-based message
162		exchanges that are, for purposes of this specification, also the ultimate destination of the
163		exchanged messages, i.e. <i>SOAP endpoints</i> . In <a href="#">[SOAPv1.1]</a> , SOAP nodes are referred
164		to as <i>SOAP endpoints</i> , or simply <i>endpoints</i> . This specification uses the SOAPv1.2
165		terminology.
166	system entity	An active element of a computer/network system. For example, an automated process
167		or set of processes, a subsystem, a person or group of persons that incorporates a distinct
168		set of functionality <a href="#">[SAMLGloss]</a> .

169	user identifier	<i>AKA user name or Principal.</i>
170	web service	Generically, a <i>service</i> defined in terms of an <i>XML</i> -based protocol, often transported over <i>SOAP</i> , and/or a service whose instances, and possibly data objects managed therein, are concisely addressable via <i>URIs</i> .
171		
172		
173		As specifically used in Liberty specifications, usually in terms of <i>WSCs</i> and <i>WSPs</i> ,
174		it means a web service that's defined in terms of the <i>ID</i> -* "stack", and thus utilizes
175		<a href="#">[LibertySOAPBinding]</a> , <a href="#">[LibertySecMech]</a> , and is "discoverable" <a href="#">[LibertyDisco]</a> .
176	Web Service Consumer	<i>A role donned by a system entity when it makes a request to a web service.</i>
177	Web Service Provider	<i>A role donned by a system entity when it provides a web service.</i>

## 178 4. Authentication Protocol

179 This section defines the ID-WSF Authentication Protocol. This protocol facilitates authentication between two ID-\*  
180 entities, and is a profile of SASL [RFC2222].

### 181 4.1. Conceptual Model

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

185 The authentication exchange is comprised of SOAP-bound ID-\* messages [LibertySOAPBinding], and can involve an  
186 arbitrary number of round trips, dictated by the particular SASL mechanism employed [RFC2222]. The WSC may  
187 have out-of-band knowledge of the server's supported SASL mechanisms, or it may send the server its own list of  
188 supported SASL mechanisms and allow the server to choose one from among them.

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

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

194 **Note:**

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

### 196 4.2. Schema Declarations

197 The XML schema [Schema1] normatively defined in this section is constituted in the XML Schema file:  
198 `lib-arch-authn-svc.xsd`, entitled " [Liberty ID-WSF Authentication Service XSD](#) " (see [Appendix C](#)).

199 In addition, the [Liberty ID-WSF Authentication Service XSD](#) explicitly includes, in the XML Schema sense, the  
200 Liberty ID-WSF Utility XSD file (see [Appendix D](#)), whose filename is: `lib-arch-iwsf-utility.xsd`.

### 201 4.3. SOAP Header Blocks and SOAP Binding

202 This specification does not define any SOAP header blocks. [Section 4.3.1](#), below, constitutes the SOAP binding  
203 statement for this specification.

#### 204 4.3.1. SOAP Binding

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

209 **Note:**

210 Implementations of this specification MUST use the `<sb:Correlation>` SOAP header block defined in  
211 [LibertySOAPBinding] to establish a *message thread* and thus correlate their authentication exchanges. See  
212 [Section 5.5: Authentication Service Interaction Example](#) for an example.

### 213 4.4. SASL Profile Particulars

214 The ID-WSF Authentication Protocol is based on SASL [RFC2222], and thus "profiles" SASL. Section 4 of  
215 [RFC2222] specifies SASL's "profiling requirements". This section of this specification addresses some particulars  
216 of profiling SASL that are not otherwise addressed in the sections defining the protocol messages (Section 4.6:  
217 Protocol Messages ), and their sequencing (Section 4.7: Sequencing of the Authentication Exchange ).

#### 218 4.4.1. SASL "Service Name"

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

#### 220 4.4.2. Composition of SASL Mechanism Names

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

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

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

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

### 230 4.5. Authentication Exchange Security

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

236 Thus it is RECOMMENDED that the authentication protocol exchange defined herein (Section 4.7: Sequencing of  
237 the Authentication Exchange ) be employed over a TLS/SSL channel [RFC2246] as amended by [RFC3546]. This  
238 will ensure the integrity and confidentiality of the authentication protocol messages. Additionally, clients SHOULD  
239 authenticate the server via TLS/SSL validation procedures. This will help guard against man-in-the-middle attacks.

### 240 4.6. Protocol Messages

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

#### 244 4.6.1. The <SASLRequest> Message

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

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

- 250 • **authzID** [Optional] — The `authzID`, also known as *user identifier* or *username* or *Principal*, that the client  
 251 wishes to establish as the "authorization identity" per [\[RFC2222\]](#). It is only included on the *initial SASL request*.
  
- 252 • **advisoryAuthnID** [Optional] — The `advisoryAuthnID` may be used to advise the server what authentication  
 253 identity will be asserted by the client via the selected SASL mechanism; i.e. it is a "hint". It is only included on  
 254 the initial SASL request. The `advisoryAuthnID` provides a means for server implementations to optimize their  
 255 behavior on a per authentication identity basis. E.g. if a client requests to execute a certain SASL mechanism  
 256 on behalf of some given authentication identity (represented by `advisoryAuthnID`) and authorization identity  
 257 (represented by `authzID`) pair, the server can decide whether to proceed without having to execute the SASL  
 258 mechanism (execution of which might involve more than a single round-trip). Server implementations that make  
 259 use of the optional `advisoryAuthnID` attribute, **SHOULD** be capable of processing initial `<SASLRequest>`  
 260 messages that do not include the `advisoryAuthnID` attribute.
  
- 261 • **id** [Optional] — identifies a `<SASLRequest>` message element instance. This attribute **MUST** be used when the  
 262 message is signed as described in [\[LibertySecMech\]](#), and the element instance is to be included as one of the set  
 263 of signed message components.

```

264
265
266 <xs:element name="SASLRequest">
267   <xs:complexType>
268     <xs:sequence>
269
270       <xs:element name="Data" minOccurs="0">
271         <xs:complexType>
272           <xs:simpleContent>
273             <xs:extension base="xs:base64Binary" />
274           </xs:simpleContent>
275         </xs:complexType>
276       </xs:element>
277
278       <xs:element ref="lib:RequestAuthnContext"
279         minOccurs="0"/>
280
281     </xs:sequence>
282
283     <xs:attribute name="mechanism"
284       type="xs:string"
285       use="required" />
286
287     <xs:attribute name="authzID"
288       type="xs:string"
289       use="optional" />
290
291     <xs:attribute name="advisoryAuthnID"
292       type="xs:string"
293       use="optional" />
294
295     <xs:attribute name="id"
296       type="xs:ID"
297       use="optional" />
298
299   </xs:complexType>
300 </xs:element>
301
302

```

303 **Figure 1. <SASLRequest> Message Element — Schema Fragment**

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

- 305 • **<Data>** — This element is used by the client to send SASL mechanism data to the server. In [\[RFC2222\]](#) parlance,  
306 this data is termed a "client response". Its content model is base64-encoded data.
- 307 • **<RequestAuthnContext>** — This element is used by the client to convey to the server a desired authentication  
308 context. It is used only on the initial SASL request (see [Section 4.7: Sequencing of the Authentication](#)  
309 [Exchange](#) ). If present, the server uses the information in the `<RequestAuthnContext>` in combination  
310 with mechanism attribute when choosing the SASL mechanism to execute. The background use case for  
311 `<RequestAuthnContext>` is presented in [Section 5.1: Authentication Service: Conceptual Model](#) . See also:  
312 [\[LibertyAuthnContext\]](#) and [\[LibertyProtSchema\]](#).

```
313
314
315 <?xml version="1.0" encoding="UTF-8"?>
316
317 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
318     xmlns:sa="urn:liberty:sa:2004-04"
319     xmlns:sb="urn:liberty:wsf:soap-bind:1.0"
320     xmlns:pp="urn:liberty:id-sis-pp:2003-08">
321
322   <S:Header>
323
324     <sb:Correlation S:mustUnderstand="1"
325         sb:id="A13454...245"
326         S:actor="http://schemas.../next"
327         sb:messageID="uuid:efefefef-aaaa-ffff-cccc-eeeeffffbbbb"
328         sb:timestamp="2112-03-15T11:12:12Z"/>
329
330   </S:Header>
331
332   <S:Body>
333
334     <sa:SASLRequest sa:mechanism="foo">
335       <sa:Data>
336         qwYGHhSWpjQu5yq.....vUulvONmOZtfzgFz
337       <sa:Data>
338     </sa:SASLRequest>
339
340   </S:Body>
341
342 </S:Envelope>
343
344
```

345 **Example 1. A SASLRequest Bound into a SOAP Message**

#### 346 4.6.1.1. **<SASLRequest> Usage**

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

- 348 • list of one or more client-supported SASL mechanism names,

349 ..in combination with optional:

- 350 • `authzID` attribute, and/or,
- 351 • `advisoryAuthnID` attribute, and/or,
- 352 • `<RequestAuthnContext>` element.

353 In the case where a single SASL mechanism name is conveyed, the <SASLRequest> message can contain a so-called  
354 *initial response* (see Section 5.1 of [RFC2222]) in the <Data> element.

355 If the server's subsequent <SASLResponse> message signals that the authentication exchange should continue—and  
356 thus contains a server "challenge"—the client will send another <SASLRequest> message, with the <Data> element  
357 containing the client's "response" to the challenge. This sequence of server challenges and client responses continues  
358 until the server signals a successful completion or aborts the exchange.

