



Liberty IDP Service Specification

Version: 1.0

Editors:

Conor P. Cahill, Intel Corporation

Contributors:

Shelagh Callahan, Intel Corporation

Jane Dashevsky, Intel Corporation

Tapio Kaukonen, Nokia

Sampo Kellomäki, Symlabs, Inc.

Hubert Le Van Gong, Sun

Andrew Lindsay-Stewart, Vodafone

Paul Madsen, NTT

Paul Miller, Gemplus

Hiroyoshi Takiguchi, NTT

Pierre Vannel, Gemplus

Sean Walker, Axalto

Abstract:

This specification describes the ID-WSF IDP Service.

Filename: liberty-idwsf-idp-v1.0.pdf

1 **Notice**

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

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

15 Copyright © 2007 Adobe Systems; Agencia Catalana De Certificacio; America Online, Inc.; American Express
16 Company; Amsoft Systems Pvt Ltd.; Avatier Corporation; BIPAC; BMC Software, Inc.; Bank of America
17 Corporation; Beta Systems Software AG; British Telecommunications plc; Computer Associates International, Inc.;
18 Credentica; Dan Combs; Danish National IT and Telecom Agency; DataPower Technology, Inc.; Deutsche Telekom
19 AG, T-Com; Diamelle Technologies, Inc.; Diversinet Corp.; Drummond Group Inc.; Enosis Group LLC; Entrust,
20 Inc.; Epok, Inc.; Ericsson; Falkin Systems LLC; Fidelity Investments; Forum Systems, Inc.; France Télécom; French
21 Government Agence pour le développement de l'administration électronique (ADAE); Fugen Solutions, Inc; Fulvens
22 Ltd.; GSA Office of Governmentwide Policy; Gamefederation; Gemalto; General Motors; GeoFederation; Giesecke
23 & Devrient GmbH; Guy Huntington; Hewlett-Packard Company; Hochhauser & Co., LLC; IBM Corporation; Intel
24 Corporation; Intuit Inc.; Kantega; Kayak Interactive; Livo Technologies; Luminance Consulting Services; Mark
25 Wahl; Mary Ruddy; MasterCard International; MedCommons Inc.; Mobile Telephone Networks (Pty) Ltd; Mortgage
26 Bankers Association (MBA); NEC Corporation; NTT DoCoMo, Inc.; Netegrity, Inc.; NHK (Japan Broadcasting
27 Corporation) Science & Technical Research Laboratories; Neustar, Inc.; New Zealand Government State Services
28 Commission; Nippon Telegraph and Telephone Corporation; Nokia Corporation; Novell, Inc.; OpenNetwork; Oracle
29 Corporation; Ping Identity Corporation; Postsecondary Electronic Standards Council (PESC); RSA Security Inc.;
30 Reach; Reactivity Inc.; Rob Marano; Royal Mail Group plc; SAP AG; SanDisk Corporation; Senforce; Sharp
31 Laboratories of America; Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Supremacy Financial
32 Corporation; Symlabs, Inc.; Telecom Italia S.p.A.; Telefónica Móviles, S.A.; Telenor R&D; Thales e-Security;
33 Trusted Network Technologies; UNINETT AS; UTI; VeriSign, Inc.; Vodafone Group Plc.; Wave Systems Corp.
34 Wells Fargo; All rights reserved.

35 Liberty Alliance Project
36 Licensing Administrator
37 c/o IEEE-ISTO
38 445 Hoes Lane
39 Piscataway, NJ 08855-1331, USA
40 info@projectliberty.org

41 Contents

42	1. Introduction	4
43	1.1. Notation and Conventions	4
44	1.1.1. XML Namespaces	4
45	2. Data Definitions	5
46	2.1. Common Provider ID	5
47	2.2. Minting Assertion (MING)	5
48	2.2.1. MING Creation Rules	8
49	2.2.2. MING Processing Rules	8
50	2.3. Minted Assertion (MED)	9
51	2.3.1. MED Creation Rules	10
52	2.3.2. MED Processing Rules	10
53	2.4. <ProviderInfo> element	10
54	3. Identity Provider Service (IdP)	12
55	3.1. Service URIs	12
56	3.2. Status Codes	12
57	3.3. Operation: <i>GetAssertion</i>	12
58	3.3.1. wsa:Action values for <i>GetAssertion</i> Messages	12
59	3.3.2. <i>GetAssertion</i> Message	12
60	3.3.3. <i>GetAssertionResponse</i> Message	15
61	3.3.4. <i>GetAssertion</i> Processing Rules	18
62	3.4. Operation: <i>GetProviderInfo</i>	19
63	3.4.1. wsa:Action values for <i>GetProviderInfo</i> Messages	19
64	3.4.2. <i>GetProviderInfo</i> Message	20
65	3.4.3. <i>GetProviderInfoResponse</i> Message	21
66	3.4.4. <i>GetProviderInfo</i> Processing Rules	21
67	3.5. Operation: <i>CreatedStatus</i>	22
68	3.5.1. wsa:Action values for <i>CreatedStatus</i> Messages	22
69	3.5.2. <i>CreatedStatus</i> Message	22
70	3.5.3. <i>CreatedStatusResponse</i> Message	23
71	3.5.4. <i>CreatedStatus</i> Processing Rules	23
72	4. IDP Service Schema	25
73	5. IDP Service WSDL	29
74	References	31

75 **1. Introduction**

76 IdP Service Introduction

77 **1.1. Notation and Conventions**

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

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

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

85 **1.1.1. XML Namespaces**

86 The following XML namespaces are referred to in this document:

87 • The prefix *idp:* represents the IdP Service namespace. This namespace is the default for instance fragments, type
88 names, and element names in this document. In schema listings, and in examples of IdP Service messages and
89 fragments thereof, this is the default namespace *when* no prefix is shown:

90 *urn:liberty:idp:2007-09*

91 • The prefix *saml2:* stands for the SAMLv2 assertion namespace [\[SAMLCore2\]](#):

92 *urn:oasis:names:tc:SAML:2.0:assertion*

93 • The prefix *samlp2:* stands for the SAMLv2 protocol namespace [\[SAMLCore2\]](#):

94 *urn:oasis:names:tc:SAML:2.0:protocol*

95 • The prefix *xs:* stands for the W3C XML schema namespace [\[Schema1-2\]](#):

96 *http://www.w3.org/2001/XMLSchema*

97 • The prefix *xsi:* stands for the W3C XML schema instance namespace:

98 *http://www.w3.org/2001/XMLSchema-instance*

99 2. Data Definitions

100 2.1. Common Provider ID

101 The URI *urn:liberty:idp:2007-09:ProviderID:Common* is a reserved URI which represents a globally shared identifier
102 which may be used by clients in place of a unique *ProviderID* when the execution context of a request requires the
103 privacy protections offered by the common identifier.

104 The Liberty ID-WSF Advanced Client Technologies Overview ([\[LibertyACT\]](#)) contains a complete discussion about
105 the privacy and maintenance concerns surrounding the selection of a *ProviderID* for clients issuing authentication
106 assertions.

107 2.2. Minting Assertion (MING)

108 The **minting assertion** (MING) is a SAML assertion issued by an IdP granting a TM permission to mint assertions on
109 behalf of the IdP. This assertion will be built from standard SAML Assertion elements including:

- 110 • The `<saml2:Subject>` **MUST** be the TM, with the TM's *ProviderID* in the `<saml2:NameID>` sub-element and
111 with one of the TM's public keys in the `<saml2:SubjectConfirmation>` sub-element.

112 An example `<Subject>`:

```
113     <saml2:Subject>
114       <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
115         urn:liberty:idp:2007-09:ProviderID:Common
116       </saml2:NameID>
117       <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
118         <saml2:SubjectConfirmationData xsi:type="saml2:KeyInfoConfirmationDataType">
119           <ds:KeyInfo> TM Public Key Info </ds:KeyInfo>
120         </saml2:SubjectConfirmationData>
121       </saml2:SubjectConfirmation>
122     </saml2:Subject>
123
```

124 In this case, the TM is using the common provider ID (likely due to some privacy requirement).

- 125 • The SAML 2.0 standard `<saml2:Condition>`s apply to the MING itself (and indirectly impact the MED since
126 the MED is not valid unless the MING is valid). In addition, there are new, MING-specific, conditions added to
127 restrict some of the properties of the MED.

128 The new condition types are:

- 129 • `<idp:SubjectRestrictionType>` - this condition restricts the possible subjects that can be specified in any
130 MED minted using this MING. The schema for the `<idp:SubjectRestrictionType>` is as follows:

```
131     <!-- SubjectRestriction - MING condition restricting Subjects in MED -->
132
133     <xs:complexType name="SubjectRestrictionType">
134       <xs:complexContent>
135         <xs:extension base="saml2:ConditionAbstractType">
136           <xs:sequence>
137             <xs:element ref="saml2:Subject" maxOccurs="unbounded"/>
138           </xs:sequence>
139         </xs:extension>
140       </xs:complexContent>
141     </xs:complexType>
142
```

143 Multiple `<saml2:Subject>`s **MAY** be specified allowing a single subject-restricted MING to be used for
144 different MEDs for different subjects at different providers.

