



Liberty ID-WSF Discovery Service Specification

Version: 1.2

Editors:

Jonathan Sergent, Sun Microsystems, Inc.

Contributors:

Robert Aarts, Nokia Corporation
John Beatty, Sun Microsystems, Inc.
Conor Cahill, AOL Time Warner, Inc.
Gary Ellison, Sun Microsystems, Inc.
Jukka Kainulainen, Nokia Corporation
John Kemp, IEEE-ISTO
Paul Madsen, Entrust, Inc.
Greg Whitehead, Trustgenix, Inc.
Emily Xu, Sun Microsystems, Inc.

Abstract:

Specification from the Liberty Alliance Project Identity Web Services Framework for describing and discovering identity services.

Filename: liberty-idwsf-disco-svc-v1.2.pdf

1 **Notice**

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

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

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

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

31 Contents

32	1. Introduction	4
33	2. Common Types	6
34	3. Service Instance Description	9
35	4. Resource Offering Description	12
36	5. Discovery Service	14
37	6. SAML AttributeDesignator for Discovery ResourceOffering	23
38	7. Option Value for Response Authentication	24
39	8. Including keys in ModifyResponse	25
40	9. Version 1.2 Core XSD	26
41	10. Version 1.2 Extension XSD	29
42	11. WSDL	30
43	References	32

44 1. Introduction

45 This specification defines a framework for describing and discovering identity services. A conceptual model with
46 terminology is first provided to set the context for the rest of the specification.

47 1.1. Conceptual Model and Terminology

48 An *identity service* is an abstract notion of a web service that acts upon some resource to either retrieve information
49 about an identity, update information about an identity, or perform some action for the benefit of some identity.

50 There are different types of identity services, each of which has a unique *service type*, identified by a URI. This
51 service type identifier maps to exactly one *abstract WSDL* definition of a service, which contains the `wsdl:types`,
52 `wsdl:message`, and `wsdl:portType` elements of a WSDL 1.1 description. An example of a service type is a
53 "calendar service," which could be identified by a URI such as `urn:example:services:calendar`.

54 A service instance is the physical instantiation of a particular type of identity service. A service instance maps to a
55 *concrete WSDL* document (which includes at least the `wsdl:binding`, `wsdl:service`, and `wsdl:port` elements)
56 that contains the *protocol endpoint* and additional information necessary for a client to communicate with the particular
57 service instance (e.g., this information may include security policy information). Each service instance is hosted by
58 some provider, which is identified by a URI. An example of a service instance is a SOAP-over-HTTP endpoint offering
59 a calendar service.

60 A service instance exposes a protocol interface to a set of resources. A *resource* in this specification is either data
61 related to some identity or a service acting for the benefit of some identity. An example of a resource is a calendar
62 containing appointments for a particular identity. When a client sends a request message to a service instance, it
63 includes the resource identifier (i.e., a URI) for the resource it wishes the service instance to act upon.

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

67 The discovery service defined here is not intended to be exclusive. Some identity services meeting the conceptual
68 model may be exposed via other discovery mechanisms. For example, [LibertyPAOS] defines an equivalent discovery
69 mechanism.

70 1.2. Scope

71 This specification contains:

- 72 • Schemas for service instance enumeration and resource offering description.
- 73 • Specification of a discovery service that facilitates discovery and invocation of resource offerings.
- 74 • A SAML (see [SAMLCore]) attribute designator so that a resource offering for the discovery service itself can be
75 conveyed via SAML assertions.

76 **1.3. Notation and Conventions**

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

79 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT",
80 "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

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

84 The following namespaces are referred to in this document:

- 85 • The prefix disco: stands for the Discovery Service namespace. This namespace is the default for instance
86 fragments, type names, and element names in this document. In schema listings, this is the default namespace
87 and no prefix is shown.
- 88 • The prefix ds11: stands for the Discovery Service 1.1 extensions namespace. This namespace is used for new
89 elements that were added in version 1.1 of this specification for use in Extension elements from the base schema.
- 90 • The prefix wsdl: stands for the primary WSDL namespace (<http://schemas.xmlsoap.org/wsdl/>).
- 91 • The prefix wsdlsoap: stands for the namespace of the WSDL-SOAP binding (<http://schemas.xmlsoap.org/wsdl/soap/>).
- 92 • The prefix xs: stands for the W3C XML schema namespace (<http://www.w3.org/2001/XMLSchema>).
- 93 • The prefix xsi: stands for the W3C XML schema instance namespace ([http://www.w3.org/2001/XMLSchema-](http://www.w3.org/2001/XMLSchema-instance)
94 instance).

95 2. Common Types

96 Several XML Schema ComplexType and element declarations are used throughout this specification.

97 2.1. ServiceType

98 The `ServiceType` element is used to identify a service type. This URI needs be constant across all implementations
99 of a service to enable interoperability. Therefore, it is RECOMMENDED that this URI be the same as the
100 targetNamespace URI of the abstract WSDL description for the service.

```
101 <xs:element name="ServiceType" type="xs:anyURI"/>  
102
```

103 Some example of possible `ServiceType` URIs:

```
104 urn:liberty:disco:2003-08  
105 urn:liberty:id-sis-pp:2003-08  
106 http://example.com/my-service-wsdl-ns  
107 http://example.com/wsdl/my-service.wsdl  
108  
109
```

110 2.2. ResourceID

111 The `ResourceID` element contains a URI used to identify a particular resource. Resource identifiers can be registered
112 with the discovery service. Queries on the discovery service return the resource identifiers of matching resources
113 (along with a service instance description that describes how to access the resource). The format of resource IDs will
114 vary from service instance to service instance. Clients should never need to construct resource IDs; clients will obtain
115 them from the discovery service.

116 The following constraints are imposed on the acceptable values for `ResourceID`:

- 117 • It MUST not be a relative URI (see [\[RFC2396\]](#).)
- 118 • The resource URI SHOULD contain a domain name which is owned by the provider hosting the resource.
- 119 • If a resource is exposed via multiple `ResourceOffering` elements (for example, if there are multiple service
120 instances used as a front end to the same data), all such `ResourceOffering` elements SHOULD have the same
121 `ResourceID` value.

122 One special value of ResourceID is defined: "urn:liberty:isf:implied-resource". This resource identifier is to be used
 123 in circumstances where the resource in question is implicitly identified because there is only one resource that could
 124 be operated upon at the service instance being contacted. In some circumstances, the use of this resource identifier can
 125 eliminate the need for contacting the discovery service to access the resource. (See [LibertyPAOS] for some example
 126 uses.)

127 The ResourceID element is optional (minOccurs="0"). If the ResourceID element is not present, the message should
 128 be processed the same way as if the ResourceID element was present and its value was "urn:liberty:isf:implied-
 129 resource". Note that in version 1.0 of this specification ResourceID was not optional and so older implementations
 130 may not process messages which have omitted ResourceID instead of including it with this value.

```

131
132 <xs:complexType name="ResourceIDType">
133   <xs:simpleContent>
134     <xs:extension base="xs:anyURI">
135       <xs:attribute name="id" type="xs:ID" use="optional"/>
136     </xs:extension>
137   </xs:simpleContent>
138 </xs:complexType>
139
140 <xs:complexType name="EncryptedResourceIDType">
141   <xs:sequence>
142     <xs:element ref="xenc:EncryptedData"/>
143     <xs:element ref="xenc:EncryptedKey"/>
144   </xs:sequence>
145 </xs:complexType>
146
147 <xs:element name="ResourceID" type="ResourceIDType"/>
148 <xs:element name="EncryptedResourceID" type="EncryptedResourceIDType"/>
149
150 <!--
151   if not present, equivalent to
152   <ResourceID>urn:liberty:isf:implied-resource</ResourceID>
153   (see specification text for details)
154   -->
155
156 <xs:group name="ResourceIDGroup">
157   <xs:sequence>
158     <xs:choice minOccurs="0" maxOccurs="1">
159       <xs:element ref="ResourceID"/>
160       <xs:element ref="EncryptedResourceID"/>
161     </xs:choice>
162   </xs:sequence>
163 </xs:group>
164
165
  
```

166 Some examples of possible resource IDs are:

```

167
168 http://example.com/disco/d0CQF8e1JTDLmzEo
169 http://profile-provider.com/profiles/14m0B82k15csaUxs
170 urn:liberty:isf:implied-resource
171
  
```

172 2.3. EncryptedResourceID

173 The schema also defines an element EncryptedResourceID for the transport of obfuscated resource identifiers. The
 174 EncryptedResourceID contains a ResourceID that has been encrypted using XML encryption, and an encrypted
 175 key that was used to encrypt the ResourceID. Use of EncryptedResourceID is often necessary for privacy reasons.

