



Liberty ID-WSF Discovery Service Specification

Version: 2.0-24

Editors:

Jeff Hodges, NeuStar, Inc.

Conor Cahill, Intel Corporation

Contributors:

John Beatty, Sun Microsystems, Inc.

Jonathan Sergent, Sun Microsystems, Inc.

Robert Aarts, Nokia Corporation

Carolina Canales-Valenzuela, Ericsson

Darryl Champagne, IEEE-ISTO

Gary Ellison, Sun Microsystems, Inc.

Jukka Kainulainen, Nokia Corporation

John Kemp, Nokia Corporation

Paul Madsen, Entrust, Inc.

Greg Whitehead, Trustgenix, Inc.

Emily Xu, Sun Microsystems, Inc.

Abstract:

This specification defines mechanisms for describing and discovering identity web services.

Filename: draft-liberty-idwsf-disco-svc-v2.0-24.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 © 2006 Adobe Systems; America Online, Inc.; American Express Company; Amsoft Systems Pvt Ltd.;
16 Avatier Corporation; Axalto; Bank of America Corporation; BIPAC; BMC Software, Inc.; Computer Associates
17 International, Inc.; DataPower Technology, Inc.; Diversinet Corp.; Enosis Group LLC; Entrust, Inc.; Epok, Inc.;
18 Ericsson; Fidelity Investments; Forum Systems, Inc.; France Télécom; French Government Agence pour le
19 développement de l'administration électronique (ADAE); Gamefederation; Gemplus; General Motors; Giesecke &
20 Devrient GmbH; GSA Office of Governmentwide Policy; Hewlett-Packard Company; IBM Corporation; Intel
21 Corporation; Intuit Inc.; Kantega; Kayak Interactive; MasterCard International; Mobile Telephone Networks (Pty)
22 Ltd; NEC Corporation; Netegrity, Inc.; NeuStar, Inc.; Nippon Telegraph and Telephone Corporation; Nokia
23 Corporation; Novell, Inc.; NTT DoCoMo, Inc.; OpenNetwork; Oracle Corporation; Ping Identity Corporation;
24 Reactivity Inc.; Royal Mail Group plc; RSA Security Inc.; SAP AG; Senforce; Sharp Laboratories of America;
25 Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Supremacy Financial Corporation; Symlabs, Inc.;
26 Telecom Italia S.p.A.; Telefónica Móviles, S.A.; Trusted Network Technologies; Trustgenix; UTI; VeriSign, Inc.;
27 Vodafone Group Plc.; Wave Systems Corp. All rights reserved.

28 Liberty Alliance Project
29 Licensing Administrator
30 c/o IEEE-ISTO
31 445 Hoes Lane
32 Piscataway, NJ 08855-1331, USA
33 info@projectliberty.org

34 Contents

35	1. Introduction	5
36	1.1. Conceptual Model and Terminology	5
37	1.2. Scope	6
38	1.3. Notation and Conventions	6
39	1.3.1. XML Namespaces	6
40	2. Discovery Service Information Model	8
41	2.1. Overview of Discovery Service Information Model	8
42	2.2. Versioning in ID-WSF	9
43	2.3. ID-WSF Endpoint References (EPRs)	9
44	2.3.1. EPR Profiling Attributes	10
45	2.3.2. EPR Profiling Elements	11
46	2.3.3. ID-WSF Web Services Addressing EPR Profile	15
47	2.3.4. Effective Web Services Addressing EPR	19
48	2.3.5. Example Liberty ID-WSF EPRs	20
49	2.4. Service Metadata	22
50	2.4.1. Service Metadata element	22
51	2.4.2. Minting ID-WSF EPRs based upon Service Metadata	25
52	2.4.3. Service Metadata Example	25
53	3. Discovery Service	29
54	3.1. Service URIs	29
55	3.2. Status Codes	30
56	3.3. Operation: <i>DiscoveryQuery</i>	31
57	3.3.1. wsa:Action values for <i>DiscoveryQuery</i> Messages	31
58	3.3.2. <Query> Message	31
59	3.3.3. QueryResponse	35
60	3.3.4. DiscoveryQuery Processing Rules	37
61	3.4. Operation: <i>MDAssociationAdd</i>	37
62	3.4.1. wsa:Action values for <i>MDAssociationAdd</i> Messages	37
63	3.4.2. SvcMDAssociationAdd Message	37
64	3.4.3. SvcMDAssociationAddResponse Message	38
65	3.4.4. MDAssociation Add Processing Rules	39
66	3.5. Operation: <i>MDAssociationQuery</i>	39
67	3.5.1. wsa:Action values for <i>MDAssociationQuery</i> Messages	39
68	3.5.2. SvcMDAssociationQuery Message	40
69	3.5.3. SvcMDAssociationQueryResponse Message	40
70	3.5.4. MDAssociation Query Processing Rules	41
71	3.6. Operation: <i>MDAssociationDelete</i>	41
72	3.6.1. wsa:Action values for <i>MDAssociationDelete</i> Messages	41
73	3.6.2. SvcMDAssociationDelete Message	41
74	3.6.3. SvcMDAssociationDeleteResponse Message	42
75	3.6.4. MDAssociation Delete Processing Rules	42
76	3.7. Operation: <i>MetadataRegister</i>	43
77	3.7.1. wsa:Action values for <i>MetadataRegister</i> Messages	43
78	3.7.2. SvcMDRegister Message	43
79	3.7.3. SvcMDRegisterResponse Message	44
80	3.7.4. Metadata Register Processing Rules	44
81	3.8. Operation: <i>MetadataQuery</i>	45
82	3.8.1. wsa:Action values for <i>MetadataQuery</i> Messages	45
83	3.8.2. SvcMDQuery Message	45
84	3.8.3. SvcMDQueryResponse Message	46
85	3.8.4. Metadata Query Processing Rules	47
86	3.9. Operation: <i>MetadataReplace</i>	47

87	3.9.1. wsa:Action values for MetadataReplace Messages	48
88	3.9.2. SvcMDReplace Message	48
89	3.9.3. SvcMDReplaceResponse Message	48
90	3.9.4. Metadata Replace Processing Rules	49
91	3.10. Operation: <i>MetadataDelete</i>	49
92	3.10.1. wsa:Action values for MetadataDelete Messages	49
93	3.10.2. SvcMDDelete Message	50
94	3.10.3. SvcMDDeleteResponse Message	50
95	3.10.4. Metadata Delete Processing Rules	51
96	3.11. Option Value for Response Authentication	51
97	3.12. Including Keys in the ModifyResponse Message	52
98	4. Discovery Service ID-WSF EPR conveyed via a Security Token	54
99	4.1. EPR Generation Rules	54
100	4.2. SAML 2.0 Security Tokens	54
101	4.3. SAML 1.x (Liberty ID-FF) Security Tokens	55
102	5. ID-WSF 1.x Resource Offering conveyed in an EPR	57
103	6. Acknowledgments	59
104	References	60
105	A. Discovery Service Version 2.0 XSD	62
106	B. Discovery Service WSDL	69

107 1. Introduction

108 This specification defines a framework for describing and discovering web services in general and *identity web services*
109 in particular. The conceptual model and terminology is first provided to set the context for the rest of the specification.
110 Next, the data types for the information maintained by a Discovery Service are specified. Then the Discovery Service
111 itself is specified.

112 1.1. Conceptual Model and Terminology

113 An *identity web service* is defined as a type of web service whose operations are indexed by *identity*. Such services
114 maintain information about, or on behalf of, *Principals* — as represented by their *identities* — and/or perform actions
115 on behalf of Principals. They are also sometimes referred to as simply *identity services*.

116 There are various types of services, each of which is assigned a unique *service type* identifier, encoded as a *URI*
117 (Uniform Resource Identifier). This *service type URI* maps to exactly one version of an *abstract WSDL* definition of
118 a service, which contains the `<wsdl:types>`, `<wsdl:message>`, and `<wsdl:portType>` elements of a WSDL 1.1
119 description [[WSDLv1.1](#)].

120 An example of a type of identity web service is a Principal's "calendar service," which could be identified by a URI
121 such as `urn:example:services:calendar:2006-12`. Note the use of the year/month in the service type to identify the
122 version of the abstract WSDL.

123 A *service instance* is a deployed physical instantiation of a particular type of service. A *service provider* may deploy
124 one or more concrete service instances in the act of deploying a service.

125 A service instance may be described by a *concrete WSDL* document (including at least the `<wsdl:binding>`,
126 `<wsdl:service>`, and `<wsdl:port>` elements) which contains the *protocol endpoint* and additional information
127 necessary for a client to communicate with a particular service instance. An example of such "additional information"
128 is communication security policy information.

129 A service instance is hosted by some *provider*, identified by a URI. An example of a service instance is a SOAP-over-
130 HTTP endpoint offering a calendar service, being hosted by some provider.

131 Thus, a service instance exposes a protocol interface to a set of logical resources, nominally indexed by Principal. A
132 *resource* in this specification is either data related to some *Principal's identity* or a service acting on behalf of some
133 Principal. An example of a resource is a calendar containing appointments for a particular Principal. When a client
134 sends a request message to a service instance, information in the message serves to implicitly identify the resource
135 being acted upon. This is accomplished in one of the following fashions:

- 136 • Implicitly (e.g. PAOS exchange [[LibertyPAOS](#)]).
- 137 • Via a `<TargetIdentity>` header block [[LibertySOAPBinding](#)].
- 138 • Via supplied security token: it is presumed that a resource of the security token subject, i.e. the Principal itself, is
139 to be accessed.
- 140 • Via the endpoint. A service may choose to offer different endpoints for every resource. The simplest case of this
141 is to represent the resource as a part of the query string.

142 Caution should be exercised when using this unique endpoint solution as the use of unique endpoints for every
143 resource can release enough information to allow collusion across providers as to the identity of a principal (if
144 multiple providers get the same unique endpoint reference for their local principal, they can figure out that the
145 local principal on their respective environment is the same principal).

146 A resource commonly has access control policies associated with it. These access control policies are typically under
147 the purview of the entity or entities associated with the resource (in common language, the entity or entities could be
148 said to "own", or "manage", the resource). The access control policies associated with a resource must be enforced by
149 the service instance.

150 The Discovery Service defined here is not intended to be exclusive. Some identity services meeting the conceptual
151 model may be exposed via other discovery mechanisms. For example, [LibertyPAOS] defines an equivalent discovery
152 mechanism.

153 1.2. Scope

154 This specification:

- 155 • Specifies service instance endpoint description and enumeration via a profile of W3C Web Services Addressing
156 [WSAv1.0].
- 157 • Specifies a Discovery Service facilitating discovery and invocation of service instances.
- 158 • A SAML (see [SAMLCore2]) <Attribute> element defined such that an Endpoint Reference (EPR) for the
159 Discovery Service itself can be conveyed via SAML assertions. This is known as a *Discovery EPR* or *DS EPR* and
160 also colloquially as the *discovery bootstrap*.

161 1.3. Notation and Conventions

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

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

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

169 1.3.1. XML Namespaces

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

- 171 • The prefix *ds:* represents the Discovery Service namespace. This namespace is the default for instance fragments,
172 type names, and element names in this document. In schema listings, and in examples of Discovery Service
173 messages and fragments thereof, this is the default namespace *when* no prefix is shown:

174 *urn:liberty:disco:2005-11*

- 175 • The prefix *saml2:* stands for the SAMLv2 assertion namespace [SAMLCore2]:

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

- 177 • The prefix *samlp2:* stands for the SAMLv2 protocol namespace [SAMLCore2]:

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

- 179 • The prefix *sb:* stands for the Liberty Soap Bindings namespace [LibertySOAPBinding]:

180 *urn:liberty:sb:2005-11*

-
- 181 • The prefix *sbf*: stands for the Liberty Soap Bindings Framework namespace [[LibertySOAPBinding](#)]:
182 *urn:liberty:sb*
- 183 • The prefix *sec*: stands for the Liberty Security Mechanisms namespace [[LibertySecMech](#)]:
184 *urn:liberty:security:2005-11*
- 185 • The prefix *wsa*: stands for the W3C Web Services Addressing (WSA) namespace [[WSAv1.0](#)]:
186 *http://www.w3.org/@@@/@/@/addressing*
- 187 • The prefix *wSDL*: stands for the primary WSDL v1.1 namespace [[WSDLv1.1](#)]:
188 *http://schemas.xmlsoap.org/wSDL/*
- 189 • The prefix *wSDLsoap*: stands for the namespace of the WSDL-SOAP binding [[WSDLv1.1](#)]:
190 *http://schemas.xmlsoap.org/wSDL/soap/*
- 191 • The prefix *xs*: stands for the W3C XML schema namespace [[Schema1](#)]:
192 *http://www.w3.org/2001/XMLSchema*
- 193 • The prefix *xsi*: stands for the W3C XML schema instance namespace:
194 *http://www.w3.org/2001/XMLSchema-instance*

195 2. Discovery Service Information Model

196 This section describes the Discovery Service information model. This model comprises the various data types, and
197 thus information, that are maintained and managed by the Discovery Service, as well as the manner and format in
198 which this information is exchanged between the Discovery Service and its clients.

199 First, there is a brief non-normative overview describing how *service instances* are referenced, as well as the
200 interactions between the Discovery Service and the various other roles donned by system entities in the ID-WSF
201 framework. Next are the normative definitions of the various elements defined in this specification and used in
202 referencing service instances. Lastly is the Discovery Service WSA Profile, which normatively defines WSA EPRs
203 profiled for use in referencing ID-WSF service instances.

204 2.1. Overview of Discovery Service Information Model

205 A *service instance* is a web service at a distinct protocol endpoint. Information about service instances needs to
206 be communicated in various contexts. This specification defines a profile of WSA Endpoint References (EPRs)
207 [WSAv1.0][WSAv1.0-SOAP] such that they can be used to convey service instance information needed by entities
208 wishing to communicate with said service instances. Such "profiled EPRs" are termed "ID-WSF EPRs" in the
209 remainder of this specification.

210 The general model for ID-WSF system entity interactions from a Principal's perspective is as follows:

- 211 • A Principal wielding some *user agent* interacts with some *service provider* and is authenticated in some Liberty-
212 compliant fashion, such that the service provider obtains possession of a *discovery bootstrap* assertion for the
213 Principal. This assertion contains a pointer to the Principal's Discovery Service instance in the form of an ID-WSF
214 EPR.
 - 215 • Now, the service provider, acting as a *web service consumer* (WSC) and using the ID-WSF EPR obtained above,
216 queries the Principal's Discovery Service for a pointer to some other desired service of the Principal—e.g. the
217 Principal's Profile Service or Calendar Service.
 - 218 • The Discovery Service returns one or more ID-WSF EPRs to the querying WSC, pointing to the Principal's service
219 instance(s), of the requested type, if any.
 - 220 • The WSC now employs the returned ID-WSF EPR(s) to interact with the identified service instance(s), which
221 themselves will be acting in the role of a *web service provider*. The WSC returns results as appropriate to the
222 Principal's user agent.
- 223 There are various permutations of this general interaction model. For example, the Principal's user agent may
224 itself act in the role of a WSC. Or, a Principal may not be actively involved in a given interaction—a WSC is
225 simply interacting with a WSP on a Principal's behalf. For example, it may be renewing some contract, such as a
226 magazine subscription.

227 In order to enable the above Principal's-perspective model, there is a parallel model from the *web service provider's*
228 (WSP) perspective, which is as follows:

- 229 • A service instance(s), acting as a WSP, is deployed at some addressable endpoint(s). In this example, the WSP is
230 providing some service(s) on behalf of one or more Principals, e.g. a profile or calendar service.
- 231 • The WSP registers itself with the Discovery Service by inserting Service Metadata into the DS using the Service
232 Metadata maintenance operations (defined later in this specification). The Service Metadata describes the WSP's
233 service instance(s) such that the Discovery Service has the necessary information to mint ID-WSF EPRs for a
234 WSC to invoke that WSP.
- 235 • The Service Metadata, using appropriate Discovery Service protocol operations (defined later in this specification),
236 is then "associated" with a principal'.

- 237 • The above Principal’s-perspective model is now enabled.
- 238 There are various permutations of this general WSP-perspective service instance registration model. For example,
239 the same administrative entity may be deploying the both the Discovery Service and the other services and so
240 may employ alternative means, e.g. bulk configuration, to effect service metadata registration with their Discovery
241 Service.

242 2.2. Versioning in ID-WSF

243 Versioning applies to both the communications framework and the service itself within Liberty. The Discovery
244 Service is at the center of versioning in Liberty because it is the entity that matches the version capabilities of the
245 WSC to that of the WSP.

246 The specific areas of versioning include:

- 247 • **Service Versioning** — the version of the Service APIs that are available from a service instance.
- 248 The service version is carried in the `<ServiceType>` URI which actually carries the basic type of the service
249 (e.g. a Profile service or a Discovery Service) as well as the specific version of the service. For example, the
250 service type URI: `urn:liberty:disco:2005-11` represents the version of the Discovery Service defined in
251 this specification.
- 252 This URI is also typically used as the XML Namespace for the XML schema for that service, so the version
253 identifier typically shows up there as well – although this is NOT a normative requirement.
- 254 • **Framework Versioning** — the version of the communications framework used for ID-WSF messaging. Each ID-
255 WSF message has a potential collection of SOAP headers defined by the various ID-WSF specifications which are
256 tied together by the [[LibertyIDWSF20SCR](#)]. The [[LibertySOAPBinding](#)] specification defines the `<Framework>`
257 element which carries a description of the framework. As of this release that consists primarily of a `version`
258 attribute. [[LibertyIDWSF20SCR](#)] defines a particular version string to represent each concrete version of the
259 specifications.
- 260 The `Framework description` is included in ID-WSF messages, ID-WSF minted EPRs and in Discovery
261 Service `<Query>` operations (in other words, the framework description is actively specified at each stage of
262 the ID-WSF interaction model).

263 To ensure that the WSC communicates appropriately (from a versioning point of view) with the WSP, the WSC
264 specifies both the service and framework versions that it supports during discovery and the Discovery Service matches
265 the WSC capabilities with the appropriate registered service instances in order to return an EPR that the WSC can use.

266 2.3. ID-WSF Endpoint References (EPRs)

267 The general form of an EPR is illustrated in [Example 1](#).

```
268  
269 <wsa:EndpointReference ...>  
270   <wsa:Address>...some URI here...</wsa:Address>  
271  
272   <wsa:ReferenceParameters>.....</wsa:ReferenceParameters>  
273  
274   <wsa:Metadata>...some metadata here... </wsa:Metadata>  
275 </wsa:EndpointReference>  
276
```

277 Example 1. General Form of an EPR

278 The EPRs are profiled, as specified below in [Section 2.3.3](#), by placing Liberty-specific attributes and elements into
279 the EPR. Specifically, a few attributes on the EPR itself and some sub-elements within `<wsa:Metadata>` element
280 of the EPR. These Liberty-specific components are defined below in [Section 2.3.1: EPR Profiling Attributes](#) and

281 [Section 2.3.2: EPR Profiling Elements](#) . These profiled EPRs are referred to as "ID-WSF EPRs", [Example 2](#) illustrates
282 an ID-WSF EPR.

283 **Note**

284 The use of these profiled EPRs does not necessarily replace WSDL; rather, they may be used either in conjunction
285 with WSDL or without. This is also described in [Section 2.3.3](#).

```
286  
287 <wsa:EndpointReference  
288     notOnOrAfter="2005-08-15T23:18:56Z"  
289     ...>  
290   <wsa:Address>  
291     https://profile-provider.com/profiles/someFoobarProfile  
292   </wsa:Address>  
293  
294   <wsa:Metadata>  
295     <ds:Abstract>  
296       This is a personal profile containing common name information.  
297     </ds:Abstract>  
298  
299     <ds:ProviderID>http://profile-provider.com/</ds:ProviderID>  
300  
301     <ds:ServiceType>&PS1Namespace;</ds:ServiceType>  
302  
303     <ds:Framework version="2.0" />  
304  
305     <ds:SecurityContext>  
306       <ds:SecurityMechID>  
307         urn:liberty:security:2005-02:TLS:SAML  
308       </ds:SecurityMechID>  
309  
310       <sec:Token>  
311         <!-- some security token goes here -->  
312       </sec:Token>  
313     </ds:SecurityContext>  
314  
315     <ds:Options>  
316       <ds:Option>urn:liberty:id-sis-pp</ds:Option>  
317       <ds:Option>urn:liberty:id-sis-pp:cn</ds:Option>  
318       <ds:Option>urn:liberty:id-sis-pp:can</ds:Option>  
319       <ds:Option>urn:liberty:id-sis-pp:can:cn</ds:Option>  
320     </ds:Options>  
321   </wsa:Metadata>  
322 </wsa:EndpointReference>  
323
```

324 **Example 2. An Instantiated ID-WSF EPR**

325 **2.3.1. EPR Profiling Attributes**

326 This section defines the attributes that are used to profile EPRs as defined below in [Section 2.3.3: ID-WSF Web](#)
327 [Services Addressing EPR Profile](#) . The full Discovery Service schema is given in [Appendix A: Discovery Service](#)
328 [Version 2.0 XSD](#) .