145 • <idp:AuthnContextRestrictionType> - this condition restricts the possible authentica-
146 tion contexts that can be specified in any MED minted using this MING. The schema for the
147 <idp:AuthnContextRestrictionType> is as follows:

```
148 <!-- AuthnContextRestriction - Ming Condition - restricts authncontext in MED -->
149
150 <xs:complexType name="AuthnContextRestrictionType">
151   <xs:complexContent>
152     <xs:extension base="saml2:ConditionAbstractType">
153       <xs:sequence>
154         <xs:element ref="saml2:AuthnContext" maxOccurs="unbounded"/>
155       </xs:sequence>
156     </xs:extension>
157   </xs:complexContent>
158 </xs:complexType>
159
```

160 Multiple <saml2:AuthnContext>s MAY be specified allowing a single AuthnContext-restricted MING to
161 be used for different MEDs with different AuthnContexts.

162 These new types are used by specifying a <saml2:Conditions> element with the xsi:type attribute
163 set to one of the listed types above (and then the content of that <saml2:Conditions> element follows the
164 definition of the extended type above.

165 • An <saml2:AuthzDecisionStatement> MUST be included with the following contents:

166 • The Resource attribute MUST be set to "urn:oasis:names:tc:SAML:2.0:assertion".

167 • The Decision attribute MUST be set to "Permit".

168 • The <Action> element's Namespace attribute MUST be set to "urn:liberty:idp:2007-09:Actions" and the
169 value of the element MUST be "Mint".

170 An example <saml2:AuthzDecisionStatement>:

```
171 <saml2:AuthzDecisionStatement
172   Resource="urn:oasis:names:tc:SAML:2.0:assertion"
173   Decision="Permit">
174   <saml2:Action Namespace="urn:liberty:idp:2007-09:Actions">Mint</saml2:Action>
175 </saml2:AuthzDecisionStatement>
176
```

177 An example MING:

```
178 <saml2:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
179   ID="sxJu9gvvLG9sAN9bKp8q0NKU"
180   IssueInstant="2006-05-01T17:20:30.213Z"
181   Version="2.0">
182
183 <saml2:Issuer>http://idp.example.com</saml2:Issuer>
184 <ds:Signature>IdP's signature data goes here</ds:Signature>
185
186 <saml2:Subject>
187   <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
188     urn:liberty:idp:2007-09:ProviderID:Common
189   </saml2:NameID>
190   <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
191     <saml2:SubjectConfirmationData>
192       <ds:KeyInfo>PublicKey info for TM</ds:KeyInfo>
193     </saml2:SubjectConfirmationData>
194   </saml2:SubjectConfirmation>
195 </saml2:Subject>
196
197 <saml2:Conditions NotBefore="2006-05-01T17:20:30.213Z"
198   NotOnOrAfter="2006-05-02T17:20:30.213Z">
```

```

199     <saml2:AudienceRestriction>
200         <saml2:Audience>Provider1</saml2:Audience>
201         <saml2:Audience>Provider2</saml2:Audience>
202     </saml2:AudienceRestriction>
203 </saml2:Conditions>
204
205 <saml2:AuthzDecisionStatement
206     Resource="urn:oasis:names:tc:SAML:2.0:assertion"
207     Decision="Permit">
208     <saml2:Action Namespace="urn:liberty:idp:2007-09:Actions">Mint</saml2:Action>
209 </saml2:AuthzDecisionStatement>
210 </saml2:Assertion>
211
212

```

213 Notes about this example:

- 214 • The TM is using a common ProviderID.
- 215 • The MING is restricted to either of 2 audiences.
- 216 • There is no Subject Restriction so the TM can use this MING for any valid subject it has for the specified relying
- 217 parties.

218 An example MING with a subject and AuthnContext restrictions:

```

219 <saml2:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
220     ID="sxJu9gvvLG9sAN9bKp8q0NKU"
221     IssueInstant="2006-05-01T17:20:30.213Z"
222     Version="2.0">
223
224     <saml2:Issuer>http://idp.example.com</saml2:Issuer>
225     <ds:Signature>IdP's signature data goes here</ds:Signature>
226
227     <saml2:Subject>
228         <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
229             urn:liberty:idp:2007-09:ProviderID:Common
230         </saml2:NameID>
231         <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
232             <saml2:SubjectConfirmationData>
233                 <ds:KeyInfo> PublicKey info for TM </ds:KeyInfo>
234             </saml2:SubjectConfirmationData>
235         </saml2:SubjectConfirmation>
236     </saml2:Subject>
237
238     <saml2:Conditions NotBefore="2006-05-01T17:20:30.213Z"
239         NotOnOrAfter="2006-05-02T17:20:30.213Z">
240         <saml2:Condition xsi:type="idp:SubjectRestrictionType">
241             <saml2:Subject>
242                 <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
243                     SPNameQualifier="Provider1">NameIDatProvider1</saml2:NameID>
244             </saml2:Subject>
245             <saml2:Subject>
246                 <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
247                     SPNameQualifier="Provider2">NameIDatProvider2</saml2:NameID>
248             </saml2:Subject>
249         </saml2:Condition>
250         <saml2:Condition xsi:type="idp:AuthnContextRestrictionType">
251             <saml2:AuthnContext>
252                 <saml2:AuthnContextClassRef>
253                     urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
254                 </saml2:AuthnContextClassRef>
255             </saml2:AuthnContext>
256             <saml2:AuthnContext>
257                 <saml2:AuthnContextClassRef>
258                     urn:oasis:names:tc:SAML:2.0:ac:classes:SmartcardPKI

```

```
259     </saml2:AuthnContextClassRef>
260   </saml2:AuthnContext>
261 </saml2:Condition>
262 </saml2:Conditions>
263
264 <saml2:AuthzDecisionStatement
265   Resource="urn:oasis:names:tc:SAML:2.0:assertion"
266   Decision="Permit">
267   <saml2:Action Namespace="urn:liberty:idp:2007-09:Actions">Mint</saml2:Action>
268 </saml2:AuthzDecisionStatement>
269 </saml2:Assertion>
270
271
```

272 Notes about this example:

- 273 • The TM is using a CommonProviderID.
- 274 • The MING is restricted to either of 2 Subjects, one at Provider1 and one at Provider2.
- 275 • There is no `<saml2:AudienceRestriction>` necessary as the `<saml2:Subject>` elements included the
276 SPNameQualifier attribute which has same result.
- 277 • The NameID elements were not encrypted. Normally, when NameIDs for multiple providers are included in an
278 assertion that may be sent to multiple providers, an EncryptedID would be used. Doing so here would have made
279 the example kind of big and ugly, so we didn't encrypt.
- 280 • The TM may generate MEDs that show the user authenticated with a password or with a Smartcard plus pin (the
281 IdP is depending upon the TM to do the right thing when selecting which of those classes to use). Note that, in
282 some cases, the TM, itself, can be within a Smartcard (or the equivalent) and this can be considered part of the
283 authentication context of the user.

284 2.2.1. MING Creation Rules

285 When creating a MING, the following rules apply:

- 286 • All of the SAML 2.0 rules for the generation of an assertion MUST be observed.
- 287 • The `<saml2:Subject>` of the MING MUST contain the ProviderID of the TM and the
288 `<saml2:SubjectConfirmation>` MUST be one that supports proof-of-possession via a digital signa-
289 ture (such as *urn:oasis:names:tc:SAML:2.0:cm:holder-of-key*)
- 290 • If desired, include, at most, one `<idp:SubjectRestriction>` `<idp:SubjectRestriction>` and/or, at
291 most, one `<idp:AuthnContextRestriction>` .
- 292 • If the assertion is to include a `<idp:SubjectRestriction>` condition and a `<saml2:AudienceRestriction>`
293 condition, there SHOULD be at least one `<saml2:Subject>` in the `<idp:SubjectRestriction>` for every
294 `<saml2:Audience>`.
- 295 • `<saml2:Subject>` elements within a `<idp:SubjectRestriction>` element MUST NOT include a
296 `<saml2:SubjectConfirmation>` element.