176 If a non-predictable nonce is used for the EncryptedKey, each discovery service client will get a different identifier.
 177 This will prevent discovery service clients from colluding with other discovery service clients about the identity of the

178 Principal on the basis of the ResourceID for one of the Principal's services. (Note that strictly speaking, the key does
179 not need to be a nonce, rather just unique for each client.)

- 180 • The `xenc:EncryptedData`, when successfully decrypted, MUST contain a `ResourceID` element.
- 181 • The `xenc:EncryptedData` MUST be encrypted using the `xenc:EncryptedKey` that is present in the same
182 element.
- 183 • The key used to encrypt the key in `xenc:EncryptedKey` MUST be the public key of the provider that hosts the
184 resource.
- 185 • [[LibertyMetadata](#)] defines a mechanism to retrieve the provider's public key if the Provider ID is known; the
186 Provider ID is available in the resource's service instance description (see [Section 3](#)). [[LibertyMetadata](#)] can also
187 be used to ensure that the encryption algorithm being used is supported by both parties.
- 188 • The `xenc:EncryptedKey` MUST exhibit nonce-like semantics, so that it does not circumvent the privacy require-
189 ment that the `EncryptedResourceID` mechanism is intended to address. However, an `xenc:EncryptedKey`
190 MAY be used multiple times with the same client, so long as the same encrypted identifier is never delivered to
191 more than one party.
- 192 • Because the `xenc:EncryptedKey` element is used for key transport, the `xenc:Algorithm` attribute of the
193 `xenc:EncryptionMethod` element must be one of the URIs designated for key transport as defined in [[xmlenc-
194 core](#)].

195 2.4. Status Codes

196 The following status code QNames are defined in the Discovery Service namespace:

- 197 • *OK*: message processing succeeded
- 198 • *Failed*: general failure code
- 199 • *RemoveEntry*: an entry being removed does not exist
- 200 • *Forbidden*: the request was denied based on policy
- 201 • *NoResults*: no results could be found
- 202 • *Directive*: a directive was supplied in `InsertEntry` that was not understood or not supported

203 These QNames are expected to appear in the "code" attribute of `Status` elements used in Discovery Service protocol
204 messages. Specific uses for the status codes are defined in the processing rules for individual messages. The "ref"
205 attribute on the `Status` element is not used in this specification, so it MUST not appear on `Status` elements in
206 Discovery Service protocol messages. The contents of the "comment" attribute are not defined by this specification,
207 but it may be used for additional descriptive text intended for human consumption (for example, to carry information
208 that will aid debugging).

209 3. Service Instance Description

210 A service instance is a running web service at a distinct protocol endpoint. Information about service instances
211 needs to be communicated in various contexts. For example, the Discovery Service defined in this specification is
212 an identity service which provides an enumeration of resource offerings (each of which includes a service instance
213 description). This specification defines a schema for service instance description that can be used in a variety of
214 protocol interactions. Note that this description schema does not replace WSDL; rather, it is to be used in conjunction
215 with WSDL. In essence, it wraps WSDL with additional information and allows for enumeration of various service
216 instances described with WSDL.

```
217 <xs:complexType name="DescriptionType">
218   <xs:sequence>
219     <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="1" maxOccurs="unbounded"/>
220     <xs:element name="CredentialRef" type="xs:IDREF" minOccurs="0" maxOccurs="unbounded"/>
221     <xs:choice>
222       <xs:group ref="WsdRef"/>
223       <xs:group ref="BriefSoapHttpDescription"/>
224     </xs:choice>
225   </xs:sequence>
226   <xs:attribute name="id" type="xs:ID" use="optional"/>
227 </xs:complexType>
228
229 <xs:complexType name="ServiceInstanceType">
230   <xs:sequence>
231     <xs:element ref="ServiceType"/>
232     <xs:element name="ProviderID" type="md:entityIDType"/>
233     <xs:element name="Description" type="DescriptionType" minOccurs="1" maxOccurs="unbounded"/>
234   </xs:sequence>
235 </xs:complexType>
236
```

237 *ServiceType* contains the URI defining the type of service this service instance implements. Because *ServiceType*
238 maps one-to-one with abstract WSDL (see [Section 1.1](#)), the *ServiceType* element enables a potential requester to
239 know the abstract WSDL implemented by the service instance.

240 *ProviderID* contains the URI of the provider of the service instance. This is useful for resolving metadata (e.g.,
241 trust metadata) necessary for invoking the service instance. Note that a single physical provider may have multiple
242 provider IDs. At registration time, the provider can ensure that the *ServiceInstance* element contains the
243 appropriate *ProviderID* element given the context (for example, based on the circle of trust that the discovery service
244 instance is in). *ProviderID* MUST correspond with the constraints for the *entityIDType* data type as specified in
245 [\[LibertyMetadata\]](#).

246 The *Description* element contains the necessary concrete service description (see [Section 1.1](#)) is included in the
247 *Description* via a choice between a *WsdRef* group (which references an external concrete WSDL resource) or a
248 *BriefSoapHttpDescription* group (which provides inline the information necessary to invoke basic SOAP-over-
249 HTTP-based service instances without using WSDL). The *BriefSoapHttpDescription* group MUST be chosen if and
250 only if it is possible to logically compute the concrete WSDL from the abstract WSDL (as referred to by the service
251 type URI) and the information contained in the *BriefSoapHttpDescription*. (See [Section 3.2](#) below.) Otherwise, the
252 *WsdRef* group must be used. The purpose of having the *BriefSoapHttpDescription* choice is to ease the burden
253 of *ServiceInstance* processors from having to retrieve and parse WSDL in common cases.

254 The *Description* contains an optional *id* attribute. The *id* attribute need not be specified unless it is necessary to refer
255 to the description elsewhere. In particular, it may be needed to refer to a particular *Description* from a directive
256 (see [Section 5.2.1.1](#)) in the *Modify* request.

257 The *Description* contains one or more *SecurityMechID* URIs. These URIs identify the security mechanisms sup-
258 ported by the service instance. Other specifications, such as [\[LibertySecMech\]](#) define the actual security mechanisms
259 along with their identifiers. These security mechanisms refer to the way a web service client authenticates to the web
260 service provider. The service instance description SHOULD list of all of the security mechanisms that the service

261 instance supports. The client SHOULD pick the first mechanism (in the order listed) that it supports; the description
 262 SHOULD list them in order of preference, to avoid situations where the client fails to gain access to the service because
 263 it picked the wrong security mechanism.

264 The Description may also contain CredentialRef elements. These elements refer to security credentials (which
 265 are contained elsewhere; see Section 5.1.2) which the client may need in order to invoke the service using the given
 266 security mechanism.

267 Multiple Description elements are allowed in case the WSDL binding varies between security mechanisms. (For
 268 example, many web servers will require a different endpoint URI to be used for SOAP/HTTP clients authenticating
 269 using client TLS certificates than for clients which do not authenticate using client TLS certificates.) Any single
 270 SecurityMechID URI MUST NOT appear in more than one Description in a particular service instance description.
 271 In other words, each service instance may only specify one WSDL binding per supported security mechanism. The
 272 descriptions SHOULD appear in the order of the service's preference, and the client SHOULD use the first description
 273 in the list that it is capable of accessing.