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

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

#### 363 **4.6.1.2. Values for mechanism attribute of <SASLRequest>**

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

##### 366 **Note:**

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

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

```
373  
374  
375 <SASLRequest mechanism="GSSAPI OTP PLAIN" />
```

376  
377  
378 **Example 2. <SASLRequest> Specifying Multiple Client-supported Mechanism Names**

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

```
381  
382  
383 <SASLRequest mechanism="GSSAPI">  
384 <Data>  
385 Q29ub3IgcQ2FoaWxsIGNhc3VhbGx5IG1hbmdsZXZzcGFzZ3dvcnRzCg==  
386 </Data>  
387 </SASLRequest>
```

388  
389  
390 **Example 3. <SASLRequest> Specifying a Single Mechanism Name**

391 3. **A NULL string ("" )** — This indicates to the authentication server that the client wishes to abort the authentica-  
392 tion exchange. See [Example 4](#).

```
393  
394  
395 <SASLRequest mechanism="" />
```

396  
397  
398 **Example 4. <SASLRequest> Message Aborting the SASL Authentication Exchange**

## 399 4.6.2. The <SASLResponse> Message

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

- 402 • **serverMechanism** [Optional] — The server's choice of SASL mechanism from among the list sent by the client.
- 403 • **id** [Optional] — identifies a <SASLResponse> message element instance. This attribute MUST be used when the  
404 message is signed as described in [\[LibertySecMech\]](#), and the element instance is to be included as one of the set  
405 of signed message components.

```
406  
407  
408 <xs:element name="SASLResponse">  
409   <xs:complexType>  
410     <xs:sequence>  
411  
412       <xs:element ref="Status"/>  
413  
414       <xs:element ref="PasswordTransforms" minOccurs="0"/>  
415  
416       <xs:element name="Data" minOccurs="0">  
417         <xs:complexType>  
418           <xs:simpleContent>  
419             <xs:extension base="xs:base64Binary"/>  
420           </xs:simpleContent>  
421         </xs:complexType>  
422       </xs:element>  
423  
424       <xs:element ref="disco:ResourceOffering"  
425         minOccurs="0"  
426         maxOccurs="unbounded"/>  
427  
428       <xs:element name="Credentials" minOccurs="0">  
429         <xs:complexType>  
430           <xs:sequence>  
431             <xs:any namespace="##any"  
432               processContents="lax"  
433               minOccurs="0"  
434               maxOccurs="unbounded"/>  
435           </xs:sequence>  
436         </xs:complexType>  
437       </xs:element>  
438  
439     </xs:sequence>  
440  
441     <xs:attribute name="serverMechanism"  
442       type="xs:string"  
443       use="optional"/>  
444  
445     <xs:attribute name="id"  
446       type="xs:ID"  
447       use="optional"/>  
448  
449   </xs:complexType>  
450 </xs:element>  
451  
452
```

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

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

- 455 • **<Status>** — This element is from [Liberty ID-WSF Utility XSD](#) and is used to convey status from the server to  
456 the client. See below.
- 457 • **<PasswordTransforms>** — This element is used to convey to the client any required password transformations.  
458 See [Section 7: Password Transformations: The PasswordTransforms Element](#) .
- 459 • **<Data>** — This element is used to return SASL mechanism data to the client. Its content model is base64-encoded  
460 data.
- 461 • **<disco:ResourceOffering>** — This element is to convey to the client a resource offering for the server, in its  
462 role as a WSP, upon a successful authentication exchange completion. See [Section 5: Authentication Service](#).
- 463 • **<Credentials>** — This element is used to convey to the client credentials authorizing it to interact with  
464 the server (who is acting as a WSP) upon a successful authentication exchange completion. See [Section 5:  
465 Authentication Service](#).

#### 466 4.6.2.1. <SASLResponse> Usage

467 This message is sent by the server in response to a client <SASLRequest> message. It is used to convey "server  
468 challenges", in [\[RFC2222\]](#) parlance, to the client during an authentication exchange. So-called "client responses"  
469 are correspondingly conveyed to the server via the <SASLRequest> message, defined above. A given authentication  
470 exchange may occur in one "round-trip", or it may involve several round-trips. This depends on the SASL mechanism  
471 being executed.

472 The first <SASLResponse> sent by the server is explicitly distinguished from subsequent <SASLResponse> messages  
473 in terms of child elements and attributes. The final <SASLResponse> sent by the server in an authentication exchange  
474 is similarly distinguished, although with its own particular characteristics. These details are specified below in  
475 [Section 4.7: Sequencing of the Authentication Exchange](#) .

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

```
479  
480  
481 <xs:element name="Status" type="StatusType"/>  
482  
483 <xs:sequence>  
484 <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>  
485 </xs:sequence>  
486 <xs:attribute name="code" type="xs:QName" use="required"/>  
487 <xs:attribute name="ref" type="xs:NCName" use="optional"/>  
488 <xs:attribute name="comment" type="xs:string" use="optional"/>  
489 </xs:complexType>  
490  
491
```

492 **Figure 3. <Status> Element and Type - Schema Fragment (from lib-arch-iwsf-utility.xsd)**

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

##### 495 4.6.2.1.1. Values for the `code` attribute of <Status>

496 If the value of `code` is:

- 497 • **"sa:continue"** — the server expects the client to craft and send a new <SASLRequest> message containing data  
498 appropriate for whichever step the execution of the SASL mechanism is at.

- 499 • **"sa:abort"** — the server is aborting the authentication exchange. It will not send any more messages on this  
500 message thread.
- 501 • **"sa:OK"** — the server considers the authentication exchange to have completed successfully.  
502 The <SASLResponse> message will typically contain <disco:ResourceOffering> element(s) and a  
503 <Credentials> element, as described below in [Section 5.3: Rules for Authentication Service Providers](#) ,  
504 enabling the client to interact further with this provider, for example to invoke another ID-WSF service such as  
505 the Discovery Service.  
506 Additionally, the <SASLResponse> message can contain <disco:ResourceOffering> element(s) and  
507 <Credentials> content for other providers.  
508 See [Section 4.7: Sequencing of the Authentication Exchange](#) for the normative specification of the composition  
509 of the <SASLResponse> message in this case. See also [Section 5.3: Rules for Authentication Service Providers](#) .

#### 510 4.6.2.1.2. Returning the Server's Selected SASL Mechanism

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

```
514  
515  
516 <SASLResponse serverMechanism="DIGEST-MD5">  
517   <Status code="sa:continue" />  
518   <Data>  
519     Q29ub3IgcQ2FoaWxsIGNhc3VhbGx5IG1hbmdsZXMgcGFzc3dvcmRzCg==  
520   </Data>  
521 </SASLResponse>  
522  
523
```

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

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

```
529  
530  
531 <SASLResponse>  
532   <Status code="sa:abort" />  
533 </SASLResponse>  
534  
535
```

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

## 537 4.7. Sequencing of the Authentication Exchange

538 The authentication exchange is sequenced as follows:

- 539 1. The authentication exchange **MUST** begin by the client sending the server a <SASLRequest> message. This  
540 message:

- 541       • MUST contain a `mechanism` attribute whose value is a string containing one or more SASL mechanisms  
542       the client supports and is prepared to negotiate (see [Section 4.6.1.2: Values for mechanism attribute of](#)  
543       `<SASLRequest>` ).
- 544       • MAY contain a `<Data>` element containing an initial response, specific to the cited SASL mechanism, if the  
545       `mechanism` attribute contains only a single SASL mechanism. See section 5.1 of [\[RFC2222\]](#).
- 546       • MAY contain a `<RequestAuthnContext>` element.
- 547       • SHOULD contain an `authzID` attribute whose value is an identifier string for the Principal being authenti-  
548       cated.
- 549       • MAY contain an `advisoryAuthnID` attribute whose value is an identifier asserted by the client to represent  
550       the authentication identity being established by this authentication event.
- 551       • MAY contain an `id` attribute.
- 552       2. If the server is prepared to execute, with this client, at least one of the SASL mechanism(s) cited by the client in  
553       the previous step, then processing continues with step 4.
- 554       3. Otherwise, the server does not support, or is not prepared to negotiate, any of the SASL mechanisms cited by the  
555       client. The server MUST respond to the client with a `<SASLResponse>` message containing:
- 556       • A `<Status>` element with a `code` attribute with a value of "**sa:abort**".
- 557       • No `<PasswordTransforms>` element.
- 558       • No `<Data>` element.
- 559       • No `<disco:ResourceOffering>` element.
- 560       • No `<Credentials>` element.
- 561       • No `serverMechanism` attribute.
- 562       The `<SASLResponse>` message MAY have an `id` attribute. After this message is sent to the client, processing  
563       continues with step 7.
- 564       4. The server sends to the client a `<SASLResponse>` message.  
565       If this message is the first `<SASLResponse>` sent to the client in this authentication exchange, this message:
- 566       • MUST contain a `serverMechanism` attribute whose value is a single SASL mechanism name, chosen by  
567       the server from the list sent by the client.
- 568       • MAY contain a `<Data>` element containing a SASL mechanism-specific challenge.
- 569       • MAY contain a `<PasswordTransforms>` element. See [Section 7: Password Transformations: The](#)  
570       `PasswordTransforms` Element for details on the client's subsequent obligations in this case.
- 571       • MAY contain a `<id>` attribute.
- 572       • MUST contain a `<Status>` element with a `code` attribute whose value is given by either item [A](#), or [B](#), or [C](#):