297 2.2.2. MING Processing Rules

298 The processing rules for accepting a MING include:

- 299 • All of the SAML 2.0 rules for processing and accepting an assertion **MUST** be observed (e.g., if the current time
300 is beyond the value in the `NotOnOrAfter` attribute on the `<saml2:Conditions>` element, the MING **MUST** be
301 considered invalid).
- 302 • If the `<saml2:Conditions>` element contains a `<idp:SubjectRestriction>` element, the
303 `<saml2:Subject>` of the MED (not the MING) **MUST strongly match** one of the `<saml2:Subject>`s
304 within the `<idp:SubjectRestriction>`. If this is not the case, the MED **MUST** be considered invalid.
- 305 • If the `<saml2:Conditions>` element contains a `<idp:AuthnContextRestriction>` . element, the
306 `<saml2:AuthnContext>` of the MED (not the MING) **MUST** match one of the `<saml2:AuthnContext>`s
307 within the `<idp:AuthnContextRestriction>`. If this is not the case, the MED **MUST** be considered invalid.
- 308 • The `<saml2:Conditions>` on the MED (not MING) **MUST** be at least as restrictive as the
309 `<saml2:Conditions>` on the MING (e.g., the MED **MUST NOT** have a `NotOnOrAfter` which is later
310 than the `NotOnOrAfter` in the MING). If any of the conditions are less restrictive, the MED **MUST** be
311 considered invalid.

312 2.3. Minted Assertion (MED)

313 The **minted assertion** (MED) is a SAML authentication assertion issued by a TM which includes a MING granting
314 the TM permission to mint the MED. This assertion is built from standard SAML elements including:

- 315 • The `<saml2:Issuer>` **MUST** contain the TM's ProviderID. The possibilities for the TM ProviderID are
316 discussed in the Client Identity section within the Liberty ID-WSF Advanced Client Technologies Overview
317 ([\[LibertyACT\]](#)).
- 318 • The `<saml2:Subject>` **SHOULD** contain a `<saml2:NameID>` which identifies the principal (not the TM). The
319 `NameQualifier` attribute **SHOULD** identify the IdP (as the identifier is issued in the namespace of the IdP, not
320 the TM).
- 321 • The `<saml2:Conditions>` **MAY** contain any values permitted by the MING (including, for example, a validity
322 period that is within the validity period of the MING).
- 323 • The MING **MUST** be included within the `<saml2:Advice>` element.
- 324 • All other SAML elements **MAY** be used, including Attribute Statements.

325 An example MED:

```

326 <saml2:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
327     ID="ID_1"
328     IssueInstant="2006-05-02T11:43:12.723Z"
329     Version="2.0">
330 <saml2:Issuer>urn:liberty:idp:2007-09:ProviderID:Common</saml2:Issuer>
331 <ds:Signature>
332     TM's signature data goes here - MUST be signed using the
333     the key identified by the keyinfo in the "MING"
334 </ds:Signature>
335 <saml2:Subject>
336     <saml2:NameID
337         NameQualifier="IdP's ProviderID"
338         SPNameQualifier="SP's Provider ID"
339         Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">
340         Principal's NameID at SP
341     </saml2:NameID>
342     <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
343         <saml2:SubjectConfirmationData InResponseTo="AuthnRequestID" />
344     </saml2:SubjectConfirmation>
345 </saml2:Subject>
346 <saml2:Conditions NotBefore="2006-05-02T11:43:12.723Z"

```

```
347         NotOnOrAfter="2006-05-02T11:48:12.723Z">
348     <saml2:AudienceRestriction>
349     <saml2:Audience>
350         SP's Provider ID
351     </saml2:Audience>
352 </saml2:AudienceRestriction>
353 </saml2:Conditions>
354 <saml2:Advice>
355     <saml2:Assertion ... >
356         MING assertion data here
357     </saml2:Assertion>
358 </saml2:Advice>
359 <saml2:AuthnStatement AuthnInstant="2006-05-02T11:43:12.723Z">
360     Authentication information here
361 </saml2:AuthnStatement>
362
363 </saml2:Assertion>
364
```

365 2.3.1. MED Creation Rules

366 When creating a MED, the following rules apply:

- 367 • All of the SAML 2.0 rules for the generation of an assertion **MUST** be observed.
- 368 • Restrict the `<saml2:Conditions>` and `<saml2:AuthnStatement>` element information to be within the
369 limits contained within MING.
- 370 • Include the MING within the `<saml2:Advice>` element.
- 371 • Sign the assertion (the MED) using the private key associated with the public key identified in the
372 `<saml2:SubjectConfirmation>` element of the MING.

373 2.3.2. MED Processing Rules

374 The processing rules for accepting a MED include:

- 375 • All of the SAML 2.0 rules for processing and accepting an assertion **MUST** be observed (e.g., if the current time
376 is beyond the value in the `NotOnOrAfter` attribute on the `<saml2:Conditions>` element, the MED **MUST** be
377 considered invalid).
- 378 • The relying party, seeing a `<saml2:Issuer>` it does not know, **SHOULD** locate the MING within the
379 `<saml2:Advice>` and validate the MING independently. If the MING is not valid, the MED **SHOULD NOT** be
380 considered valid.
- 381 • The MING must be present in the `<saml2:Advice>` and the MING itself **MUST** validate (also using all SAML2.0
382 rules for validation).
- 383 • If the MING `<saml2:Conditions>` element contains a `<idp:AuthnContextRestriction>` . element, the
384 `<saml2:AuthnContext>` of the MED (not the MING) **MUST** match one of the `<saml2:AuthnContext>`s
385 within the `<idp:AuthnContextRestriction>` in the MING. If this is not the case, the MED **MUST** be
386 considered invalid.
- 387 • If any of the `<saml2:Conditions>` in the MED are **LESS** restrictive (e.g., a later `NotOnOrAfter`) than the
388 `<saml2:Conditions>` in the MING, the MED **MUST** be considered invalid.

389 **2.4. <ProviderInfo> element**

390 The <ProviderInfo> element contains information describing a provider (name, ProviderID, etc.). This
391 information is commonly used by clients to provide the user with a selection list of the providers with which the
392 user may want to interact.

393 The <ProviderInfo> element contains the following attributes/elements:

394 • providerID [required]

395 The ProviderID for the provider.

396 • name [optional]

397 The name of the provider (typically used by the caller to allow the principal to select which providers that they
398 anticipate using as the name is more understandable than the providerID).

399 The schema for the <ProviderInfo> element is shown below.

```
400 <!-- ProviderInfo - Information about providers returned by GetProviderInfo -->  
401  
402 <xs:element name="ProviderInfo" type="ProviderInfoType"/>  
403  
404 <xs:complexType name="ProviderInfoType">  
405   <xs:attribute name="providerID" type="xs:anyURI" use="required"/>  
406   <xs:attribute name="name" type="xs:string" use="optional"/>  
407 </xs:complexType>  
408
```

409 **Figure 1. <ProviderInfo> — Schema Fragment**

410 An example <ProviderInfo> element for the Acme Profile Service follows.

```
411 <idp:ProviderInfo providerID="http://services.acme.com/Profile"  
412   name="ACME Profile Service" />  
413
```

414 **Example 1. Example <ProviderInfo> Element**

415 **3. Identity Provider Service (IdP)**

416 An IdP Service is a web service interface to an Identity Provider. This service provides interfaces which provide
417 assertions and provider information to a client so that the client can facilitate SSO operations with relying parties.

418 An abstract WSDL definition for the IdP Service is included in this document, see [Section 5: IDP Service WSDL](#) .
419 This WSDL document defines all of the "WSDL operations" for the IdP Service.

420 The complete schema for the IdP Service is included in this document, see [Section 4: IDP Service Schema](#) .

421 **3.1. Service URIs**

422 **Table 1. IdP Service URIs**

Use	URI
Service Type	<i>urn:liberty:idp:2007-09</i>
GetAssertion wsa:Action	<i>urn:liberty:idp:2007-09:GetAssertion</i>
GetProviderInfo wsa:Action	<i>urn:liberty:idp:2007-09:GetProviderInfo</i>

423 **3.2. Status Codes**

424 The following status code strings are defined:

- 425 • *OK*: message processing succeeded.
- 426 • *Failed*: general failure code.
- 427 • *Partial*: some portion of the request succeed and some failed. See the response element for more details.
- 428 • *NoResults*: the query had no matching results.
- 429 • *NotFound*: the specified item(s) were not found.

430 These strings are expected to appear in the "code" attribute of <Status> elements used in SOAP-bound IdP Service
431 protocol messages [LibertySOAPBinding]. Specific uses for the status codes are defined in the processing rules for
432 individual messages. The "ref" attribute on the <Status> element is not used in this specification, so it MUST NOT
433 appear on Status elements in IdP Service protocol messages. The contents of the comment attribute are not defined by
434 this specification, but it may be used for additional descriptive text intended for human consumption (for example, to
435 carry information that will aid debugging).

436 3.3. Operation: *GetAssertion*

437 The *GetAssertion* operation is used by the TM to obtain Minting Assertions from the IdP.

438 3.3.1. wsa:Action values for *GetAssertion* Messages

439 <GetAssertion> messages MUST include a <wsa:Action> SOAP header with the value of
440 "urn:liberty:idp:2007-09:GetAssertion."

441 <GetAssertionResponse> messages MUST include a <wsa:Action> SOAP header with the value of
442 "urn:liberty:idp:2007-09:GetAssertionResponse."