329 **2.3.1.1. wsu:Id — unique identifier for xml references to an EPR.**

330 The `wsu:Id` attribute ([Figure 1](#)) is used when other elements in the XML document (e.g. message) need to refer to
331 this EPR (for example, when this element is referenced in an XML signature).

```
332
333
334     <!-- wsu:Id attribute for EPR - for xml document references to EPR -->
335
336     <xs:attribute ref="wsu:Id" use="optional" />
337
```

338 **Figure 1. wsu:Id — Schema Fragment**

339 2.3.1.2. reqRef — request reference

340 The reqRef attribute ([Figure 2](#)) identifies which <RequestedServiceType> element in the Discovery Service
341 <Query> request that this EPR was minted in response to. In other words this is used to associate the EPR in
342 the <QueryResponse> with the <RequestedServiceType> in the <Query> request.

```
343
344     <!-- Request Reference - to id which Request generated EPR -->
345
346     <xs:attribute name="reqRef" type="xs:string" use="optional"/>
347
```

348 **Figure 2. reqRef — Schema Fragment**

349 2.3.1.3. notOnOrAfter

350 The notOnOrAfter attribute states the expiration timestamp for the EPR with which it is associated ([Figure 3](#)). See
351 [Example 2](#), above, for an instantiated EPR example.

352 Values of the notOnOrAfter attribute MUST be expressed in accordance with Liberty ID-WSF time value restric-
353 tions.

354 Liberty system entities SHOULD NOT rely on time resolution finer than milliseconds. Implementations MUST NOT
355 generate time instants that specify leap seconds.

```
356
357     <!-- EPR Expiration Timestamp -->
358
359     <xs:attribute name="NotOnOrAfter" type="xs:dateTime"/>
360
361
```

362 **Figure 3. notOnOrAfter — Schema Fragment**

363 2.3.2. EPR Profiling Elements

364 This section defines the elements that are used to profile EPRs as defined below in [Section 2.3.3: ID-WSF Web](#)
365 [Services Addressing EPR Profile](#) . The full Discovery Service schema is given in [Appendix A: Discovery Service](#)
366 [Version 2.0 XSD](#) .

367 2.3.2.1. Abstract

368 The <Abstract> element ([Figure 4](#)) is used for conveying a textual, natural language description of the service
369 instance.

```
370
371 <!-- Abstract: natural-language description of service -->
372
373 <xs:element name="Abstract" type="xs:string"/>
374
375
```

376 **Figure 4. <Abstract> — Schema Fragment**

377 2.3.2.2. Provider ID

378 The <ProviderID> element (Figure 5) contains the URI of the provider of this service instance.

```
379
380 <!-- Provider ID -->
381
382 <xs:element name="ProviderID" type="xs:anyURI"/>
383
384
```

385 **Figure 5. <ProviderID> — Schema Fragment**

386 2.3.2.3. Service Type

387 The <ServiceType> element (Figure 6) is used to identify a service type and version. This URI needs be constant
388 across all implementations of a service to enable interoperability. Therefore, it is RECOMMENDED that this URI be
389 the same as the targetNamespace URI of the abstract WSDL description for the service.

```
390
391 <!-- Service Type -->
392
393 <xs:element name="ServiceType" type="xs:anyURI"/>
394
395
```

396 **Figure 6. <ServiceType> — Schema Fragment**

397 Some examples of possible ServiceType URIs:

```
398 urn:liberty:disco:2005-11
399 urn:liberty:id-sis-pp:2003-08
400 http://myservices.com/gaming/1.0
```

401 2.3.2.4. SecurityContext

402 The <SecurityContext> element (Figure 7) is a container in which <SecurityMechID> elements and
403 <sec:Token> elements are placed and thus associated with an ID-WSF EPR. The <sec:Token> element is used to
404 either directly contain, or reference, security tokens and/or identity tokens.

405 Therefore, the <SecurityContext> element serves to denote the invocation context necessary for interacting with
406 the service instance represented by the containing ID-WSF EPR.

407 NOTE: in some cases the DS will **not** be able to generate the necessary tokens to complete the security context. This
408 will usually happen when a context needs a security token from a provider other than the DS (such as a non-related
409 IdP). In such cases, the DS will include an empty token element with the `ref` attribute set to the following URI:

410 • urn:liberty:disco:tokenref:ObtainFromIDP
411 In such cases, the WSC receiving the EPR MUST communicate with the invoking principal's IdP's SSO Service (see
412 [LibertyAuthn]) in order to obtain the necessary security token.

413 The value of the security mechanism in the security context will identify the type of security token that the WSC
414 should request from the IdP. For example, if the security mechanism was "urn:liberty:....:SAMLV2", the WSC would
415 know they needed a SAML 2.0 token with a subject confirmation of "....:holder-of-key" and would indicate so on the
416 SSO Service request.

417 An ID-WSF EPR MAY contain more than one <SecurityContext> element. This serves to denote mutually-
418 exclusive groupings of <SecurityMechID>s and <sec:Token>s, and thus different security contexts.

419 See [Section 2.3.3: ID-WSF Web Services Addressing EPR Profile](#), below, for the precise specification of the mapping
420 of <SecurityContext>, and its contents, to ID-WSF EPRs.

```
421
422 <!-- Security Context Container -->
423
424 <xs:element name="SecurityContext">
425   <xs:complexType>
426     <xs:sequence>
427       <xs:element ref="SecurityMechID"
428         minOccurs="1"
429         maxOccurs="unbounded" />
430
431       <xs:element ref="sec:Token"
432         minOccurs="0"
433         maxOccurs="unbounded" />
434     </xs:sequence>
435   </xs:complexType>
436 </xs:element>
437
438
```

439 **Figure 7. <SecurityContext> — Schema Fragment**

440 See [Example 2](#), above, for an instantiated ID-WSF EPR example, containing a <SecurityContext> element, itself
441 containing <SecurityMechID> and <sec:Token> elements.

442 **2.3.2.5. SecurityMechID**

443 The <SecurityMechID> element ([Figure 8](#)) specifies the security mechanism(s) supported by the service instance
444 represented by the ID-WSF EPR. These security mechanisms are represented as URIs, and are defined in [[Liberty-
445 SecMech](#)].

446 The <SecurityMechID> element is used within the <SecurityContext> element described above. This is detailed
447 in the [Section 2.3.3: ID-WSF Web Services Addressing EPR Profile](#) section below.

```
448
449 <!-- Security Mechanism ID -->
450
451 <xs:element name="SecurityMechID" type="xs:anyURI"/>
452
453
```

454 **Figure 8. <SecurityMechID> — Schema Fragment**

455 Some examples of possible SecurityMechID URI values (from [[LibertySecMech](#)]):

456 urn:liberty:security:2004-12:ClientTLS:SAMLV2

457 urn:liberty:security:2003-08:ClientTLS:SAML

458 urn:liberty:security:2004-12:TLS:SAMLV2

459 See [Example 2](#), above, for an instantiated ID-WSF EPR example, containing a <SecurityContext> element
460 containing <SecurityMechID> and <sec:Token> elements.

461 2.3.2.6. Framework

462 The <Framework> element ([Figure 9](#)) identifies the Liberty ID-WSF framework supported by the service instance at
463 this endpoint. There MUST be at least one <Framework> element within an EPR.

464 Multiple <Framework> elements indicate that the service instance supports any of the specified ID-WSF versions at
465 this same endpoint.

466 The structure and content of this element is defined in [[LibertySOAPBinding](#)].

```
467  
468 <!-- Framework Description -->  
469  
470 <xs:element ref="sb:Framework" maxOccurs="unbounded" />  
471  
472
```

473 **Figure 9. <Framework> — Schema Fragment**

474 2.3.2.7. Action

475 The optional multi-occurrence <Action> element ([Figure 10](#)) is used to identify the set of interfaces exposed by the
476 provider at this endpoint.

477 Each <Action> element contains a URI that MUST match one of the <wsa:Action> URIs defined for the service.

478 When there are no <Action> elements in an EPR, the EPR can be used to invoke **all** of the interfaces for the defined
479 service type.

480 This element is typically only included when the service instance specified in the EPR can only address a sub-set of the
481 service's interfaces. A service instance may do this to scale their resources across different interfaces. For example,
482 a service instance of the personal profile service may support the Query interface on a large cluster of systems, but
483 require that the less frequently called, modify operations take place on some dedicated hardware.

```
484  
485 <!-- Action(s) - the interfaces available at this service -->  
486  
487 <xs:element name="Action" type="xs:anyURI" />  
488
```

489 **Figure 10. <Action> — Schema Fragment**

490 2.3.2.8. Options

491 The <Options> element ([Figure 11](#)) expresses the "options" supported by a service instance. Thus they provide hints
492 to a potential requester whether certain data or operations may be available with a particular service instance.

493 For example, an option may be provided stating that home contact information is available. If no Options element is
494 present, it means only that the service instance does not advertise whether any options are available. Options may,

495 in fact, be employed by the service instance. For example, it may be a simple service that is not capable of updating
496 its entry in the Discovery Service when the available options change, so it avoids listing them at all. If the Options
497 element is present, but is empty, it means that the service instance explicitly advertises that no options are available.

```
498  
499 <!-- Options -->  
500  
501 <xs:element name="Options" type="OptionsType"/>  
502  
503 <xs:element name="Option" type="xs:anyURI" />  
504  
505 <xs:complexType name="OptionsType">  
506   <xs:sequence>  
507     <xs:element ref="Option" minOccurs="0" maxOccurs="unbounded"/>  
508   </xs:sequence>  
509 </xs:complexType>  
510  
511
```

512 **Figure 11. <Options> — Schema Fragment**

513 The <Options> element contains zero or more <Option> elements, each of which contains a URI identifying a
514 particular option. The set of possible URIs for an <Option> element should be defined by the service type. For
515 example, a person profile service specification would specify a set of options particular to its own domain. However,
516 one common <Option> flag related to security, and thus common to ID-WSF services, is defined in [Section 3.11](#):
517 Option Value for Response Authentication .

518 2.3.3. ID-WSF Web Services Addressing EPR Profile

519 This section specifies the profile of WSA Endpoint References (EPRs). Profiling an EPR, yielding an ID-WSF EPR,
520 is accomplished by placing various of the elements defined in [Section 2.3.2: EPR Profiling Elements](#) , above, into
521 the EPR's <wsa:Metadata> element according to the rules defined below. All ID-WSF EPRs must adhere to the
522 per-element rules in [Section 2.3.2](#), and thereupon adhere to the rules defined in the following sections, depending
523 upon the intended usage scenario for the ID-WSF EPR being minted.

524 For reference, the general form of an instantiated EPR is illustrated above in [Example 1](#), and the
525 <wsa:EndpointReference> schema fragment [[WSAv1.0-SOAP](#)] is illustrated below in [Figure 12](#).

526 An ID-WSF EPR is normatively defined as a <wsa:EndpointReference> profiled as per this section.

527 **Note**

528 Except for the <wsa:Address> and <wsa:ReferenceParameters> elements, all elements discussed in the below
 529 sections are denoted as either being "absent" or "present" as content of the <wsa:Metadata> element of the ID-WSF
 530 EPR being minted.

```

531
532 <xs:element name="EndpointReference" type="tns:EndpointReferenceType" />
533
534 <xs:complexType name="EndpointReferenceType">
535   <xs:sequence>
536     <xs:element name="Address"
537       type="tns:AttributedURIType" />
538
539     <xs:element name="ReferenceParameters"
540       type="tns:ReferenceParametersType"
541       minOccurs="0" />
542
543     <xs:element ref="tns:Metadata"
544       minOccurs="0" />
545
546     <xs:any namespace="##other"
547       processContents="lax"
548       minOccurs="0"
549       maxOccurs="unbounded" />
550   </xs:sequence>
551
552   <xs:anyAttribute namespace="##other" processContents="lax" />
553 </xs:complexType>
554
555
556
```

557 **Figure 12. <wsa:EndpointReference> — Schema Fragment**

558 **2.3.3.1. ID-WSF EPR Minting Rules**

559 ID-WSF EPRs are minted by both the Discovery Service (in response to <Query> requests) and by system entities
 560 acting as in a WSC or WSP role for inclusion in SOAP message header blocks such as the <wsa:ReplyTo> and the
 561 <wsa:FaultTo>, as discussed in [[LibertySOAPBinding](#)]. This section refers to these different parties collectively
 562 as "issuers".

563 The following rules **MUST** be observed by issuers when constructing an ID-WSF EPR:

- 564 1. A `notOnOrAfter` attribute **MAY** be present in each ID-WSF EPR. If absent, or if it has a value of `1970-01-`
 565 `01T00:00:00Z`, it means the issuer is not stipulating an expiration time for this ID-WSF EPR, and that its wielder
 566 is obliged to follow its own local policy for refreshing any cached copies. If present, the value should be set by
 567 the issuer according to local policy.
- 568 2. The value of the <wsa:Address> element **MUST** contain the endpoint address of the service instance being
 569 described by this EPR. This literally-addressed form of ID-WSF EPR is useful in order to ease the burden
 570 of WSCs from having to retrieve and parse WSDL in common cases. Additionally, the rules specified in
 571 [Section 2.3.3.2: ID-WSF EPR Specifics](#) **MUST** be adhered to.
- 572 3. A `wsu:Id` attribute **MAY** be present on the EPR root element.
- 573 4. A `reqRef` attribute **MAY** be present on the EPR root element.
- 574 5. Exactly one <Abstract> element **MAY** be present in the EPR <Metadata> element.

- 575 6. Exactly one <ProviderID> element MUST be present in the EPR <Metadata> element.
- 576 7. One or more <ServiceType> elements MUST be present in the EPR <Metadata> element.
- 577 8. One or more <Framework> elements MUST be present in the EPR <Metadata> element.
- 578 9. Optionally, one or more <Options> element(s). These are discussed in detail above, in [Section 2.3.2.8](#).
- 579 10. Optionally, one or more <Action> element(s). These are discussed in detail above, in [Section 2.3.2.7](#).
- 580 11. One or more <SecurityContext> elements SHOULD be present in each ID-WSF EPR. If so they, and their
581 content, MUST adhere to the rules below, as well as the additional specific rules in [Section 2.3.3.3: Security](#)
582 Mechanism Specifics :
- 583 a. If no security or identity tokens are to be embedded, then place all the supported security mechanisms,
584 denoted by <SecurityMechID> elements, in a single <SecurityContext> element.
- 585 b. Else, if security and/or identity tokens are to be embedded or referenced (via <sec:Token> ele-
586 ments), then one MUST group corresponding <SecurityMechID> and <sec:Token> elements into
587 the same <SecurityContext> element. In other words, all security and identity tokens within a
588 <SecurityContext> element MUST apply to ALL of the security mechanisms in the same context.
- 589 c. A security and/or identity token embedded in a <sec:Token> in a given ID-WSF EPR's
590 <SecurityContext> element MAY be referenced from other <SecurityContext> elements, whether
591 the other <SecurityContext> elements are contained within the given ID-WSF EPR or whether they are
592 in another ID-WSF EPR in the list of ID-WSF EPRs being constructed.
- 593 Such referencing is accomplished by using the ref attribute of a <sec:Token> element. When constructing
594 such a reference, the referencing <sec:Token> MUST reference the <sec:Token> element containing the
595 target embedded security token, as specified in [[LibertySecMech](#)].
- 596 d. All <sec:Token> elements included in the <SecurityContext> element MUST have the usage attribute
597 set to the appropriate value (as documented in [[LibertySecMech](#)]) indicating their intended purpose.
- 598 e. If the issuer is unable to generate a necessary token, it MUST include an empty <sec:Token> element with
599 the ref attribute set to the value urn:liberty:disco:tokenref:ObtainFromIDP

600 2.3.3.2. ID-WSF EPR Specifics

601 The information contained in an ID-WSF EPR is sufficient for making invocations for service instances. In other
602 words, the information contained in this group together with the abstract WSDL specified by the ServiceType URI is
603 sufficient to logically compute concrete WSDL with the rule set specified below.

604 The <wsa:Address> element of the ID-WSF EPR contains the URI of the endpoint. For SOAP-over-HTTP
605 endpoints, the URI scheme MUST be "http" or "https".

606 Use of this addressing form implies <wsdl:binding> and <wsdl:service> elements according to the following
607 rules (i.e., the concrete WSDL can be logically computed given the abstract WSDL and an ID-WSF EPR):

- 608 • The <wsdl:binding> contains a <wsdlsoap:binding> element. This specifies that the SOAP binding for
609 WSDL is being used.
- 610 • The style attribute of the <wsdlsoap:binding> element is "document".
- 611 • The transport attribute of the <wsdlsoap:binding> element is *http://schemas.xmlsoap.org/soap/http*.

- 612 • The abstract WSDL corresponding to the `<ServiceType>` MUST contain a single `<portType>` element. The
613 `<wsdl:binding>` element provides bindings for the operations specified in this `<wsdl:portType>`. Each
614 operation binding includes an input element and an output element, each containing a single `<wsdlsoap:body>`
615 element. The `use` attribute of the `<wsdlsoap:body>` elements is "literal".
- 616 • The `location` attribute of `<wsdlsoap:address>` is equal to `<wsa:Address>`.
- 617 • All other optional elements and attributes are not specified and thus default to the SOAP binding of WSDL.

618 2.3.3.3. Security Mechanism Specifics

619 With respect to `<SecurityMechID>` URIs: these URIs denote the security mechanisms supported by the service
620 instance described by the ID-WSF EPR. Other specifications, such as [\[LibertySecMech\]](#) define the actual security
621 mechanisms along with their identifying URIs. These security mechanisms refer to the way a WSC authenticates to a
622 WSP ("peer-entity authentication") and/or provides message security ("data-origin authentication").

623 An ID-WSF EPR SHOULD list all of the security mechanisms that the service instance supports in order of preference.
624 I.e. the most preferred security mechanism is first in the list, the next is the second-most preferred, and so on.

625 In the case that the set of supported security mechanisms varies with respect to endpoint address(es) and/or WSDL
626 binding, the system entity constructing the ID-WSF EPRs MUST construct multiple ID-WSF EPRs with each ID-WSF
627 EPR separately representing each supported mapping.

628 Also, any single `<SecurityMechID>` URI MUST NOT appear in more than one of the `<SecurityContext>`
629 elements of any of the ID-WSF EPRs so constructed. In other words, each service instance may only specify one
630 WSDL binding per supported security mechanism. If a sequence of ID-WSF EPRs is constructed, then the ID-WSF
631 EPRs SHOULD appear in the order of the constructor's preference, and the `<SecurityContext>` elements within
632 each should be in order of preference, as should the `<SecurityMechID>` elements within them—with the most
633 preferred item listed first in each case.

634 For example: many web servers will require a different endpoint URI to be used for SOAP/HTTP clients authenticating
635 using client TLS certificates than for clients which authenticate in some other fashion. See [Example 4](#).

636 2.3.3.4. Action Specifics

637 With respect to `<Action>` URIs: these URIs denote the interfaces supported by the service instance described by the
638 ID-WSF EPR. The service specific specifications, such as this document, define the actual interfaces along with their
639 identifying URIs.

640 An ID-WSF EPR SHOULD NOT list actions unless the service instance at this endpoint does not support the complete
641 set of service interfaces. In such a case, the ID-WSF EPR SHOULD list all of the available interfaces.

642 There is no preference or other significance to the ordering of the `<Action>` URIs.

643 2.3.3.5. Identity Invocation Context specifics

644 The invocation of an ID-WSF service can carry several identities as documented in [\[LibertySOAPBinding\]](#). These
645 identities include the `Sender`, the `InvocationIdentity`, the `TargetIdentity`, and the `Recipient`.

646 The Discovery Service, when minting ID-WSF EPRs, works to maintain the same identity invocation context that
647 was used to invoke it such that the same logical `Sender`, `InvocationIdentity` and `TargetIdentity` are carried
648 forth in messages invoked through the minted EPR. Of course, the `Recipient` of the subsequent invocation will be
649 different as it will be the WSP to which this EPR points.

650 The Discovery Service generates security and/or identity tokens to convey these identities in the minted ID-WSF EPR.
651 These tokens are placed into the `<sec:Token>` elements within `<SecurityContext>` element.

652 In preparing the necessary tokens to carry forth these identities, the Discovery Service may have to perform identity
653 translations to obtain pseudonymous identifiers for the interested parties at the intended `Recipient`.

654 The rules for when and how the tokens are generated when the ID-WSF EPR is minted by the Discovery Service (in
655 response to a `DiscoveryQuery` operation, see [Section 3.3](#)), are as follows:

656 • If the `Principal`, whose discovery resource is being queried, is the same as the invocation identity of the
657 `DiscoveryQuery` operation — i.e. there is not a `<sb:TargetIdentity>` header block on the `<Query>`
658 message — then the same effective invocation identity **MUST** be expressed by the Discovery Service's resultant
659 selected security tokens for the invocation identity (which are embedded in `<sec:Token>` element(s) in the
660 `<SecurityContext>` element in the ID-WSF EPR's `<wsa:Metadata>` element)

661 **Note**

662 Since the security tokens usually carry the identity of the `Sender` and that of the `InvocationIdentity` it is
663 possible that a single `<SecurityContext>` may include multiple security tokens identifying each of the parties.