- 573           A. "**sa:continue**" — the execution of the SASL mechanism is not complete; the server expects the client to  
574           process this message and respond. Processing continues with step 5.
- 575           B. "**sa:OK**" — the server declares the authentication exchange has completed successfully.  
576           In this case, this *final SASL response* message can contain, in addition to the items listed above,  
577           <disco:ResourceOffering> element(s) and a <Credentials> element. This is specified in  
578           [Section 5.3: Rules for Authentication Service Providers](#) .  
579           Processing continues with step 6.
- 580           C. "**sa:abort**" — the server declares the authentication exchange has completed unsuccessfully. For  
581           example, the user may have supplied incorrect information, such as an incorrect password. See step  
582           7, below, for additional information.  
583           In this case, this <SASLResponse> message **MUST NOT** contain any <disco:ResourceOffering>  
584           element(s) or a <Credentials> element.  
585           Processing continues with step 7.
- 586   Otherwise, this message:
- 587       • **MUST NOT** contain a `serverMechanism` attribute.
  - 588       • **MAY** contain a <Data> element containing a SASL mechanism-specific challenge.
  - 589       • **MUST NOT** contain a <PasswordTransforms> element.
  - 590       • **MAY** contain a <id> attribute.
  - 591       • **MUST** contain a <Status> element with a `code` attribute whose value is given by either item A, or B, or C:
- 592           A. "**sa:continue**" — the execution of the SASL mechanism is not complete; the server expects the client to  
593           process this message and respond. Processing continues with step 5.
- 594           B. "**sa:OK**" — the server declares the authentication exchange has completed successfully.  
595           In this case, this "final response" <SASLResponse> message can contain, in addition to the items listed  
596           above, <disco:ResourceOffering> element(s) and a <Credentials> element. This is specified in  
597           [Section 5.3: Rules for Authentication Service Providers](#) .  
598           Processing continues with step 6.
- 599           C. "**sa:abort**" — the server declares the authentication process has completed unsuccessfully. For example,  
600           the user may have supplied incorrect information, such as an incorrect password. See step 7, below, for  
601           additional information.  
602           In this case, this <SASLResponse> message **MUST NOT** contain any <disco:ResourceOffering>  
603           element(s) or a <Credentials> element.  
604           Processing continues with step 7.
- 605   5. The client sends the server a <SASLRequest> message. This message:

- 606       • SHOULD contain a `mechanism` attribute set to the same value as sent by the server, as the value of the  
607        `serverMechanism` attribute, in its first `<SASLResponse>` message (see [Section 4.6.2.1.2: Returning the](#)  
608        Server's Selected SASL Mechanism ).
- 609       **Note:**  
610        The client MAY, however, choose to abort the authentication exchange by setting the `mechanism` attribute  
611        to either a "null" string, or to a mechanism name different than the one returned by the server in its first  
612        `<SASLResponse>` message.  
613        If the client chooses to abort, processing continues with step 8.
- 614       • SHOULD contain a `<Data>` element containing data specific to the cited SASL mechanism.
- 615       • MUST NOT contain a `<RequestAuthnContext>` element.
- 616       • MUST NOT contain an `authzID` attribute.
- 617       • MUST NOT contain an `advisoryAuthnID` attribute.
- 618       • MAY contain an `id` attribute.
- 619       Processing continues with steps 4 and 5 until the server signals success, failure, or aborts — or the client aborts  
620       the exchange using the technique noted in the first bullet item, above, of this step.
- 621       6. The authentication exchange has completed successfully. The client is now authenticated in the server's view, and  
622        the server may be authenticated in the client's view, depending upon the SASL mechanism employed. [Section 5.1:](#)  
623        [Authentication Service: Conceptual Model](#) discusses what the next interaction steps between the client and  
624        server are in the ID-WSF authentication service case.
- 625       7. The authentication exchange has completed unsuccessfully due to an exception on the server side. The client  
626        SHOULD cease sending messages on this message thread.  
627        The reasons for an authentication exchange failing are manifold. Often it is simply a case of the user having  
628        supplied incorrect information, such as a password or passphrase. Or, there may have been a problem on the  
629        server's part, such as an authentication database being unavailable or unreachable.
- 630       **Note:**  
631        [\[RFC2222\]](#) and the RFCs specifying various SASL mechanisms—for example [\[RFC2245\]](#), [\[RFC2444\]](#), and  
632        [\[RFC3163\]](#)—are arguably not as clear as they could be with respect to the situation where an execution of the  
633        SASL mechanism fails for some reason. Though, Section 4, item 3, of [\[RFC2222\]](#) indicates that the server must  
634        have a means of indicating "failure of the exchange" to the client. In this version of this specification, this is  
635        handled by the server returning a status code of "**sa:abort**" to the client, as specified above in 4. Future versions  
636        of this specification may facilitate more fine-grained error reporting by the server.
- 637       8. The client aborted the authentication exchange.

## 638 5. Authentication Service

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

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

### 642 5.1. Conceptual Model

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

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

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

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

#### 656 5.1.1. Stipulating a Particular Authentication Context

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

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

666       Now, the LUAD-WSC "knows" that it needs to help the Principal reauthenticate—as her present  
667       credentials aren't being honored for the financial transaction she wishes to carry out. So the  
668       LUAD-WSC prompts the Principal for permission to reauthenticate her, and (assuming the answer  
669       was "yes") initiates the ID-WSF Authentication Protocol with the appropriate authentication service  
670       provider, and includes the supplied-by-the-bank `<RequestAuthnContext>`. The authentication  
671       service provider factors the requested authentication context into its selection of SASL mechanism  
672       for the ensuing authentication exchange. And upon successful authentication, the Principal is able  
673       to successfully make the funds transfer.

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

### 677 5.2. Service Type Declaration

678 The Service Type URI for the ID-WSF Authentication Service is:

679       `urn:liberty:as:2004-04`

## 680 5.3. Rules for Authentication Service Providers

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

- 682 1. Authentication Service Providers (AS Providers) MUST implement the ID-WSF Authentication Protocol, as  
683 defined in [Section 4: Authentication Protocol](#) . The Authentication Service Provider MUST play the role of the  
684 *authentication server*.
- 685 2. Upon successful completion of an authentication exchange the **first** <ResourceOffering> element instances  
686 contained in the *final SASL response* SHOULD refer to services at the Authentication Service provider—i.e. at  
687 the "same provider"—that said AS Provider can offer to the Authentication Service Consumer.  
688 For example, Identity Providers may often add the <disco:ResourceOffering> and <Credentials> for  
689 the Discovery Service of the Principal just authenticated, as well as <disco:ResourceOffering>s and  
690 <Credentials> for other offered services, such as an SSO Service.  
691 In deployments where the Identity Provider and Discovery Service are tightly coupled the <Credentials>  
692 element MAY be shared. See [Section 4.7: Sequencing of the Authentication Exchange](#) , Step 4. The Provider  
693 MAY also include additional <disco:ResourceOffering> element instances, and security tokens within the  
694 <Credentials> element, that refer to services offered by other providers—i.e. providers other than the AS  
695 Provider.
- 696 3. Any included credentials SHOULD be useful for a reasonable time. Even if the AS Consumer recently  
697 authenticated with the Authentication Service, i.e. an earlier issued credential for consumption by the AS  
698 Provider is still valid, the AS Provider SHOULD issue credential(s) that have later expiration times than the  
699 earlier issued credential(s). The AS Provider MAY choose to re-authenticate, using any of the available  
700 SASL mechanisms, or issue new credentials without an engaging in an authentication exchange. This can be  
701 accomplished by responding to the AS Consumer's initial SASL request with a final SASL response containing  
702 requisite <ResourceOffering>(s) and <Credentials>.  
703 **Note:**  
704 Credentials containing <saml:AuthenticationStatement>(s) should have their <saml:AuthenticationInstant>(s)  
705 set to the time when the authentication event actually took place. See [\[SAMLCore11\]](#).
- 706 4. Additionally, if the first <SASLRequest> in an exchange contains a <lib:RequestAuthnContext> element,  
707 then upon successful authentication, the Authentication Service MUST either: return <Credentials> that  
708 satisfy the <lib:RequestAuthnContext>, or, abort the authentication exchange (see also the "Single Sign-  
709 On and Federation Protocol" section in [\[LibertyProtSchema\]](#)). To satisfy the <lib:RequestAuthnContext>,  
710 any returned <Credentials> MUST be created according to the following rules:
- 711 a. If one or more <lib:AuthnContextClassRef> or <lib:AuthnContextStatementRef> elements are  
712 present in the lib:RequestAuthnContext, then the resulting authentication statement in the assertion  
713 (if any) MUST contain an authentication statement that conforms to the class or statement specified.  
714 Additionally, the set of supplied elements MUST be evaluated as an ordered set, where the first element  
715 is the most preferred authentication context class or statement. If none of the specified classes or statements  
716 can be satisfied, the identity provider MUST NOT include a credential and abort.
- 717 b. Additionally, if an <lib:AuthnContextComparison> element is supplied, and one or more  
718 <lib:AuthnContextStatementRef> or <lib:AuthnContextClassRef> elements are included,  
719 then the resulting authentication statement in the assertion (if any) MUST follow the rule specified in the  
720 <lib:AuthnContextComparison> element. If this requirement cannot be satisfied, the identity provider  
721 MUST NOT include a credential and abort.