443 3.3.2. *GetAssertion* Message

444 The <GetAssertion> is called to obtain assertions for the TM. These assertions may be used for hoarding or
445 minting depending upon the request parameters.

446 The <GetAssertion> contains the following attributes/elements:

447 •purpose [Required]

448 The intended purpose for the requested assertions. The following two values are defined by this specification:

449 •urn:liberty:idp:2007-09:purpose:minting

450 The assertions issued in response to this request will be used for the purpose of creating (minting) authentica-
451 tion assertions by the caller.

452 When this purpose is specified, the <saml2p:AuthnRequest> incorporates the following
453 restrictions/interpretations:

454 1. The <saml2:Subject> element MUST be specified and MUST ONLY include a
455 <saml2:SubjectConfirmation> element.

456 The identity of the subject MUST be taken from the *Target Identity* in the invocation context for the call.

457 2. The subject confirmation method MUST be a method which requires proof of possession of a key (such
458 as urn:oasis:names:tc:SAML:2.0:cm:holder-of-key).

459 3. The <saml2:SubjectConfirmationData> element SHOULD carry the necessary information to
460 identify the key that the TM plans to use to bind the minting assertion to any generated assertions
461 associated with this provider.

462 4. The <saml2p:NameIDPolicy> MAY be present.

- 463 5. The `<saml2:Conditions>` element MUST be present and MUST include at least one
 464 `<saml2:Audience>` element within the `<saml2:AudienceRestriction>` element. The
 465 `<saml2:Audience>` elements specify the provider ID(s) for the providers for which the TM desires a
 466 Minting Assertion.
- 467 If multiple `<saml2:Audience>` elements are specified, the TM is requesting the necessary minting
 468 assertion(s) for the entire list of providers (which may be in multiple minting assertions).
- 469 The `<saml2:Conditions>` element MAY also include other proposed conditions (such as
 470 NotOnOrAfter) that the TM would like to see reflected in the Minting Assertion(s) generated in
 471 response to this request. The IdP uses this data as input into its issuance algorithms and MAY choose
 472 conditions that are not fully in agreement with the proposed conditions.
- 473 6. The `<saml2p:RequestedAuthnContext>` MAY be present, but is interpreted differently. In this
 474 case, it specifies the AuthnContext(s) that the TM intends to use in MED(s). This gives the IdP guidance
 475 on what it may want to put into any `<idp:AuthnContextRestriction>` within the MINGs resulting
 476 from this request.
- 477 7. The `<saml2p:Scoping>` element SHOULD NOT be present.
- 478 8. The ForceAuthn, IsPassive, AssertionConsumerServiceIndex, AssertionConsumerServiceURL,
 479 AttributeConsumingServiceIndex, and ProviderName, attributes SHOULD NOT be present.
- 480 • *urn:liberty:idp:2007-09:purpose:SSO*
- 481 The assertions issued in response to this request will be used for the purpose of single sign on operations (both
 482 browser-based and web-services-based) by the caller.
- 483 • `<saml2p:AuthnRequest>` [Any Number]
- 484 The [SAMLCore2] authentication request containing the desired assertion properties.
- 485 Multiple `<saml2p:AuthnRequest>` elements MAY be specified when necessary to describe the set
 486 of expected responses. Each such element is treated as an independent request with an associated
 487 `<idp:GetAssertionResponseItem>` element in the response.
- 488 • Arbitrary Attributes
- 489 Arbitrary attributes may be added as necessary for a particular extension and/or inclusion of this element into a
 490 messaging protocol (such as an `xs:id` attribute).
- 491 The schema for the `<GetAssertion>` is shown below.

```

492 <!-- GetAssertion - get hoarding or minting assertions -->
493 <xs:element name="GetAssertion" type="GetAssertionType"/>
494 <xs:complexType name="GetAssertionType">
495   <xs:complexContent>
496     <xs:extension base="RequestAbstractType">
497       <xs:sequence>
498         <xs:element ref="saml2p:AuthnRequest" maxOccurs="unbounded"/>
499       </xs:sequence>
500       <xs:attribute name="purpose" type="xs:anyURI" use="required"/>
501     </xs:extension>
502   </xs:complexContent>
503 </xs:complexType>
504
505
506
```

507 **Figure 2. `<GetAssertion>` — Schema Fragment**

508 An example message body containing a <GetAssertion> message follows. This request gets Minting Assertions
509 for 4 different providers using 3 different public keys. Note that for provider 1 and 2, the request allows a new
510 federation handle to be created for the user.

```
511 <idp:GetAssertion purpose="urn:liberty:idp:2007-09:purpose:minting">
512
513   <saml2p:AuthnRequest ID="ID_2343823023823" Version="2.0"
514     IssueInstant="2006-06-23T15:38:46Z">
515     <saml2:Subject>
516       <saml2:SubjectConfirmation Method="...:holder-of-key">
517         <saml2:SubjectConfirmationData>
518           <ds:KeyInfo>Key info for Minting Assn (TM Public Key)</ds:KeyInfo>
519         </saml2:SubjectConfirmationData>
520       </saml2:SubjectConfirmation>
521     </saml2:Subject>
522     <saml2p:NameIDPolicy Format="...:persistent" AllowCreate="true"/>
523     <saml2:Conditions NotOnOrAfter="2006-07-23T15:38:46Z">
524       <saml2:AudienceRestriction>
525         <saml2:Audience>Provider 1</saml2:Audience>
526         <saml2:Audience>Provider 2</saml2:Audience>
527       </saml2:AudienceRestriction>
528     </saml2:Conditions>
529   </saml2p:AuthnRequest>
530
531   <saml2p:AuthnRequest ID="ID_2343823023824" Version="2.0"
532     IssueInstant="2006-06-23T15:38:46Z">
533     <saml2:Subject>
534       <saml2:SubjectConfirmation Method="...:holder-of-key">
535         <saml2:SubjectConfirmationData>
536           <ds:KeyInfo>A second public key to be used by the TM</ds:KeyInfo>
537         </saml2:SubjectConfirmationData>
538       </saml2:SubjectConfirmation>
539     </saml2:Subject>
540     <saml2p:NameIDPolicy Format="...:persistent"/>
541     <saml2:Conditions NotOnOrAfter="2006-07-23T15:38:46Z">
542       <saml2:AudienceRestriction>
543         <saml2:Audience>Provider 3</saml2:Audience>
544       </saml2:AudienceRestriction>
545     </saml2:Conditions>
546   </saml2p:AuthnRequest>
547
548   <saml2p:AuthnRequest ID="ID_2343823023825" Version="2.0"
549     IssueInstant="2006-06-23T15:38:46Z">
550     <saml2:Subject>
551       <saml2:SubjectConfirmation Method="...:holder-of-key">
552         <saml2:SubjectConfirmationData>
553           <ds:KeyInfo>A third public key to be used by the TM</ds:KeyInfo>
554         </saml2:SubjectConfirmationData>
555       </saml2:SubjectConfirmation>
556     </saml2:Subject>
557     <saml2p:NameIDPolicy Format="...:persistent"/>
558     <saml2:Conditions NotOnOrAfter="2006-07-23T15:38:46Z">
559       <saml2:AudienceRestriction>
560         <saml2:Audience>Provider 4</saml2:Audience>
561       </saml2:AudienceRestriction>
562     </saml2:Conditions>
563   </saml2p:AuthnRequest>
564 </idp:GetAssertion>
565
566
```

567 **Example 2. Example <GetAssertion> Message**

568 3.3.3. GetAssertionResponse Message

569 This response to the <GetAssertion> request contains the following elements and attributes.

570 • <lu:Status> [required] - the completion status for the request. See the processing rules for more information.

571 • <idp:GetAssertionResponseItem> [optional] the result(s) of the <saml2p:AuthnRequest>(s). If the
572 GetAssertion did not fail, there **MUST** be one <idp:GetAssertionResponseItem> in the response for each
573 successful <saml2p:AuthnRequest> in the <idp:GetAssertion>. If partial results are being returned, the
574 <idp:GetAssertionResponseItem> for failed <saml2p:AuthnRequest>s **MUST NOT** be included in the
575 response.

576 The <idp:GetAssertionResponseItem> element contains the following elements/attributes:

577 • <idp:AssertionItem> [required] (one or more) - a container used to bind together assertions with optional
578 MEDInfo (used when the assertions are MINGs). Multiple <idp:AssertionItem> elements **MAY** be
579 included if the <saml2p:AuthnRequest> results in multiple assertions which have different MEDInfo
580 parameters.

581 This element consists of the following elements:

582 • <idp:MEDInfo> [optional] Information deemed necessary for the generation of MEDs based on the
583 MINGs in this <idp:GetAssertionResponseItem> at the TM. This is a container element that
584 **MUST** contain the following elements:

585 • <saml:NameID> [required] the principal's NameID at the provider. In the case where multiple
586 providers are addressed by the same MING, multiple <saml:NameID>s will be present with the
587 associated ProviderID in the SPNameQualifier.