274 3.1. WsdRef Group

275 wsdlURI provides a URI to a WSDL resource containing the service description. This must be concrete WSDL
 276 (see Section 1.1), not abstract WSDL (see Section 1.1). The ServiceNameRef references a wsdl:service
 277 element within the WSDL resource such that ServiceNameRef is equal to the wsdl:name attribute of the proper
 278 wsdl:service element. The specified ServiceNameRef MUST refer to a wsdl:service that implements
 279 bindings to the portTypes defined by the ServiceType URI. The processor of the ServiceInstance chooses the
 280 proper wsdl:service element by this means. The specified WSDL resource MUST contain a wsdl:service with
 281 a wsdl:name attribute equal to the specified ServiceNameRef.

```
282 <xs:group name="WsdRef">
283   <xs:sequence>
284     <xs:element name="WsdURI" type="xs:anyURI"/>
285     <xs:element name="ServiceNameRef" type="xs:QName"/>
286   </xs:sequence>
287 </xs:group>
288
```

289 An example service instance description using a WSDL reference is:

```
290
291 <ServiceInstance xmlns="urn:liberty:disco:2003-08">
292   <ServiceType>http://example.com/wsdl/my-service.wsdl</ServiceType>
293   <ProviderID>http://example.com/</ProviderID>
294   <Description>
295     <SecurityMechID>urn:liberty:security:2003-08:TLS:null</SecurityMechID>
296     <SecurityMechID>urn:liberty:security:2005-02:TLS:X509</SecurityMechID>
297     <WsdURI>http://example.com/wsdl/my-service.wsdl</WsdURI>
298     <ServiceNameRef xmlns:m="http://example.com/wsdl/my-service.wsdl">
299 m:MyService</ServiceNameRef>
300   </Description>
301 </ServiceInstance>
302
```

303 3.2. BriefSoapHttpDescription Group

304 The information contained in this group is sufficient for making invocations for some service instances. In other
 305 words, the information contained in this group together with the abstract WSDL specified by the ServiceType URI is
 306 sufficient to logically compute concrete WSDL with the rule set specified below. If the service instance exposes an
 307 endpoint that is different from the logically generated concrete WSDL, the WsdRef group MUST be used instead.

308 Endpoint contains the URI of the SOAP-over-HTTP endpoint. The URI scheme MUST be "http" or "https".
309 SoapAction contains the equivalent of the wsdlsoap:soapAction attribute of the wsdlsoap:operation element
310 in WSDL-based description.

311 Use of this group implies wsdl:binding and wsdl:service elements according to the following rules (i.e., the
312 concrete WSDL can be logically computed given the abstract WSDL and the BriefSoapHttpDescription group):

313 • The wsdl:binding contains a wsdlsoap:binding element. This specifies that the SOAP binding for WSDL is
314 being used.

315 • The style attribute of the wsdlsoap:binding element is *document*.

316 • The transport attribute of the wsdlsoap:binding element is *http://schemas.xmlsoap.org/soap/http*.

317 • The abstract WSDL corresponding to the ServiceType MUST contain a single portType element. The
318 wsdl:binding element provides bindings for the operations specified in this wsdl:portType. Each operation
319 binding includes an input element and an output element, each containing a single wsdlsoap:body element. The
320 use attribute of the wsdlsoap:body elements is "literal".

321 • The soapAction attribute of wsdlsoap:operation is equal to SoapAction if provided, otherwise it is omitted.

322 • The location attribute of wsdlsoap:address is equal to Endpoint.

323 • All other optional elements and attributes are not specified and thus default to the SOAP binding of WSDL.

```
324 <xs:group name="BriefSoapHttpDescription">
325   <xs:sequence>
326     <xs:element name="Endpoint" type="xs:anyURI" />
327     <xs:element name="SoapAction" type="xs:anyURI" minOccurs="0" />
328   </xs:sequence>
329 </xs:group>
330
```

331 An example ServiceInstance using the BriefSoapHttpDescription is:

```
332
333 <ServiceInstance xmlns="urn:liberty:disco:2003-08">
334   <ServiceType>urn:liberty:id-sis-pp:2003-08</ServiceType>
335   <ProviderID>http://profile-provider.com/</ProviderID>
336   <Description>
337     <SecurityMechID>urn:liberty:security:2003-08:TLS:null</SecurityMechID>
338     <SecurityMechID>urn:liberty:security:2005-02:TLS:X509</SecurityMechID>
339     <Endpoint>https://soap.profile-provider.com/soap/</Endpoint>
340   </Description>
341   <Description>
342     <SecurityMechID>urn:liberty:security:2003-08:ClientTLS:null</SecurityMechID>
343     <Endpoint>https://soap-auth.profile-provider.com/soap/</Endpoint>
344   </Description>
345 </ServiceInstance>
346
```

347 4. Resource Offering Description

348 A *resource offering* is the association of a resource and a service instance. This association is necessary as there is
349 a many-to-many relationship between resources and service instances. Typically, a single service instance will serve
350 many resources. For example, a personal profile service provider would typically serve up many profiles behind
351 a single service instance, as having a separate protocol endpoint for each profile would be impractical. Thus, a
352 `ResourceOffering` element is defined to associate a resource with a service instance that provides access to that
353 resource.

```
354 <xs:element name="ResourceOffering" type="ResourceOfferingType"/>
355 <xs:complexType name="ResourceOfferingType">
356   <xs:sequence>
357     <xs:group ref="ResourceIDGroup"/>
358     <xs:element name="ServiceInstance" type="ServiceInstanceType"/>
359     <xs:element ref="Options" minOccurs="0"/>
360     <xs:element name="Abstract" type="xs:string" minOccurs="0"/>
361   </xs:sequence>
362   <xs:attribute name="entryID" type="IDType" use="optional"/>
363 </xs:complexType>
364
```

365 The `ResourceID` element provides the URI for the resource that the requester can use in a request to the described
366 `ServiceInstance`. The `EncryptedResourceID` MAY appear in place of the `ResourceID`. Therefore, service
367 instances need to handle the case where the client presents an `EncryptedResourceID` instead of a `ResourceID`.
368 The `ServiceInstance` contains the description of the service instance that is providing access to the resource.

369 The `Abstract` element contains a human-readable description of the resource offering.

370 The `Options` element expresses the "options" available for the resource offering, which provides hints to a potential
371 requester whether certain data or operations may be available with a particular resource offering. For example, an
372 option may be provided stating that home contact information is available. If no `Options` element is present, it means
373 that the service instance does not advertise whether any options are available (for example, it may be a simple service
374 that is not capable of updating its entry in the discovery service when the available options change, so it avoids listing
375 them at all.) If the `Options` element is present, but it is empty, it means that the service instance explicitly advertises
376 that none of the options are available.

377 The `Options` element contains zero or more `Option` elements, each of which contain a URI identifying the particular
378 option. The set of possible URIs for an `Option` element should be defined by the service type (e.g., a person profile
379 service specification standardize a set of options). However, one common `Option` flag related to security and common
380 to ID-WSF services is defined in [Section 7](#).

```
381 <xs:element name="Options" type="OptionsType"/>
382 <xs:complexType name="OptionsType">
383   <xs:sequence>
384     <xs:element name="Option" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
385   </xs:sequence>
386 </xs:complexType>
387
```

388 An example `ResourceOffering` is:

```
389
390 <ResourceOffering xmlns="urn:liberty:disco:2003-08">
391   <ResourceID>http://profile-provider.com/profiles/14m0B82k15csaUxs</ResourceID>
392   <ServiceInstance xmlns="urn:liberty:disco:2003-08">
393     <ServiceType>urn:liberty:id-sis-pp:2003-08</ServiceType>
394     <ProviderID>http://profile-provider.com/</ProviderID>
395     <Description>
396       <SecurityMechID>urn:liberty:security:2003-08:TLS:null</SecurityMechID>
397       <SecurityMechID>urn:liberty:security:2005-02:TLS:X509</SecurityMechID>
398     <Endpoint>https://soap.profile-provider.com/soap</Endpoint>

```

```
399     </Description>
400     <Description id="saml-profile-description">
401         <SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</SecurityMechID>
402         <Endpoint>https://soap.profile-provider.com/soap/</Endpoint>
403     </Description>
404     <Description>
405         <SecurityMechID>urn:liberty:security:2003-08:ClientTLS:null</SecurityMechID>
406         <Endpoint>https://soap-auth.profile-provider.com/soap/</Endpoint>
407     </Description>
408 </ServiceInstance>
409 <Options>
410     <Option>urn:liberty:id-sis-pp</Option>
411     <Option>urn:liberty:id-sis-pp:cn</Option>
412     <Option>urn:liberty:id-sis-pp:can</Option>
413     <Option>urn:liberty:id-sis-pp:can:cn</Option>
414 </Options>
415 <Abstract>
416     This is a personal profile containing common name information.
417 </Abstract>
418 </ResourceOffering>
419
```

420 5. Discovery Service

421 The Discovery Service is an identity service that allows requesters to discover resource offerings. Thus, the Discovery
422 Service is essentially a web service interface for "discovery resources", each of which can be viewed as a registry of
423 resource offerings. Entities can place resource offerings in a discovery resource, and this will allow other entities to
424 discover these resource offerings. A common use case is that a user places his or her personal profile, calendar, and so
425 on a discovery resource so that these other resources can be discovered by other entities.