- 722 c. If an `<lib:AuthnContextComparison>` is specified and set to "exact", then the resulting authentication  
723 statement in the assertion (if any) **MUST** be the exact match of at least one of the authentication contexts  
724 specified.  
725 If `<lib:AuthnContextComparison>` is specified and set to "minimum", then the resulting authentication  
726 statement in the assertion (if any) **MUST** be at least as strong (as deemed by the Authentication Service  
727 provider) as one of the authentication contexts specified.  
728 If `AuthnContextComparison` is specified and set to "better", then the resulting authentication statement  
729 in the assertion (if any) **MUST** be stronger (as deemed by the identity provider) than any specified in the  
730 supplied authentication contexts. If `AuthnContextComparison` is specified and set to "maximum", then  
731 the resulting authentication statement in the assertion (if any) **MUST** be as strong as possible (as deemed by  
732 the identity provider) without exceeding the strength of at least one of the authentication contexts specified.
- 733 5. An Authentication Service instance **SHOULD** be deployed such that the security mechanism [[LibertySecMech](#)]:  
734 `urn:liberty:security:2003-08:TLS:null`  
735 can be used by the WSC.  
736 **Note:**  
737 In practice this means that the Authentication Service should be exposed on an endpoint for which the URL  
738 should have `https` as the protocol field.
- 739 6. An Authentication Service implementation **SHOULD** support the following SASL mechanisms [[SASLReg](#)]:  
740 PLAIN, CRAM-MD5.

## 741 5.4. Rules for Authentication Service Consumers

- 742 WSCs implementing the client-side of the ID-WSF Authentication Protocol, and thus also known as *Authentication*  
743 *Service Consumers* (AS Consumers), **MUST** adhere to the following rules:
- 744 1. AS Consumers **MUST** implement the ID-WSF Authentication Protocol, as defined in [Section 4: Authentication](#)  
745 Protocol in the role of the client.  
746 **Note:**  
747 The AS Consumer may include various SOAP header blocks, e.g. a `<wsse:Security>` element [[Liberty-](#)  
748 [SecMech](#)] which can house a security token(s) obtained earlier from an Authentication Service or Discovery  
749 Service [[LibertyDisco](#)]. In such a case, the Authentication Service **SHOULD** evaluate the presented security to-  
750 ken(s) in combination with applicable policy, as a part of the overall authentication event. This provides a means,  
751 for example, of "security token renewal".
- 752 2. In case the AS Consumer has not been provisioned with the `<disco:SecurityMechID>` for the Authentication  
753 Service instance that it uses, the AS Consumer **SHOULD** assume that the required security mechanism is:  
754 `urn:liberty:security:2003-08:TLS:null`  
755 **Note:**  
756 `<disco:SecurityMechID>` is a subelement of `<disco:Description>`, which is a subelement of  
757 `<disco:ServiceInstance>`, which is a part of `<disco:ResourceOffering>` [[LibertyDisco](#)].  
758 Only when the endpoint URL of the Authentication Service is prescribed to have `https` as the protocol **MAY** the  
759 WSC presume a security mechanism of: `urn:liberty:security:2003-08:null:null`
- 760 3. It is **RECOMMENDED** that the WSC support the password transformations specified in [Appendix B](#) .

## 761 5.5. Authentication Service Interaction Example

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

```
765  
766 <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">  
767   <s:Header>  
768     <sb:Correlation s:mustUnderstand="1" xmlns:sb="urn:liberty:sb:2003-08"  
769       id="thisCorrHdr.1234"  
770       messageID="-1909129211480725265-1455516650932757068"  
771       timestamp="2004-02-03T22:12:26Z"  
772       xmlns:sb="urn:liberty:sb:2003-08"/>  
773   </s:Header>  
774   <s:Body>  
775     <SASLRequest mechanism="CRAM-MD5"  
776       advisoryAuthnID="358408021451"  
777       xmlns="urn:liberty:sa:2004-04" />  
778   </s:Body>  
779 </s:Envelope>  
780
```

781 **Example 7. The WSC sends a <SASLRequest> on behalf of a Principal, asserting that the authentication identity is**  
782 **"358408021451" and indicates it desire to use the "CRAM-MD5" SASL mechanism.**

```
783  
784 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">  
785   <S:Header>  
786     <sb:Correlation s:mustUnderstand="1" xmlns:sb="urn:liberty:sb:2003-08"  
787       id="thisCorrHdr.2345"  
788       messageID="i48b4353f50aca1494665d61b93498c885449c868"  
789       refToMessageID="-1909129211480725265-1455516650932757068"  
790       timestamp="2004-02-03T22:12:27Z" />  
791   </S:Header>  
792   <S:Body>  
793     <SASLResponse serverMechanism="CRAM-MD5"  
794       xmlns="urn:liberty:sa:2004-04">  
795     <Status code="continue" />  
796     <Data>  
797       ... a CRAM-MD5 challenge here...  
798     </Data>  
799   </SASLRequest>  
800 </s:Body>  
801 </s:Envelope>  
802
```

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

```
804
805 <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
806   <s:Header>
807     <sb:Correlation s:mustUnderstand="1" xmlns:sb="urn:liberty:sb:2003-08"
808       id="thisCorrHdr.3456"
809       messageID="-411835766977623870327144762785172067"
810       refToMessageID="i48b4353f50aca1494665d61b93498c885449c868"
811       timestamp="2004-02-03T22:12:28Z" />
812   </s:Header>
813   <s:Body>
814     <SASLRequest mechanism="CRAM-MD5"
815       xmlns="urn:liberty:sa:2004-04">
816       <Data>
817         ...some CRAM-MD5 response here...
818       </Data>
819     </SASLRequest>
820   </s:Body>
821 </s:Envelope>
822
```

823

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

824

```
825 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
826   <S:Header>
827     <sb:Correlation s:mustUnderstand="1" xmlns:sb="urn:liberty:sb:2003-08"
828       id="thisCorrHdr.4567"
829       messageID="ic8d255ebe4b286ea0e7645ac3e9c558e11e8e8f1"
830       refToMessageID="-411835766977623870327144762785172067"
831       timestamp="2004-02-03T22:12:29Z"/>
832   </S:Header>
833   <S:Body id="msgBody">
834     <sa:SASLResponse xmlns:sa="urn:liberty:sa:2004-04"
835       xmlns:disco="urn:liberty:disco:2003-08">
836       <Status code="sa:OK"/>
837       <disco:ResourceOffering>
838         <disco:ResourceID>
839           http://tg2.example.com:8080/tfs/local/358408021451
840         </disco:ResourceID>
841         <disco:ServiceInstance>
842           <disco:ServiceType>
843             urn:liberty:disco:2003-08
844           </disco:ServiceType>
845           <disco:ProviderID>
846             http://tg2.example.com:8080/tfs
847           </disco:ProviderID>
848           <disco:Description>
849             <disco:SecurityMechID>
850               urn:liberty:security:2005-02:null:Bearer
851             </disco:SecurityMechID>
852             <disco:CredentialRef>
853               ilb42508103cab657f34e5ef189f28ea10dd86926
854             </disco:CredentialRef>
855             <disco:Endpoint>
856               http://tg2.example.com:8080/tfs-soap/IdPDiscoveryService
857             </disco:Endpoint>
858             <disco:SoapAction>
859               urn:liberty:disco:2003-08
860             </disco:SoapAction>
861           </disco:Description>
862         </disco:ServiceInstance>
863       </disco:ResourceOffering>
864       <sa:Credentials>
865         <saml:Assertion
866           AssertionID="ilb42508103cab657f34e5ef189f28ea10dd86926"
867           IssueInstant="2004-02-03T22:12:33Z"
868           Issuer="http://tg2.trustgenix.com:8080/tfs"
869           ....
870           ....
871         </saml:Assertion>
872       </sa:Credentials>
873     </sa:SASLResponse>
874   </S:Body>
875 </S:Envelope>
876
```

877

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

878

## 879 6. Single Sign-On Service

880 The ID-WSF Single Sign-On Service (SSO Service, or SSOS) provides Web Service Consumers (WSCs) an ID-  
881 WSF-based means to obtain *Liberty authentication assertions* enabling them to interact with *Service Providers* (SPs)  
882 [[LibertyProtSchema](#)].

883 This section first outlines the ID-WSF SSO Service's conceptual model and then defines the SSO Service in terms of  
884 rules for Providers and Consumers of the service.

### 885 6.1. Conceptual Model

886 In the Liberty architecture, it is conceivable for any concrete *system entity* to don any architectural *role* that it is  
887 physically capable of bearing. For example, a Liberty *Service Provider* (SP) is essentially just a Liberty ID-FF-enabled  
888 website. Such Service Providers can also be simultaneously cast into WSC and WSP roles.

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

893 Similarly to the conundrum outlined in [Section 5: Authentication Service](#), a LUAD acting as a WSC one moment (a  
894 "LUAD-WSC") and as a vanilla browser the next, will need the means to obtain authentication assertions and security  
895 tokens as necessary.

896 As noted in [Section 5](#), a (LUAD-)WSCs needing to obtain security tokens in order to interact with a Discovery  
897 Service can utilize an ID-WSF Authentication Service to obtain requisite security tokens. However, in ID-FF,  
898 the user agent is assumed to be a vanilla browser, and Identity Providers vouch for browser-wielding Principals by  
899 sending authentication assertions, or "pointers" to authentication assertions (AKA "SAML artifacts" [[SAMLCore11](#)]),  
900 to Service Providers "through" Principals' browsers (e.g. via HTTP "redirects").