664 • Else, if the `Principal`, whose discovery resource is being queried, is not the same as the invocation identity of
665 the `DiscoveryQuery` operation — i.e. a `<sb:TargetIdentity>` header block appears in the header of the
666 `<Query>` message — then the invocation identity to be conveyed in the ID-WSF EPR is expressed as denoted in
667 the bullet item above, and additionally, a identity token denoting the target identity (per [\[LibertySecMech\]](#) and
668 [\[LibertySecMech20SAML\]](#)) is also embedded in a `<sec:Token>` element in the `<SecurityContext>` element
669 in the ID-WSF EPR's `<wsa:Metadata>` element.

670 The rules for when and how the above identity tokens are included as above when the ID-WSF EPR is minted by a
671 WSC or WSP (refer to [Section 2.3.3.1](#), above, for context), are as follows:

672 • If the intended target identity is to be the same as that of the intended invocation identity, then the intended
673 invocation identity **MUST** be expressed in the minted ID-WSF EPR as detailed in the rules above (first bullet
674 item).

675 • If the intended target identity is to be different than the intended invocation identity, then the intended invocation
676 identity and the intended target identity both **MUST** be expressed in the minted ID-WSF EPR as detailed in the
677 rules above (second bullet item).

678 The recipient of an ID-WSF EPR distinguishes between the various tokens contained within a `<sec:Token>` element
679 via the `usage` attribute as follows:

680 • A token with the `usage` attribute set to `urn:liberty:security:tokenusage:2005-11:SecurityToken`
681 contains a security token that **MUST** be placed into the `<wsse:Security>` header block (according to [\[Liberty-](#)
682 [SecMech\]](#) and its related profiles) when a message is generated for the target of the ID-WSF EPR.

683 If multiple `<sec:Token>`s are included in a single `<ds:SecurityContext>`, they **MUST ALL** be placed into
684 the same `<wsse:Security>` header block.

685 • A token with the `usage` attribute set to `urn:liberty:security:tokenusage:2005-11:TargetIdentity`
686 contains an identity token that **MUST** be placed into the `<sb:TargetIdentity>` header block (according to
687 [\[LibertySOAPBinding\]](#)) when a message is generated for the target of the ID-WSF EPR.

688 2.3.4. Effective Web Services Addressing EPR

689 The net effect of the ID-WSF profile of the EPR is as if the `EndpointReferenceType` were defined with the schema
690 fragment below. There are several things to note about this schema including:

691 • There is no normative XML schema defined as such, this is just an approximation of what the schema could look
692 like.

- 693 • While the elements within the <Metadata> element appear to be ordered, they can all appear in any order and can
694 have other elements appear between the listed elements. This is why they are contained within a multi-occurrence
695 <xs:choice>.
- 696 • Four attributes have been added to the EPR element itself: the notOnOrAfter timestamp and three different IDs
697 (for use in different circumstances).
- 698 • Seven sub-elements were added to the <Metadata> element.
- 699 • The <Metadata> sub-elements: <disco:ProviderID>, and <disco:ServiceType> **MUST** all appear exactly
700 once in an ID-WSF EPR, even though the schema below does not enforce that requirement (because of a limitation
701 in XML Schema – or perhaps in the author’s understanding of XML schema).

```

702
703 ...
704 <xs:element name="EndpointReference" type="tns:EndpointReferenceType"/>
705 <xs:complexType name="EndpointReferenceType" mixed="false">
706   <xs:sequence>
707     <xs:element name="Address" type="tns:AttributedURIType"/>
708     <xs:element name="ReferenceParameters"
709       type="tns:ReferenceParametersType" minOccurs="0"/>
710     <xs:element ref="tns:Metadata" minOccurs="0"/>
711     <xs:any namespace="##other" processContents="lax"
712       minOccurs="0" maxOccurs="unbounded"/>
713   </xs:sequence>
714   <xs:attribute name="notOnOrAfter" type="xs:dateTime" use="optional" />
715   <xs:attribute ref="wsu:Id" use="optional" />
716   <xs:attribute name="reqRef" type="xs:string" use="optional" />
717   <xs:anyAttribute namespace="##other" processContents="lax"/>
718 </xs:complexType>
719
720 <xs:complexType name="ReferenceParametersType" mixed="false">
721   <xs:sequence>
722     <xs:any namespace="##any" processContents="lax"
723       minOccurs="0" maxOccurs="unbounded"/>
724   </xs:sequence>
725   <xs:anyAttribute namespace="##other" processContents="lax"/>
726 </xs:complexType>
727
728 <xs:element name="Metadata" type="tns:MetadataType"/>
729 <xs:complexType name="MetadataType" mixed="false">
730   <xs:sequence>
731     <xs:choice minOccurs="0" maxOccurs="unbounded">
732       <xs:element ref="disco:Abstract" minOccurs="0" />
733       <xs:element ref="sbf:Framework" maxOccurs="unbounded" />
734       <xs:element ref="disco:ProviderID" />
735       <xs:element ref="disco:ServiceType" />
736       <xs:element ref="disco:SecurityContext" minOccurs="0" maxOccurs="unbounded" />
737       <xs:element ref="disco:Options" minOccurs="0" maxOccurs="unbounded" />
738       <xs:element ref="disco:Action" minOccurs="0" maxOccurs="unbounded" />
739     <xs:any namespace="##other" processContents="lax"
740       minOccurs="0" maxOccurs="unbounded"/>
741   </xs:choice>
742 </xs:sequence>
743   <xs:anyAttribute namespace="##other" processContents="lax"/>
744 </xs:complexType>
745

```

746 **Example 3. Effective ID-WSF EPR Schema**

747 2.3.5. Example Liberty ID-WSF EPRs

```

748
749 <wsa:EndpointReference
750     notOnOrAfter="2005-08-15T23:18:56Z"
751     ...>
752 <wsa:Address>
753     http://profile-provider.com/profiles/someFoobarProfileAddr
754 </wsa:Address>
755
756 <wsa:Metadata>
757     <ds:Abstract>
758         This is a personal profile containing common name information.
759     </ds:Abstract>
760
761     <ds:ProviderID>http://profile-provider.com/</ds:ProviderID>
762
763     <ds:ServiceType>urn:liberty:id-sis-pp:2003-08</ds:ServiceType>
764
765     <sbef:Framework version="2.0" />
766
767     <ds:SecurityContext>
768         <ds:SecurityMechID>
769             urn:liberty:security:2004-12:ClientTLS:SAMLV2
770         </ds:SecurityMechID>
771
772         <ds:SecurityMechID>
773             urn:liberty:security:2005-02:ClientTLS:SAML
774         </ds:SecurityMechID>
775
776         <sec:Token wsu:id="_10"
777             usage="urn:liberty:security:tokenusage:2005-11:SecurityToken">
778             <!-- some security token goes here -->
779         </sec:Token>
780     </ds:SecurityContext>
781
782     <ds:SecurityContext >
783         <ds:SecurityMechID>
784             urn:liberty:security:2005-02:ClientTLS:X509
785         </ds:SecurityMechID>
786
787         <sec:Token wsu:id="_20"
788             usage="urn:liberty:security:tokenusage:2005-11:InvocationIdentity">
789             <!-- Identity Token goes here -->
790         </sec:Token>
791     </ds:SecurityContext>
792
793     <ds:Options>
794         <ds:Option>urn:liberty:id-sis-pp</ds:Option>
795         <ds:Option>urn:liberty:id-sis-pp:cn</ds:Option>
796         <ds:Option>urn:liberty:id-sis-pp:can</ds:Option>
797         <ds:Option>urn:liberty:id-sis-pp:can:cn</ds:Option>
798     </ds:Options>
799 </wsa:Metadata>
800 </wsa:EndpointReference>
801
802 <wsa:EndpointReference
803     notOnOrAfter="2005-08-15T23:18:56Z"
804     ...>
805 <wsa:Address>
806     http://profile-provider.com/profiles/anotherFoobarProfileEndpointAddr
807 </wsa:Address>
808
809 <wsa:Metadata>
810     <ds:Abstract>
811         This is a personal profile containing common name information.
812     </ds:Abstract>
813
814     <ds:ProviderID>http://profile-provider.com/</ds:ProviderID>

```

```

815
816     <ds:ServiceType>urn:liberty:id-sis-pp:2003-08</ds:ServiceType>
817
818     <sb:Framework version="2.0" />
819
820     <ds:SecurityContext>
821         <ds:SecurityMechID>
822             urn:liberty:security:2004-12:TLS:SAMLV2
823         </ds:SecurityMechID>
824
825         <sec:Token ref="_10" usage="urn:liberty:security:tokenusage:2005-11:SecurityToken" />
826     </ds:SecurityContext>
827
828     <ds:Options>
829         <ds:Option>urn:liberty:id-sis-pp</ds:Option>
830         <ds:Option>urn:liberty:id-sis-pp:cn</ds:Option>
831         <ds:Option>urn:liberty:id-sis-pp:can</ds:Option>
832         <ds:Option>urn:liberty:id-sis-pp:can:cn</ds:Option>
833     </ds:Options>
834 </wsa:Metadata>
835 </wsa:EndpointReference>
836

```

837 **Example 4. Instantiated List of ID-WSF EPRs Illustrating Multiple <SecurityContext> Elements with both Embedded**
838 **and Referenced <sec:Token> Elements**

839 2.4. Service Metadata

840 The discovery Service mints the ID-WSF EPRs described in the previous section using information provided by the
841 WSP in the WSP's registered Service Metadata.

842 2.4.1. Service Metadata element

843 The Service Metadata is used to describe a single instance of a service hosted by a WSP as it applies to all principals
844 (i.e. the principal independent information related to an instance).

845 This single instance can include multiple endpoints, multiple security mechanisms, and even multiple service
846 types. Multiple service types SHOULD only be included in a single Service Metadata element if the WSP
847 considers those service types to be different versions of the same service (for example, *urn:liberty:disco:2005-11*
848 and *urn:liberty:disco:2003-08* are two different versions of the Liberty ID-WSF Discovery Service).

849 Most of the fields present in the Service Metadata have the same purpose and meaning as the elements of the same
850 name in the ID-WSF EPR (as this is where the Discovery service gets those elements for the ID-WSF EPR).

851 When fields permit multiple values, the order of entries in the SvcMD is significant with higher preference items
852 coming first. This comes into plan should the WSC request a subset of the possible results when querying the
853 Discovery Service (in which case the entries with the higher preference – those listed first – would be used to mint the
854 ID-WSF EPRs in the response).

```

855
856 <!-- Service Metadata (SvcMD) - metadata about service instance -->
857
858 <xs:element name="SvcMD" type="SvcMetadataType"/>
859 <xs:complexType name="SvcMetadataType">
860   <xs:sequence>
861     <xs:element ref="Abstract"           />
862     <xs:element ref="ProviderID"        />
863     <xs:element ref="ServiceContext"    maxOccurs="unbounded" />
864   </xs:sequence>
865   <xs:attribute name="svcMDID" type="xs:string" use="optional" />
866 </xs:complexType>
867
868 <!-- ServiceContext - describes service type/option/endpoint context -->
869 <xs:element name="ServiceContext" type="ServiceContextType"/>
870 <xs:complexType name="ServiceContextType">
871   <xs:sequence>
872     <xs:element ref="ServiceType"      maxOccurs="unbounded" />
873     <xs:element ref="Options"          minOccurs="0"
874                                     maxOccurs="unbounded" />
875     <xs:element ref="EndpointContext" maxOccurs="unbounded" />
876   </xs:sequence>
877 </xs:complexType>
878
879 <!-- EndpointContext - describes endpoints used to access service -->
880 <xs:element name="EndpointContext" type="EndpointContextType"/>
881 <xs:complexType name="EndpointContextType">
882   <xs:sequence>
883     <xs:element ref="Address"          maxOccurs="unbounded" />
884     <xs:element ref="sbf:Framework"  maxOccurs="unbounded" />
885     <xs:element ref="SecurityMechID"  maxOccurs="unbounded" />
886     <xs:element ref="Action"          minOccurs="0"
887                                     maxOccurs="unbounded" />
888   </xs:sequence>
889 </xs:complexType>
890
891 <!-- SvcMD ID element used to refer to Service Metadata elements -->
892 <xs:element name="SvcMDID" type="xs:string" />
893

```

894 **Figure 13. Service Metadata — Schema Fragment**

895 **2.4.1.1. svcMDID**

896 The `svcMDID` attribute is a unique identifier assigned by the Discovery Service during service metadata registration
897 and used on later principal registrations.

898 The value of the identifier **MUST** be unique across all registered service metadata for the registering WSP at the DS
899 and **MAY** be unique across all WSPs.

900 **2.4.1.2. Abstract**

901 A text description of the service.

902 **2.4.1.3. ProviderID**

903 The URI of the provider of this service instance.

904 **2.4.1.4. ServiceContext**

905 The `<ServiceContext>` describes the set of service versions and options that are available at a particular set of
906 endpoints. A Service Metadata description may have multiple `<ServiceContext>`s when they support a particular
907 version (or set of options) of the service at one set of endpoints and another version at a different set of endpoints.

908 The elements contained within a `<ServiceContext>`s are discussed below:

909 **2.4.1.4.1. ServiceType**

910 The URI of which defines the type of service.

911 Note that there may be multiple service types defined in a service metadata indicating that multiple distinct services
912 are available at the same endpoint. This typically occurs when multiple versions of the same general type of service
913 are available at the same endpoint although it is possible that very different services could be at the same endpoint.

914 **2.4.1.4.2. Option**

915 The Option(s) supported by this service instance.

916 Multiple options may be specified indicating that this service instance supports all of the listed options.

917 **2.4.1.4.3. EndpointContext**

918 While not explicitly in an ID-WSF EPR, the contents of this element show up in various locations within the IPR
919 and/or guide the generation of the contents of the EPR.

920 Multiple `<EndpointContext>` elements may appear if the same service is available via different, incompatible
921 combinations of the contents (such as a TLS and a non-TLS endpoint at different addresses).

922 The sub-elements include:

923 **2.4.1.4.3.1. Address**

924 A URI describing the address to which messages should be sent to communicate with this provider.

925 If multiple addresses are specified they are all considered equally valid addresses for this same service (such that if a
926 Discovery Service were to mint all of the possible EPRs for this case, there would be a separate EPR for each address
927 specified since an EPR can only include a single address).

928 In the case where the Discovery service has been asked to mint a subset of the possible EPRs (see [Section 3.3](#)), the
929 Discovery service is free to select any of the specified addresses using whatever local policy it chooses.

930 **2.4.1.4.3.2. Framework**

931 The SOAP Bindings ([\[LibertySOAPBinding\]](#)) `<sbef:Framework>` element describing the version of the ID-WSF
932 framework supported at this endpoint..

933 Multiple `<Framework>` elements may be specified if they can be used at each of the `<Address>` URIs within this
934 `<EndpointContext>`.

935 **2.4.1.4.3.3. SecurityMechID**

936 The Security Mechanism URI(s) (defined in [\[LibertySecMech\]](#) and its related profiles) supported by this endpoint.

937 Multiple `<SecurityMechID>` elements may be specified indicating that any of these mechanisms can be used at this
938 endpoint.

939 Note that while a particular security mechanism may need a particular form of a security token, the registering WSP
940 cannot provide such tokens. It is up to the Discovery service to mint the necessary token, or indicate to the WSC that
941 they need to obtain the token from their IdP.

942 **2.4.1.4.3.4. Action**

943 The URI indicating the supported service action at this endpoint. This is typically used when only a sub-set of the
944 entire service's operations are available at this endpoint.

945 Multiple `<Action>` elements may be specified to indicate that there are multiple operations available at this endpoint.

946 If no `<Action>` element is specified, all service operations are available at this endpoint.

947 **2.4.2. Minting ID-WSF EPRs based upon Service Metadata**

948 Service Metadata is stored in the Discovery Service in order to guide the minting of ID-WSF EPRs by the Discovery
949 Service in response to queries from WSCs.

950 One can visualize that the entire set of elements within a single Service Metadata can result in a large number of
951 possible EPRs based upon the possible combinations of those elements.

952 The Discovery Service MUST mint ID-WSF EPRs as if the following process took place (there is NOT a normative
953 requirement to implement this exact process, just a requirement that the results generated by whatever process is used
954 by the DS MUST result in the set of data that would result from this process).

955 1. Eliminate portions of the Service Metadata that do not conform to the search requirements (such as unsupported
956 (by the WSC) security mechanisms or framework versions, or undesired service types).

957 2. If an `<EndpointContext>` element had all occurrences of a given sub-element (such as `<Framework>`)
958 eliminated, eliminate the context.

959 3. For each remaining `<Address>` element within a remaining `<EndpointContext>` element, an EPR SHOULD
960 be minted.

961 4. For each EPR, assign one `<Address>` element to the `<wsa:Address>` element in the EPR and use the rest of
962 the `<EndpointContext>` that contained this address to build the necessary `<Metadata>` sub-elements for the
963 ID-WSF EPR (`<SecurityContext>`(s), `<Action>`(s), `<Framework>`(s), etc.).

964 5. Fill out the rest of the ID-WSF EPR using the service wide elements (`<Abstract>`, `<ProviderID>`,
965 `<ServiceType>`(s), etc.).

966 6. If necessary, generate any security and/or identity tokens and place them into the appropriate
967 `<SecurityContext>` element(s).

968 The set of EPRs generated by this process may be further restricted by the request parameters on the DiscoveryQuery
969 operation, see [Section 3.3](#)).

970 **2.4.3. Service Metadata Example**

971 Some examples to help show how service metadata works.

972 **2.4.3.1. A simple service**

973 This is an example of a simple service that has a single endpoint, supports a single framework version (2.0), and only
974 supports a single security mechanism.

```
975
976 <ds:svcMD svcMDID="1234">
977   <ds:Abstract>This is a simple service metadata definition</ds:abstract>
978   <ds:ProviderID>http://simpler.providers.com</ds:ProviderID>
979   <ds:ServiceContext>
980     <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
981     <ds:EndpointContext>
982       <ds:Address>https://simple.providers.com/PP</ds:Address>
983       <sb:Framework version="2.0" />
984       <ds:SecurityMechID>
985         urn:liberty:security:2003-08:TLS:Bearer
986       </ds:SecurityMechID>
987     </ds:EndpointContext>
988   </ds:ServiceContext>
989 </ds:SvcMD>
990
```

991 **Figure 14. Service Metadata example: A simple service**

992 2.4.3.2. A complex service

993 This is an example of a service metadata definition with a number of complex attributes including:

- 994 • Multiple service versions **and** multiple framework versions on the same endpoint.
- 995 • There are two service contexts, one for one version of the service and one for a different version of the service.
996 So, for example, the 2003-08 version of the service is only available at the URL *https://old.providers.com/PP* and
997 only for framework version *1.1*
- 998 • Multiple interfaces on different endpoints with different security mechanisms
- 999 • There are multiple, redundant, addresses for the TLS endpoint for the *2007-11* version of the service.

```

1000
1001 <!-- Service Metadata Example: A complex service -->
1002 <ds:SvcMD svcMDID="4567">
1003   <ds:Abstract>This is a complex service metadata definition</ds:abstract>
1004   <ds:ProviderID>http://complex.providers.com</ds:Provider ID>
1005   <ds:ServiceContext>>
1006     <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
1007     <ds:EndpointContext>
1008       <ds:Address>https://old.providers.com/PP</ds:Address>
1009       <sb:Framework version="1.1" />
1010       <ds:SecurityMechID>
1011         urn:liberty:security:2003-08:TLS:Bearer
1012       </ds:SecurityMechID>
1013     </ds:EndpointContext>
1014   </ds:ServiceContext>>
1015   <ds:ServiceContext>>
1016     <ds:ServiceType>urn:liberty:pp:2007-11</ds:ServiceType>
1017     <ds:EndpointContext>
1018       <ds:Address>https://complex.providers.com/PP</ds:Address>
1019       <ds:Address>https://backup.complex.providers.com/PP</ds:Address>
1020       <sb:Framework version="2.0" />
1021       <ds:SecurityMechID>
1022         urn:liberty:security:2003-08:TLS:Bearer
1023       </ds:SecurityMechID>
1024     </ds:EndpointContext>
1025     <ds:EndpointContext>
1026       <ds:Address>http://complex.providers.com/PP</ds:Address>
1027       <sb:Framework version="2.0" />
1028       <ds:SecurityMechID>
1029         urn:liberty:security:2003-08:null:SAMLV2
1030       </ds:SecurityMechID>
1031     </ds:EndpointContext>
1032   </ds:ServiceContext>>
1033 </ds:SvcMD>
1034

```

1035 **Figure 15. Service Metadata example: A complex service**

1036 2.4.3.3. Another complex service

1037 This is an example of a service metadata definition where the service has some of its operations at one endpoint and
 1038 others at a different endpoint (thus splitting the service operations across different instances).