588 The TM **MUST** place the appropriate <saml:NameID> into the <saml:Subject> of any MEDs
589 generated using the associated MING(s) (MINGs within the same <idp:AssertionItem> ele-
590 ment).

591 Note that though the schema does not require this element to be present (for validation reasons), **at**
592 **least one** occurrence of this element **MUST** be present in the <idp:MEDInfo> element.

593 • Other [optional] zero or more assertion statements (such as an <saml:AttributeStatement>).
594 If present, the TM **MUST** place any such statements into any MED generated using this MING.

595 This mechanism **MUST NOT** be used to insert statements whose generation is the responsibility of the
596 TM such as the <saml:Subject>, <saml:Conditions>, or <saml:AuthnStatement>.

597 • <saml2:Assertion> [required] - the assertion(s) resulting from the referenced <saml2p:AuthnRequest>.

598 • created [Optional] - a boolean attribute set to *true* if the NameID in the enclosed assertion(s) and/or
599 <idp:MEDInfo> element was newly created for this response

600 If this attribute is not present, the assumed value is false.

601 If the value is *true* the invoker of this <GetAssertion> **MUST** report whether or no this data was used in
602 a transaction as soon as is reasonably possible following use of the data or expiration of the data, whichever
603 comes first.

604 • id [Optional] - an identifier for this assertion item. This attribute **MUST** be specified if the created
605 attribute is *true*. The value of this attribute **MUST** be included in the ref attribute on any subsequent
606 <CreatedStatus> request(s) reporting on the usage status of the newly created NameID.

607 • ref: [Required] - the value from the ID attribute on the <saml2p:AuthnRequest> element in the request
608 whose results are included in this <idp:AssertionItem> element.

- 609 • Arbitrary Attributes
- 610 Arbitrary attributes may be added as necessary for a particular extension and/or inclusion of this element into
- 611 a messaging protocol (such as an `xs:id` attribute).

```
612 <!-- GetAssertionResponse - the response for the GetAssertion request -->
613
614 <xs:element name="GetAssertionResponse" type="GetAssertionResponseType"/>
615
616 <xs:complexType name="GetAssertionResponseType">
617   <xs:complexContent>
618     <xs:extension base="ResponseAbstractType">
619       <xs:sequence>
620         <xs:element ref="GetAssertionResponseItem" minOccurs="0"
621           maxOccurs="unbounded"/>
622       </xs:sequence>
623     </xs:extension>
624   </xs:complexContent>
625 </xs:complexType>
626
627 <xs:element name="GetAssertionResponseItem" type="GetAssertionResponseItemType"/>
628 <xs:complexType name="GetAssertionResponseItemType">
629   <xs:sequence>
630     <xs:element ref="AssertionItem" minOccurs="0" maxOccurs="unbounded"/>
631   </xs:sequence>
632   <xs:attribute name="ref" type="xs:string" use="required"/>
633   <xs:anyAttribute namespace="##other" processContents="lax"/>
634 </xs:complexType>
635
636 <xs:element name="AssertionItem" type="AssertionItemType"/>
637 <xs:complexType name="AssertionItemType">
638   <xs:sequence>
639     <xs:element ref="MEDInfo" minOccurs="0"/>
640     <xs:element ref="saml2:Assertion" maxOccurs="unbounded"/>
641   </xs:sequence>
642   <xs:attribute name="created" type="xs:boolean" use="optional"/>
643   <xs:attribute name="id" type="xs:string" use="optional"/>
644   <xs:anyAttribute namespace="##other" processContents="lax"/>
645 </xs:complexType>
646
647 <xs:element name="MEDInfo" type="MEDInfoType"/>
648 <xs:complexType name="MEDInfoType">
649   <xs:sequence>
650     <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
651   </xs:sequence>
652 </xs:complexType>
653
654
```

655 **Figure 3. <GetAssertionResponse> — Schema Fragment**

```
656 <idp:GetAssertionResponse>
657 <lu:Status code="OK" />
658 <idp:GetAssertionResponseItem ref="2343823023823">
659 <idp:AssertionItem>
660 <idp:MEDInfo>
661 <saml2:NameID>..Provider 1 NameID...</saml2:NameID>
662 <saml2:AttributeStatement>
663 <saml2:Attribute Name="http://idp.com/MySpecialAttribute"
664 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
665 <saml2:AttributeValue>attribute-value</saml2:AttributeValue>
666 </saml2:Attribute>
667 </saml2:AttributeStatement>
668 </idp:MEDInfo>
669 <saml2:Assertion>..Provider 1 Minting Assertion.. </saml2:Assertion>
670 </idp:AssertionItem>
671 <idp:AssertionItem created="true">
672 <idp:MEDInfo>
673 <saml2:NameID>..Provider 2 NameID...</saml2:NameID>
674 </idp:MEDInfo>
675 <saml2:Assertion>..Provider 2 Minting Assertion.. </saml2:Assertion>
676 </idp:AssertionItem>
677 </idp:GetAssertionResponseItem>
678
679 <idp:GetAssertionResponseItem ref="2343823023824">
680 <idp:AssertionItem>
681 <idp:MEDInfo>
682 <saml2:NameID>..Provider 3 NameID...</saml2:NameID>
683 </idp:MEDInfo>
684 <saml2:Assertion>..Provider 3 Minting Assertion.. </saml2:Assertion>
685 </idp:AssertionItem>
686 </idp:GetAssertionResponseItem>
687
688 <idp:GetAssertionResponseItem ref="2343823023825">
689 <idp:AssertionItem>
690 <idp:MEDInfo>
691 <saml2:NameID>..Provider 4 NameID...</saml2:NameID>
692 </idp:MEDInfo>
693 <saml2:Assertion>..Provider 4 Minting Assertion.. </saml2:Assertion>
694 </idp:AssertionItem>
695 </idp:GetAssertionResponseItem>
696 </idp:GetAssertionResponse>
697
```

698 **Example 3. <GetAssertionResponse> Message**

699 3.3.4. GetAssertion Processing Rules

- 700 • Each <saml2p:AuthnRequest> MUST be processed independently of the others, succeed-
701 ing or failing on its own. The results of each <saml2p:AuthnRequest> is returned in a
702 <idp:GetAssertionResponseItem> element within the <GetAssertionResponse> message.
- 703 • The IdP MUST include exactly one <idp:GetAssertionResponseItem> in the <GetAssertionResponse>
704 for each successful <saml2p:AuthnRequest> element in the request.
- 705 • If a particular <saml2p:AuthnRequest> is not successful, the IdP MUST NOT include a
706 <GetAssertionResponseItem> referencing that <saml2p:AuthnRequest>
- 707 • The IdP may refuse to create new federations for the principal through this interface. In such cases the
708 <saml2p:AuthnRequest> that would need a new federation to successfully complete MUST be treated as
709 a failure. In such a case, if the IdP is providing second level error codes, the error code MUST be *NoCreated*.

- 710 • If the IdP is willing to create new federations for the principal but the `AllowCreate` attribute on the
711 `<saml2p:NameIDPolicy>` is not set to `true`, the `<saml2p:AuthnRequest>` **MUST** be treated as a failure. In
712 such a case, if the IdP is providing second level error codes, the error code **MUST** be *NotFederated*.
- 713 • If the `created` attribute is set to `true` on the `<idp:AssertionItem>`, the recipient **MUST NOT** use the data
714 contained within as part an identity transaction with the relying party unless the relying party has indicated
715 that it is willing to accept a newly federated identity (such as by setting the `AllowCreate` attribute on its
716 `<saml2p:AuthnRequest>` to the recipient.
- 717 • If the IdP created a new persistent NameID to represent a new federation for the principal at the relying party:
- 718 • The IdP **MUST NOT** treat this as a successfully completed federation until the IdP receives confirmation from
719 the invoker that the new NameID has been used in a transaction.
- 720 This is driven by the fact that, as far as the IdP knows, the Relying party did not request that the federation be
721 generated (the invoker of this messages is not the relying party).
- 722 • Should the IdP receive a federation request through another channel (such as a SAML2 `AuthnRequest` via
723 browser SSO from the relying party) while the assertions resulting from this request are still valid, the IdP
724 **MUST** use the same NameID.
- 725 This prevents the IdP from issuing two different persistent NameIDs for the same principal at the same relying
726 party.
- 727 • If the IdP created a new NameID to represent a new federation for the principal at the relying party, the IdP **MUST**
728 not treat this as a successfully completed federation until the IdP receives confirmation from the invoker that the
729 new NameID has been used in a transaction.
- 730 • The IdP **MUST ONLY** include `<idp:MedInfo>` elements in `<idp:AssertionItem>` elements that contain
731 **minting assertions**.
- 732 • If the invoker's request for permission to mint assertions is not granted, the IdP must treat that
733 `<saml2p:AuthnRequest>` as a failure and **NOT** return a `<GetAssertionResponseItem>` for that
734 `<saml2p:AuthnRequest>` – in other words, the IDP **MUST NOT** generate a Minting assertion with an
735 authorization decision of "Deny."
- 736 • If request processing succeeded, the top-level status code **MUST** be
- 737 • *OK* - if **all**. `<saml2p:AuthnRequest>`s succeeded.
- 738 • *Partial* - if some of the requests succeeded and some failed. The individual request status will be disclosed in
739 the `<saml2p:Status>` element in each `<saml2p:Response>`.
- 740 • *Failed* - if **all**. `<saml2p:AuthnRequest>`s failed.
- 741 • If the top-level status code is *Failed*, the response **MAY** also contain *Forbidden* as a second-level status code. The
742 IdP Service instance may not wish to reveal the reason for failure, in which case no second-level status code will
743 appear.