901 (LUAD-)WSCs thus need some means to cause authentication assertions to be conveyed to SPs they wish to interact  
902 with. Remember that not all SPs will be able to don a WSP role, so simply authenticating via the Authentication  
903 Service, either at some Identity Provider or with an SP/WSP providing an Authentication Service, is not a solution for  
904 this use case.

905 The ID-WSF Single Sign-On Service addresses this issue. It is a profile of the ID-FF Single Sign-On and Federation  
906 Protocol [[LibertyProtSchema](#)]. It provides the means for (LUAD-)WSCs to interact with SPs. See also [[LibertyClient-](#)  
907 Profiles] for additional background information.

908 The overall mechanism is based on two steps. First, a (LUAD-)WSC wishing to interact with some SP can use  
909 the Authentication Service at an Identity Provider to obtain security tokens. Next, the (LUAD-)WSC invokes the  
910 Single Sign-On Service at the Identity Provider in order to obtain an authentication assertion to convey to the SP, thus  
911 enabling Liberty-SSO-enabled, vanilla, web-based interactions with that SP.

912 For example, if a (LUAD-)WSC successfully authenticates with an Identity Provider (IdP) via the IdP's Au-  
913 thentication Service ([Section 5](#)), the IdP can ensure that the LUAD-WSC will have in its possession a  
914 <disco:ResourceOffering> and necessary credentials for the ID-WSF Single Sign-On Service at the very  
915 same IdP. Thus the (LUAD-)WSC may obtain an authentication assertion via the IdP's the latter Service.

916 Additionally, the IdP can, at the same time, ensure that the (LUAD-)WSC possesses a <disco:ResourceOffering>  
917 and necessary credentials for the Discovery Service (DS) of the Principal wielding the LUAD-WSC — thus enabling  
918 the LUAD-WSC to simultaneously utilize ID-FF- and ID-WSF-based services on behalf of the Principal, based on  
919 one sign-on interaction, from the Principal's perspective.

920 In yet another plausible scenario, some web service provider(s) might not be ID-WSF-based. Rather, they could be  
921 *generic Web Service Providers* (gWSPs).

922 A (LUAD-)WSC consuming services from gWSPs may need to obtain security tokens satisfying whichever security  
923 paradigm the gWSPs employ. It is plausible that such a paradigm will accommodate use of Liberty authentication  
924 assertions as security tokens — for example, see [wss-sms] and [wss-saml]. Note that the SSO Service can address  
925 this use case.

## 926 6.2. Service Type Declaration

927 The Service Type URI for the ID-WSF SSO Service is:  
928 `urn:liberty:ssos:2004-04`

## 929 6.3. Rules for SSO Service Providers

930 SSO Service Providers (SSOS Providers) MUST adhere to the following rules:

- 931 1. Unless stated otherwise below the SSOS Provider SHOULD adhere to the [LibertyProtSchema] and [Liberty-  
932 BindProf] specifications.
- 933 2. The SSOS Provider SHOULD offer an ID-WSF Authentication Service, as defined in Section 5: Authentication  
934 Service.  
935 Upon successful authentication the SSOS Provider will respond to the SSOS Consumer with a SASLResponse  
936 as specified in Section 5. Returned <disco:ResourceOffering> elements referring to SSO Service instances  
937 MUST use the Service Type URI defined in Section 6.2 above.
- 938 3. The SSOS Provider SHOULD adhere to the SOAP binding as specified in [LibertySOAPBinding]; in case  
939 of conflict with the SOAP binding as specified in [LibertyBindProf] the [LibertySOAPBinding] shall take  
940 precedence.
- 941 4. SSOS Providers SHOULD advertise in their SSO capability in metadata [LibertyMetadata] it provides the  
942 SSO Service. To accomplish this, it MUST include a <md:SingleSignOnServiceProfile> element in  
943 its metadata, with a value of:  
944 `urn:liberty:iff:profiles:id-wsf`
- 945 5. The SSOS Provider MAY, when it receives an <lib:AuthnRequest> that has its ProtocolProfile element  
946 set to `urn:liberty:iff:profiles:id-wsf`, respond with an ID-WSF message, containing relevant  
947 header blocks as specified in [LibertySOAPBinding].  
948 **Note:**  
949 SSOS Providers MAY take advantage of various optional header blocks defined in [LibertySOAPBind-  
950 ing]. For example, instead of attempting to establish a local session via an HTTP cookie, which is  
951 likely to be ignored, the SSOS Provider may include a <sec:ServiceSessionContext> element in a  
952 <sb:ServiceInstanceUpdate> header block. The WSC that sent the original <lib:AuthnRequest> must  
953 of course understand these header blocks.
- 954 6. SSOS Providers SHOULD NOT respond with any content other than SOAP. For example, the MIME type of the  
955 HTTP response must be set according to [LibertySOAPBinding]  
956 **Note:**  
957 This is different from the LECP profile [LibertyBindProf] where an IdP is allowed to respond with any content  
958 that is acceptable to the requester (i.e. the LECP).
- 959 7. Upon successful processing of the <lib:AuthnRequest>, the SSOS Provider SHOULD respond with a SOAP-  
960 bound <lib:AuthnResponse> message, constructed according to [LibertyProtSchema] in combination with  
961 [LibertySOAPBinding].  
962 **Note:**  
963 This is different from the LECP profile [LibertyBindProf] where an IdP is expected to respond with an  
964 <lib:AuthnRequestEnvelope>.

## 965 **6.4. Rules for SSO Service Consumers**

966 Consumers of the ID-WSF SSO Service MUST adhere to the following rules:

- 967 1. Unless stated otherwise below the WSC SHOULD adhere to the rules for active intermediaries as specified in  
968 [\[LibertyProtSchema\]](#) and [\[LibertyBindProf\]](#).
- 969 2. The WSC SHOULD adhere to the SOAP binding as specified in [\[LibertySOAPBinding\]](#); in case of conflict  
970 with the SOAP binding as specified in [\[LibertyBindProf\]](#) the [\[LibertySOAPBinding\]](#) shall take precedence.  
971 For example, the WSC must include a proper `<sb:Correlation>` header block in its messages to the SSOS  
972 Provider.  
973 **Note:**  
974 The WSC MAY include various other header blocks, e.g. a `<wsse:Security>` header block [\[LibertySecMech\]](#)  
975 [\[wss-sms\]](#). Such a header block could contain a security token obtained from an ID-WSF Authentication Service  
976 Provider.
- 977 3. The WSC MUST set the `<lib:ProtocolProfile>` element of the `<lib:AuthnRequest>` to:  
978 `urn:liberty:iff:profiles:id-wsf`  
979 **Note:**  
980 Depending on the application and deployment model the WSC may have to construct the `<lib:AuthnRequest>`  
981 by itself, unlike LECP implementations that merely repack a `<lib:AuthnRequest>` message that was con-  
982 structed by an SP. Obviously, a WSC will not be able to sign `<lib:AuthnRequest>` messages on behalf of the  
983 party that will consume the `<lib:AuthnResponse>`. See the discussion in [Section 6.1: Conceptual Model](#) for  
984 context.
- 985 4. When the WSC receives security tokens, in the form of `<saml:Assertion>` elements or derivatives thereof,  
986 it MUST NOT send these to any other party than the intended audience, as indicated in the assertion's  
987 `<saml:Audience>` element.

## 988 7. Password Transformations: The PasswordTransforms Element

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

991 For example, an authentication server may have been configured such that it presumes that the strings users enter as  
992 their passwords have been pre-processed in some fashion before being further processed and/or stored. For example  
993 the passwords may be truncated to a given length, and all upper case characters may be folded to lower case, and  
994 whitespace may be eliminated. The authentication server can communicate these requirements dynamically to clients  
995 using the <PasswordTransforms> element in an initial <SASLResponse>. See [Figure 4](#).

```
996
997 <xs:element name="PasswordTransforms">
998
999   <xs:annotation>
1000     <xs:documentation>
1001       Contains ordered list of sequential password transformations
1002     </xs:documentation>
1003   </xs:annotation>
1004
1005   <xs:complexType>
1006     <xs:sequence>
1007
1008       <xs:element name="Transform" maxOccurs="unbounded">
1009         <xs:complexType>
1010           <xs:sequence>
1011
1012             <xs:element name="Parameter"
1013               minOccurs="0"
1014               maxOccurs="unbounded">
1015               <xs:complexType>
1016                 <xs:simpleContent>
1017                   <xs:extension base="xs:string">
1018                     <xs:attribute name="name"
1019                       type="xs:string"
1020                       use="required" />
1021                   </xs:extension>
1022                 </xs:simpleContent>
1023               </xs:complexType>
1024             </xs:element>
1025
1026           </xs:sequence>
1027
1028           <xs:attribute name="name"
1029             type="xs:anyURI"
1030             use="required" />
1031
1032           <xs:attribute name="id"
1033             type="xs:ID"
1034             use="optional" />
1035
1036         </xs:complexType>
1037       </xs:element>
1038     </xs:sequence>
1039   </xs:complexType>
1040 </xs:element>
1041
```

1042

**Figure 4. The PasswordTransforms element**

1043

```
1044 <PasswordTransforms>
1045   <Transform name="urn:liberty:sa:pm:truncate">
1046     <Parameter name="length">8</Parameter>
1047   </Transform>
1048   <Transform name="urn:liberty:sa:pm:lowercase" />
1049 </PasswordTransforms>
1050
```

1051

**Figure 5. Example of a PasswordTransforms**

1052 Servers MAY include a <PasswordTransforms> element along with their **initial** <SASLResponse> to a client. A  
1053 <PasswordTransforms> element contains one or more <Transform> elements. Each <Transform> is identified  
1054 by the value of the name attribute which must be a URI [RFC2396]. This URI MUST specify a particular transforma-  
1055 tion on the password. Transforms are specified elsewhere, for example in configuration data at implementation- and/or  
1056 deployment-time. A basic set is specified in [Appendix B: Password Transformations](#).