1039 This service is still defined with a single service context since the endpoints all expose the same service type.

```
1040
1041 <!-- Service Metadata Example: A simple service -->
1042 <ds:SvcMD svcMDID="8901">
1043   <ds:Abstract>Another example complex service</ds:abstract>
1044   <ds:ProviderID>http://split.providers.com</ds:ProviderID>
1045   <ds:ServiceContext>>
1046     <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
1047     <ds:EndpointContext>
1048       <ds:Address>https://cluster1.split.providers.com/PP</ds:Address>
1049       <sb:Framework version="2.0" />
1050       <ds:SecurityMechID>
1051         urn:liberty:security:2003-08:TLS:Bearer
1052       </ds:SecurityMechID>
1053       <ds:Action>urn:liberty:pp:2003-08:Query</ds:Action>
1054     </ds:EndpointContext>
1055     <ds:EndpointContext>
1056       <ds:Address>https://writer.split.providers.com/PP</ds:Address>
1057       <sb:Framework version="2.0" />
1058       <ds:SecurityMechID>
1059         urn:liberty:security:2003-08:TLS:Bearer
1060       </ds:SecurityMechID>
1061       <ds:Action>urn:liberty:pp:2003-08:Modify</ds:Action>
1062     </ds:EndpointContext>
1063   </ds:ServiceContext>>
1064 </ds:SvcMD>
1065
```

1066

Figure 16. Service Metadata example: Another complex service

1067 3. Discovery Service

1068 A Discovery Service is a web service providing both identity based and non-identity based operations.

1069 The identity based Discovery Service interfaces facilitate requesters' discovery of identity service instances on a
1070 per-identity basis, and acquisition of ID-WSF Endpoint References (ID-WSF EPRs) "pointing" to the discovered
1071 service instances. These ID-WSF EPRs provide requesters with the information necessary to invoke discovered service
1072 instances.

1073 The non-identity based Discovery Service interfaces provide a WSP with principal-independent management of their
1074 metadata stored at the Discovery Service (which is, through an identity-based interface, associated with a principal).

1075 Thus in an abstract sense, the Discovery Service is essentially a web service interface to per-identity "discovery
1076 resources", each of which can be viewed as a registry of ID-WSF EPRs. The notion of "discovery resources" is an
1077 abstract way of referring to what are concretely "identity-indexed Discovery Service instances".

1078 The Discovery Service can also be used as a non-identity service to discover and obtain ID-WSF EPRs for non-identity
1079 services. For example, the Discovery Service could be used to locate the available Authentication Services before a
1080 principal identity has been established.

1081 Entities can register ID-WSF EPRs, pointing to their identity services, with a discovery resource, and this will allow
1082 other entities to discover them. A common use case is that a Principal places references (aka ID-WSF EPRs) to his
1083 or her personal profile, calendar, and so on, in a discovery resource so that they may be discovered by other entities,
1084 e.g. web service providers who wish to provide the Principal with value-added services.

1085 When invoked as an identity service, the act of discovering service instances is implicitly on a per-identity basis. This
1086 occurs in a number of fashions in ID-WSF including:

1087 • When a Principal authenticates to a service provider using a SAMLv2 profile (or similarly via
1088 ID-FF), the identity provider conveys, within the authentication assertion, an ID-WSF EPR pointing
1089 explicitly to the *Principal's* discovery service resource, which the SP may then use to discover the
1090 Principal's various services.

1091 • A Principal's (LUAD-)WSC authenticates via the Authentication Service (see [\[LibertyAuthn\]](#)),
1092 which will likely return an ID-WSF EPR for the Principal's Discovery Service resource.

1093 • Any Identity Token (see [\[LibertySecMech\]](#)), or security token may contain a Discovery Service
1094 bootstrap ID-WSF EPR (see [Section 4: Discovery Service ID-WSF EPR conveyed via a Security Token](#)
1095) which contains the necessary information to access the Principal's Discovery Service resource.

1096 The Discovery service is identified by ID-WSF EPRs, which themselves have been crafted (typically by an identity
1097 provider) such that they identify the discovery service resource (aka Discovery Service instance) mapped to the
1098 Principal in question.

1099 The Discovery Service is intended to be used in conjunction with other ID-WSF specifications. For example, security
1100 mechanisms are not specified here, because they are defined in [\[LibertySecMech\]](#). At the same time, the Discovery
1101 Service is specified such that it could be used with other security mechanisms, not yet defined.

1102 The Discovery Service is designed to be describable by WSDL [\[WSDLv1.1\]](#), and an abstract WSDL definition
1103 is included in this document, see [Appendix B: Discovery Service WSDL](#). This WSDL document defines two
1104 "WSDL operations" for the Discovery Service. The first is the *DiscoveryQuery* operation. This operation returns
1105 an enumeration of ID-WSF EPRs for a given search criteria.

1106 To enforce access control policies, security tokens may need to be presented by the client when interacting with a
1107 Discovery Service instance. While the definition of these security tokens is outside the scope of this specification, it
1108 is common for the same provider that is hosting the Discovery Service to also be the entity that generates the security

1109 tokens necessary to access the service. To avoid extra network round-trips, arrangements are made here so that
 1110 security tokens may be provided as part of the Discovery Service lookup response.

1111 3.1. Service URIs

1112 **Table 1. Discovery Service URIs**

Use	URI
Service Type	<i>urn:liberty:disco:2005-11</i>
Query wsa:Action	<i>urn:liberty:disco:2005-11:Query</i>
QueryResponse wsa:Action	<i>urn:liberty:disco:2005-11:QueryResponse</i>
SvcMDAssociationAdd wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDAssociationAdd</i>
SvcMDAssociationAddResponse wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDAssociationAddResponse</i>
SvcMDAssociationQuery wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDAssociationQuery</i>
SvcMDAssociationQueryResponse wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDAssociationQueryResponse</i>
SvcMDAssociationDelete wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDAssociationDelete</i>
SvcMDAssociationDeleteResponse wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDAssociationDeleteResponse</i>
SvcMDQuery wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDQuery</i>
SvcMDQueryResponse wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDQueryResponse</i>
SvcMDRegister wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDRegister</i>
SvcMDRegisterResponse wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDRegisterResponse</i>
SvcMDReplace wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDReplace</i>
SvcMDReplaceResponse wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDReplaceResponse</i>
SvcMDDelete wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDDelete</i>
SvcMDDeleteResponse wsa:Action	<i>urn:liberty:disco:2005-11:SvcMDDeleteResponse</i>

1113 3.2. Status Codes

1114 The following status code strings are defined:

- 1115 • *OK*: message processing succeeded
- 1116 • *Failed*: general failure code
- 1117 • *Forbidden*: the request was denied based on policy
- 1118 • *Duplicate*: the request was denied because it would result in duplicate data in the service
- 1119 • *LogicalDuplicate*: the request was denied because it would result in logically duplicate data in the service
- 1120 • *NoResults*: the query had no matching results
- 1121 • *NotFound*: the specified item(s) were not found

1122 These strings are expected to appear in the "code" attribute of <Status> elements used in SOAP-bound Discovery
1123 Service protocol messages [[LibertySOAPBinding](#)]. Specific uses for the status codes are defined in the processing
1124 rules for individual messages. The "ref" attribute on the <Status> element is not used in this specification, so it MUST
1125 NOT appear on Status elements in Discovery Service protocol messages. The contents of the comment attribute are
1126 not defined by this specification, but it may be used for additional descriptive text intended for human consumption
1127 (for example, to carry information that will aid debugging).

1128 **3.3. Operation: *DiscoveryQuery***

1129 The *DiscoveryQuery* WSDL operation enables a requester to obtain an enumeration of ID-WSF EPRs (see [Section 2:](#)
1130 [Discovery Service Information Model](#)) — the requester sends a <Query> message and receives a <QueryResponse>
1131 message in return. Also, because a provider hosting a Discovery Service may also be playing other roles on behalf
1132 of Principals (such as a *Policy Decision Point* or an *Authentication Authority*), the *DiscoveryQuery* operation can also
1133 function as a security token service, providing the requester with an efficient means of obtaining security tokens that
1134 may be necessary to invoke service instances described in the <QueryResponse>.

1135 **3.3.1. wsa:Action values for DiscoveryQuery Messages**

1136 <Query> messages MUST include a <wsa:Action> SOAP header with the value of urn:liberty:disco:2005-11:Query.

1137 <QueryResponse> messages MUST include a <wsa:Action> SOAP header with the value of
1138 urn:liberty:disco:2005-11:QueryResponse.

1139 **3.3.2. <Query> Message**

1140 A <Query> request is an attempt to retrieve ID-WSF EPRs suitable for use in the same identity context that was used
1141 to make the request. In particular, the Target Identity (See [[LibertySOAPBinding](#)]), if applicable, is used to restrict the
1142 results to just those for the specified principal. The Invocation Identity will be verified against an access control list to
1143 ensure that they have access to the requested results.

1144 A <Query> request message is empty in the minimal case. Such a request indicates the requester is requesting all
1145 available ID-WSF EPRs, regardless of security mechanisms or service types. The result set is dependant upon the
1146 local access control policies of the discovery service instance.

1147 Alternatively, a request can be qualified with a set of <RequestedService> elements, which enables the requester to
1148 specify that all ID-WSF EPRs returned must be offered via one or more service instances complying with the specified
1149 search criteria. For each <RequestedService> specified, the requester specifies the search criteria for the DS to use
1150 in determining if there is a matching instance. The search criteria includes zero or more of any of the following:

1151 • <ServiceType> the requested type of service. Service Type URIs are defined by the individual service
1152 specifications and contain both service class and service versioning information.

1153 Multiple <ServiceType>s MAY be specified in a single <RequestedService>s element in order to allow the
1154 WSC to specify what the WSC considers to be different versions of the same service.

1155 When multiple entries are listed, the order of such entries is an indication of the preference as to which
1156 <ServiceType> the WSC would prefer to see in the results, with the first being the most preferred. This
1157 typically only impacts a request where the WSC indicates that they only want a subset of the results returned (see
1158 [resultsType](#) below).

1159 When a request results in multiple ID-WSF EPRs in a response, the preference order specified by the WSC on the
1160 request MAY have no impact on the order of results returned by the Discovery Services. The Discovery Service
1161 is free to return the results of the request in whatever order it chooses.

1162 If not specified, then any service instance would be considered a match for this criteria.

1163 A service instance must support at least one of the specified service types in order to be considered a match for
1164 this criteria.

- 1165 • `<ProviderID>` the requested provider ID(s). This is used when the WSC wants to communicate with a particular
1166 WSP. Frequently such requests are made without specifying a `<ServiceType>` element in the request, but doing
1167 so is not prohibited.
- 1168 If not specified, then any service instance would be considered a match for this criteria.
- 1169 A service instance must contain at least one of the specified providerIDs in order to be considered a match for this
1170 criteria.
- 1171 The order of the `<ProviderID>` elements is an indication as to the preference of the requester with the first such
1172 element being the most desired (declining preference order). The Discovery Services is free to return the results
1173 of the request in whatever order it chooses.
- 1174 • `<Options>` — an optional multi-occurrence element defining options SETs desired for the service.
- 1175 If not specified, any service instance will be considered a match for this criteria.
- 1176 An option SET is defined within each `<Options>` element and contains a list of the desired options. The service
1177 instance MUST support ALL of the options within the option SET in order to be considered a match for this
1178 request.
- 1179 If more than one `<Options>` element is specified (thus defining multiple option SETs), service instances that
1180 match ANY of the SETS are considered a match for this request. As noted above, to match a SET, you have to
1181 match ALL of the entries within the SET.
- 1182 Service instance EPRs registered without an `<Options>` element are always considered a match from the point of
1183 view of any possible `<Options>` search criteria.
- 1184 The order of the `<options>` elements is an indication as to the preference of the requester with the first such
1185 element being the most desired (declining preference order). The Discovery Services is free to return the results
1186 of the request in whatever order it chooses.
- 1187 • `<SecurityMechID>` - an optional multi-occurrence element specifying the security mechanism identifier(s) (see
1188 [[LibertySecMech](#)]) that the WSC is willing to use to invoke the WSP. If not specified, any security mechanism
1189 registered for a service will be considered a match for this criteria.
- 1190 A service instance MUST support at least one of the requested security mechanisms in order to be considered a
1191 match for this request.
- 1192 The order of the `<SecurityMechID>` elements is an indication as to the preference of the requester with the first
1193 such element being the most desired (declining preference order). The Discovery Services is free to return the
1194 results of the request in whatever order it chooses.
- 1195 • `<Framework>` — an optional multi-occurrence element specifying the framework description(s) supported by the
1196 WSC.
- 1197 If not specified, the Discovery Service SHOULD use the value of the framework description used in the ID-WSF
1198 framework layer for the current request (e.g. if the call to the Discovery Service was made using an ID-WSF
1199 version 2.0 message the request SHOULD be treated as if a `<disco:Framework>` element was present and
1200 contained the value specified in the `<sbf:Framework>` SOAP header.
- 1201 Multiple `<disco:Framework>` elements MAY be specified, indicating that the WSC has the capability to support
1202 ANY of the specified versions. The order of elements in such a case indicates the WSC's preference with the most
1203 preferred coming first.
- 1204 Note that while both the `<disco:Framework>` and the `<sbf:Framework>` elements are of the same type
1205 (`<sbf:FrameworkType>`), the elements themselves are in different namespaces. The element within the
1206 `<RequestedService>` is in the Discover Service Namespace, while the element within any ID-WSF EPRs and
1207 the SOAP header block on an ID-WSF message are in the SOAP Bindings namespace.

- 1208 • `<Action>` — an optional multi-occurrence element specifying the `wsa:Action` value(s) for the interfaces of the
1209 service that the WSC intends to make use of.
- 1210 If not specified, the Discovery Service SHOULD treat this request as a request for all of the interfaces at the
1211 requested specified instance.
- 1212 Unlike the other sub-elements of the `<RequestedService>` element, if multiple `<Action>` elements are
1213 specified it indicates that the WSC intends to invoke **all** of the specified interfaces and the Discovery Service
1214 SHOULD return the set of EPRs that are necessary to reach the complete set of specified interfaces.
- 1215 Services registered without an `<Action>` element (which is the norm) are treated as exposing **all** of the interfaces
1216 defined for that type of service.
- 1217 The Discovery Service will return the set of EPRs for service instances that intersect with the search criteria specified
1218 in the `<RequestedService>` element. This may result in a single EPR in the response or it may result in a multitude
1219 of EPRs, depending upon the search criteria and the service instance definitions available (see [Section 2.3.3: ID-WSF](#)
1220 [Web Services Addressing EPR Profile](#)).
- 1221 The result set of EPRs generated in response to a particular `<RequestedService>` element can be further controlled
1222 using the following attributes:
- 1223 • `reqID` — an optional attribute identifying this `<RequestedService>` request. Typically only used when
1224 multiple `<RequestedService>` elements are included in a single Discovery Service `<Query>`.
- 1225 If present the value of this attribute will be placed into the `reqRef` attribute in any EPRs that result from this
1226 `<RequestedService>` element (see [Section 2.3.1.2](#) above).
- 1227 The value of `reqID` SHOULD be different for all `<RequestedService>` elements in a given `<Query>`.
- 1228 • `resultsType` — an optional attribute describing the results desired by the requestor. This value may be set to:
- 1229 • **best** — the Discovery Service SHOULD return what **it** considers the best match (under local policy) for the
1230 given search criteria.
- 1231 • **all** — the Discovery Service SHOULD return all of the matching entries for the given search criteria.
1232 This would typically be used when the client wants to choose which EPRs within the DS database it should
1233 use.
1234 This option should be used with caution as it can cause the DS to perform substantial work in order to mint all
1235 of the matching EPRs and the necessary security tokens for those EPRs.
- 1236 • **only-one** — The Discovery Service SHOULD return what it considers the one (1) EPR that **it** considers (under
1237 local policy) to be the best match for the search criteria. A client would typically specify this option if it was
1238 going to ignore anything other than the first entry.
- 1239 If `resultsType` is not specified the Discovery Service may make its own determination (under local policy) as to
1240 which set of results to return.

1241 Requestors SHOULD include at least one <ServiceType> or <ProviderID> element, and MAY include any number
1242 of both of them.

1243 Requesters SHOULD construct a Query to be as qualified as possible, as the Discovery Service instance may have to
1244 perform significant work for each item in the result set, especially if security tokens will be generated.

```

1245
1246 <!-- Query Message Element & Type -->
1247
1248 <xs:element name="Query" type="QueryType"/>
1249
1250 <xs:complexType name="QueryType">
1251   <xs:sequence>
1252     <xs:element name="RequestedService"
1253       type="RequestedServiceType"
1254       minOccurs="0"
1255       maxOccurs="unbounded"/>
1256   </xs:sequence>
1257
1258   <xs:attribute ref="wsu:Id" use="optional"/>
1259 </xs:complexType>
1260
1261 <xs:complexType name="RequestedServiceType">
1262   <xs:sequence>
1263     <xs:element ref="ServiceType" minOccurs="0" maxOccurs="unbounded" />
1264
1265     <xs:element ref="ProviderID" minOccurs="0" maxOccurs="unbounded" />
1266
1267     <xs:element ref="Options" minOccurs="0" maxOccurs="unbounded"/>
1268
1269     <xs:element ref="SecurityMechID" minOccurs="0" maxOccurs="unbounded"/>
1270
1271     <xs:element ref="Framework" minOccurs="0" maxOccurs="unbounded"/>
1272
1273     <xs:element ref="Action" minOccurs="0" maxOccurs="unbounded" />
1274
1275     <xs:any namespace="##other"
1276       processContents="lax"
1277       minOccurs="0"
1278       maxOccurs="unbounded" />
1279   </xs:sequence>
1280
1281   <xs:attribute name="reqID" type="xs:string" use="optional" />
1282   <xs:attribute name="resultsType" type="xs:string" use="optional" />
1283 </xs:complexType>
1284
1285
1286
1287

```

1288 **Figure 17. Query Message — Schema Fragment**

```

1289
1290 <soap:Envelope
1291   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
1292
1293   <soap:Header>
1294     ...
1295   </soap:Header>
1296
1297   <soap:Body>
1298     <Query xmlns="&DS2Namespace;">
1299       <RequestedService>
1300         <ServiceType>urn:liberty:id-sis-pp:2003-08</ds:ServiceType>
1301
1302         <SecurityMechID>urn:liberty:security:2004-12:ClientTLS:SAMLV2</SecurityMechID>
1303         <SecurityMechID>urn:liberty:security:2005-02:ClientTLS:SAML</SecurityMechID>
1304         <SecurityMechID>urn:liberty:security:2004-12:TLS:SAMLV2</SecurityMechID>
1305         <Framework version="2.0" />
1306
1307       </RequestedService>
1308     </Query>
1309   </soap:Body>
1310 </soap:Envelope>
1311

```

1312 **Example 5. SOAP message containing a Query**

1313 3.3.3. QueryResponse

1314 A <QueryResponse> message conveys the results of the query as a set of ID-WSF EPRs, i.e. profiled
1315 <wsa:EndpointReference> elements (see [Section 2.3.3: ID-WSF Web Services Addressing EPR Profile](#)).

1316 As specified in [Section 2.3.3](#), security tokens, appropriate for subsequent invocation(s) of the service instances
1317 represented by the returned ID-WSF EPRs, MAY be provided within the ID-WSF EPRs in the response.

1318 A status code is also included in the response.

```

1319
1320 <!-- QueryResponse Message Element & Type -->
1321
1322 <xs:element name="QueryResponse" type="QueryResponseType"/>
1323
1324 <xs:complexType name="QueryResponseType">
1325   <xs:sequence>
1326     <xs:element ref="lu:Status"/>
1327
1328     <xs:element ref="wsa:EndpointReference"
1329       minOccurs="0"
1330       maxOccurs="unbounded"/>
1331   </xs:sequence>
1332
1333   <xs:attribute name="id"
1334     type="xs:ID"
1335     use="optional"/>
1336 </xs:complexType>
1337
1338

```

1339 **Figure 18. <QueryResponse> — Schema Fragment**

1340 An example SOAP message containing a <QueryResponse> message is illustrated in [Example 6](#). This example
1341 includes a security token embedded in the returned ID-WSF EPR. Parts of the security token have been omitted due
1342 to size.

```

1343
1344 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
1345
1346   <soap:Header>
1347     ...
1348   </soap:Header>
1349
1350   <soap:Body>
1351     <QueryResponse xmlns="&DS2Namespace;">
1352       <Status code="OK"/>
1353
1354       <wsa:EndpointReference
1355         notOnOrAfter="2005-08-15T23:18:56Z" >
1356         <wsa:Address>http://example.com/pip/bob</wsa:Address>
1357
1358         <wsa:Metadata>
1359           <ds:Abstract>
1360             Bob's personal profile
1361           </ds:Abstract>
1362
1363           <ds:ProviderID>http://example.com/</ds:ProviderID>
1364
1365           <ds:ServiceType>urn:liberty:id-sis-pp:2003-08</ds:ServiceType>
1366
1367           <ds:FrameworkVersion="2.0" />
1368
1369           <ds:SecurityContext>
1370             <ds:SecurityMechID>urn:liberty:security:2004-12:ClientTLS:SAMLV2</ds:SecurityMechID>
1371             <ds:SecurityMechID>urn:liberty:security:2005-02:ClientTLS:SAML</ds:SecurityMechID>
1372
1373             <sec:Token usage="urn:liberty:security:tokenusage:2005-11:SecurityToken" >
1374               <saml2:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
1375                 ID="sxJu9g/vvLG9sAN9bKp/8q0NKU="
1376                 Issuer="idp.example.com"
1377                 IssueInstant="2003-09-09T16:58:33.173Z">
1378                 <saml2:Subject>
1379                   <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
1380                     http://serviceprovider.com/
1381                   </saml2:NameID>
1382
1383                   <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
1384                     <saml2:SubjectConfirmationData xsi:type="saml:KeyInfoConfirmationDataType">
1385                       <ds:KeyInfo>
1386                         <ds:KeyName>
1387                           CN=serviceprovider.com,
1388                           OU=Services R US,O=Service Nation,...
1389                         </ds:KeyName>
1390                       </ds:KeyInfo>
1391                     </saml2:SubjectConfirmationData>
1392                   </saml2:SubjectConfirmation>
1393                 </saml2:Subject>
1394
1395                 <saml2:AuthnStatement AuthnInstant="2003-09-09T16:57:30.000Z"
1396                   SessionIndex="..."
1397                   SessionNotOnOrAfter="..."
1398                 >
1399                   <saml2:AuthnContext
1400                     ...
1401                   </saml2:AuthnContext>
1402                 </saml2:AuthnStatement>
1403
1404                 <ds:Signature>...</ds:Signature>
1405               </saml2:Assertion>
1406             </sec:Token>
1407           </ds:SecurityContext>
1408         </wsa:Metadata>
1409       </wsa:EndpointReference>

```

```
1410     </QueryResponse>
1411 </soap:Body>
1412 </soap:Envelope>
1413
```

1414 **Example 6. SOAP-bound <QueryResponse> Message with Embedded Security Token**

1415 3.3.4. DiscoveryQuery Processing Rules

1416 The discovery Service returns entries based on the requester's search criteria (interpreted as described above in
1417 [Section 3.3.2: <Query> Message](#)), the policies of the discovery resource, and the contents of the discovery resource.