744 **3.4. Operation: *GetProviderInfo***

745 The *GetProviderInfo* operation is used to obtain Provider Information for providers with which the IdP has relation-
746 ships.

747 **3.4.1. *wsa:Action* values for *GetProviderInfo* Messages**

748 <GetProviderInfo> messages MUST include a <wsa:Action> SOAP header with the value of
749 "urn:liberty:idp:2007-09:GetProviderInfo."

750 <GetProviderInfoResponse> messages MUST include a <wsa:Action> SOAP header with the value of
751 "urn:liberty:idp:2007-09:GetProviderInfoResponse."

752 **3.4.2. *GetProviderInfo* Message**

753 The <GetProviderInfo> is called to obtain provider information about the providers with which an IdP has a
754 relationship (so that the TM can select the providers with which it will interact).

755 The <GetProviderInfo> contains the following attributes/elements:

756 • all [optional]

757 A boolean flag indicating whether the TM wants to get back all of the providers or just the providers with which
758 the current user has a relationship.

759 If present and set to "true," the IdP returns data for all available providers. Otherwise the IdP only returns the
760 providers for which the principal has an established federation.

761 If not present, the request is interpreted as if all was set to "false."

762 This attribute has no effect if specific providers are requested using the <ProviderID> element.

763 • <ProviderID> [optional]

764 zero or more Provider IDs for providers for whom the provider info is requested. If this element appears, the
765 results are limited to the providers for which the IdP has a relationship which are also in this list.

766 The schema for the <GetProviderInfo> is shown below.

```
767 <!-- GetProviderInfo - Get list of providers visible to user/IdP -->
768
769 <xs:element name="GetProviderInfo" type="GetProviderInfoType"/>
770
771 <xs:complexType name="GetProviderInfoType">
772   <xs:complexContent>
773     <xs:extension base="RequestAbstractType">
774       <xs:sequence>
775         <xs:element ref="ProviderID" minOccurs="0" maxOccurs="unbounded"/>
776       </xs:sequence>
777       <xs:attribute name="all" type="xs:boolean" use="optional"/>
778     </xs:extension>
779   </xs:complexContent>
780 </xs:complexType>
781
782 <xs:element name="ProviderID" type="xs:anyURI"/>
783
```

784 **Figure 4. <GetProviderInfo> — Schema Fragment**

785 An example message body containing a <GetProviderInfo> message follows. This is a request for information
786 about all of the providers associated with the current principal.

```
787 <idp:GetProviderInfo all="true"/>
788
```

789 **Example 4. Example <GetProviderInfo> Message**

790 3.4.3. GetProviderInfoResponse Message

791 This response to the <GetProviderInfo> request contains the following elements and attributes.

- 792 • <lu:Status>: Contains status code; see processing rules.
- 793 • One or more <shps:ProviderInfo> elements if the call did not fail.

```
794 <!-- GetProviderInfoResponse - response for the GetProviderInfo request -->
795
796 <xs:element name="GetProviderInfoResponse" type="GetProviderInfoResponseType"/>
797
798 <xs:complexType name="GetProviderInfoResponseType">
799   <xs:complexContent>
800     <xs:extension base="ResponseAbstractType">
801       <xs:sequence>
802         <xs:element ref="ProviderInfo" minOccurs="0"
803           maxOccurs="unbounded"/>
804       </xs:sequence>
805     </xs:extension>
806   </xs:complexContent>
807 </xs:complexType>
808
```

809 **Figure 5. <GetProviderInfoResponse> — Schema Fragment**

```
810 <idp:GetProviderInfoResponse >
811   <lu:Status code="OK" />
812   <idp:ProviderInfo providerID="http://services.acme.com/Profile"
813     name="ACME Profile Service" />
814   <idp:ProviderInfo providerID="http://ps.services.Rus.com/"
815     name="Services-R-Us People Service" />
816 </idp:GetProviderInfoResponse>
817
```

818 **Example 5. <GetProviderInfoResponse> Message**

819 3.4.4. GetProviderInfo Processing Rules

- 820 • If <shps:ProviderID> elements are specified on the request, the IdP MUST return <shps:ProviderInfo>
821 elements for each provider in the intersection of the IdPs list of acceptable providers and the provided list of
822 requested providers (only providers on both lists are included in the response).
- 823 • If no <shps:ProviderID> elements are specified in the request, the IdP should examine the all attribute. If
824 this attribute is true, the IDP SHOULD return information about all providers on its list of acceptable providers.
825 On the other hand, if the all attribute is not present or is set to "false," the IDP should only return information
826 about providers which the principal has established federations with.
- 827 • The IDP MAY use other local policies to control and/or restrict the visibility of providers to the caller (including
828 allowing the user to control this).
- 829 • If request processing succeeded, the top-level status code MUST be

- 830 • *OK* - if the call succeeded (which may include cases where no `<shps:ProviderID>` elements are returned
831 – such as a principal which has not yet established any federations).
- 832 • *Failed* - if the call failed (because `<shps:ProviderID>` elements were specified on the request and **none**.
833 of them are known to the IdP.
- 834 • If the top-level status code is *Failed*, the response MAY also contain *Forbidden* as a second-level status code. The
835 IdP Service instance may not wish to reveal the reason for failure, in which case no second-level status code will
836 appear.

837 **3.5. Operation: *CreatedStatus***

838 The *CreatedStatus* operation is used to report to the IdP that a NameID which represents a new federation has been
839 used.

840 **3.5.1. `wsa:Action` values for *CreatedStatus* Messages**

841 `<CreatedStatus>` messages MUST include a `<wsa:Action>` SOAP header with the value of
842 "urn:liberty:idp:2007-09:CreatedStatus."

843 `<CreatedStatusResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of
844 "urn:liberty:idp:2007-09:CreatedStatusResponse."

845 **3.5.2. *CreatedStatus* Message**

846 The `<CreatedStatus>` request is called to report the use of data from a `<GetAssertionResponse>` which had
847 the `created` attribute set to *true*.

848 The `<CreatedStatus>` contains the following one or more `<CreatedStatusItem>` elements, each of which
849 contain the following attributes/elements:

- 850 • `ref` [**required**] - a reference to the `<AssertionItem>` whose status is being reported. The value of the `id`
851 attribute on the `<AssertionItem>` is specified here.
- 852 • `used` [**required**] - the status of the item (whether or not it was used). If the assertion was used this attribute
853 MUST be set to *true* and otherwise it MUST be set to *false*.
- 854 • `firstUsed` [**optional**] - the date/time when the data was first used.

855 The schema for the `<CreatedStatus>` is shown below.

```
856 <!-- CreatedStatus - report on use of a new federation -->
857
858 <xs:element name="CreatedStatus" type="CreatedStatusType"/>
859
860 <xs:complexType name="CreatedStatusType">
861   <xs:complexContent>
862     <xs:extension base="RequestAbstractType">
863       <xs:sequence>
864         <xs:element ref="CreatedStatusItem" maxOccurs="unbounded"/>
865       </xs:sequence>
866     </xs:extension>
867   </xs:complexContent>
868 </xs:complexType>
869
870 <xs:element name="CreatedStatusItem" type="CreatedStatusItemType"/>
871
872 <xs:complexType name="CreatedStatusItemType">
873   <xs:attribute name="ref" type="xs:anyURI" use="required"/>
874   <xs:attribute name="used" type="xs:boolean" use="required"/>
875   <xs:attribute name="firstUsed" type="xs:dateTime" use="optional"/>
876 </xs:complexType>
877
```

878 **Figure 6. `<CreatedStatus>` — Schema Fragment**

879 An example message body containing a `<CreatedStatus>` message follows. This is a reporting on the use of a
880 `<saml2:NameID>` that had been returned as part of a MING response.

```
881 <idp:CreatedStatus>
882   <idp:CreatedStatusItem used="true"
883     ref="uuid:9230230-238023-2309238-83494" firstUsed="2007-01-11T22:17:37Z"/>
884 </idp:CreatedStatus>
885
```