1057 A client receiving an **initial** <SASLResponse> message containing a <PasswordTransforms> element MUST  
1058 apply the specified transformations to any password that is used as input for the SASL mechanism indicated in the  
1059 <SASLResponse>.

1060 The client MUST apply the transformations in the order given in the <PasswordTransforms> element, and MUST  
1061 apply each transform to the result of the preceding transform. Of course, the first transform MUST be applied to the  
1062 raw password.

1063 Unless the specification of a <Transform> states otherwise, it is specified in terms of [Unicode] *abstract characters*.  
1064 An abstract character is a character as rendered to a user. Since an abstract character may require more than one  
1065 octet to represent, there is not necessarily a one-to-one mapping between an abstract character, or sequence of abstract  
1066 characters, and its corresponding *coded character representation*.

1067 For example, if a truncation transform indicates, "truncate after the first eight characters", the characters after the  
1068 eighth abstract character should be removed; in some languages and character encodings this could mean that more  
1069 than 8 octets remain.

1070 See also [Appendix B](#).

## 1071 **8. Acknowledgments**

1072 This spec leverages techniques and ideas from draft-nystrom-http-sasl-xx (an IETF Internet-Draft), RFC3080,  
1073 RFC2251, RFC2829, RFC2830, et al (all are various IETF Requests For Comments). The authors of those specs  
1074 are gratefully acknowledged. Thanks also to Alexy Melnikov, Paul Madsen, Scott Cantor, and RL "Bob" Morgan  
1075 for their feedback and insights. The docbook source code for this specification was hand set to the tunes of Brad,  
1076 Bob Mould, Weather Report, Miles Davis, John Coltrane, Liz Phair, The Wallflowers, Alan Holdsworth, Chick Corea,  
1077 Jennifer Trynin, Elisa Korenne, The Cowboy Junkies, Fugazi, Blues Traveler, Blink-182, CSN, Pearl Jam, and various  
1078 others. Thanks also to whatever deities are responsible for the existence of coffee, dark chocolate, and fermented  
1079 cereals.

# References

1080

## Normative

1081

1082 [LibertyAuthnContext] Madsen, Paul, eds. "Liberty ID-FF Authentication Context Specification," Version 1.3, Liberty  
1083 Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>

1084 [LibertyBindProf] Cantor, Scott, Kemp, John, Champagne, Darryl, eds. "Liberty ID-FF Bindings and  
1085 Profiles Specification," Version 1.2-errata-v2.0, Liberty Alliance Project (12 September 2004).  
1086 <http://www.projectliberty.org/specs>

1087 [LibertyClientProfiles] Aarts, Robert, eds. (14 December 2004). "Liberty ID-WSF Profiles for Liberty enabled User  
1088 Agents and Devices," version 1.1, Liberty Alliance Project <http://www.projectliberty.org/specs/>

1089 [LibertyDisco] Sergent, Jonathan, eds. "Liberty ID-WSF Discovery Service Specification," Version 1.2, Liberty  
1090 Alliance Project (12 December 2004). <http://www.projectliberty.org/specs>

1091 [LibertyInteract] Aarts, Robert, eds. "Liberty ID-WSF Interaction Service Specification," Version 1.1, Liberty Alliance  
1092 Project (14 December 2004). <http://www.projectliberty.org/specs>

1093 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 1.1, Liberty  
1094 Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>

1095 [LibertyPAOS] Aarts, Robert, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version 1.1, Liberty  
1096 Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>

1097 [LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version  
1098 1.2-errata-v3.0, Liberty Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>

1099 [LibertySecMech] Ellison, Gary, eds. "Liberty ID-WSF Security Mechanisms," Version 1.2, Liberty Alliance Project  
1100 (14 December 2004). <http://www.projectliberty.org/specs>

1101 [LibertyGlossary] "Liberty Technical Glossary," Version 1.4, Liberty Alliance Project (14 Dec 2004).  
1102 <http://www.projectliberty.org/specs> Hodges, Jeff, eds.

1103 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, eds. "Liberty ID-WSF SOAP Binding Specification  
1104 ," Version 1.2, Liberty Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>

1105 [RFC2119] Bradner, S., eds. "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet  
1106 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt> [March 1997].

1107 [RFC2222] "Simple Authentication and Security Layer (SASL)," John G. Myers (October 1997). RFC 2222, Internet  
1108 Engineering Task Force <http://www.ietf.org/rfc/rfc2222.txt> [October 1997].

1109 [RFC2246] Dierks, T., Allen, C., eds. (January 1999). "The TLS Protocol," Version 1.0 RFC 2246, Internet  
1110 Engineering Task Force <http://www.ietf.org/rfc/rfc2246.txt> [January 1999].

1111 [RFC2396] Berners-Lee, T., Fielding, R., Masinter, L., eds. (August 1998). "Uniform Resource Identifiers (URI):  
1112 Generic Syntax," RFC 2396, The Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2396.txt>  
1113 [August 1998].

1114 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., eds. (June  
1115 1999). "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, The Internet Engineering Task Force  
1116 <http://www.ietf.org/rfc/rfc2616.txt> [June 1999].

1117 [RFC3066] Alvestrand, H., eds. (January 2001). "Tags for the Identification of Languages," RFC 3066., Internet  
1118 Engineering Task Force <http://www.ietf.org/rfc/rfc3066.txt> [January 2001].

- 1119 [RFC3546] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., Wright, T., eds. (June 2003).  
1120 "Transport Layer Security (TLS) Extensions," RFC 3546, The Internet Engineering Task Force  
1121 <http://www.ietf.org/rfc/rfc3546.txt> [June 2003].
- 1122 [SAMLGloss] Hodges, Jeff, Maler, Eve, eds. (05 November 2002). "Glossary for the OASIS Security Assertion  
1123 Markup Language (SAML)," Version 1.0, OASIS Standard, Organization for the Advancement of Structured  
1124 Information Standards <http://www.oasis-open.org/committees/security/#documents>
- 1125 [SASLReg] "Simple Authentication and Security Layer (SASL) Mechanisms ," Internet Assigned Numbers Authority  
1126 (IANA) <http://www.iana.org/assignments/sasl-mechanisms>
- 1127 [Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May  
1128 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium  
1129 <http://www.w3.org/TR/xmlschema-1/>
- 1130 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David , Kakivaya, Gopal, Layman,  
1131 Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C  
1132 Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 1133 [Unicode] The Unicode Consortium (2003). "The Unicode Standard, version 4.0," Addison-Wesley *Unicode 4.0.0*  
1134 [<http://www.unicode.org>]
- 1135 [WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Mered-  
1136 ith, Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001).  
1137 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- 1138 **Informational**
- 1139 [IANA] "The Internet Assigned Numbers Authority," <http://www.iana.org/>
- 1140 [LibertyIDPP] Kellomaki, Sampo, eds. "Liberty Identity Personal Profile Service Specification," Version 1.0, Liberty  
1141 Alliance Project (12 November 2003). <http://www.projectliberty.org/specs>
- 1142 [LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework  
1143 Overview," Version 1.1, Liberty Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 1144 [Merriam-Webster] "Merriam-Webster Dictionary," <http://www.merriam-webster.com/>
- 1145 [RFC2245] "Anonymous SASL Mechanism," C. Newman (November 1997). RFC 2245, Internet Engineering Task  
1146 Force <http://www.ietf.org/rfc/rfc2245.txt> [November 1997].
- 1147 [RFC2289] "A One-Time Password System," N. Haller C. Metz P. Nessner M. Straw (February 1998). RFC 2289,  
1148 Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2289.txt> [February 1998].
- 1149 [RFC2444] Newman, C., eds. (October 1998). "The One-Time-Password SASL Mechanism," RFC 2444, The Internet  
1150 Engineering Task Force <http://www.ietf.org/rfc/rfc2444.txt> [October 1998].
- 1151 [RFC2828] Shirey, R., eds. (May 2000). "Internet Security Glossary," RFC 2828., Internet Engineering Task Force  
1152 <http://www.ietf.org/rfc/rfc2828.txt> [May 2000].
- 1153 [RFC3163] Zuccherato, R., Nystrom, M., eds. (August 2001). "ISO/IEC 9798-3 Authentication SASL Mechanism,"  
1154 RFC 3163, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc3163.txt> [August 2001].
- 1155 [SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (27 May 2003). "Assertions and Protocol  
1156 for the OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification,  
1157 version 1.1, Organization for the Advancement of Structured Information Standards [http://www.oasis-  
1158 open.org/committees/documents.php?wg\\_abbrev=security](http://www.oasis-open.org/committees/documents.php?wg_abbrev=security)