1418 For each <RequestedService> element in a <Query> message, the matching rules MUST applied independently
1419 (as if the other <RequestedService> elements were not present (potentially returning equivalent EPRs in response
1420 to different <RequestedService> elements).

1421 The Discovery Service SHOULD, when possible, provide the security tokens necessary for the security mechanism(s)
1422 identified in the ID-WSF EPRs in the response. If the Discovery Service is not able to generate the necessary security
1423 token, it should indicate so by including an empty <sec:Token> element with the `ref` attribute set to the value:

```
1424     urn:liberty:disco:tokenref:ObtainFromIDP
```

1425 The Discovery Service SHOULD mint new EPRs such that they carry the same identity context that was used to invoke
1426 the Discovery Service in the invocation context for the targeted WSP.

1427 The Discovery Service MAY order <wsa:EndpointReference> elements as it sees fit. If the Discovery Service is
1428 rank ordering the entries, it MUST use descending rank order. This enables the requester to assume that if the results
1429 were ordered, the first result is the most relevant.

1430 The following rules specify the status code in the response:

1431 • If request processing succeeded, the top-level status code MUST be *OK*. Otherwise, the top-level status code
1432 MUST be *Failed*.

1433 • If the top-level status code is *Failed*, the response MAY also contain *Forbidden* or *NoResults* as a second-level
1434 status code.

1435 The service may not wish to reveal the reason for failure, in which case no second-level status code will appear.

1436 **3.4. Operation: *MDAssociationAdd***

1437 The *MDAssociationAdd* operation is used by the WSP to add an association of the principal to the specified metadata.

1438 **3.4.1. *wsa:Action* values for *MDAssociationAdd* Messages**

1439 <SvcMDAssociationAdd> messages MUST include a <wsa:Action> SOAP header with the value of
1440 "urn:liberty:disco:2005-11:SvcMDAssociationAdd".

1441 <SvcMDAssociationAddResponse> messages MUST include a <wsa:Action> SOAP header with the value of
1442 "urn:liberty:disco:2005-11:SvcMDAssociationAddResponse".

1443 **3.4.2. *SvcMDAssociationAdd* Message**

1444 The <SvcMDAssociationAdd> is called with one or more <SvcMDID> elements to add associations to these service
1445 metadata descriptions for the principal.

1446 A WSP SHOULD NOT associate the same <SvcMD> (or different SvcMD element that carry metadata for the "same"
1447 service) to a principal multiple times without first removing the previous entry.

1448 The values in the <SvcMDID> element(s) must have been obtained via one of the service metadata operations discussed
1449 later in this specification.

```
1450
1451 <!-- SvcMDAssociationAdd operation -->
1452
1453 <xs:element name="SvcMDAssociationAdd" type="SvcMDAssociationAddType" />
1454
1455 <xs:complexType name="SvcMDAssociationAddType">
1456   <xs:sequence>
1457     <xs:element ref="SvcMDID" maxOccurs="unbounded" />
1458   </xs:sequence>
1459 </xs:complexType>
1460
```

1461 **Figure 19. <SvcMDAssociationAdd> — Schema Fragment**

1462 An example message body containing a <SvcMDAssociationAdd> message follows. This request adds a new
1463 association for the current principal (note that the identity of the principal is carried in the invocation context and not
1464 in the body of the message).

```
1465
1466 <ds:SvcMDAssociationAdd>
1467   <ds:SvcMDID>2323872</ds:SvcMDID>
1468 </ds:SvcMDAssociationAdd>
1469
```

1470 **Example 7. <SvcMDAssociationAdd> Message**

1471 **3.4.3. *SvcMDAssociationAddResponse* Message**

1472 This response to the <SvcMDAssociationAdd> request contains the following elements and attributes.

1473 • <lu:Status>: Contains status code; see processing rules.

```

1474
1475 <!-- Response for SvcMDAssociationAdd operation -->
1476
1477 <xs:element name="SvcMDAssociationAddResponse"
1478           type="SvcMDAssociationAddResponseType"/>
1479
1480 <xs:complexType name="SvcMDAssociationAddResponseType">
1481   <xs:sequence>
1482     <xs:element ref="lu:Status" />
1483   </xs:sequence>
1484 </xs:complexType>
1485

```

1486 **Figure 20. <SvcMDAssociationAddResponse> — Schema Fragment**

```

1487
1488 <ds:SvcMDAssociationAddResponse>
1489   <lu:Status code="OK" />
1490 </ds:SvcMDAssociationAddResponse>
1491

```

1492 **Example 8. <SvcMDAssociationAddResponse> Message**

1493 3.4.4. MDAssociation Add Processing Rules

- 1494 • Once the association is added by the WSP, the Discovery Service MUST consider this metadata (subject to local
1495 policy) when responding to subsequent DiscoveryQuery operations and should the associated metadata meet the
1496 requirements of the query, mint the necessary ID-WSF EPRs based upon the requirements of the WSC and the
1497 WSP.
 - 1498 • The Discovery Service SHOULD reject attempts to associate a <SvcMDID> that has already been associated with
1499 the principal by this WSP. In such cases the Discovery service MAY set the second level status code in the response
1500 to *Duplicate*.
 - 1501 • The Discovery Service MAY similarly reject attempts to associate a <SvcMDID> that references the same service
1502 type and WSP that is in one of the already associated service metadata descriptions. In such cases the Discovery
1503 service MAY set the second level status code in the response to *LogicalDuplicate*.
 - 1504 • The Discovery Service MUST reject attempts to associate a <SvcMDID> that does not exist or is not owned by the
1505 WSP invoking the call. In such cases the Discovery service MAY set the second level status code in the response
1506 to *NotFound*.
 - 1507 • If request processing succeeded, the top-level status code MUST be *OK*. Otherwise, the top-level status code
1508 MUST be *Failed*.
 - 1509 • If the top-level status code is *Failed*, the response MAY also contain *Forbidden*, *Duplicate*, *LogicalDuplicate*, or
1510 *NotFound* as a second-level status code. The Discovery Service instance may not wish to reveal the reason for
1511 failure, in which case no second-level status code will appear.
 - 1512 • A Discovery Service MAY provide some programmatic or browser based interface which allows the principal to
1513 manage the service associations that have been added to their resource at the Discovery Service. A principal may
1514 be able to use such interfaces to change or even remove service associations made by the WSP without the WSP's
1515 permission (it is the principal's resource) and perhaps, even without notification to the WSP.
- 1516 Such interfaces are out-of-scope for this specification, but are mentioned here to remind the WSP that they may
1517 exist.

1518 **3.5. Operation: *MDAssociationQuery***

1519 The *MDAssociationQuery* operation is used by the WSP to query the Discovery Service for any previously added
1520 associations related to the principal.

1521 **3.5.1. *wsa:Action* values for *MDAssociationQuery* Messages**

1522 <SvcMDAssociationQuery> messages MUST include a <wsa:Action> SOAP header with the value of
1523 "urn:liberty:disco:2005-11:SvcMDAssociationQuery".

1524 <SvcMDAssociationQueryResponse> messages MUST include a <wsa:Action> SOAP header with the value of
1525 "urn:liberty:disco:2005-11:SvcMDAssociationQueryResponse".

1526 **3.5.2. *SvcMDAssociationQuery* Message**

1527 The <SvcMDAssociationQuery> is called with zero or more <SvcMDID> elements to query associations to these
1528 service metadata descriptions. If no <SvcMDID> elements are specified, ALL associations between the WSP's service
1529 metadata and the principal are returned.

```
1530  
1531 <!-- SvcMDAssociationQuery operation -->  
1532  
1533 <xs:element name="SvcMDAssociationQuery" type="SvcMDAssociationQueryType" />  
1534  
1535 <xs:complexType name="SvcMDAssociationQueryType" >  
1536   <xs:sequence>  
1537     <xs:element ref="SvcMDID" minOccurs="0" maxOccurs="unbounded" />  
1538   </xs:sequence>  
1539 </xs:complexType>  
1540
```

1541 **Figure 21. <SvcMDAssociationQuery> — Schema Fragment**

1542 An example message body containing a <SvcMDAssociationQuery> message follows. This request asks for all
1543 associations.

```
1544  
1545 <ds:SvcMDAssociationQuery />  
1546
```

1547 **Example 9. <SvcMDAssociationQuery> Message**

1548 **3.5.3. *SvcMDAssociationQueryResponse* Message**

1549 This response to the <SvcMDAssociationQuery> request contains the following elements and attributes.

- 1550
- <lu:Status>: Contains status code; see processing rules.
- 1551
- <SvcMDID>: the associated service metadata ID(s). If <SvcMDID>s were specified on the
1552 <SvcMDAssociationQuery> the response will be limited to at most those IDs (if they have been associ-
1553 ated with the principal).


```
1554
1555 <!-- Response for SvcMDAssociationQuery operation -->
1556
1557 <xs:element name="SvcMDAssociationQueryResponse"
1558           type="SvcMDAssociationQueryResponseType" />
1559
1560 <xs:complexType name="SvcMDAssociationQueryResponseType">
1561   <xs:sequence>
1562     <xs:element ref="lu:Status" />
1563     <xs:element ref="SvcMDID" minOccurs="0" maxOccurs="unbounded" />
1564   </xs:sequence>
1565 </xs:complexType>
1566
```

1567 **Figure 22. <SvcMDAssociationQueryResponse> — Schema Fragment**

```
1568
1569 <ds:SvcMDAssociationQueryResponse>
1570   <lu:Status code="OK" />
1571   <ds:SvcMDID>2323872</ds:SvcMDID>
1572 </ds:SvcMDAssociationQueryResponse>
1573
```

1574 **Example 10. <SvcMDAssociationQueryResponse> Message**

1575 3.5.4. MDAssociation Query Processing Rules

- 1576 • The Discovery Service MUST limit the operation to only those associations added by the WSP to the current
1577 principal's resource (a WSP MUST NOT be able to query associations added at the same Discovery Service by
1578 other WSPs or associations added to a different principal). There MUST NOT be any indication on the response
1579 as to whether or not other such elements exist.
- 1580 • If request processing succeeded, the top-level status code MUST be *OK*. Otherwise, the top-level status code
1581 MUST be *Failed*.
- 1582 • If the top-level status code is *Failed*, the response MAY also contain *Forbidden* or *NotFound* as a second-level
1583 status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no
1584 second-level status code will appear.

1585 3.6. Operation: *MDAssociationDelete*

1586 The *MDAssociationDelete* operation is used by the WSP to delete a previously added association of the principal to
1587 the specified metadata.

1588 3.6.1. *wsa:Action* values for *MDAssociationDelete* Messages

1589 <SvcMDAssociationDelete> messages MUST include a <wsa:Action> SOAP header with the value of
1590 "urn:liberty:disco:2005-11:SvcMDAssociationDelete".

1591 <SvcMDAssociationDeleteResponse> messages MUST include a <wsa:Action> SOAP header with the value
1592 of "urn:liberty:disco:2005-11:SvcMDAssociationDeleteResponse".

1593 3.6.2. *SvcMDAssociationDelete* Message

1594 The <SvcMDAssociationDelete> is called with one or more <SvcMDID> elements to delete associations to these
1595 service metadata descriptions.

1596 Note that the service metadata description is not impacted by this call. Only the principal's association with the
 1597 metadata is impacted.

```

1598
1599 <!-- SvcMDAssociationDelete operation -->
1600
1601 <xs:element name="SvcMDAssociationDelete" type="SvcMDAssociationDeleteType"/>
1602
1603 <xs:complexType name="SvcMDAssociationDeleteType">
1604   <xs:sequence>
1605     <xs:element ref="SvcMDID" maxOccurs="unbounded" />
1606   </xs:sequence>
1607 </xs:complexType>
1608
  
```

1609 **Figure 23. <SvcMDAssociationDelete> — Schema Fragment**

1610 An example message body containing a <SvcMDAssociationDelete> message follows. This request deletes a
 1611 single association for the current principal (note that the identity of the principal is carried in the invocation context
 1612 and not in the body of the message).

```

1613
1614 <ds:SvcMDAssociationDelete>
1615   <ds:SvcMDID>2323872</ds:SvcMDID>
1616 </ds:SvcMDAssociationDelete>
1617
  
```

1618 **Example 11. <SvcMDAssociationDelete> Message**

1619 3.6.3. SvcMDAssociationDeleteResponse Message

1620 This response to the <SvcMDAssociationDelete> request contains the following elements and attributes.

- 1621 • <lu:Status>: Contains status code; see processing rules.

```

1622
1623 <!-- Response for SvcMDAssociationDelete operation -->
1624
1625 <xs:element name="SvcMDAssociationDeleteResponse"
1626   type="SvcMDAssociationDeleteResponseType"/>
1627
1628 <xs:complexType name="SvcMDAssociationDeleteResponseType">
1629   <xs:sequence>
1630     <xs:element ref="lu:Status" />
1631   </xs:sequence>
1632 </xs:complexType>
1633
  
```

1634 **Figure 24. <SvcMDAssociationDeleteResponse> — Schema Fragment**

```

1635
1636 <ds:SvcMDAssociationDeleteResponse>
1637   <lu:Status code="OK" />
1638 </ds:SvcMDAssociationDeleteResponse>
1639
  
```

1640 **Example 12. <SvcMDAssociationDeleteResponse> Message**

1641 3.6.4. MDAssociation Delete Processing Rules

- 1642 • Once deleted, the association **MUST NOT** be subsequently used by the DS to mint ID-WSF EPRs in response to
1643 queries relative to this principal. However, WSPs should be prepared to receive requests from WSCs from clients
1644 who previously obtained ID-WSF EPRs minted from the associaton which haven't expired.
- 1645 • The Discovery Service **MUST** limit the operation to only those associations added by the WSP to the current
1646 principal's resource (a WSP **MUST NOT** be able to delete associations added at the same Discovery Service by
1647 other WSPs or associations added to a different principal). There **MUST NOT** be any indication on the response
1648 as to whether or not other such elements exist.
- 1649 • If request processing succeeded, the top-level status code **MUST** be *OK*. Otherwise, the top-level status code
1650 **MUST** be *Failed*.
- 1651 • If the top-level status code is *Failed*, the response **MAY** also contain *Forbidden* or *NotFound* as a second-level
1652 status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no
1653 second-level status code will appear.

1654 **3.7. Operation: *MetadataRegister***

1655 The *MetadataRegister* operation is used to register a new service metadata description with the Discovery Service.

1656 **3.7.1. wsa:Action values for *MetadataRegister* Messages**

1657 <SvcMDRegister> messages **MUST** include a <wsa:Action> SOAP header with the value of
1658 "urn:liberty:disco:2005-11:SvcMDRegister".

1659 <SvcMDRegisterResponse> messages **MUST** include a <wsa:Action> SOAP header with the value of
1660 "urn:liberty:disco:2005-11:SvcMDRegisterResponse".

1661 **3.7.2. *SvcMDRegister* Message**

1662 The <SvcMDRegister> is called with one or more service metadata descriptions to be registered at the Discovery
1663 Service on behalf of the WSP.

```

1664 <!-- Register operation for Service Metadata -->
1665
1666 <xs:element name="SvcMDRegister" type="SvcMDRegisterType"/>
1667
1668 <xs:complexType name="SvcMDRegisterType">
1669   <xs:sequence>
1670     <xs:element ref="SvcMD" maxOccurs="unbounded" />
1671   </xs:sequence>
1672 </xs:complexType>
1673
1674
1675
```

1676 **Figure 25. <SvcMDRegister> — Schema Fragment**

1677 An example message body containing a <SvcMDRegister> message follows. This request registers a new service
1678 metadata description. Note that the WSP has not set the *svcMDID* attribute on the <SvcMD> element – this will be
1679 assigned by the DS and returned in the response to the WSP.

```

1680
1681 <ds:SvcMDRegister>
1682   <ds:SvcMD>
1683     <ds:Abstract>Profile Service</ds:abstract>
1684     <ds:ProviderID>http://profile.com</ds:ProviderID>
1685     <ds:ServiceContext>
1686       <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
1687       <ds:EndpointContext>
1688         <ds:Address>https://profile.com/</ds:Address>
1689         <sb:Framework version="2.0" />
1690         <ds:SecurityMechID>
1691           urn:liberty:security:2003-08:TLS:Bearer
1692         </ds:SecurityMechID>
1693       </ds:EndpointContext>
1694     </ds:ServiceContext>
1695   </ds:SvcMD>
1696 </ds:SvcMDRegister>
1697

```

1698 **Example 13. <SvcMDRegister> Message**

1699 3.7.3. SvcMDRegisterResponse Message

1700 This response to the <SvcMDRegister> request contains the following elements and attributes.

- 1701 • <lu:Status>: Contains status code; see processing rules.
- 1702 • One or more <SvcMDID> if the call was successful (status code is OK). One SvcMDID is returned for each service
1703 metadata element registered.
- 1704 • <Keys>: Contains the key descriptors for the keys used by the Discovery Service to sign security tokens (see
1705 [Section 3.12](#) for a description of when and why this may be necessary).

```

1706
1707 <!-- Response for SvcMDRegister operation -->
1708
1709 <xs:element name="SvcMDRegisterResponse"
1710   type="SvcMDRegisterResponseType"/>
1711
1712 <xs:complexType name="SvcMDRegisterResponseType">
1713   <xs:sequence>
1714
1715     <xs:element ref="lu:Status" />
1716     <xs:element ref="SvcMDID" minOccurs="0" maxOccurs="unbounded" />
1717     <xs:element ref="Keys" minOccurs="0" maxOccurs="unbounded" />
1718
1719   </xs:sequence>
1720 </xs:complexType>
1721
1722

```

1723 **Figure 26. <SvcMDRegisterResponse> — Schema Fragment**

```

1724
1725 <ds:SvcMDRegisterResp>
1726   <lu:Status code="OK" />
1727   <ds:SvcMDID>2323872</ds:SvcMDID>
1728 </ds:SvcMDRegister>
1729

```

1730 **Example 14. <SvcMDRegisterResponse> Message**

1731 **3.7.4. Metadata Register Processing Rules**

- 1732 • This operation **MUST** be processed in the context of the WSP, (as opposed to the context of the principal) so that
1733 the WSP can maintain a single set of service metadata across all principals at the same Discovery Service.
- 1734 Even if this operation is invoked with an invocation identity of a principal, the Discovery Service **MUST** use
1735 the Sender's identity (the WSP) when processing this call. The Discovery Service **MAY** refuse to process the
1736 operation if the identity of the Sender cannot be established to the Discovery Service's satisfaction.
- 1737 • The transaction unit for this operation is the entire set of <SvcMD> elements; they either all succeed or all fail. The
1738 Discovery Service **MUST** enforce this atomicity.
- 1739 • For each <SvcMD> element, the Discovery Service instance **MAY** store the metadata provided such that it can be
1740 used (subject to policy) to mint ID-WSF EPRs in response to future *DiscoveryQuery* operations should that service
1741 metadata be associated with a principal's resource at the Discovery Service.
- 1742 If the Discovery Service instance does not store the metadata, it **MUST** return a *Failed* status code for the operation,
1743 and therefore not register any of the other entries provided.
- 1744 If the Discovery Service does store the metadata, it **MUST** assign a permanent identifier for the metadata usable
1745 by the WSP to subsequently reference the metadata. This identifier **MUST** be unique across all metadata objects
1746 stored by a WSP and **MAY** be unique across all metadata objects stored by all WSPs at that Discovery Service.
1747 This identifier is provided to the WSP in the response and can be subsequently used by the WSP to associate this
1748 metadata with a principal or to manage the metadata using one of the other metadata operations.
- 1749 • A WSP **MAY** register multiple service metadata descriptions that for all intents and purposes, appear to be fully
1750 equal. The Discovery Service **MUST NOT** generate an error solely because it thinks the descriptions are equal.
1751 The Discovery Service **MUST** treat these records as independent registrations and assign the associated unique
1752 SvcMDID values.
- 1753 • The Discovery Service **MAY** have some policy driven limit on the number of service metadata descriptions that it
1754 will allow a WSP to register. If a WSP attempts to register a new service metadata description that would exceed
1755 such a limit, the DS **SHOULD** include a secondary-level status code of *LimitExceeded*.
- 1756 A WSP should exercise care to only register new service metadata descriptions when an existing, registered,
1757 description that meets the WSP's needs is not available.
- 1758 • If request processing succeeded, the top-level status code **MUST** be *OK*. Otherwise, the top-level status code
1759 **MUST** be *Failed*.
- 1760 • If the top-level status code is *Failed*, the response **MAY** also contain *Forbidden* or *OverLimit* as a second-level
1761 status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no
1762 second-level status code will appear.