886 **Example 6. Example `<CreatedStatus>` Message**

887 **3.5.3. CreatedStatusResponse Message**

888 This response to the `<CreatedStatus>` request contains the following elements and attributes.

889 • `<lu:Status>`: Contains status code; see processing rules.

```
890 <!-- CreatedStatusResponse - response to the CreatedStatus request -->
891
892 <xs:element name="CreatedStatusResponse" type="CreatedStatusResponseType" />
893
894 <xs:complexType name="CreatedStatusResponseType">
895   <xs:complexContent>
896     <xs:extension base="ResponseAbstractType" />
897   </xs:complexContent>
898 </xs:complexType>
899
```

900 **Figure 7. <CreatedStatusResponse> — Schema Fragment**

```
901 <idp:CreatedStatusResponse>
902   <lu:Status code="OK"/>
903 </idp:CreatedStatusResponse>
904
```

905 **Example 7. <CreatedStatusResponse> Message**

906 3.5.4. CreatedStatus Processing Rules

- 907 • Upon receipt of this call, the IDP should consider the federation process to be complete and should maintain the
908 persistent identifier as appropriate.
- 909 • If the specified `<saml2:NameID>` or `<saml2:Assertion>` were not issued by the IdP to this invoker, the
910 request **MUST** be treated as a failure. In such cases, if the IdP is reporting second level status codes, the error
911 code **MUST** be *NotIssued*.
- 912 • If request processing succeeded, the top-level status code **MUST** be
 - 913 • *OK* - if the call succeeded.
 - 914 • *Partial* - if some of the requests succeeded and some failed. The individual request status will be disclosed in
915 the `<saml2p:Status>` element in each `<saml2p:Response>`.
 - 916 • *Failed* - if all of the `<idp:CreatedStatusItem>` elements failed.
- 917 • If the top-level status code is *Failed*, the response **MAY** also contain *Forbidden* as a second-level status code. The
918 IdP Service instance may not wish to reveal the reason for failure, in which case no second-level status code will
919 appear.

920 **4. IDP Service Schema**

```
921 <?xml version="1.0" encoding="UTF-8"?>
922 <xs:schema targetNamespace="urn:liberty:idp:2007-09"
923   xmlns:lu="urn:liberty:util:2006-08"
924   xmlns:xs="http://www.w3.org/2001/XMLSchema"
925   xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
926   xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
927   xmlns="urn:liberty:idp:2007-09"
928   elementFormDefault="qualified"
929   attributeFormDefault="unqualified"
930 >
931
932 <xs:import namespace="urn:liberty:util:2006-08"
933   schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
934
935 <xs:import namespace="urn:oasis:names:tc:SAML:2.0:assertion"
936   schemaLocation="http://docs.oasis-open.org/security/saml/v2.0/saml-schema-assertion-2.0.xsd"/>
937 <xs:import namespace="urn:oasis:names:tc:SAML:2.0:protocol"
938   schemaLocation="http://docs.oasis-open.org/security/saml/v2.0/saml-schema-protocol-2.0.xsd"/>
939
940 <!-- Data Definitions { -->
941
942
943
944 <!-- ProviderInfo - Information about providers returned by GetProviderInfo -->
945
946 <xs:element name="ProviderInfo" type="ProviderInfoType"/>
947
948 <xs:complexType name="ProviderInfoType">
949   <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
950   <xs:attribute name="name" type="xs:string" use="optional"/>
951 </xs:complexType>
952
953
954 <!-- AuthnContextRestriction - Ming Condition - restricts authncontext in MED -->
955
956 <xs:complexType name="AuthnContextRestrictionType">
957   <xs:complexContent>
958     <xs:extension base="saml2:ConditionAbstractType">
959       <xs:sequence>
960         <xs:element ref="saml2:AuthnContext" maxOccurs="unbounded"/>
961       </xs:sequence>
962     </xs:extension>
963   </xs:complexContent>
964 </xs:complexType>
965
966
967 <!-- SubjectRestriction - MING condition restricting Subjects in MED -->
968
969 <xs:complexType name="SubjectRestrictionType">
970   <xs:complexContent>
971     <xs:extension base="saml2:ConditionAbstractType">
972       <xs:sequence>
973         <xs:element ref="saml2:Subject" maxOccurs="unbounded"/>
974       </xs:sequence>
975     </xs:extension>
976   </xs:complexContent>
977 </xs:complexType>
978
979
980
981 <!-- End of Data Definitions } -->
982
983 <!-- Interface Definitions { -->
984
985
```

```
986
987 <!-- GetAssertion - get hoarding or minting assertions -->
988
989 <xs:element name="GetAssertion" type="GetAssertionType"/>
990
991 <xs:complexType name="GetAssertionType">
992   <xs:complexContent>
993     <xs:extension base="RequestAbstractType">
994       <xs:sequence>
995         <xs:element ref="saml2p:AuthnRequest" maxOccurs="unbounded"/>
996       </xs:sequence>
997       <xs:attribute name="purpose" type="xs:anyURI" use="required"/>
998     </xs:extension>
999   </xs:complexContent>
1000 </xs:complexType>
1001
1002
1003 <!-- GetAssertionResponse - the response for the GetAssertion request -->
1004
1005 <xs:element name="GetAssertionResponse" type="GetAssertionResponseType"/>
1006
1007 <xs:complexType name="GetAssertionResponseType">
1008   <xs:complexContent>
1009     <xs:extension base="ResponseAbstractType">
1010       <xs:sequence>
1011         <xs:element ref="GetAssertionResponseItem" minOccurs="0"
1012           maxOccurs="unbounded"/>
1013       </xs:sequence>
1014     </xs:extension>
1015   </xs:complexContent>
1016 </xs:complexType>
1017
1018 <xs:element name="GetAssertionResponseItem" type="GetAssertionResponseItemType"/>
1019 <xs:complexType name="GetAssertionResponseItemType">
1020   <xs:sequence>
1021     <xs:element ref="AssertionItem" minOccurs="0" maxOccurs="unbounded"/>
1022   </xs:sequence>
1023   <xs:attribute name="ref" type="xs:string" use="required"/>
1024   <xs:anyAttribute namespace="##other" processContents="lax"/>
1025 </xs:complexType>
1026
1027 <xs:element name="AssertionItem" type="AssertionItemType"/>
1028 <xs:complexType name="AssertionItemType">
1029   <xs:sequence>
1030     <xs:element ref="MEDInfo" minOccurs="0"/>
1031     <xs:element ref="saml2:Assertion" maxOccurs="unbounded"/>
1032   </xs:sequence>
1033   <xs:attribute name="created" type="xs:boolean" use="optional"/>
1034   <xs:attribute name="id" type="xs:string" use="optional"/>
1035   <xs:anyAttribute namespace="##other" processContents="lax"/>
1036 </xs:complexType>
1037
1038 <xs:element name="MEDInfo" type="MEDInfoType"/>
1039 <xs:complexType name="MEDInfoType">
1040   <xs:sequence>
1041     <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
1042   </xs:sequence>
1043 </xs:complexType>
1044
1045
1046
1047 <!-- GetProviderInfoResponse - response for the GetProviderInfo request -->
1048
1049 <xs:element name="GetProviderInfoResponse" type="GetProviderInfoResponseType"/>
1050
1051 <xs:complexType name="GetProviderInfoResponseType">
1052   <xs:complexContent>
```

```

1053     <xs:extension base="ResponseAbstractType">
1054     <xs:sequence>
1055         <xs:element ref="ProviderInfo"          minOccurs="0"
1056             maxOccurs="unbounded"/>
1057     </xs:sequence>
1058 </xs:extension>
1059 </xs:complexContent>
1060 </xs:complexType>
1061
1062
1063 <!-- GetProviderInfo - Get list of providers visible to user/IdP -->
1064
1065 <xs:element name="GetProviderInfo" type="GetProviderInfoType"/>
1066
1067 <xs:complexType name="GetProviderInfoType">
1068     <xs:complexContent>
1069         <xs:extension base="RequestAbstractType">
1070             <xs:sequence>
1071                 <xs:element ref="ProviderID" minOccurs="0" maxOccurs="unbounded"/>
1072             </xs:sequence>
1073             <xs:attribute name="all" type="xs:boolean" use="optional"/>
1074         </xs:extension>
1075     </xs:complexContent>
1076 </xs:complexType>
1077
1078 <xs:element name="ProviderID" type="xs:anyURI"/>
1079
1080
1081 <!-- CreatedStatus - report on use of a new federation -->
1082
1083 <xs:element name="CreatedStatus" type="CreatedStatusType"/>
1084
1085 <xs:complexType name="CreatedStatusType">
1086     <xs:complexContent>
1087         <xs:extension base="RequestAbstractType">
1088             <xs:sequence>
1089                 <xs:element ref="CreatedStatusItem" maxOccurs="unbounded"/>
1090             </xs:sequence>
1091         </xs:extension>
1092     </xs:complexContent>
1093 </xs:complexType>
1094
1095 <xs:element name="CreatedStatusItem" type="CreatedStatusItemType"/>
1096
1097 <xs:complexType name="CreatedStatusItemType">
1098     <xs:attribute name="ref"          type="xs:anyURI" use="required"/>
1099     <xs:attribute name="used"        type="xs:boolean" use="required"/>
1100     <xs:attribute name="firstUsed"   type="xs:dateTime" use="optional"/>
1101 </xs:complexType>
1102
1103
1104 <!-- CreatedStatusResponse - response to the CreatedStatus request -->
1105
1106 <xs:element name="CreatedStatusResponse" type="CreatedStatusResponseType"/>
1107
1108 <xs:complexType name="CreatedStatusResponseType">
1109     <xs:complexContent>
1110         <xs:extension base="ResponseAbstractType" />
1111     </xs:complexContent>
1112 </xs:complexType>
1113
1114
1115 <!-- RequestAbstractType - common request message structure -->
1116
1117 <xs:complexType name="RequestAbstractType" abstract="true">
1118     <xs:anyAttribute namespace="##other" processContents="lax"/>
1119 </xs:complexType>