426 Note that the Discovery Service itself is an identity service like any other. Also note that other discovery mechanisms
427 are possible; this specification formalizes one particular mechanism that can be used in a wide variety of applications.

428 The Discovery Service is meant to be used in conjunction with the lower-layer ID-WSF specifications. For example,
429 security mechanisms are not specified here, because they are defined in [\[LibertySecMech\]](#). At the same time, the
430 Discovery Service is specified such that it can be used with other security mechanisms, not yet defined.

431 The Discovery Service is defined by the abstract WSDL in this document (see [\[WSDLv1.1\]](#)), which defines two
432 `wsdl:operation` definitions. The *DiscoveryLookup* operation returns an enumeration of `ResourceOffering`
433 elements given search criteria. The *DiscoveryUpdate* operation enables maintenance of a discovery resource,
434 accommodating inserts and removals of resource offerings.

435 The discovery service MAY return `EncryptedResourceID` elements instead of the `ResourceID` in the
436 `QueryResponse` (except in the case of the `EncryptResourceID` directive (see [Section 5.2.1.1](#)). Note that it
437 may decide to encrypt the identifier based on policy, or it may decide not to do so because of implementation
438 limitations.

439 However, some identity services may not wish to register with discovery services that do not encrypt resource IDs for
440 whatever reason. The `EncryptResourceID` directive is provided to allow the identity service to require the discovery
441 service to either always encrypt the resource ID or refuse the registration. The intent is that, in general, the discovery
442 service that is describing the resource makes the decision, not the service instance hosting the resource in question, so
443 even if this directive is not present, the discovery service may still decide to encrypt the resource ID.

444 Therefore, identity services which are registered with the discovery service MUST be prepared to accept
445 `EncryptedResourceID` from their clients instead of `ResourceID`. (Identity services which do not wish to
446 expose the `ResourceID` to the discovery service unless it will encrypt it for privacy reasons may do so by using the
447 `EncryptResourceID` directive (see [Section 5.2.1.1](#)).

448 To enforce access control policies, security credentials may need to be presented by the client. While the definition
449 of these security credentials is outside the scope of this specification, it is common in many cases for the same entity
450 that is hosting the discovery service to also be the entity that generates the credentials necessary to access the service.
451 To avoid extra network round-trips, arrangements are made here so that credentials may be provided as part of the
452 discovery service lookup response.

453 The Discovery Service service type URI is *urn:liberty:disco:2003-08*.

454 5.1. Operation: DiscoveryLookup

455 The *DiscoveryLookup* operation enables a requester to obtain an enumeration of `ResourceOffering` elements. The
456 requester sends a `Query` and receives a `QueryResponse` in return. Also, because a provider hosting a Discovery
457 Service may also be playing other roles for an identity (such as a *Policy Decision Point* or an *Authentication*
458 *Authority*), the *DiscoveryLookup* operation can also function as a security credential service, providing the requester
459 with an efficient means of obtaining credentials that may be necessary to invoke service instances described in the
460 `QueryResponse`.

461 5.1.1. Query

462 The `Query` minimally contains the `ResourceID` element, which describes which discovery resource is being
 463 requested. A request with only the `ResourceID` element indicates the requester is requesting all available resource
 464 offerings. The set of results is dependant upon local access control policy of the discovery resource.

465 The request can be qualified with a set of `RequestedServiceType` elements, which enables the requester to specify
 466 that all resource offerings returned must be offered via a service instance complying with one of the specified service
 467 types. For each `ServiceType` specified, the requester can also specify `Options` (see [Section 4](#)) the returned resource
 468 offering should support. Note that returned resource offerings are not guaranteed to support the requester-specified
 469 options, as some discovery service instances and/or resource offering registrations may not support options registration.

470 Requesters **SHOULD** construct a `Query` to be as qualified as possible, as the discovery service provider may have to
 471 perform significant work for each result in the response, especially if credentials are going to be generated.

```

472 <xs:element name="Query" type="QueryType"/>
473 <xs:complexType name="QueryType">
474   <xs:sequence>
475     <xs:group ref="ResourceIDGroup"/>
476     <xs:element name="RequestedServiceType" minOccurs="0" maxOccurs="unbounded">
477       <xs:complexType>
478         <xs:sequence>
479           <xs:element ref="ServiceType"/>
480           <xs:element ref="Options" minOccurs="0"/>
481         </xs:sequence>
482       </xs:complexType>
483     </xs:element>
484   </xs:sequence>
485   <xs:attribute name="id" type="xs:ID" use="optional"/>
486 </xs:complexType>
487
```

488 An example SOAP message containing a `Query` follows:

```

489 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
490   <soap:Header>
491     <Correlation id="e3Bvm4gNw_IrVJiEpU50"
492       soap:mustUnderstand="1"
493       soap:actor="http://schemas.xmlsoap.org/soap/actor/next"
494       messageId="NK44V79NdfPaE5jCwlk_"
495       xmlns="urn:liberty:sb:2003-08"/>
496   </soap:Header>
497   <soap:Body>
498     <Query xmlns="urn:liberty:disco:2003-08">
499       <ResourceID>http://example.com/disco/d0CQF8elJTDLmzEo</disco:ResourceID>
500       <RequestedServiceType>
501         <ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>
502       </RequestedServiceType>
503     </Query>
504   </soap:Body>
505 </soap:Envelope>
506
507
```

508 5.1.2. QueryResponse

509 The `QueryResponse` element contains the set of results of the query as a set of `ResourceOffering` elements. Each
 510 `ResourceOffering` element **MUST** contain an `entryID` attribute, to be used in the `Modify` message. This `entryID`
 511 **MUST** be unique across all entries in the discovery resource being queried.

512 A set of credentials may be provided within the `Credentials` element in the response. All credentials **MUST**
 513 have an attribute of type `ID`, so that they can be referred to via an `IDREF`. The relevant credentials for each
 514 `ResourceOffering` are referenced with a set of zero or more `CredentialRef` elements contained in the service
 515 instance description (see [Section 3](#)).