- 1159 [SOAPv1.2] "SOAP Version 1.2 Part 1: Messaging Framework," Gudgin, Martin, Hadley, Marc, Mendelsohn,  
1160 Noah, Moreau, Jean-Jacques, Nielsen, Henrik Frystyk, eds. World Wide Web Consortium W3C Proposed  
1161 Recommendation (07 May 2003). <http://www.w3.org/TR/2003/PR-soap12-part1-20030507/>
- 1162 [TrustInCyberspace] Schneider, Fred B., eds. "Trust in Cyberspace," National Research Council (1999).  
1163 <http://www.nap.edu/readingroom/books/trust/>
- 1164 [WooLam92] "Authentication for Distributed Systems," Thomas Y. C. Woo Simon S. Lam (January, 1992). IEEE  
1165 <http://citeseer.nj.nec.com/woo92authentication.html>
- 1166 [wss-sms] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (January, 2004). "Web  
1167 Services Security: SOAP Message Security," OASIS Standard V1.0 [OASIS 200401], Organization for the  
1168 Advancement of Structured Information Standards [http://docs.oasis-open.org/wss/2004/01/oasis-200401-](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)  
1169 [wss-soap-message-security-1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)
- 1170 [wss-saml] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (December 1, 2004). Orga-  
1171 nization for the Advancement of Structured Information Standards [http://docs.oasis-open.org/wss/oasis-wss-](http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf)  
1172 [saml-token-profile-1.0.pdf](http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf) "Web Services Security: SAML Token Profile," OASIS Standard V1.0 [OASIS  
1173 200412],
- 1174 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C.M., Maler, Eve, eds. (Oct 2000). "Extensible  
1175 Markup Language (XML) 1.0 (Second Edition)," Recommendation, World Wide Web Consortium  
1176 <http://www.w3.org/TR/2000/REC-xml-20001006>

## 1177 A. Listing of Simple Authentication and Security Layer (SASL) Mecha- 1178 nisms

1179 Ref: [\[SASLReg\]](#)

### 1180 Note:

1181 The file listed below IS SUBJECT TO CHANGE! It is presented here as non-normative background  
1182 information only. Implementers and deployers should always retrieve the a fresh copy of this file from  
1183 [\[IANA\]](#).

1184

1185

1186 SIMPLE AUTHENTICATION AND SECURITY LAYER (SASL) MECHANISMS

1187 -----

1188

1189 (last updated 2004-01-21)

1190

1191 The Simple Authentication and Security Layer (SASL) [RFC2222] is a  
1192 method for adding authentication support to connection-based  
1193 protocols. To use this specification, a protocol includes a command  
1194 for identifying and authenticating a user to a server and for  
1195 optionally negotiating a security layer for subsequent protocol  
1196 interactions. The command has a required argument identifying a SASL  
1197 mechanism.

1198

1199 SASL mechanisms are named by strings, from 1 to 20 characters in  
1200 length, consisting of upper-case letters, digits, hyphens, and/or  
1201 underscores. SASL mechanism names must be registered with the IANA.  
1202 Procedures for registering new SASL mechanisms are given in the  
1203 section "Registration procedures" of RFC2222.

1204

1205 MECHANISMS	USAGE	REFERENCE	OWNER
1206 -----	-----	-----	-----
1207 KERBEROS_V4	LIMITED	[RFC2222]	IESG <iesg@ietf.org>
1208			
1209 GSSAPI	COMMON	[RFC2222]	IESG <iesg@ietf.org>
1210			
1211 SKEY	OBSOLETE	[RFC2444]	IESG <iesg@ietf.org>
1212			
1213 EXTERNAL	COMMON	[RFC2222]	IESG <iesg@ietf.org>
1214			
1215 CRAM-MD5	LIMITED	[RFC2195]	IESG <iesg@ietf.org>
1216			
1217 ANONYMOUS	COMMON	[RFC2245]	IESG <iesg@ietf.org>
1218			
1219 OTP	COMMON	[RFC2444]	IESG <iesg@ietf.org>
1220			
1221 GSS-SPNEGO	LIMITED	[Leach]	Paul Leach <paulle@microsoft.com>
1222			
1223 PLAIN	COMMON	[RFC2595]	IESG <iesg@ietf.org>
1224			
1225 SECURID	COMMON	[RFC2808]	Magnus Nystrom <magnus@rsasecurity.com>
1226			
1227 NTLM	LIMITED	[Leach]	Paul Leach <paulle@microsoft.com>
1228			
1229 NMAS_LOGIN	LIMITED	[Gayman]	Mark G. Gayman <mgayman@novell.com>
1230			
1231 NMAS_AUTHEN	LIMITED	[Gayman]	Mark G. Gayman <mgayman@novell.com>
1232			
1233 DIGEST-MD5	COMMON	[RFC2831]	IESG <iesg@ietf.org>
1234			
1235 9798-U-RSA-SHA1-ENC	COMMON	[RFC3163]	robert.zuccherato@entrust.com
1236			
1237 9798-M-RSA-SHA1-ENC	COMMON	[RFC3163]	robert.zuccherato@entrust.com

1238  
1239 9798-U-DSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com  
1240  
1241 9798-M-DSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com  
1242  
1243 9798-U-ECDSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com  
1244  
1245 9798-M-ECDSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com  
1246  
1247 KERBEROS\_V5 COMMON [Josefsson] Simon Josefsson <simon@josefsson.org>  
1248  
1249  
1250 References  
1251 -----  
1252  
1253 [RFC2222] Myers, J., "Simple Authentication and Security Layer  
1254 (SASL)", RFC 2222, Netscape Communications, October 1997.  
1255  
1256 [RFC2195] Klensin, J., Catoe, R., Krumviede, P. "IMAP/POP AUTHorize  
1257 Extension for Simple Challenge/Response", RFC 2195, MCI,  
1258 September 1997.  
1259  
1260 [RFC2245] Newman, C., "Anonymous SASL Mechanism", RFC 2245, Innosoft,  
1261 November 1997.  
1262  
1263 [RFC2444] Newman, C., "The One-Time-Password SASL Mechanism", RFC  
1264 2444, October 1998.  
1265  
1266 [RFC2595] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595,  
1267 Innosoft, June 1999.  
1268  
1269 [RFC2808] Nystrom, M., "The SecurID(r) SASL Mechanism", RFC 2808,  
1270 April 2000.  
1271  
1272 [RFC2831] Leach, P. and C. Newman, "Using Digest Authentication as a  
1273 SASL Mechanism", RFC 2831, May 2000.  
1274  
1275  
1276 [RFC3163] R. Zuccherato and M. Nystrom, "ISO/IEC 9798-3 Authentication  
1277 SASL Mechanism", RFC 3163, August 2001.  
1278  
1279  
1280  
1281 People  
1282 -----  
1283  
1284 [Gayman] Mark G. Gayman, <mgayman@novell.com>, September 2000.  
1285  
1286 [Josefsson] Simon Josefsson, <simon@josefsson.org>, January 2004.  
1287  
1288 [Leach] Paul Leach, <paulle@microsoft.com>, December 1998, June 2000.  
1289  
1290 []  
1291  
1292

## 1293 B. Password Transformations

1294 This section defines a number of password transformations.

### 1295 1. Truncation

1296 The `urn:liberty:sa:pw:truncate` transformation instructs processors to remove all (Unicode abstract) sub-  
1297 sequent characters after a given number of characters have been obtained (from the user). Subsequent processing  
1298 MUST take only the given number of characters as input. The number of characters that shall remain is given in a  
1299 `<Parameter>` element with name "length".

```
1300  
1301 <Transform name="urn:liberty:sa:pw:truncate">  
1302   <Parameter name="length">8</Parameter>  
1303 </Transform>  
1304
```

1305 **Figure B.1. Example of truncation transformation**

### 1306 2. Lowercase

1307 The `urn:liberty:sa:pw:lowercase` transformation instructs processors to replace all uppercase characters with  
1308 lowercase characters. Characters that do not have case must remain unchanged. This transformation has no parameters.  
1309 Note that the "case" of the abstract Unicode character is decisive, i.e. only characters that have the `UPPERCASE`  
1310 property should be replaced with equivalent characters with the `Lowercase` property. This mapping from `UPPERCASE`  
1311 to `lowercase` should confirm to the relevant sections (e.g. 4.2) of [\[Unicode\]](#).

```
1312  
1313 <Transform name="urn:liberty:sa:pw:lowercase" />  
1314
```

1315 **Figure B.2. Example of lowercase transformation**

### 1316 3. Uppercase

1317 The `urn:liberty:sa:pw:uppercase` transformation instructs processors to replace all lowercase characters with  
1318 uppercase characters. Characters that do not have case must remain unchanged. This transformation has no parameters.  
1319 Note that the "case" of the abstract Unicode character is decisive, i.e. only characters that have the `Lowercase`  
1320 property should be replaced with equivalent characters with the `UPPERCASE` property. This mapping from `lowercase`  
1321 to `UPPERCASE` should confirm to the relevant sections (e.g. 4.2) of [\[Unicode\]](#).

```
1322  
1323 <Transform name="urn:liberty:sa:pw:uppercase" />  
1324
```

1325 **Figure B.3. Example of uppercase transformation**

### 1326 4. Select

1327 The `urn:liberty:sa:pw:select` transformation instructs processors to remove all characters except those spec-  
1328 ified in the "allowed" parameter. Note that the allowed characters refer to abstract Unicode characters. In the  
1329 message that contains the `<Transform>` element these characters are encoded with the same encoding as used for the  
1330 xml document that contains the message (usually UTF-8).

```
1331  
1332 <Transform name="urn:liberty:sa:pw:select">  
1333   <Parameter name="allowed">0123456789abcdefghijklmnopqrstvwxyz</Parameter>  
1334 </Transform>  
1335
```