1763 **3.8. Operation: *MetadataQuery***

1764 The *MetadataQuery* operation is used to query the Discovery Service for existing, registered, service metadata
1765 descriptions.

1766 **3.8.1. wsa:Action values for *MetadataQuery* Messages**

- 1767 <SvcMDQuery> messages **MUST** set the value of the <wsa:Action> header to "urn:liberty:disco:2005-11:SvcMDQuery".
- 1768 <SvcMDQueryResponse> messages **MUST** include a <wsa:Action> SOAP header with the value of
1769 "urn:liberty:disco:2005-11:SvcMDQueryResponse".

1770 **3.8.2. SvcMDQuery Message**

1771 The <SvcMDQuery> is called with zero or more <SvcMDID> elements to retrieve the specified list of service metadata
1772 descriptions. If no <SvcMDID>s are specified, ALL of the metadata stored at the Discovery service by the invoking
1773 WSP will be returned.

```
1774  
1775 <!-- Query operation on Service Metadata -->  
1776  
1777 <xs:element name="SvcMDQuery" type="SvcMDQueryType"/>  
1778  
1779 <xs:complexType name="SvcMDQueryType">  
1780 <xs:sequence>  
1781 <xs:element ref="SvcMDID"  
1782 minOccurs="0"  
1783 maxOccurs="unbounded"/>  
1784 </xs:sequence>  
1785 </xs:complexType>  
1786  
1787  
1788
```

1789 **Figure 27. <SvcMDQuery> — Schema Fragment**

1790 An example message body containing a <SvcMDQuery> message follows. This request queries for a specific service
1791 metadata description by providing the ID of the desired metadata in the <SvcMDID> element.

```
1792  
1793 <ds: SvcMDQuery>  
1794 <ds: SvcMDID>2323872</ds: SvcMDID>  
1795 </ds: SvcMDQuery>  
1796
```

1797 **Example 15. <SvcMDQuery> Message**

1798 3.8.3. SvcMDQueryResponse Message

1799 This response to the <SvcMDQuery> request contains the following elements and attributes.

- 1800 • <lu: Status>: Contains status code; see processing rules.
- 1801 • One or more <SvcCMD> elements if the call was successful (status code is OK).

```
1802
1803 <!-- Response for Query operation on Service Metadata -->
1804
1805 <xs:element name="SvcMDQueryResponse" type="SvcMDQueryResponseType" />
1806
1807 <xs:complexType name="SvcMDQueryResponseType">
1808   <xs:sequence>
1809     <xs:element ref="lu:Status" />
1810     <xs:element ref="SvcMD" minOccurs="0" maxOccurs="unbounded" />
1811   </xs:sequence>
1812 </xs:complexType>
1813
1814
```

1815 **Figure 28. <SvcMDQueryResponse> — Schema Fragment**

```
1816
1817 <ds:SvcMDQueryResponse>
1818   <lu:Status code="OK" />
1819   <ds:SvcMD svcMDID="2323872" >
1820     <ds:Abstract>Profile Service</ds:Abstract>
1821     <ds:ProviderID>http://profile.com</ds:ProviderID>
1822     <ds:ServiceContext>
1823       <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
1824       <ds:EndpointContext>
1825         <ds:Address>https://profile.com/</ds:Address>
1826         <sb:Framework version="2.0" />
1827         <ds:SecurityMechID>
1828           urn:liberty:security:2003-08:TLS:Bearer
1829         </ds:SecurityMechID>
1830       </ds:EndpointContext>
1831     </ds:ServiceContext>
1832   </ds:SvcMD>
1833 </ds:SvcMDQueryResponse>
1834
```

1835 **Example 16. <SvcMDQueryResponse> Message**1836 **3.8.4. Metadata Query Processing Rules**

- 1837 • This operation **MUST** be processed in the context of the WSP, (as opposed to the context of the principal) so that
1838 the WSP can maintain a single set of service metadata across all principals at the same Discovery Service.
- 1839 Even if this operation is invoked with an invocation identity of a principal, the Discovery Service **MUST** use
1840 the Sender's identity (the WSP) when processing this call. The Discovery Service **MAY** refuse to process the
1841 operation if the identity of the Sender cannot be established to the Discovery Service's satisfaction.
- 1842 • The Discovery Service **MUST** limit the results to only those metadata elements stored by the WSP (a WSP **MUST**
1843 **NOT** be able to retrieve metadata elements stored at the same Discovery Service by other WSPs). There **MUST**
1844 **NOT** be any indication on the response as to whether or not other such elements exist.
- 1845 • The Discovery Service **SHOULD** treat a request that matches a subset of the `svcMDID` values specified in the
1846 request as a successful request returning the entries that were found and nothing for the missing entries. The
1847 WSP will be able to distinguish which entries were found by examining the `svcMDID` attribute on the `<SvcMD>`
1848 element(s) in the response.
- 1849 • If request processing succeeded **AND** results are returned, the top-level status code **MUST** be *OK*. Otherwise, the
1850 top-level status code **MUST** be *Failed*.

1851 • If the top-level status code is *Failed*, the response MAY also contain *Forbidden* or *NoResults* as a second-level
 1852 status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no
 1853 second-level status code will appear.

1854 3.9. Operation: *MetadataReplace*

1855 The *MetadataReplace* operation is used by a WSP to replace previously stored metadata in the Discovery Service.
 1856 This is how the WSP updates their metadata without having to reassociate with a principal.

1857 3.9.1. *wsa:Action* values for *MetadataReplace* Messages

1858 <SvcMDReplace> messages MUST include a <wsa:Action> SOAP header with the value of
 1859 "urn:liberty:disco:2005-11:SvcMDReplace".

1860 <SvcMDReplaceResponse> messages MUST include a <wsa:Action> SOAP header with the value of
 1861 "urn:liberty:disco:2005-11:SvcMDReplaceResponse".

1862 3.9.2. *SvcMDReplace* Message

1863 The <SvcMDReplace> is called with one or more replacement <SvcMD> elements each of which must include the
 1864 *svcMDID* attribute set to the ID of the respective metadata element they are to replace.

```
1865
1866 <!-- Replace operation on Service Metadata -->
1867
1868 <xs:element name="SvcMDReplace" type="SvcMDReplaceType" />
1869
1870 <xs:complexType name="SvcMDReplaceType" >
1871   <xs:sequence>
1872     <xs:element ref="SvcMD" maxOccurs="unbounded" />
1873   </xs:sequence>
1874 </xs:complexType>
1875
1876
```

1877 **Figure 29. <SvcMDReplace> — Schema Fragment**

1878 An example message body containing a <SvcMDReplace> message follows. This request replaces an existing
 1879 metadata element to update the endpoint for the service.

```
1880
1881 <ds:SvcMDReplace>
1882   <ds:SvcMD svcMDID="2323872" >
1883     <ds:Abstract>Profile Service</ds:abstract>
1884     <ds:ProviderID>http://profile.com</ds:ProviderID>
1885     <ds:ServiceContext>
1886       <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
1887       <ds:EndpointContext>
1888         <ds:Address>https://newaddr.com/</ds:Address>
1889         <sb:Framework version="2.0" />
1890         <ds:SecurityMechID>
1891           urn:liberty:security:2003-08:TLS:Bearer
1892         </ds:SecurityMechID>
1893       </ds:EndpointContext>
1894     </ds:ServiceContext>
1895   </ds:SvcMD>
1896 </ds:SvcMDReplace>
1897
```

1898 **Example 17. <SvcMDReplace> Message**

1899 3.9.3. SvcMDReplaceResponse Message

1900 This response to the <SvcMDReplace> request contains the following elements and attributes.

- 1901 • <lu:Status>: Contains status code; see processing rules.

```
1902
1903 <!-- Response for SvcMDReplace operation -->
1904
1905 <xs:element name="SvcMDReplaceResponse" type="SvcMDReplaceResponseType" />
1906
1907 <xs:complexType name="SvcMDReplaceResponseType" >
1908   <xs:sequence>
1909     <xs:element ref="lu:Status" />
1910   </xs:sequence>
1911 </xs:complexType>
1912
1913
```

1914 **Figure 30. <SvcMDReplaceResponse> — Schema Fragment**

```
1915
1916 <ds:SvcMDReplaceResponse>
1917   <lu:Status code="OK" />
1918 </ds:SvcMDReplaceResponse>
1919
```

1920 **Example 18. <SvcMDReplaceResponse> Message**

1921 3.9.4. Metadata Replace Processing Rules

- 1922 • This operation **MUST** be processed in the context of the WSP, (as opposed to the context of the principal) so that
1923 the WSP can maintain a single set of service metadata across all principals at the same Discovery Service.
- 1924 Even if this operation is invoked with an invocation identity of a principal, the Discovery Service **MUST** use
1925 the Sender's identity (the WSP) when processing this call. The Discovery Service **MAY** refuse to process the
1926 operation if the identity of the Sender cannot be established to the Discovery Service's satisfaction.
- 1927 • The Discovery Service **MUST** limit the operation to only those metadata elements stored by the WSP (a WSP
1928 **MUST NOT** be able to replace metadata elements stored at the same Discovery Service by other WSPs). There
1929 **MUST NOT** be any indication on the response as to whether or not other such elements exist.
- 1930 • The transaction unit for this operation is the entire set of <SvcCMD> elements; they either all succeed or all fail. The
1931 Discovery Service **MUST** enforce this atomicity.
- 1932 • Once replaced, the previous service metadata element **MUST NOT** be subsequently used by the DS to mint ID-
1933 WSF EPRs. However, WSPs should be prepared to receive requests from WSCs from clients who previously
1934 obtained ID-WSF EPRs minted from the prior metadata which haven't expired.
- 1935 • If request processing succeeded, the top-level status code **MUST** be *OK*. Otherwise, the top-level status code
1936 **MUST** be *Failed*.
- 1937 • If the top-level status code is *Failed*, the response **MAY** also contain *Forbidden* or *NotFound* as a second-level
1938 status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no
1939 second-level status code will appear.

1940 **3.10. Operation: *MetadataDelete***

1941 The *MetadataDelete* operation is used by the WSP to delete previously registered metadata elements in the Discovery
1942 Service.

1943 **3.10.1. wsa:Action values for MetadataDelete Messages**

1944 <SvcMDelete> messages MUST include a <wsa:Action> SOAP header with the value of
1945 "urn:liberty:disco:2005-11:SvcMDelete".

1946 <SvcMDeleteResponse> messages MUST include a <wsa:Action> SOAP header with the value of
1947 "urn:liberty:disco:2005-11:SvcMDeleteResponse".

1948 **3.10.2. SvcMDelete Message**

1949 The <SvcMDelete> is called with one or more <SvcMDID> elements to delete the specified list of service metadata
1950 descriptions.

```
1951  
1952 <!-- Delete operation on Service Metadata -->  
1953  
1954 <xs:element name="SvcMDelete" type="SvcMDeleteType" />  
1955  
1956 <xs:complexType name="SvcMDeleteType">  
1957   <xs:sequence>  
1958     <xs:element ref="SvcMDID" maxOccurs="unbounded" />  
1959   </xs:sequence>  
1960 </xs:complexType>  
1961  
1962
```

1963 **Figure 31. <SvcMDelete> — Schema Fragment**

1964 An example message body containing a <SvcMDelete> message follows. This request deletes a single service
1965 metadata description.

```
1966  
1967 <ds:SvcMDelete>  
1968   <ds:SvcMDID>2323872</ds:SvcMDID>  
1969 </ds:SvcMDelete>  
1970
```

1971 **Example 19. <SvcMDelete> Message**

1972 **3.10.3. SvcMDeleteResponse Message**

1973 This response to the <SvcMDelete> request contains the following elements and attributes.

- 1974 • <lu:Status>: Contains status code; see processing rules.

```

1975
1976 <!-- Response for delete operation on Service Metadata -->
1977
1978 <xs:element name="SvcMDDeleteResponse" type="SvcMDDeleteResponseType" />
1979
1980 <xs:complexType name="SvcMDDeleteResponseType">
1981   <xs:sequence>
1982     <xs:element ref="lu:Status" />
1983   </xs:sequence>
1984 </xs:complexType>
1985
1986

```

1987 **Figure 32. <SvcMDDeleteResponse> — Schema Fragment**

```

1988
1989 <ds:SvcMDDeleteResponse>
1990   <lu:Status code="OK" />
1991 </ds:SvcMDDeleteResponse>
1992

```

1993 **Example 20. <SvcMDDeleteResponse> Message**

1994 3.10.4. Metadata Delete Processing Rules

- 1995 • This operation **MUST** be processed in the context of the WSP, (as opposed to the context of the principal) so that
1996 the WSP can maintain a single set of service metadata across all principals at the same Discovery Service.
- 1997 Even if this operation is invoked with an invocation identity of a principal, the Discovery Service **MUST** use
1998 the Sender's identity (the WSP) when processing this call. The Discovery Service **MAY** refuse to process the
1999 operation if the identity of the Sender cannot be established to the Discovery Service's satisfaction.
- 2000 • If the service metadata being deleted is still associated with one or more principals, the Discovery Service
2001 **SHOULD** automatically remove such associations (i.e. the delete of metadata cascades to delete the associations).
- 2002 • Once deleted, the service metadata element **MUST NOT** be subsequently used by the DS to mint ID-WSF EPRs.
2003 However, WSPs should be prepared to receive requests from WSCs from clients who previously obtained ID-WSF
2004 EPRs minted from the metadata which haven't expired.
- 2005 • The Discovery Service **MUST** limit the operation to only those metadata elements stored by the WSP (a WSP
2006 **MUST NOT** be able to delete metadata elements stored at the same Discovery Service by other WSPs). There
2007 **MUST NOT** be any indication on the response as to whether or not other such elements exist.
- 2008 • If request processing succeeded, the top-level status code **MUST** be *OK*. Otherwise, the top-level status code
2009 **MUST** be *Failed*.
- 2010 • If the top-level status code is *Failed*, the response **MAY** also contain *Forbidden* or *NotFound* as a second-level
2011 status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no
2012 second-level status code will appear.

2013 **3.11. option Value for Response Authentication**

2014 The ID-WSF EPR `<SecurityContext>` element provides a way for services to indicate to clients what mecha-
2015 nisms are necessary for the client to authenticate itself to the service via the `<SecurityMechID>` element. The
2016 `<SecurityMechID>` values defined by [LibertySecMech] also indicate whether the service uses peer entity authenti-
2017 cation (for example, server-side SSL/TLS). However, a web service client may need to know whether the service will
2018 use message authentication (that is, whether the service will sign the response message) and may not be willing to use
2019 a service which does not sign its responses.

2020 To avoid situations where a client requests data and then discovers it does not trust it because it is not signed, an
2021 `<Option>` value is defined:

2022 *urn:liberty:disco:2005-11:options:security-response-x509*

2023 If a service instance always authenticates its response messages according to the "X.509 v3 Certificate Message
2024 Authentication" mechanism in [LibertySecMech], registrations of ID-WSF EPRs describing the service instance
2025 SHOULD include this option value. Otherwise, its registered ID-WSF EPRs MUST NOT include this option value.
2026 Clients MAY include this option value in `<Query>` messages in order to locate only services which always authenticate
2027 their response messages. A service MAY authenticate its response messages even if this option value was not included
2028 in its description at the Discovery Service instance.

2029 In case the service also supports a previous version of the security mechanism specification [LibertySecMech11],
2030 it should be able to register two different endpoints at the Discovery Service, each of them with different Options
2031 values—one according to [LibertySecMech], the other one according to [LibertySecMech11]. This information
2032 will aid the client in determining which version of the WSS-SMS specification ([wss-sms-draft] and/or [wss-sms]) is
2033 supported by the service, and the service will act accordingly, depending on the ID-WSF EPR used by the client. Note
2034 that this behavior only applies to the case when the client's request does not use message authentication mechanisms.

2035 Otherwise, it should be possible for the service to determine the version of the WSS-SMS specification supported by
2036 the client by simply analyzing the `<wsse:Security>` header present in the request.

2037 In general, it is recommended that services do not sign their responses unless they positively know that clients are able
2038 to perform message authentication and are aware of the version of the WSS-SMS spec used by that client.

2039 **3.12. Including Keys in the ModifyResponse Message**

2040 The Discovery Service instance may need to generate signed security tokens in `<QueryResponse>` messages for the
2041 ID-WSF EPRs in question (which are later included in a message to a WSP). The WSP which receives the signed
2042 security tokens from a client needs to be able to verify the Discovery service instance's signature on the security tokens.
2043 Typically the metadata (see [SAMLMeta2]) for the Discovery service instance is sufficient for such information. In
2044 certain situations, such as when the Discovery service instance is hosted on a LUAD (see [LibertyClientProfiles]), it
2045 may not be feasible to assign the LUAD a ProviderID with which to obtain metadata. However, the key material
2046 still needs to be made available to service instances which register ID-WSF EPRs with the Discovery Service which
2047 include security mechanisms requiring such tokens.

2048 The Discovery Service instance may include a `<Keys>` element in the `<ModifyResponse>` in order to provide such
2049 keys.

2050 The Discovery Service instance SHOULD ONLY include the `<Keys>` element in `<ModifyResponse>` messages if
2051 it has no `<ProviderID>` and the `<Modify>` message included an ID-WSF EPR for which the Discovery Service
2052 instance intends to generate signed security tokens.

```
2053
2054 <!-- Keys Element - For use in ModifyResponse -->
2055
2056 <xs:element name="Keys" type="KeysType"/>
2057
2058 <xs:complexType name="KeysType">
2059   <xs:sequence>
2060     <xs:element ref="md:KeyDescriptor"
2061       minOccurs="1"
2062       maxOccurs="unbounded"/>
2063   </xs:sequence>
2064 </xs:complexType>
2065
2066
```

2067 **Figure 33. <keys> — Schema Fragment**

2068 The <Keys> element appears as a child of the <ModifyResponse> element. It contains one or more
2069 <KeyDescriptor> elements.

2070 4. Discovery Service ID-WSF EPR conveyed via a Security Token

2071 In both single sign-on and web services environments, many recipients of a security token find the need to subsequently
2072 invoke the identified principal's Discovery Service in order to discover and invoke identity services on behalf of said
2073 principal. For example, a SAML SP upon receiving an SSO assertion may want to discover and invoke the principals
2074 Profile Service and would need the Discovery Service ID-WSF EPR in order to do so.

2075 In the SSO environment, this concept is often referred to as the "Discovery Service Bootstrap" in that the SP is using
2076 the data in the SSO assertion to bootstrap into the ID-WSF environment.

2077 The need for this Discovery Service ID-WSF EPR is not restricted to SSO environments as any WSP that is invoked
2078 by a WSC may in turn need to act as a WSC and invoke other WSPs in order to fulfill the requested operation. For
2079 example, a Profile Service WSP may need to invoke the Interaction Service in order to request consent from the user
2080 before releasing data to a WSC.

2081 This section describes the recommended interoperable method for an Identity Provider and/or Discovery Service
2082 can embed an ID-WSF EPR for the Discovery Service within security and/or Identity tokens that they issue.
2083 Unfortunately, because of the variance in structure and formats of various tokens, the model used tends to be specific
2084 to the format of the security token. The remainder of this section documents how this is accomplished within some
2085 specific token formats.

2086 4.1. EPR Generation Rules

2087 The Discovery Service Bootstrap ID-WSF EPR which is placed into any security token must be generated according
2088 to the following rules:

2089 • The `<wsa:EndpointReference>` that MAY contain `<SecurityContext>` element(s) in turn containing
2090 `<sec:Token>` elements containing embedded security tokens, which are necessary to access the Discovery Ser-
2091 vice instance(s).

2092 • The `<sec:Token>` element MAY instead include a reference to an external security token using a
2093 `<wsse:SecurityTokenReference>` containing a non-relative URI reference to a security token.