516 A status code is also included in the response.

```

517 <xs:element name="QueryResponse" type="QueryResponseType"/>
518 <xs:complexType name="QueryResponseType">
519   <xs:sequence>
520     <xs:element ref="Status"/>
521     <xs:element ref="ResourceOffering" minOccurs="0" maxOccurs="unbounded"/>
522     <xs:element name="Credentials" minOccurs="0">
523       <xs:complexType>
524         <xs:sequence>
525           <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
526         </xs:sequence>
527       </xs:complexType>
528     </xs:element>
529   </xs:sequence>
530   <xs:attribute name="id" type="xs:ID" use="optional"/>
531 </xs:complexType>
532

```

533 An example SOAP message containing a QueryResponse follows. This example includes a credential and a
 534 ResourceOffering which references the credential. Parts of the credential have been omitted due to size.

```

535 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
536   <soap:Header>
537     <Correlation id="HE-XCr0005_3rXLDqKWM"
538       soap:mustUnderstand="1"
539       soap:actor="http://schemas.xmlsoap.org/soap/actor/next"
540       refToMessageId="NK44V79NdfPaE5jCwlk_"
541       messageId="Xl1-dyCnQzfRW-BD9Ngx"
542       xmlns="urn:liberty:sb:2003-08" />
543   </soap:Header>
544   <soap:Body>
545     <QueryResponse xmlns="urn:liberty:disco:2003-08">
546       <Status code="OK"/>
547       <ResourceOffering entryID="1">
548         <ResourceID>http://example.com/pip/bob</ResourceID>
549         <ServiceInstance>
550           <ServiceType>urn:liberty:id-sis-pp:2003-08</ServiceType>
551           <ProviderID>http://example.com/</ProviderID>
552           <Description>
553             <SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</SecurityMechID>
554             <CredentialRef>2sxJu9g/vvLG9sAN9bKp/8q0NKU=</CredentialRef>
555             <Endpoint>https://soap.example.com:443/soap</Endpoint>
556           </Description>
557         </ServiceInstance>
558         <Abstract>Bob's personal profile</Abstract>
559       </ResourceOffering>
560       <Credentials>
561         <saml:Assertion
562           xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
563           AssertionID="2sxJu9g/vvLG9sAN9bKp/8q0NKU="
564           Issuer="idp.example.com"
565           IssueInstant="2003-09-09T16:58:33.173Z">
566           <saml:AuthenticationStatement
567             AuthenticationMethod="urn:ietf:rfc:2246"
568             AuthenticationInstant="2003-09-09T16:57:30.000Z">
569             <saml:Subject>
570               <saml:NameIdentifier format="urn:liberty:iff:nameid:entityID">
571                 http://serviceprovider.com/
572               </saml:NameIdentifier>
573             </saml:Subject>
574           </saml:AuthenticationStatement>
575           <ResourceAccessStatement
576             xmlns="urn:liberty:sec:2003-08"
577

```



```

580         xmlns:disco="urn:liberty:disco:2003-08">
581         <saml:Subject>
582             <saml:NameIdentifier format="urn:liberty:iff:nameid:entityID">
583                 http://serviceprovider.com/
584             </saml:NameIdentifier>
585             <saml:SubjectConfirmation>
586                 <saml:ConfirmationMethod>
587                     urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
588                 </saml:ConfirmationMethod>
589                 <ds:KeyInfo>
590                     <ds:KeyName>
591                         CN=serviceprovider.com,OU=Services R US,O=Service Nation,...
592                     </ds:KeyName>
593                     <ds:KeyValue>...</ds:KeyValue>
594                 </ds:KeyInfo>
595             </saml:SubjectConfirmation>
596         </saml:Subject>
597
598         <disco:ResourceID>http://example.com/pip/bob</disco:ResourceID>
599         <SessionContext xmlns="urn:liberty:sec:2003-08"
600             AuthenticationInstant=" " AssertionIssueInstant=" ">
601             <SessionSubject>
602                 <saml:NameIdentifier ID="..."
603                     Format="urn:liberty:iff:nameid:encrypted">
604                     HJJWbvqW9E84vJVQkjjLLA6nNvBX7mY00TZhwBdFNDElgscSXZ5Ekw
605                     A23B45C569UkjjLLA6nNvBX7mY00TZhwBdFNDElgscSXZ5Ekw89XR3==
606                 </saml:NameIdentifier>
607             </SessionSubject>
608             <AuthnContext xmlns="http://www.projectliberty
609 .org/schemas/authctx/2002/05">
610                 ...
611             </AuthnContext>
612             <ProviderID>http://serviceprovider.com/</ProviderID>
613         </SessionContext>
614     </ResourceAccessStatement>
615     <ds:Signature>...</ds:Signature>
616 </saml:Assertion>
617 </Credentials>
618 </QueryResponse>
619 </soap:Body>
620 </soap:Envelope>
621

```

622 5.1.3. Processing Rules

623 The discovery service provider returns entries based on the requester's criteria, the policies of the discovery resource,
624 and the contents of the discovery resource. For each RequestedServiceType, the following matching rules MUST
625 be followed in determining the subset of result that will be returned to the requester:

- 626 • If no Options element is present in the ResourceOffering, all entries in the discovery resource with the specified
627 service type match.
- 628 • A ResourceOffering in the discovery resource matches the query if each Option element value in the
629 RequestedServiceType is also present in the ResourceOffering. Note that this means that if there are
630 no Option elements in the RequestedServiceType, the resource matches. (See [Section 4](#).)

631 The discovery service provider SHOULD provide credentials in the response if it knows those credentials are necessary
632 based on the directives provided when the resources being discovered were registered.

633 The discovery service provider MAY order `ResourceOffering` elements as it sees fit. If the discovery service is
634 rank ordering the entries, it MUST use descending rank order. This enables the requester to assume that if the results
635 were ordered, the first result is the most relevant.

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

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

639 • If the top-level status code is *Failed*, the response MAY also contain *Forbidden* or *NoResults* as a second-level
640 status code. *Forbidden* MUST be only be used if the request was denied based on policy such that no future
641 Modify request would put the resource in a state that any results could be returned. *NoResults* MUST only be used
642 if there are no `ResourceOffering` elements in the response, but there might be some `ResourceOffering` elements if
643 an entry matching the criteria was later inserted.

644 The service may not wish to reveal the reason for failure, in which case no second-level status code will appear.
645 Other second-level status codes from the Discovery Service namespace MUST NOT be used. Other second-level
646 status codes from other namespaces MAY appear. Clients MAY ignore status codes from other namespaces if they
647 are not understood.

648 5.2. Operation: DiscoveryUpdate

649 The `DiscoveryUpdate` operation enables a requester to insert new resource offering entries into a discovery resource
650 and remove existing entries from a discovery resource. The `DiscoveryUpdate` allows multiple insertions and removals
651 to be made in a single request. Updates to existing entries are performed by removing an existing entry and inserting
652 a new entry in a single operation.

653 5.2.1. Modify

654 The `Modify` element contains a set of zero or more `InsertEntry` elements, each containing exactly one
655 `ResourceOffering` element, and a set of zero or more `RemoveEntry` elements, each containing an *entryID* at-
656 tribute.

657 `ResourceOffering` elements being inserted MUST NOT contain *entryID* attributes.

658 Note that the `InsertEntry` definition contains an *any* element. This allows the requester to include directives
659 about the `ResourceOffering` being inserted. For example, access control policy for the resource offering could
660 be specified. This specification defines several standard directives that can be used in this placeholder.

```
661 <xs:complexType name="InsertEntryType">
662   <xs:sequence>
663     <xs:element ref="ResourceOffering"/>
664     <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
665   </xs:sequence>
666 </xs:complexType>
667
668 <xs:complexType name="RemoveEntryType">
669   <xs:attribute name="entryID" type="IDReferenceType" use="required"/>
670 </xs:complexType>
671
672 <xs:element name="Modify" type="ModifyType"/>
673 <xs:complexType name="ModifyType">
674   <xs:sequence>
675     <xs:group ref="ResourceIDGroup"/>
676     <xs:element name="InsertEntry" type="InsertEntryType" minOccurs="0" maxOccurs="unbounded"/>
```

```

677     <xs:element name="RemoveEntry" type="RemoveEntryType" minOccurs="0" maxOccurs="unbounded"/>
678 </xs:sequence>
679 <xs:attribute name="id" type="xs:ID" use="optional"/>
680 </xs:complexType>
681