1336 **Figure B.4. Example of select transformation**

## 1337 C. lib-arch-authn-svc.xsd Schema Listing

```
1338
1339     <?xml version="1.0" encoding="UTF-8"?>
1340
1341 <xs:schema
1342   targetNamespace="urn:liberty:sa:2004-04"
1343   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
1344   xmlns:sa="urn:liberty:sa:2004-04"
1345   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1346   xmlns:lib="urn:liberty:iff:2003-08"
1347   xmlns:disco="urn:liberty:disco:2003-08"
1348   xmlns="urn:liberty:sa:2004-04"
1349   elementFormDefault="qualified"
1350   attributeFormDefault="unqualified"
1351   version="07"
1352 >
1353
1354 <!-- Filename: lib-arch-authn-svc.xsd -->
1355 <!-- $Id: lib-arch-authn-svc.xsd,v 1.4.4.1 2005/01/25 17:25:52 dchampagne Exp $ -->
1356 <!-- Author: Jeff Hodges -->
1357 <!-- Last editor: $Author: dchampagne $ -->
1358 <!-- $Date: 2005/01/25 17:25:52 $ -->
1359 <!-- $Revision: 1.4.4.1 $ -->
1360
1361
1362 <xs:import
1363   namespace="urn:liberty:iff:2003-08"
1364   schemaLocation="liberty-idff-protocols-schema-1.2-errata-v3.0.xsd"/>
1365
1366 <xs:import
1367   namespace="urn:liberty:disco:2003-08"
1368   schemaLocation="liberty-idwsf-disco-svc-v1.2.xsd"/>
1369
1370 <xs:include schemaLocation="liberty-idwsf-utility-v1.1.xsd"/>
1371
1372 <xs:annotation>
1373   <xs:documentation>
1374     Liberty ID-WSF Authentication Service XSD
1375   </xs:documentation>
1376   <xs:documentation>
1377     The source code in this XSD file was excerpted verbatim from:
1378
1379     Liberty ID-WSF Authentication Service Specification
1380     Version 1.1
1381     14 Dec 2004
1382
1383     Copyright (c) 2003-2005 Liberty Alliance participants,
1384     see http://www.projectliberty.org/specs/idwsf\_1\_1\_copyrights.php
1385
1386   </xs:documentation>
1387 </xs:annotation>
1388
1389
1390 <!-- SASLRequest and SASLResponse ID-* messages -->
1391
1392 <xs:element name="SASLRequest">
1393   <xs:complexType>
1394     <xs:sequence>
1395
1396       <xs:element name="Data" minOccurs="0">
1397         <xs:complexType>
1398           <xs:simpleContent>
1399             <xs:extension base="xs:base64Binary"/>
1400           </xs:simpleContent>
1401         </xs:complexType>
1402       </xs:element>

```

```
1403
1404     <xs:element ref="lib:RequestAuthnContext"
1405         minOccurs="0" />
1406
1407 </xs:sequence>
1408
1409 <xs:attribute name="mechanism"
1410     type="xs:string"
1411     use="required" />
1412
1413 <xs:attribute name="authzID"
1414     type="xs:string"
1415     use="optional" />
1416
1417 <xs:attribute name="advisoryAuthnID"
1418     type="xs:string"
1419     use="optional" />
1420
1421 <xs:attribute name="id"
1422     type="xs:ID"
1423     use="optional" />
1424
1425 </xs:complexType>
1426 </xs:element>
1427
1428
1429 <xs:element name="SASLResponse">
1430     <xs:complexType>
1431         <xs:sequence>
1432
1433             <xs:element ref="Status" />
1434
1435             <xs:element ref="PasswordTransforms" minOccurs="0" />
1436
1437             <xs:element name="Data" minOccurs="0">
1438                 <xs:complexType>
1439                     <xs:simpleContent>
1440                         <xs:extension base="xs:base64Binary" />
1441                     </xs:simpleContent>
1442                 </xs:complexType>
1443             </xs:element>
1444
1445             <xs:element ref="disco:ResourceOffering"
1446                 minOccurs="0"
1447                 maxOccurs="unbounded" />
1448
1449             <xs:element name="Credentials" minOccurs="0">
1450                 <xs:complexType>
1451                     <xs:sequence>
1452                         <xs:any namespace="##any"
1453                             processContents="lax"
1454                             minOccurs="0"
1455                             maxOccurs="unbounded" />
1456                     </xs:sequence>
1457                 </xs:complexType>
1458             </xs:element>
1459
1460         </xs:sequence>
1461
1462         <xs:attribute name="serverMechanism"
1463             type="xs:string"
1464             use="optional" />
1465
1466         <xs:attribute name="id"
1467             type="xs:ID"
1468             use="optional" />
1469     </xs:complexType>
</xs:element>
```

```
1470     </xs:complexType>
1471 </xs:element>
1472
1473
1474 <!-- Password Transformations -->
1475
1476 <xs:element name="PasswordTransforms">
1477
1478     <xs:annotation>
1479         <xs:documentation>
1480             Contains ordered list of sequential password transformations
1481         </xs:documentation>
1482     </xs:annotation>
1483
1484     <xs:complexType>
1485         <xs:sequence>
1486
1487             <xs:element name="Transform" maxOccurs="unbounded">
1488                 <xs:complexType>
1489                     <xs:sequence>
1490
1491                         <xs:element name="Parameter"
1492                             minOccurs="0"
1493                             maxOccurs="unbounded">
1494                             <xs:complexType>
1495                                 <xs:simpleContent>
1496                                     <xs:extension base="xs:string">
1497                                         <xs:attribute name="name"
1498                                             type="xs:string"
1499                                             use="required" />
1500                                     </xs:extension>
1501                                 </xs:simpleContent>
1502                             </xs:complexType>
1503                         </xs:element>
1504
1505                     </xs:sequence>
1506
1507                     <xs:attribute name="name"
1508                         type="xs:anyURI"
1509                         use="required" />
1510
1511                     <xs:attribute name="id"
1512                         type="xs:ID"
1513                         use="optional" />
1514
1515                 </xs:complexType>
1516             </xs:element>
1517         </xs:sequence>
1518     </xs:complexType>
1519 </xs:element>
1520
1521
1522
1523 </xs:schema>
1524
1525
1526
```

## 1527 D. lib-arch-iwsf-utility.xsd Schema Listing

```
1528
1529     <?xml version="1.0" encoding="UTF-8"?>
1530 <!-- filename: liberty-idwsf-utility-v1.1.xsd -->
1531
1532 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
1533     elementFormDefault="qualified" attributeFormDefault="unqualified">
1534     <xs:annotation>
1535         <xs:documentation>
1536             Liberty Alliance Project utility schema. A collection of common
1537             Identity Web Services Framework (ID-WSF) elements and types.
1538             This schema is intended for use in ID-WSF schemas.
1539
1540             Copyright 2003-2005 Liberty Alliance Project, see
1541             http://www.projectliberty.org/specs/idwsf_1_1_copyrights.php
1542
1543             This file intended for inclusion, rather than importation,
1544             into other schemas.
1545
1546             This version: 2004-12-14
1547
1548         </xs:documentation>
1549     </xs:annotation>
1550     <xs:simpleType name="IDType">
1551         <xs:annotation>
1552             <xs:documentation>This type should be used to provided IDs to components that have IDs that
1553 may not be scoped within the local xml instance document. </xs:documentation>
1554         </xs:annotation>
1555         <xs:restriction base="xs:string"/>
1556     </xs:simpleType>
1557     <xs:simpleType name="IDReferenceType">
1558         <xs:annotation>
1559             <xs:documentation> This type can be used when referring to elements that are
1560             identified using an IDType </xs:documentation>
1561         </xs:annotation>
1562         <xs:restriction base="xs:string"/>
1563     </xs:simpleType>
1564     <xs:element name="Status" type="StatusType">
1565         <xs:annotation>
1566             <xs:documentation> A standard Status type</xs:documentation>
1567         </xs:annotation>
1568     </xs:element>
1569     <xs:complexType name="StatusType">
1570         <xs:annotation>
1571             <xs:documentation> A type that may be used for status codes. </xs:documentation>
1572         </xs:annotation>
1573         <xs:sequence>
1574             <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
1575         </xs:sequence>
1576         <xs:attribute name="code" type="xs:QName" use="required"/>
1577         <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
1578         <xs:attribute name="comment" type="xs:string" use="optional"/>
1579     </xs:complexType>
1580     <xs:complexType name="EmptyType">
1581         <xs:annotation>
1582             <xs:documentation> This type may be used to create an empty element </xs:documentation>
1583         </xs:annotation>
1584         <xs:complexContent>
1585             <xs:restriction base="xs:anyType"/>
1586         </xs:complexContent>
1587     </xs:complexType>
1588     <xs:element name="Extension" type="extensionType">
1589         <xs:annotation>
1590             <xs:documentation>An element that contains arbitrary content extensions from other
1591 namespaces</xs:documentation>
1592         </xs:annotation>
```

```
1593     </xs:element>
1594     <xs:complexType name="extensionType">
1595         <xs:annotation>
1596             <xs:documentation>A type for arbitrary content extensions from other_
1597 namespaces</xs:documentation>
1598         </xs:annotation>
1599         <xs:sequence>
1600             <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded" />
1601         </xs:sequence>
1602     </xs:complexType>
1603 </xs:schema>
1604
1605
```