2094 • The `<sec:Token>` element's `ref` attribute MAY instead refer to local security token available elsewhere in the
2095 same security token (such as another ID-WSF EPR within the security token). These references SHOULD only
2096 refer to elements within the security token carrying the ID-WSF EPR so that the reference will remain valid if the
2097 security token is separated from any message carrying the token.

2098 It is even possible (and in some cases typical) for the reference to be to the enveloping security token itself (the
2099 security token that contains this ID-WSF EPR) In such cases, the enveloping security token SHOULD carry the
2100 necessary information to support its consumption at the Discovery Service (as well as the information necessary
2101 for consumption at its primary relying party (the SP/WSP)).

2102 For example, with a SAML Assertion, this includes:

2103 • A second `<Audience>` element with the Discovery Service's ProviderID.

2104 • A subject confirmation method that the relying party can meet. This will frequently be
2105 `urn:oasis:names:tc:SAML:2.0:cm:bearer` in which case the same confirmation can be used by
2106 both parties. However, the assertion could contain multiple confirmation methods one for the initial party to
2107 use when invoking the relying party and one for the relying party to use when invoking the DS.

2108 This will allow the Discovery Service to validate the assertion using the normal assertion processing rules without
2109 having to manage some form of exception for self issued assertions.

2110 4.2. SAML 2.0 Security Tokens

2111 In a SAML 2.0 Assertion, the Discovery Service ID-WSF EPR SHOULD be conveyed as an XML element within the
2112 <saml2:AttributeStatement> element in a <saml2:Assertion>.

2113 The <saml2:AttributeStatement> SHOULD be constructed according to the following rules:

2114 • The Name attribute of the <saml2:Attribute> element MUST be:

2115 *urn:liberty:disco:2005-11:DiscoveryEPR*

2116 • The NameFormat attribute of the <saml2:Attribute> element MUST be:

2117 *urn:oasis:names:tc:SAML:2.0:attrname-format:uri*

2118 • One or more <saml2:AttributeValue> elements MUST be included which each containing a single
2119 <wsa:EndpointReference> element identifying a Discovery Service instance(s). These Discovery Ser-
2120 vice instances SHOULD offer identity services for the Principal identified in the Subject element inside the
2121 <saml2:Assertion>.

2122 An example <saml2:AttributeStatement> that might be found in a SAMLv2 <saml2:Assertion> follows.
2123 The example includes a <sec:Token> element which has a reference to the surrounding assertion.

```
2124
2125 <AttributeStatement xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
2126   <Attribute Name="urn:liberty:disco:2005-11:DiscoveryEPR"
2127     NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
2128     <AttributeValue>
2129       <wsa:EndpointReference>
2130         <wsa:Address>https://example.com/disco/</wsa:Address>
2131
2132         <wsa:Metadata>
2133           <Abstract>
2134             The Principal's Discovery Service Resource
2135           </Abstract>
2136
2137           <ServiceType>urn:liberty:disco:2005-11</ServiceType>
2138
2139           <ProviderID>http://example.com/</ProviderID>
2140
2141           <SecurityContext>
2142             <SecurityMechID>urn:liberty:security:2005-02:TLS:bearer</SecurityMechID>
2143             <sec:Token ref="..." usage="urn:liberty:security:tokenusage:2005-11:SecurityToken">
2144             </SecurityContext>
2145           </wsa:Metadata>
2146         </wsa:EndpointReference>
2147       </AttributeValue>
2148     </Attribute>
2149   </AttributeStatement>
2150
```

2151 **Example 21. <AttributeStatement> that might be found in a SAMLv2 AuthnResponse**

2152 In all cases, this <AttributeStatement> MUST carry an ID-WSF EPR for the Liberty Discovery Service. Any
2153 other ID-WSF EPRs are to be discovered by contacting the Discovery Service.

2154 4.3. SAML 1.x (Liberty ID-FF) Security Tokens

2155 In a SAML 1.x Assertion, the Discovery Service ID-WSF EPR SHOULD be conveyed as an XML element within the
2156 <saml:AttributeStatement> element in a <saml:Assertion>.

2157 The <saml:AttributeStatement> SHOULD be constructed according to the following rules:

- 2158 • For the <saml:Attribute> element:
 - 2159 • The AttributeName attribute MUST be "DiscoveryEPR".
 - 2160 • The AttributeNamespace attribute MUST be "urn:liberty:disco:2005-11".
- 2161 • The <Subject> element of the <saml:AttributeStatement> element MUST carry the identity of the
 2162 principal whose Discovery Service is referenced by this EPR and SHOULD be the same identity in the subject of
 2163 the other statements in the <saml:Assertion>.
- 2164 • One or more <saml:AttributeValue> elements MUST be included which each containing a single
 2165 <wsa:EndpointReference> element identifying a Discovery Service instance(s). These Discovery Ser-
 2166 vice instances SHOULD offer identity services for the Principal identified in the Subject element inside this
 2167 <saml:AttributeStatment>.

2168 An example <saml:AttributeStatement> that might be found in a SAML 1.1 <saml:Assertion> follows. The
 2169 example includes a <sec:Token> element which has a reference to the surrounding assertion.

```

2170
2171 <AttributeStatement xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
2172   <Subject>
2173     <NameIdentifier Format="urn:liberty:iff:nameid:federated">
2174       d0CQF8elJTDLmzEO
2175     </NameIdentifier>
2176   </Subject>
2177   <Attribute AttributeName="DiscoveryEPR"
2178     AttributeNamespace="urn:liberty:disco:2005-11">
2179     <AttributeValue>
2180       <wsa:EndpointReference>
2181         <wsa:Address>https://example.com/disco/</wsa:Address>
2182
2183         <wsa:Metadata>
2184           <Abstract>
2185             The Principal's Discovery Service Resource
2186           </Abstract>
2187
2188           <ServiceType>urn:liberty:disco:2005-11</ServiceType>
2189
2190           <ProviderID>http://example.com/</ProviderID>
2191
2192           <SecurityContext>
2193             <SecurityMechID>urn:liberty:security:2005-02:TLS:bearer</SecurityMechID>
2194             <sec:Token ref="..." usage="urn:liberty:security:tokenusage:2005-11:SecurityToken">
2195               </SecurityContext>
2196           </wsa:Metadata>
2197         </wsa:EndpointReference>
2198       </AttributeValue>
2199     </Attribute>
2200   </AttributeStatement>
2201
  
```

2202 **Example 22. <AttributeStatement> that might be found in a SAML 1.1 AuthnResponse**

2203 In all cases, this <AttributeStatement> MUST only carry an ID-WSF EPR for the Liberty Discovery Service.
 2204 Any other ID-WSF EPRs are to be discovered by contacting the Discovery Service.

2205 5. ID-WSF 1.x Resource Offering conveyed in an EPR

2206 In order to support the discovery and subsequent invocation of ID-WSF 1.0 and 1.1 services it may be necessary for
2207 the Discovery Service to carry the ID-WSF 1.x Resource Offering information within the ID-WSF EPR.

2208 The process involves taking the fields that would normally be present in the Resource Offering and placing them into
2209 the appropriate fields within the EPR according to the following rules:

- 2210 • The <ResourceID> element and/or the <EncryptedResourceID> element are placed into the <Metadata>
2211 element as-is.
- 2212 • The <ServiceType> element in the <ServiceInstance> element is placed into the <Metadata> element.
- 2213 • The <ProviderID> element in the <ServiceInstance> element is placed into the <Metadata> element.
- 2214 • The <SecurityMechID> element in ServiceInstance/Description is placed into the
2215 <SecurityContext> element (and will be combined with other SecurityMechIDs based upon whether or
2216 not they share the same endpoint *and* credential (or do not use a credential)).
- 2217 • The data from the <Endpoint> element in ServiceInstance/Description is placed into the <Address>
2218 element. Note that if there are multiple distinct <Endpoint>s they must be placed into different ID-WSF EPRs
2219 rather than being able to be placed into a single EPR like they were in an RO.
- 2220 • The <SoapAction> element in ServiceInstance/Description is placed into the <Metadata> element.
- 2221 • Options are placed into the <Metadata> element.
- 2222 • Abstract is placed into the <Metadata> element.
- 2223 • Credentials, which in the days of the Resource Offering were carried elsewhere in the message and referenced
2224 from the ServiceInstance/Description element are now carried directly within the EPR in a <sec:Token>
2225 element in the <SecurityContext> element.

2226 In addition, the ID-WSF EPR MUST also include at least one <sbfl:Framework> element with the appropriate value
2227 (1.0 or 1.1) in the version attribute for the ID-WSF version being used.

2228 As an example, let's start with an example ID-WSF 1.x Resource Offering:

```
2229  
2230 <ResourceOffering>  
2231   <ResourceID> 123 </ResourceID>  
2232   <ServiceInstance>  
2233     <ServiceType>urn:liberty:idsis-pp:2003-08</ServiceType>  
2234     <ProviderID>http://pp.services.aol.com</ProviderID>  
2235     <Description CredentialRef="1">  
2236       <SecurityMechID>urn:liberty:security:2004-04:TLS:Bearer</SecurityMechID>  
2237       <Endpoint>https://ep1.pp.service.aol.com</Endpoint>  
2238     </Description>  
2239     <Description>  
2240       <SecurityMechID>urn:liberty:security:2004-04:Client-TLS:Null</SecurityMechID>  
2241       <Endpoint>https://ep1.pp.service.aol.com</Endpoint>  
2242     </Description>  
2243   </ServiceInstance>  
2244 </ResourceOffering>  
2245 <Credentials>  
2246   <saml1:Assertion AssertionID="1" ...>  
2247     Assertion data goes here  
2248   </saml1:Assertion>  
2249 </Credentials>  
2250
```

2251 Translating this using the above rules would result in the following ID-WSF EPR:

```

2252
2253 <wsa:EndpointReference>
2254 <wsa:Address>https://ep1.pp.services.aol.com</wsa:Address >
2255 <wsa:Metadata>
2256 <ds1:ResourceID>123</ds1:ResourceID>
2257 <ds2:ProviderID>http://pp.services.aol.com</ds2:ProviderID>
2258 <ds2:ServiceType>urn:liberty:idsis-pp:2003-08</ds2:ServiceType>
2259 <ds2:FrameworkVersion="1.1" />
2260 <ds2:SecurityContext>
2261 <ds2:SecurityMechID>urn:liberty:security:2004-04:TLS:Bearer</ds2:SecurityMechID>
2262 <sec:Token usage="urn:liberty:security:tokenusage:2005-11:SecurityToken">
2263 <saml1:Assertion AssertionID="1" ... >
2264 ... assertion data goes here ...
2265 </saml1:Assertion>
2266 </sec:Token>
2267 </ds2:SecurityContext>
2268 <ds2:SecurityContext>
2269 <ds2:SecurityMechID>
2270 urn:liberty:security:2004-04:Client-TLS:Null
2271 </ds2:SecurityMechID>
2272 </ds2:SecurityContext>
2273 </wsa:Metadata>
2274 </wsa:EndpointReference>
2275
  
```

2276 And subsequently, the invocation of the ID-WSF 1.x service would look to be something along the lines of (assuming
 2277 that the WSC chose to use the "...:TLS:bearer" Security Mechanism):

```

2278
2279 <?xml version="1.0" encoding="utf-8" ?>
2280 <S:Envelope...
2281 <S:Header>
2282 <sb:Correlation S:mustUnderstand="1"
2283 messageID=uuid:958312848-29348938-232342121
2284 timestamp="2003-06-06T18:29:18Z" />
2285 <wsse:Security>
2286 <saml1:Assertion AssertionID="1" ... >
2287 ... assertion data goes here ...
2288 </saml1:Assertion>
2289 </wsse:Security>
2290 </S:Header>
2291 <S:Body>
2292 <pp:Query>
2293 <pp:ResourceID>123</pp:ResourceID>
2294 <pp:QueryItem>
2295 ... query data goes here ...
2296 </pp:QueryItem>
2297 </pp:Query>
2298 </S:Body>
2299 </S:Envelope>
2300
  
```

2301 **6. Acknowledgments**

2302 Many people have made contributions to this specification as it has evolved over time. The original specification was
2303 written by John Beatty with subsequent versions "inked" by Jonathan Sergent and later, Jeff Hodges and now myself.

2304 The changes made in this latest release of the specification are due in a large part to the work of Jeff Hodges, Robert
2305 Aarts, John Kemp, Gary Ellison and Greg Whitehead. Many others, including those that are listed as contributors
2306 on the cover page, have also played a part in this and earlier releases of the specification. Many thanks to all who
2307 participated (and apologies if I have forgotten to mention your name).

2308 References

2309 Normative

- 2310 [LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0-05, Liberty Alliance Project (28
2311 March 2006). <http://www.projectliberty.org/specs>
- 2312 [LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version v2.0-18, Liberty
2313 Alliance Project (28 March 2006). <http://www.projectliberty.org/specs>
- 2314 [LibertySecMech11] Ellison, Gary, eds. "Liberty ID-WSF Security Mechanisms," Version 1.1, Liberty Alliance
2315 Project (18 April 2004). <http://www.projectliberty.org/specs/>
- 2316 [LibertySecMech20SAML] Hirsch, Frederick, eds. "ID-WSF 2.0 SecMech SAML Profile," Version v2.0-14, Liberty
2317 Alliance Project (28 March 2006). <http://www.projectliberty.org/specs>
- 2318 [LibertyAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. "Liberty ID-WSF Authentication,
2319 Single Sign-On, and Identity Mapping Services Specification," Version v2.0-16, Liberty Alliance Project
2320 (27 March 2006). <http://www.projectliberty.org/specs>
- 2321 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Lib-
2322 erty ID-WSF SOAP Binding Specification," Version 2.0-14, Liberty Alliance Project (28 March, 2006).
2323 <http://www.projectliberty.org/specs>
- 2324 [LibertyIDWSF20SCR] Whitehead, Greg, eds. Version 1.0-05, Liberty Alliance Project (27 March 2006).
2325 <http://www.projectliberty.org/specs>
- 2326 [RFC2119] Bradner, S., eds. "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
2327 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>
- 2328 [RFC3986] Berners-Lee, T., Fielding, R., Masinter, L., eds. (January 2005). "Uniform Resource Identifier
2329 (URI): Generic Syntax," RFC 3986 (Obsoletes RFC2732, RFC2396, RFC1808) (Updates RFC1738) (Also
2330 STD0066) (Status: STANDARD), The Internet Engineering Task Force <http://www.ietf.org/rfc/rfc3986.txt>
- 2331 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions
2332 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
2333 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-
2334 open.org/security/saml/v2.0/saml-core-2.0-os.pdf)
- 2335 [SAMLMeta2] Cantor, Scott, Moreh, Jahan, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Metadata for the OA-
2336 SIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for
2337 the Advancement of Structured Information Standards [http://docs.oasis-open.org/security/saml/v2.0/saml-
metadata-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-
2338 metadata-2.0-os.pdf)
- 2339 [Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May
2340 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium
2341 <http://www.w3.org/TR/xmlschema-1/>
- 2342 [Schema2] Biron, Paul V., Malhotra, Ashok, eds. (May 2002). "XML Schema Part 2: Datatypes," Recommendation,
2343 World Wide Web Consortium <http://www.w3.org/TR/xmlschema-2/>
- 2344 [WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, eds. World Wide Web
2345 Consortium W3C Candidate Recommendation (17 August 2005). [http://www.w3.org/TR/2005/CR-ws-addr-
core-20050817/](http://www.w3.org/TR/2005/CR-ws-addr-
2346 core-20050817/)

- 2347 [WSAv1.0-SOAP] "WS-Addressing 1.0 SOAP Binding," Gudgin, Martin, Hadley, Marc, eds. World Wide Web
2348 Consortium W3C Candidate Recommendation (17 August 2005). [http://www.w3.org/TR/2005/CR-ws-addr-
soap-20050817/](http://www.w3.org/TR/2005/CR-ws-addr-
2349 soap-20050817/)
- 2350 [WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Meredith,
2351 Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001).
2352 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- 2353 [wss-sms] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (January, 2004). "Web
2354 Services Security: SOAP Message Security," OASIS Standard V1.0 [OASIS 200401], Organization for
2355 the Advancement of Structured Information Standards [http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-
2356 wss-soap-message-security-1.0.pdf)
- 2357 [wss-sms-draft] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (June 30,
2358 2003). "Web Services Security: SOAP Message Security," Draft WSS-SOAPMessageSecurity-14-
2359 06-2003, Organization for the Advancement of Structured Information Standards [http://www.oasis-
open.org/committees/download.php/2757/WSS-SOAPMessageSecurity-14-063003-merged.pdf](http://www.oasis-
2360 open.org/committees/download.php/2757/WSS-SOAPMessageSecurity-14-063003-merged.pdf)
- 2361 [xmlenc-core] Eastlake, Donald, Reagle, Joseph, eds. (10 December 2002). "XML Encryption Syntax and Process-
2362 ing," W3C Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlenc-core/>

2363 Informative

- 2364 [LibertyPAOS] Aarts, Robert, Kemp, John, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version
2365 2.0-04, Liberty Alliance Project (28 March 2006). <http://www.projectliberty.org/specs>
- 2366 [LibertyClientProfiles] Aarts, Robert, Kainulainen, Jukka, Kemp, John, eds. Version v2.0-04, Liberty Alliance Project
2367 (27 Mar 2006). <http://www.projectliberty.org/specs>

2368 A. Discovery Service Version 2.0 XSD

```
2369
2370 <?xml version="1.0" encoding="UTF-8"?>
2371 <xs:schema targetNamespace="urn:liberty:disco:2005-11"
2372     xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
2373     xmlns:sb="urn:liberty:sb:2005-11"
2374     xmlns:sbf="urn:liberty:sb"
2375     xmlns:sec="urn:liberty:security:2005-11"
2376     xmlns:lu="urn:liberty:util:2005-11"
2377     xmlns:wsa="http://www.w3.org/2005/08/addressing"
2378     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
2379     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
2380     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
2381     xmlns:xs="http://www.w3.org/2001/XMLSchema"
2382
2383     xmlns="urn:liberty:disco:2005-11"
2384     elementFormDefault="qualified"
2385     attributeFormDefault="unqualified"
2386 >
2387
2388 <xs:import namespace="urn:liberty:util:2005-11"
2389     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
2390
2391 <xs:import namespace="urn:liberty:sb:2005-11"
2392     schemaLocation="liberty-idwsf-soap-binding-v2.0.xsd"/>
2393
2394 <xs:import namespace="urn:liberty:sb"
2395     schemaLocation="liberty-idwsf-soap-binding.xsd"/>
2396
2397 <xs:import namespace="http://www.w3.org/2005/08/addressing"
2398     schemaLocation="ws-addr-1.0.xsd"/>
2399
2400 <xs:import namespace="urn:oasis:names:tc:SAML:2.0:metadata"
2401     schemaLocation="saml-schema-metadata-2.0.xsd"/>
2402
2403 <xs:import namespace="urn:liberty:security:2005-11"
2404     schemaLocation="liberty-idwsf-security-mechanisms-v2.0.xsd"/>
2405
2406 <xs:import namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-s
2407 ecect-1.0.xsd"
2408     schemaLocation="wss-secext-1.0.xsd"/>
2409
2410 <xs:import namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
2411 ty-utility-1.0.xsd"
2412     schemaLocation="wss-util-1.0.xsd"/>
2413
2414 <xs:annotation>
2415     <xs:documentation>
2416         XML Schema from Liberty Discovery Service Specification.
2417     </xs:documentation>
2418     <xs:documentation>### NOTICE ###
2419
2420         Copyright (c) 2006 Liberty Alliance participants, see
2421         http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2422     </xs:documentation>
2423 </xs:annotation>
2424
2425
2426
2427 <!-- **** Discovery Service Data Elements & Types **** -->
2428
2429 <!-- The data elements and types in this section are used to
2430 embellish WS-Addressing Endpoint References (EPRs).
2431 They are placed in the /wsa:EndpointReference/Metadata
2432 element. Specific usage and cardinalities are stipulated
2433 in the Discovery Service v2.0 Specification. -->
```

```

2434
2435 <!-- Abstract: natural-language description of service -->
2436
2437 <xs:element name="Abstract" type="xs:string"/>
2438
2439 <!-- Provider ID -->
2440
2441 <xs:element name="ProviderID" type="xs:anyURI"/>
2442
2443 <!-- Service Type -->
2444
2445 <xs:element name="ServiceType" type="xs:anyURI"/>
2446
2447 <!-- Framework Description -->
2448
2449 <xs:element name="Framework" type="sbf:FrameworkType" />
2450
2451 <!-- EPR Expiration Timestamp -->
2452
2453 <xs:attribute name="NotOnOrAfter" type="xs:dateTime"/>
2454
2455 <!-- Security Context Container -->
2456
2457 <xs:element name="SecurityContext">
2458   <xs:complexType>
2459     <xs:sequence>
2460       <xs:element ref="SecurityMechID"
2461         minOccurs="1"
2462         maxOccurs="unbounded" />
2463
2464       <xs:element ref="sec:Token"
2465         minOccurs="0"
2466         maxOccurs="unbounded" />
2467     </xs:sequence>
2468   </xs:complexType>
2469 </xs:element>
2470
2471 <!-- Security Mechanism ID -->
2472
2473 <xs:element name="SecurityMechID" type="xs:anyURI"/>
2474
2475 <!-- Options -->
2476
2477 <xs:element name="Options" type="OptionsType"/>
2478
2479 <xs:element name="Option" type="xs:anyURI" />
2480
2481 <xs:complexType name="OptionsType">
2482   <xs:sequence>
2483     <xs:element ref="Option" minOccurs="0" maxOccurs="unbounded" />
2484   </xs:sequence>
2485 </xs:complexType>
2486
2487 <!-- Address -->
2488
2489 <xs:element name="Address" type="xs:anyURI"/>
2490
2491 <!-- Action(s) - the interfaces available at this service -->
2492
2493 <xs:element name="Action" type="xs:anyURI" />
2494 <!-- Keys Element - For use in ModifyResponse -->
2495
2496 <xs:element name="Keys" type="KeysType"/>
2497
2498 <xs:complexType name="KeysType">
2499   <xs:sequence>
2500     <xs:element ref="md:KeyDescriptor"