```

682 An example SOAP message containing a Modify follows. This request removes an existing resource (see the
683 QueryResponse example) and replaces it with a different one.

```

684
685 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
686   <soap:Header>
687     <Correlation id="CkXlpwJuNxn9ssDsEt51"
688       soap:mustUnderstand="1"
689       soap:actor="http://schemas.xmlsoap.org/soap/actor/next"
690       messageId="DNtVj92yVliICR47w1QV"
691       xmlns="urn:liberty:sec:2003-08"/>
692   </soap:Header>
693   <soap:Body>
694     <Modify xmlns="urn:liberty:disco:2003-08">
695       <ResourceID>http://example.com/disco/d0CQF8elJTDLmzEo</disco:ResourceID>
696       <InsertEntry>
697         <ResourceOffering xmlns="urn:liberty:disco:2003-08">
698           <ResourceID>http://profile-provider.com/profiles/14m0B82k15csaUxs</Resour
699 ceID>
700           <ServiceInstance xmlns="urn:liberty:disco:2003-08">
701             <ServiceType>urn:liberty:id-sis-pp:2003-08</ServiceType>
702             <ProviderID>http://profile-provider.com/</ProviderID>
703             <Description id="clientTLS">
704               <SecurityMechID>urn:liberty:security:2003-08:Client
705 TLS:null</SecurityMechID>
706               <Endpoint>https://soap-auth.profile-provider.com/soap/</Endpoint>
707             </Description>
708             <Description id="saml">
709               <SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</SecurityMechID>
710               <Endpoint>https://soap.profile-provider.com/soap/</Endpoint>
711             </Description>
712             <Description>
713               <SecurityMechID>urn:liberty:security:2005-02:TLS:X509</SecurityMechID>
714               <SecurityMechID>urn:liberty:security:2003-08:TLS:null</SecurityMechID>
715               <Endpoint>https://soap.profile-provider.com/soap/</Endpoint>
716             </Description>
717             <Description>
718               <SecurityMechID>urn:liberty:security:2003-08:null:null</SecurityMechID>
719               <Endpoint>http://soap.profile-provider.com/soap/</Endpoint>
720             </Description>
721           </ServiceInstance>
722           <Options>
723             <Option>urn:liberty:id-sis-pp</Option>
724             <Option>urn:liberty:id-sis-pp:cn</Option>
725             <Option>urn:liberty:id-sis-pp:can</Option>
726             <Option>urn:liberty:id-sis-pp:can:cn</Option>
727           </Options>
728           <Abstract>
729             This is a personal profile containing common name information.
730           </Abstract>
731         </ResourceOffering>
732         <AuthenticateRequester descriptionIDRefs="saml"/>
733         <AuthorizeRequester descriptionIDRefs="saml clientTLS"/>
734       </InsertEntry>
735       <RemoveEntry entryID="1"/>
736     </Modify>
737   </soap:Body>
738 </soap:Envelope>
739

```

740 5.2.1.1. Directives

741 Six policy-related directives are also defined: `AuthenticateRequester`, `AuthorizeRequester`,
742 `AuthenticateSessionContext`, `EncryptResourceID`, `ds11:SendSingleLogOut`, and `ds11:GenerateBearerToken`.

743 The directives all contain an optional `descriptionIDRefs` attribute. If the `descriptionIDRefs` attribute is not present
744 in a directive element, the directive is to be taken to apply to all `Description` elements provided in the
745 `ResourceOffering`. If the `descriptionIDRefs` attribute is present in a directive element, it MUST contain a list
746 of IDREFs which refer to `Description` elements in the `ResourceOffering` the directive is associated with. If the at-
747 tribute is present, the directive MUST be taken to apply only to those descriptions referred to in the `descriptionIDRefs`
748 list. This may be useful if certain directives are incompatible with certain security mechanisms.

749 If the `AuthenticateRequester` directive is specified for a resource, the discovery service provider SHOULD
750 include a SAML assertion containing an `AuthenticationStatement` in any future `QueryResponse` messages for
751 the resource to enable the client sending the `Query` message to authenticate to the service instance hosting the
752 resource. If the client which sends the `Query` authenticates using a `ProviderID`, the statement SHOULD be
753 a `saml:AuthenticationStatement` as defined in [LibertySecMech]. In this case, the statement's subject is
754 the client's `ProviderID`. If the client which sends the `Query` is a user agent which does not authenticate using a
755 `ProviderID` (that is, if no `Provider` header is present on the request message), the `AuthenticationStatement` included
756 SHOULD be an ID-FF `lib:AuthenticationStatement`. In this case, the statement should identify the principal
757 using the user agent using an ID-FF name identifier. (Note that since provider metadata cannot be accessed in
758 this case, the statement will not be useful with the holder of key confirmation method and therefore not useful with
759 [LibertySecMech]'s SAML security mechanism. However, it might be used with the bearer security mechanism in
760 conjunction with the `ds11:GenerateBearerToken` directive.)

761 If the `AuthorizeRequester` directive is specified for a resource, the discovery service provider SHOULD in-
762 clude a SAML assertion containing a `ResourceAccessStatement` (as defined in [LibertySecMech]) in any future
763 `QueryResponse` messages for the resource. The `AuthenticateSessionContext` directive is identical to the
764 `AuthorizeRequester` directive except that the appropriate statement is a `SessionContextStatement`.

765 However, if both `AuthorizeRequester` and `AuthenticateSessionContext` are present, the discovery service
766 provider SHOULD NOT generate both a `ResourceAccessStatement` and a `SessionContextStatement`; instead, it
767 SHOULD generate a SAML assertion containing a `ResourceAccessStatement`, and the `ResourceAccessStatement`
768 SHOULD contain a `SessionContext` element.

769 If credentials are provided in response to these directives, they MUST comply with the processing rules defined in
770 [LibertySecMech].

771 The `AuthenticateRequester` directive MUST be used with any descriptions including the security mechanisms
772 from [LibertySecMech] which use SAML for message authentication.

773 If the `EncryptResourceID` directive is included, the discovery service MUST NOT reveal the unencrypted resource
774 ID to clients (i.e. when returning it in a `QueryResponse`). If the discovery service is not willing to honor the directive
775 (for example, because it is not willing to encrypt the resource ID for policy reasons, or because it does not support
776 resource ID encryption), it MUST fail the `Modify` request.

777 The `ds11:SendSingleLogOut` directive is provided for use with the `AuthenticateSessionContext` directive.
778 The credentials received as a result of `AuthenticateSessionContext` may be used by some service instances to
779 establish session state for principal. This directive allows the service instance to request that the provider issuing
780 the credential for `AuthenticateSessionContext` deliver an ID-FF `LogoutRequest` message when the session state
781 being conveyed is no longer valid. If the `ds11:SendSingleLogOut` directive is included, the discovery service
782 MUST cause an ID-FF `LogoutRequest` to be delivered to the ID-FF Service Provider identified by the `ProviderID`
783 in the `ResourceOffering` any time a session ends for which `SessionContext` has been sent to the provider by way of
784 returning credentials in a `Query` for the provider.

785 The `ds11:GenerateBearerToken` directive is provided for use with the Bearer Token Authentication mechanism
786 defined in [LibertySecMech]. It modifies the directives which are defined for use with the SAML authentication
787 mechanism; in particular, the `AuthenticateRequester`, `AuthorizeRequester`, and `AuthenticateSessionContext` directives.
788 If the `ds11:GenerateBearerToken` directive is specified for a resource, any SAML statements generated for the
789 above directives MUST have `ConfirmationMethod` set to "urn:oasis:names:tc:SAML:1.0:cm:bearer". The resulting
790 assertion is to be used as a token with the Bearer Token Authentication mechanism defined in [LibertySecMech].
791 (Note that the rules defined for the SAML Assertion Message Authentication mechanism do not apply to the token
792 generated when `ds11:GenerateBearerToken` is present, even though it happens to be a SAML assertion.)

```
793 <xs:complexType name="DirectiveType">
794   <xs:attribute name="descriptionIDRefs" type="xs:IDREFS" use="optional"/>
795 </xs:complexType>
796 <xs:element name="AuthenticateRequester" type="DirectiveType"/>
797 <xs:element name="AuthorizeRequester" type="DirectiveType"/>
798 <xs:element name="AuthenticateSessionContext" type="DirectiveType"/>
799 <xs:element name="EncryptResourceID" type="DirectiveType"/>
800
801 <xs:element name="SendSingleLogOut" type="disco:DirectiveType"/>
802
803 <xs:element name="GenerateBearerToken" type="disco:DirectiveType"/>
804
```