```

```
1120
1121
1122 <!-- ResponseAbstractType - common message response structure -->
1123
1124 <xs:complexType name="ResponseAbstractType" abstract="true">
1125   <xs:sequence>
1126     <xs:element ref="lu:Status"/>
1127   </xs:sequence>
1128   <xs:anyAttribute namespace="##other" processContents="lax"/>
1129 </xs:complexType>
1130
1131
1132
1133 <!-- End of Interface Definitions } -->
1134
1135 </xs:schema>
1136
```

1137 5. IDP Service WSDL

```

1138 <?xml version="1.0"?>
1139 <definitions name="idp-svc"
1140   targetNamespace="urn:liberty:idp:2007-09"
1141   xmlns:tns="urn:liberty:idp:2007-09"
1142   xmlns="http://schemas.xmlsoap.org/wsdl/"
1143   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
1144   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
1145   xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
1146   xmlns:idp="urn:liberty:idp:2007-09"
1147   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1148   xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
1149     http://schemas.xmlsoap.org/wsdl/
1150     http://www.w3.org/2006/02/addressing/wsdl
1151     http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
1152
1153   <types>
1154     <xsd:schema>
1155       <xsd:import namespace="urn:liberty:idp:2007-09"
1156         schemaLocation="liberty-idwsf-idp-v1.0.xsd"/>
1157     </xsd:schema>
1158   </types>
1159
1160   <message name="GetAssertion">
1161     <part name="body" element="idp:GetAssertion"/>
1162   </message>
1163   <message name="GetAssertionResponse">
1164     <part name="body" element="idp:GetAssertionResponse"/>
1165   </message>
1166
1167   <message name="GetProviderInfo">
1168     <part name="body" element="idp:GetProviderInfo"/>
1169   </message>
1170   <message name="GetProviderInfoResponse">
1171     <part name="body" element="idp:GetProviderInfoResponse"/>
1172   </message>
1173
1174   <message name="CreatedStatus">
1175     <part name="body" element="idp:CreatedStatus"/>
1176   </message>
1177   <message name="CreatedStatusResponse">
1178     <part name="body" element="idp:CreatedStatusResponse"/>
1179   </message>
1180
1181   <portType name="IDPPort">
1182
1183     <operation name="GetAssertion">
1184       <input message="tns:GetAssertion"
1185         wsaw:Action="urn:liberty:idp:2007-09:GetAssertion" />
1186       <output message="tns:GetAssertionResponse"
1187         wsaw:Action="urn:liberty:idp:2007-09:GetAssertionResponse" />
1188     </operation>
1189
1190     <operation name="GetProviderInfo">
1191       <input message="tns:GetProviderInfo"
1192         wsaw:Action="urn:liberty:idp:2007-09:GetProviderInfo" />
1193       <output message="tns:GetProviderInfoResponse"
1194         wsaw:Action="urn:liberty:idp:2007-09:GetProviderInfoResponse" />
1195     </operation>
1196
1197     <operation name="CreatedStatus">
1198       <input message="tns:CreatedStatus"
1199         wsaw:Action="urn:liberty:idp:2007-09:CreatedStatus" />
1200       <output message="tns:CreatedStatusResponse"
1201         wsaw:Action="urn:liberty:idp:2007-09:CreatedStatusResponse" />
1202     </operation>

```

```
1203
1204 </portType>
1205
1206 <!--
1207 An example of a binding and service that can be used with this
1208 abstract service description is idpided below.
1209 -->
1210
1211 <binding name="IDPBinding" type="tns:IDPPort">
1212
1213   <soap:binding style="document"
1214     transport="http://schemas.xmlsoap.org/soap/http"/>
1215
1216   <operation name="GetAssertion">
1217     <input> <soap:body use="literal"/> </input>
1218     <output> <soap:body use="literal"/> </output>
1219   </operation>
1220
1221   <operation name="GetProviderInfo">
1222     <input> <soap:body use="literal"/> </input>
1223     <output> <soap:body use="literal"/> </output>
1224   </operation>
1225
1226   <operation name="CreatedStatus">
1227     <input> <soap:body use="literal"/> </input>
1228     <output> <soap:body use="literal"/> </output>
1229   </operation>
1230
1231 </binding>
1232
1233 <service name="IDPService">
1234
1235   <port name="IDPPort" binding="tns:IDPBinding">
1236
1237     <!-- Modify with the REAL SOAP endpoint -->
1238
1239     <soap:address location="http://example.com/idp"/>
1240
1241   </port>
1242
1243 </service>
1244
1245 </definitions>
1246
```

1247 **References**

1248 **Normative**

- 1249 [LibertyACT] Cahill, Conor P., eds. "Liberty Advanced Client Technologies," Version 1.0, Liberty Alliance Project
1250 (14 December 2007). <http://www.projectliberty.org/specs>
- 1251 [LibertyIDPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification,"
1252 Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 1253 [LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-
1254 errata-v1.0, Liberty Alliance Project (21 April, 2007). <http://www.projectliberty.org/specs>
- 1255 [LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version
1256 1.2-errata-v3.0, Liberty Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 1257 [LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.1, Liberty Alliance Project (14
1258 December, 2004). <http://www.projectliberty.org/specs>
- 1259 [LibertySOAPAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. "Liberty ID-WSF Authentication,
1260 Single Sign-On, and Identity Mapping Services Specification," v2.0-errata-1.0-01, Liberty Alliance Project
1261 (28 November, 2006). <http://www.projectliberty.org/specs>
- 1262 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Liberty
1263 ID-WSF SOAP Binding Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007).
1264 <http://www.projectliberty.org/specs>
- 1265 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
1266 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>
- 1267 [RFC2251] "Lightweight Directory Access Protocol (v3)," M. Wahl T. Howes S. Kille (December 1997). RFC 2251,
1268 Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2251.txt>
- 1269 [RFC2252] "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions," M. Wahl A.
1270 Coulbeck T. Howes S. Kille (December 1997). RFC 2252, Internet Engineering Task Force
1271 <http://www.ietf.org/rfc/rfc2252.txt>
- 1272 [RFC2891] Howes, T., Wahl, M., eds. (August 2000). "LDAP Control Extension for Server Side Sorting of Search
1273 Results," RFC 2891, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2891.txt>
- 1274 [SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (2 September 2003). "Assertions and Pro-
1275 tocol for the OASIS Security Assertion Markup Language (SAML) V1.1," SAML v1.1, OASIS
1276 Standard, Organization for the Advancement of Structured Information Standards [http://www.oasis-
open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf](http://www.oasis-
1277 open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf)
- 1278 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions
1279 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
1280 SIS Standard, Organization for the Advancement of Structured Information Standards [http://docs.oasis-
open.org/security/saml/v2.0/saml-core-2.0-os.pdf](http://docs.oasis-
1281 open.org/security/saml/v2.0/saml-core-2.0-os.pdf)
- 1282 [SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler,
1283 Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0,"
1284 SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards
1285 <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>

- 1286 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October
1287 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium
1288 <http://www.w3.org/TR/xmlschema-1/>
- 1289 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David , Kakivaya, Gopal, Layman,
1290 Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C
1291 Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 1292 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).
1293 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium
1294 <http://www.w3.org/TR/2004/REC-xml-20040204>
- 1295 [XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and
1296 Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlsig-core>
- 1297 [XPath] Clark , J., DeRose , S., eds. (16 November 1999). "XML Path Language (XPath) Version 1.0 ,"
1298 Recommendation, W3C <http://www.w3.org/TR/xpath> [August 2003].

1299 Informative

- 1300 [LibertyIDWSFGuide] Weitzel, David, eds. "Liberty ID-WSF Implementation Guide," Version 2.0-02, Liberty
1301 Alliance Project (13 January, 2005). <http://www.projectliberty.org/specs>
- 1302 [LibertyIDPPGuide] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Implementation
1303 Guidelines," Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 1304 [LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework
1305 Overview," Version 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>