```

```

2501         minOccurs="1"
2502         maxOccurs="unbounded" />
2503     </xs:sequence>
2504 </xs:complexType>
2505
2506 <!-- Service Metadata (SvcMD) - metadata about service instance -->
2507
2508 <xs:element name="SvcMD" type="SvcMetadataType" />
2509 <xs:complexType name="SvcMetadataType">
2510     <xs:sequence>
2511         <xs:element ref="Abstract" />
2512         <xs:element ref="ProviderID" />
2513         <xs:element ref="ServiceContext" maxOccurs="unbounded" />
2514     </xs:sequence>
2515     <xs:attribute name="svcMDID" type="xs:string" use="optional" />
2516 </xs:complexType>
2517
2518 <!-- ServiceContext - describes service type/option/endpoint context -->
2519 <xs:element name="ServiceContext" type="ServiceContextType" />
2520 <xs:complexType name="ServiceContextType">
2521     <xs:sequence>
2522         <xs:element ref="ServiceType" maxOccurs="unbounded" />
2523         <xs:element ref="Options" minOccurs="0"
2524             maxOccurs="unbounded" />
2525         <xs:element ref="EndpointContext" maxOccurs="unbounded" />
2526     </xs:sequence>
2527 </xs:complexType>
2528
2529 <!-- EndpointContext - describes endpoints used to access service -->
2530 <xs:element name="EndpointContext" type="EndpointContextType" />
2531 <xs:complexType name="EndpointContextType">
2532     <xs:sequence>
2533         <xs:element ref="Address" maxOccurs="unbounded" />
2534         <xs:element ref="sbf:Framework" maxOccurs="unbounded" />
2535         <xs:element ref="SecurityMechID" maxOccurs="unbounded" />
2536         <xs:element ref="Action" minOccurs="0"
2537             maxOccurs="unbounded" />
2538     </xs:sequence>
2539 </xs:complexType>
2540
2541 <!-- SvcMD ID element used to refer to Service Metadata elements -->
2542 <xs:element name="SvcMDID" type="xs:string" />
2543
2544 <!-- **** Discovery Service Protocol Messages Elements & Types **** -->
2545
2546 <!-- Query Message Element & Type -->
2547
2548 <xs:element name="Query" type="QueryType" />
2549
2550 <xs:complexType name="QueryType">
2551     <xs:sequence>
2552         <xs:element name="RequestedService"
2553             type="RequestedServiceType"
2554             minOccurs="0"
2555             maxOccurs="unbounded" />
2556     </xs:sequence>
2557
2558     <xs:attribute ref="wsu:Id" use="optional" />
2559 </xs:complexType>
2560
2561 <xs:complexType name="RequestedServiceType">
2562     <xs:sequence>
2563         <xs:element ref="ServiceType" minOccurs="0" maxOccurs="unbounded" />
2564
2565         <xs:element ref="ProviderID" minOccurs="0" maxOccurs="unbounded" />
2566
2567         <xs:element ref="Options" minOccurs="0" maxOccurs="unbounded" />

```



```

2568
2569     <xs:element ref="SecurityMechID" minOccurs="0" maxOccurs="unbounded" />
2570
2571     <xs:element ref="Framework" minOccurs="0" maxOccurs="unbounded" />
2572
2573     <xs:element ref="Action" minOccurs="0" maxOccurs="unbounded" />
2574
2575     <xs:any namespace="##other"
2576           processContents="lax"
2577           minOccurs="0"
2578           maxOccurs="unbounded" />
2579
2580 </xs:sequence>
2581
2582 <xs:attribute name="reqID" type="xs:string" use="optional" />
2583 <xs:attribute name="resultsType" type="xs:string" use="optional" />
2584
2585 </xs:complexType>
2586
2587 <!-- QueryResponse Message Element & Type -->
2588
2589 <xs:element name="QueryResponse" type="QueryResponseType" />
2590
2591 <xs:complexType name="QueryResponseType">
2592   <xs:sequence>
2593     <xs:element ref="lu:Status" />
2594
2595     <xs:element ref="wsa:EndpointReference"
2596           minOccurs="0"
2597           maxOccurs="unbounded" />
2598   </xs:sequence>
2599
2600   <xs:attribute name="id"
2601         type="xs:ID"
2602         use="optional" />
2603 </xs:complexType>
2604
2605
2606 <!-- -->
2607 <!-- DS Interfaces for SvcMD Associations -->
2608 <!-- -->
2609 <!-- These interfaces support the adding, deleting,-->
2610 <!-- querying SvcMD Associations for a principal. -->
2611 <!-- -->
2612
2613 <!-- SvcMDAssociationAdd operation -->
2614
2615 <xs:element name="SvcMDAssociationAdd" type="SvcMDAssociationAddType" />
2616
2617 <xs:complexType name="SvcMDAssociationAddType">
2618   <xs:sequence>
2619     <xs:element ref="SvcMDID" maxOccurs="unbounded" />
2620   </xs:sequence>
2621 </xs:complexType>
2622 <!-- Response for SvcMDAssociationAdd operation -->
2623
2624 <xs:element name="SvcMDAssociationAddResponse"
2625       type="SvcMDAssociationAddResponseType" />
2626
2627 <xs:complexType name="SvcMDAssociationAddResponseType">
2628   <xs:sequence>
2629     <xs:element ref="lu:Status" />
2630   </xs:sequence>
2631 </xs:complexType>
2632 <!-- SvcMDAssociationDelete operation -->
2633
2634 <xs:element name="SvcMDAssociationDelete" type="SvcMDAssociationDeleteType" />

```

```

2635
2636 <xs:complexType name="SvcMDAssociationDeleteType">
2637   <xs:sequence>
2638     <xs:element ref="SvcMDID" maxOccurs="unbounded" />
2639   </xs:sequence>
2640 </xs:complexType>
2641 <!-- Response for SvcMDAssociationDelete operation -->
2642
2643 <xs:element name="SvcMDAssociationDeleteResponse"
2644   type="SvcMDAssociationDeleteResponseType" />
2645
2646 <xs:complexType name="SvcMDAssociationDeleteResponseType">
2647   <xs:sequence>
2648     <xs:element ref="lu:Status" />
2649   </xs:sequence>
2650 </xs:complexType>
2651 <!-- SvcMDAssociationQuery operation -->
2652
2653 <xs:element name="SvcMDAssociationQuery" type="SvcMDAssociationQueryType" />
2654
2655 <xs:complexType name="SvcMDAssociationQueryType">
2656   <xs:sequence>
2657     <xs:element ref="SvcMDID" minOccurs="0" maxOccurs="unbounded" />
2658   </xs:sequence>
2659 </xs:complexType>
2660 <!-- Response for SvcMDAssociationQuery operation -->
2661
2662 <xs:element name="SvcMDAssociationQueryResponse"
2663   type="SvcMDAssociationQueryResponseType" />
2664
2665 <xs:complexType name="SvcMDAssociationQueryResponseType">
2666   <xs:sequence>
2667     <xs:element ref="lu:Status" />
2668     <xs:element ref="SvcMDID" minOccurs="0" maxOccurs="unbounded" />
2669   </xs:sequence>
2670 </xs:complexType>
2671
2672 <!--           -->
2673 <!-- DS Interfaces for Service Metadata Management -->
2674 <!--           -->
2675 <!-- These interfaces document a create, replace, -->
2676 <!-- delete, and query interface for the service -->
2677 <!-- metadata which is later associated with a -->
2678 <!-- principal.           -->
2679 <!--           -->
2680
2681 <!-- Register operation for Service Metadata -->
2682
2683 <xs:element name="SvcMDRegister" type="SvcMDRegisterType" />
2684
2685 <xs:complexType name="SvcMDRegisterType">
2686   <xs:sequence>
2687     <xs:element ref="SvcMD" maxOccurs="unbounded" />
2688   </xs:sequence>
2689 </xs:complexType>
2690
2691 <!-- Response for SvcMDRegister operation -->
2692
2693 <xs:element name="SvcMDRegisterResponse"
2694   type="SvcMDRegisterResponseType" />
2695
2696 <xs:complexType name="SvcMDRegisterResponseType">
2697   <xs:sequence>
2698     <xs:element ref="lu:Status" />
2699     <xs:element ref="SvcMDID" minOccurs="0" maxOccurs="unbounded" />
2700     <xs:element ref="Keys" minOccurs="0" maxOccurs="unbounded" />
2701

```

```
2702
2703     </xs:sequence>
2704 </xs:complexType>
2705
2706 <!-- Delete operation on Service Metadata -->
2707
2708 <xs:element name="SvcMDDelete" type="SvcMDDeleteType" />
2709
2710 <xs:complexType name="SvcMDDeleteType">
2711   <xs:sequence>
2712     <xs:element ref="SvcMDID" maxOccurs="unbounded" />
2713   </xs:sequence>
2714 </xs:complexType>
2715
2716 <!-- Response for delete operation on Service Metadata -->
2717
2718 <xs:element name="SvcMDDeleteResponse" type="SvcMDDeleteResponseType" />
2719
2720 <xs:complexType name="SvcMDDeleteResponseType">
2721   <xs:sequence>
2722     <xs:element ref="lu:Status" />
2723   </xs:sequence>
2724 </xs:complexType>
2725
2726 <!-- Query operation on Service Metadata -->
2727
2728 <xs:element name="SvcMDQuery" type="SvcMDQueryType" />
2729
2730 <xs:complexType name="SvcMDQueryType">
2731   <xs:sequence>
2732     <xs:element ref="SvcMDID"
2733       minOccurs="0"
2734       maxOccurs="unbounded" />
2735   </xs:sequence>
2736 </xs:complexType>
2737
2738 <!-- Response for Query operation on Service Metadata -->
2739
2740 <xs:element name="SvcMDQueryResponse" type="SvcMDQueryResponseType" />
2741
2742 <xs:complexType name="SvcMDQueryResponseType">
2743   <xs:sequence>
2744     <xs:element ref="lu:Status" />
2745     <xs:element ref="SvcMD" minOccurs="0" maxOccurs="unbounded" />
2746   </xs:sequence>
2747 </xs:complexType>
2748
2749 <!-- Replace operation on Service Metadata -->
2750
2751 <xs:element name="SvcMDReplace" type="SvcMDReplaceType" />
2752
2753 <xs:complexType name="SvcMDReplaceType">
2754   <xs:sequence>
2755     <xs:element ref="SvcMD" maxOccurs="unbounded" />
2756   </xs:sequence>
2757 </xs:complexType>
2758
2759 <!-- Response for SvcMDReplace operation -->
2760
2761 <xs:element name="SvcMDReplaceResponse" type="SvcMDReplaceResponseType" />
2762
2763 <xs:complexType name="SvcMDReplaceResponseType">
2764   <xs:sequence>
2765     <xs:element ref="lu:Status" />
2766   </xs:sequence>
2767 </xs:complexType>
2768
```

```
2769  
2770 </xs:schema>  
2771  
2772
```

2773 B. Discovery Service WSDL

```
2774
2775 <?xml version="1.0"?>
2776 <definitions name="disco-svc"
2777   targetNamespace="urn:liberty:disco:2005-11"
2778   xmlns:tns="urn:liberty:disco:2005-11"
2779   xmlns="http://schemas.xmlsoap.org/wsdl/"
2780   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2781   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2782   xmlns:sb="urn:liberty:sb:2005-11"
2783   xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
2784   xmlns:disco="urn:liberty:disco:2005-11"
2785   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2786   xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
2787     http://schemas.xmlsoap.org/wsdl/
2788     http://www.w3.org/2006/02/addressing/wsdl
2789     http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
2790
2791 <!-- Abstract WSDL for Liberty Discovery Service v2.0 Specification -->
2792
2793 <xsd:documentation>
2794
2795   XML Schema from Liberty Discovery Service Specification.
2796
2797   ### NOTICE ###
2798
2799   Copyright (c) 2004-2006 Liberty Alliance participants, see
2800   http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2801
2802 </xsd:documentation>
2803
2804 <types>
2805   <xsd:schema>
2806     <xsd:import namespace="urn:liberty:disco:2005-11"
2807       schemaLocation="liberty-idwsf-disco-svc-v2.0.xsd"/>
2808   </xsd:schema>
2809 </types>
2810
2811 <message name="Query">
2812   <part name="body" element="disco:Query"/>
2813 </message>
2814 <message name="QueryResponse">
2815   <part name="body" element="disco:QueryResponse"/>
2816 </message>
2817
2818 <message name="SvcMDAssociationAdd">
2819   <part name="body" element="disco:SvcMDAssociationAdd"/>
2820 </message>
2821 <message name="SvcMDAssociationAddResponse">
2822   <part name="body" element="disco:SvcMDAssociationAddResponse"/>
2823 </message>
2824
2825 <message name="SvcMDAssociationQuery">
2826   <part name="body" element="disco:SvcMDAssociationQuery"/>
2827 </message>
2828 <message name="SvcMDAssociationQueryResponse">
2829   <part name="body" element="disco:SvcMDAssociationQueryResponse"/>
2830 </message>
2831
2832 <message name="SvcMDAssociationDelete">
2833   <part name="body" element="disco:SvcMDAssociationDelete"/>
2834 </message>
2835 <message name="SvcMDAssociationDeleteResponse">
2836   <part name="body" element="disco:SvcMDAssociationDeleteResponse"/>
2837 </message>
2838
```

```

2839 <message name="SvcMDRegister">
2840   <part name="body" element="disco:SvcMDRegister" />
2841 </message>
2842 <message name="SvcMDRegisterResponse">
2843   <part name="body" element="disco:SvcMDRegisterResponse" />
2844 </message>
2845
2846 <message name="SvcMDQuery">
2847   <part name="body" element="disco:SvcMDQuery" />
2848 </message>
2849 <message name="SvcMDQueryResponse">
2850   <part name="body" element="disco:SvcMDQueryResponse" />
2851 </message>
2852
2853 <message name="SvcMDReplace">
2854   <part name="body" element="disco:SvcMDReplace" />
2855 </message>
2856 <message name="SvcMDReplaceResponse">
2857   <part name="body" element="disco:SvcMDReplaceResponse" />
2858 </message>
2859
2860 <message name="SvcMDDelete">
2861   <part name="body" element="disco:SvcMDDelete" />
2862 </message>
2863 <message name="SvcMDDeleteResponse">
2864   <part name="body" element="disco:SvcMDDeleteResponse" />
2865 </message>
2866
2867
2868 <portType name="DiscoveryPort">
2869
2870   <operation name="DiscoveryQuery">
2871     <input message="tns:Query"
2872       wsaw:Action="urn:liberty:disco:2005-11:Query" />
2873     <output message="tns:QueryResponse"
2874       wsaw:Action="urn:liberty:disco:2005-11:QueryResponse" />
2875   </operation>
2876
2877   <operation name="MDAssociationAdd">
2878     <input message="tns:SvcMDAssociationAdd"
2879       wsaw:Action="urn:liberty:disco:2005-11:SvcMDAssociationAdd" />
2880     <output message="tns:SvcMDAssociationAddResponse"
2881       wsaw:Action="urn:liberty:disco:2005-11:SvcMDAssociationAddResponse" />
2882   </operation>
2883
2884   <operation name="MDAssociationQuery">
2885     <input message="tns:SvcMDAssociationQuery"
2886       wsaw:Action="urn:liberty:disco:2005-11:SvcMDAssociationQuery" />
2887     <output message="tns:SvcMDAssociationQueryResponse"
2888       wsaw:Action="urn:liberty:disco:2005-11:SvcMDAssociationQueryResponse" />
2889   </operation>
2890
2891   <operation name="MDAssociationDelete">
2892     <input message="tns:SvcMDAssociationDelete"
2893       wsaw:Action="urn:liberty:disco:2005-11:SvcMDAssociationDelete" />
2894     <output message="tns:SvcMDAssociationDeleteResponse"
2895       wsaw:Action="urn:liberty:disco:2005-11:SvcMDAssociationDeleteResponse" />
2896   </operation>
2897
2898   <operation name="MetadataRegister">
2899     <input message="tns:SvcMDRegister"
2900       wsaw:Action="urn:liberty:disco:2005-11:SvcMDRegister" />
2901     <output message="tns:SvcMDRegisterResponse"
2902       wsaw:Action="urn:liberty:disco:2005-11:SvcMDRegisterResponse" />
2903   </operation>
2904
2905   <operation name="MetadataQuery">

```

```
2906     <input message="tns:SvcMDQuery"
2907         wsaw:Action="urn:liberty:disco:2005-11:SvcMDQuery" />
2908     <output message="tns:SvcMDQueryResponse"
2909         wsaw:Action="urn:liberty:disco:2005-11:SvcMDQueryResponse" />
2910 </operation>
2911
2912 <operation name="MetadataReplace">
2913     <input message="tns:SvcMDReplace"
2914         wsaw:Action="urn:liberty:disco:2005-11:SvcMDReplace" />
2915     <output message="tns:SvcMDReplaceResponse"
2916         wsaw:Action="urn:liberty:disco:2005-11:SvcMDReplaceResponse" />
2917 </operation>
2918
2919 <operation name="MetadataDelete">
2920     <input message="tns:SvcMDDelete"
2921         wsaw:Action="urn:liberty:disco:2005-11:SvcMDDelete" />
2922     <output message="tns:SvcMDDeleteResponse"
2923         wsaw:Action="urn:liberty:disco:2005-11:SvcMDDeleteResponse" />
2924 </operation>
2925
2926
2927 </portType>
2928
2929 <!--
2930 An example of a binding and service that can be used with this
2931 abstract service description is provided below.
2932 -->
2933
2934 <binding name="DiscoveryBinding" type="tns:DiscoveryPort">
2935
2936     <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
2937
2938     <operation name="DiscoveryQuery">
2939         <soap:operation soapAction="urn:liberty:disco:2005-11:Query" />
2940         <input> <soap:body use="literal"/> </input>
2941         <output> <soap:body use="literal"/> </output>
2942     </operation>
2943
2944     <operation name="MDAssociationAdd">
2945         <soap:operation
2946             soapAction="urn:liberty:disco:2005-11:SvcMDAssociationAdd" />
2947         <input> <soap:body use="literal"/> </input>
2948         <output> <soap:body use="literal"/> </output>
2949     </operation>
2950
2951     <operation name="MDAssociationQuery">
2952         <soap:operation
2953             soapAction="urn:liberty:disco:2005-11:SvcMDAssociationQuery" />
2954         <input> <soap:body use="literal"/> </input>
2955         <output> <soap:body use="literal"/> </output>
2956     </operation>
2957
2958     <operation name="MDAssociationDelete">
2959         <soap:operation
2960             soapAction="urn:liberty:disco:2005-11:SvcMDAssociationDelete" />
2961         <input> <soap:body use="literal"/> </input>
2962         <output> <soap:body use="literal"/> </output>
2963     </operation>
2964
2965     <operation name="MetadataRegister">
2966         <soap:operation soapAction="urn:liberty:disco:2005-11:SvcMDRegister" />
2967         <input> <soap:body use="literal"/> </input>
2968         <output> <soap:body use="literal"/> </output>
2969     </operation>
2970
2971     <operation name="MetadataQuery">
```

```
2973     <soap:operation soapAction="urn:liberty:disco:2005-11:SvcMDQuery" />
2974     <input> <soap:body use="literal" /> </input>
2975     <output> <soap:body use="literal" /> </output>
2976 </operation>
2977
2978 <operation name="MetadataReplace">
2979     <soap:operation soapAction="urn:liberty:disco:2005-11:SvcMDReplace" />
2980     <input> <soap:body use="literal" /> </input>
2981     <output> <soap:body use="literal" /> </output>
2982 </operation>
2983
2984 <operation name="MetadataDelete">
2985     <soap:operation soapAction="urn:liberty:disco:2005-11:SvcMDDelete" />
2986     <input> <soap:body use="literal" /> </input>
2987     <output> <soap:body use="literal" /> </output>
2988 </operation>
2989
2990 </binding>
2991
2992 <service name="DiscoveryService">
2993
2994     <port name="DiscoveryPort" binding="tns:DiscoveryBinding">
2995
2996         <!-- Modify with the REAL SOAP endpoint -->
2997
2998         <soap:address location="http://example.com/discovery" />
2999
3000     </port>
3001
3002 </service>
3003
3004 </definitions>
3005
3006
3007
```