805 5.2.2. ModifyResponse

806 The response contains the following elements and attributes:

- 807 • `Status`: Contains status code; see processing rules.
- 808 • `newEntryIDs`: If the status is OK, and `InsertEntry` was present in the Modify request, the `newEntryIDs` attribute
809 MUST contain the list of entry IDs assigned to the new entries. The list MUST be in the same order that the
810 `InsertEntry` elements were in.
- 811 • `Extension`: Contains future extensions in other namespaces. One such extension is defined (see [Section 8](#)).

```
812 <xs:element name="ModifyResponse" type="ModifyResponseType"/>
813 <xs:complexType name="ModifyResponseType">
814   <xs:sequence>
815     <xs:element ref="Status"/>
816     <xs:element ref="Extension" minOccurs="0" maxOccurs="1"/>
817   </xs:sequence>
818   <xs:attribute name="id" type="xs:ID" use="optional"/>
819   <xs:attribute name="newEntryIDs" use="optional">
820     <xs:simpleType>
821       <xs:list itemType="IDReferenceType"/>
822     </xs:simpleType>
823   </xs:attribute>
824 </xs:complexType>
825
```

826 An example modify response follows.

```
827
828 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
829   <soap:Header>
830     <Correlation id="YhgnUXx9EMVACSWZiDL"
831       soap:mustUnderstand="1"
832       soap:actor="http://schemas.xmlsoap.org/soap/actor/next"
833       refToMessageId="DNtVj92yV1iICR47w1QV"
834       messageId="AykqNGoN-K5kEmBzqPOU"
835       xmlns="urn:liberty:soap-bind:2003-08"/>
```

```
836     </soap:Header>
837     <soap:Body>
838         <ModifyResponse xmlns="urn:liberty:disco:2003-08" newEntryIDs="2">
839             <Status code="OK" />
840         </ModifyResponse>
841     </soap:Body>
842 </soap:Envelope>
843
```

844 5.2.3. Processing Rules

- 845 • The transaction unit for this operation is the entire set of `InsertEntry` and `RemoveEntry` elements; they either
846 all succeed or all fail. The discovery service provider **MUST** enforce this atomicity. (This is so that a client can
847 easily update a `ResourceOffering` by removing the old one and inserting a new one.)
- 848 • For each `InsertEntry` element, the service instance **MAY** store the resource offering provided such that it can be
849 retrieved (subject to policy) by future `Query` operations on the same `ResourceID`. If the service instance does not
850 store the resource, it **MUST** return a `Failed` status code for the operation (and therefore not insert or remove any of
851 the other entries provided).
- 852 • For each `RemoveEntry` element, the service instance searches for a previously inserted `ResourceOffering` with the
853 given `entryID`. If the `entryID` cannot be located, the service instance **MUST** return a `Failed` status code for the
854 operation. If the `entryID` can be located, the service instance **SHOULD** remove the entry if policy allows. If the
855 entry is not removed, the service instance **MUST** return a `Failed` status code for the operation. (Note that if a `Failed`
856 status code is returned, none of the `RemoveEntry` elements in the message may be processed.)
- 857 • Resource offerings contained in `InsertEntry` **SHOULD NOT** contain `entryID` values. If a resource offering to be
858 inserted contains an `entryID` value, the discovery service **MUST** ignore the value. The discovery service is to pick
859 a new `entryID`. The `entryID` **MUST** be unique within the discovery resource, but it **MUST NOT** be usable as a
860 pseudonym for the user, for privacy reasons (it should not be possible to correlate two discovery responses for the
861 same user from the `entryIDs`).
- 862 • If request processing succeeded, the top-level status code **MUST** be `OK`. Otherwise, the top-level status code
863 **MUST** be `Failed`.
- 864 • If the top-level status code is `Failed`, the response **MAY** also contain `RemoveEntry`, `Directive` or `Forbidden` as a
865 second-level status code. The service may not wish to reveal the reason for failure, in which case no second-level
866 status code will appear. Other second-level status codes from the Discovery Service namespace **MUST NOT** be
867 used. Other second-level status codes from other namespaces **MAY** appear. Clients **MAY** ignore status codes from
868 other namespaces if they are not understood.
- 869 • If any directives are present in a request that the discovery service does not understand, or there are directives
870 present that the discovery service understands but does not support, the discovery service **SHOULD** reject the
871 entire request and **SHOULD** include the second-level status code `Directive` to indicate the reason for failure.

872 6. SAML AttributeDesignator for Discovery ResourceOffering

873 Entities which authenticate Principals using SAML may need to discover the location of the discovery service contain-
874 ing identity services for that Principal. This can be accomplished using the existing `saml:AttributeStatement`
875 mechanism. To include a `ResourceOffering` for a Principal's discovery service in a SAML assertion, an `AttributeS-`
876 `tatement` SHOULD be included according to the following rules:

- 877 • The `saml:AttributeName` MUST be "DiscoveryResourceOffering".
- 878 • The `saml:AttributeNamespace` MUST be "urn:liberty:disco:2003-08".
- 879 • One or more `saml:AttributeValue` element MUST be included which each contain a single `ResourceOffering`
880 element for the Discovery Service specified above. The discovery service must contain identity services for the
881 Principal identified in the `Subject` element inside the `AttributeStatement`.
- 882 • The `ResourceOffering` that is inside the `AttributeStatement` may contain `CredentialRef` elements referring
883 to credentials that are necessary to access the discovery service. These IDs SHOULD resolve to an XML element
884 contained within the SAML `Advice` element of the same `Assertion`.

885 An example `AttributeStatement` that might be found in a Liberty ID-FF `AuthnResponse` follows. Note that it does
886 not include any credentials. If credentials are needed, they should be placed in the `Advice` element of the assertion
887 containing the `AttributeStatement`. In this case, the `Description` should contain `CredentialRef` elements whose values
888 are set to the IDs of the credentials.

```
889 <AttributeStatement xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
890   <Subject>
891     <NameIdentifier Format="urn:liberty:iff:nameid:federated">
892       d0CQF8elJTDLmzEo
893     </NameIdentifier>
894   </Subject>
895   <Attribute AttributeName="DiscoveryResourceOffering"
896     AttributeNamespace="urn:liberty:disco:2003-08">
897     <AttributeValue>
898       <ResourceOffering xmlns="urn:liberty:disco:2003-08">
899         <ResourceID>http://example.com/disco/d0CQF8elJTDLmzEo</ResourceID>
900         <ServiceInstance>
901           <ServiceType>urn:liberty:disco:2003-08</ServiceType>
902           <ProviderID>http://example.com/</ProviderID>
903           <Description>
904             <SecurityMechID>urn:liberty:security:2005-02:TLS:X509</SecurityMechID>
905             <SecurityMechID>urn:liberty:security:2003-08:TLS:null</SecurityMechID>
906             <Endpoint>https://soap.example.com/</Endpoint>
907             <SoapAction>urn:liberty:disco:2003-08</SoapAction>
908           </Description>
909         </ServiceInstance>
910         <Abstract>Discovery service</Abstract>
911       </ResourceOffering>
912     </AttributeValue>
913   </Attribute>
914 </AttributeStatement>
```

917 In all cases, this `AttributeStatement` MUST carry a resource offering for the Liberty discovery service defined in this
918 specification. Any other resource offerings are to be discovered by contacting the discovery service.

919 **7. Option Value for Response Authentication**

920 The service instance description provides a way for services to indicate to clients what mechanisms are necessary
921 for the client to authenticate itself to the service via the SecurityMechID element. The SecurityMechID values
922 defined by [\[LibertySecMech\]](#) also indicate whether the service uses peer entity authentication (for example, server-
923 side SSL/TLS). However, a web service client may need to know whether the service will use message authentication
924 (that is, whether the service will sign the response message) and may not be willing to use a service which does not
925 sign its responses.

926 To avoid situations where a client requests data and then discovers it does not trust it because it is not signed, an
927 `Option` value is defined: *urn:liberty:disco:2005-02:options:security-response-x509*

928 If a service instance always authenticates its response messages according to the "X.509v3 Certificate Message
929 Authentication" mechanism in [\[LibertySecMech\]](#), descriptions of the service SHOULD include this option value.
930 Otherwise, its description MUST NOT include this option value. Clients MAY include this option value in Query
931 messages in order to locate only services which always authenticate their response messages. A service MAY
932 authenticate its response messages even if this option value was not included in its description at the discovery service.

933 In case the service also supported the previous version of this specification [\[LibertySecMech11\]](#), it should be able to
934 register two different endpoints at the discovery service, each of them with different Options values (one according
935 to this version of the specification, another one according to the URN value specified in [\[LibertySecMech11\]](#)). This
936 information would aid the client to determine which version of the WS-SMS specification ([\[wss-sms-draft\]](#) and/or
937 [\[wss-sms\]](#)) is supported by the service, and the service will act accordingly, depending on the endpoint used by the
938 client. Note that this behaviour only applies to the case when the client's request does not use message authentication
939 mechanisms. Otherwise, it should be possible for the service to determine the version of the WS-SMS specification
940 supported by the client by simply analyzing the `<wsse:Security>` header present in the request.

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

943 **8. Including keys in ModifyResponse**

944 Some directives from the Modify request (see [Section 5.2.1](#)) may cause the Discovery service instance to generate
945 signed credentials in QueryResponse messages for the resource offerings in question. When the service instance
946 which included the directives receives the signed credentials from a client, it needs to be able to verify the Discovery
947 service instance's signature on the credentials. Typically the metadata (see [\[LibertyMetadata\]](#)) for the Discovery
948 service instance is sufficient for such information. In certain situations, such as when the Discovery service instance is
949 hosted on a LUAD (see [\[LibertyClientProfiles\]](#)), it may not be feasible to assign the LUAD a ProviderID with which
950 to obtain metadata. However, the key material still needs to be made available to service instances which register
951 services with the discovery service and include such directives.

952 The Discovery service instance may include a ds11:Keys extension in the ModifyResponse in order to provide such
953 keys. The Discovery service instance SHOULD NOT include the extension unless necessary (that is, in cases where
954 the Discovery service instance has no ProviderID). The Discovery service instance SHOULD include the ds11:Keys
955 extension in ModifyResponse messages if it has no ProviderID and the Modify message included a ResourceOffering
956 for which the Discovery service instance intends to generate signed credentials.

```
957 <!-- For use in ModifyResponse Extension field -->  
958 <xs:element name="Keys" type="ds11:KeysType"/>  
959 <xs:complexType name="KeysType">  
960   <xs:sequence>  
961     <xs:element ref="md:KeyDescriptor" minOccurs="1" maxOccurs="unbounded"/>  
962   </xs:sequence>  
963 </xs:complexType>  
964
```

965 The ds11:Keys element appears as a child of the Extension element. It contains one or more KeyDescriptor elements.

966 9. Version 1.2 Core XSD

```

967 <?xml version="1.0" encoding="UTF-8"?>
968 <xs:schema targetNamespace="urn:liberty:disco:2003-08"
969   xmlns:xs="http://www.w3.org/2001/XMLSchema"
970   xmlns:md="urn:liberty:metadata:2003-08"
971   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
972   xmlns="urn:liberty:disco:2003-08"
973   elementFormDefault="qualified"
974   attributeFormDefault="unqualified">
975
976   <xs:include schemaLocation="liberty-idwsf-utility-v1.1.xsd"/>
977
978   <xs:import namespace="urn:liberty:metadata:2003-08"
979     schemaLocation="liberty-metadata-v1.1.xsd"/>
980
981   <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
982     schemaLocation="http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd"/>
983   <xs:annotation>
984     <xs:documentation>
985
986       XML Schema from Liberty Discovery Service Specification.
987
988       ### NOTICE ###
989
990       Copyright (c) 2004-2005 Liberty Alliance participants, see
991       http://www.projectliberty.org/specs/idwsf_1_1_copyrights.php
992
993     </xs:documentation>
994   </xs:annotation>
995   <xs:element name="ServiceType" type="xs:anyURI"/>
996
997   <xs:complexType name="ResourceIDType">
998     <xs:simpleContent>
999       <xs:extension base="xs:anyURI">
1000         <xs:attribute name="id" type="xs:ID" use="optional"/>
1001       </xs:extension>
1002     </xs:simpleContent>
1003   </xs:complexType>
1004
1005   <xs:complexType name="EncryptedResourceIDType">
1006     <xs:sequence>
1007       <xs:element ref="xenc:EncryptedData"/>
1008       <xs:element ref="xenc:EncryptedKey"/>
1009     </xs:sequence>
1010   </xs:complexType>
1011
1012   <xs:element name="ResourceID" type="ResourceIDType"/>
1013   <xs:element name="EncryptedResourceID" type="EncryptedResourceIDType"/>
1014
1015   <!--
1016     if not present, equivalent to
1017     <ResourceID>urn:liberty:isf:implied-resource</ResourceID>
1018     (see specification text for details)
1019   -->
1020
1021   <xs:group name="ResourceIDGroup">
1022     <xs:sequence>
1023       <xs:choice minOccurs="0" maxOccurs="1">
1024         <xs:element ref="ResourceID"/>
1025         <xs:element ref="EncryptedResourceID"/>
1026       </xs:choice>
1027     </xs:sequence>
1028   </xs:group>
1029
1030   <xs:complexType name="DescriptionType">
1031     <xs:sequence>

```

```
1032     <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="1" maxOccurs="unbounded" />
1033     <xs:element name="CredentialRef" type="xs:IDREF" minOccurs="0" maxOccurs="unbounded" />
1034     <xs:choice>
1035       <xs:group ref="WsdRef" />
1036       <xs:group ref="BriefSoapHttpDescription" />
1037     </xs:choice>
1038   </xs:sequence>
1039   <xs:attribute name="id" type="xs:ID" use="optional" />
1040 </xs:complexType>
1041
1042 <xs:complexType name="ServiceInstanceType">
1043   <xs:sequence>
1044     <xs:element ref="ServiceType" />
1045     <xs:element name="ProviderID" type="md:entityIDType" />
1046     <xs:element name="Description" type="DescriptionType" minOccurs="1" maxOccurs="unbounded" />
1047   </xs:sequence>
1048 </xs:complexType>
1049 <xs:group name="WsdRef">
1050   <xs:sequence>
1051     <xs:element name="WsdURI" type="xs:anyURI" />
1052     <xs:element name="ServiceNameRef" type="xs:QName" />
1053   </xs:sequence>
1054 </xs:group>
1055 <xs:group name="BriefSoapHttpDescription">
1056   <xs:sequence>
1057     <xs:element name="Endpoint" type="xs:anyURI" />
1058     <xs:element name="SoapAction" type="xs:anyURI" minOccurs="0" />
1059   </xs:sequence>
1060 </xs:group>
1061 <xs:element name="ResourceOffering" type="ResourceOfferingType" />
1062 <xs:complexType name="ResourceOfferingType">
1063   <xs:sequence>
1064     <xs:group ref="ResourceIDGroup" />
1065     <xs:element name="ServiceInstance" type="ServiceInstanceType" />
1066     <xs:element ref="Options" minOccurs="0" />
1067     <xs:element name="Abstract" type="xs:string" minOccurs="0" />
1068   </xs:sequence>
1069   <xs:attribute name="entryID" type="IDType" use="optional" />
1070 </xs:complexType>
1071 <xs:element name="Options" type="OptionsType" />
1072 <xs:complexType name="OptionsType">
1073   <xs:sequence>
1074     <xs:element name="Option" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded" />
1075   </xs:sequence>
1076 </xs:complexType>
1077 <xs:element name="Query" type="QueryType" />
1078 <xs:complexType name="QueryType">
1079   <xs:sequence>
1080     <xs:group ref="ResourceIDGroup" />
1081     <xs:element name="RequestedServiceType" minOccurs="0" maxOccurs="unbounded">
1082       <xs:complexType>
1083         <xs:sequence>
1084           <xs:element ref="ServiceType" />
1085           <xs:element ref="Options" minOccurs="0" />
1086         </xs:sequence>
1087       </xs:complexType>
1088     </xs:element>
1089   </xs:sequence>
1090   <xs:attribute name="id" type="xs:ID" use="optional" />
1091 </xs:complexType>
1092 <xs:element name="QueryResponse" type="QueryResponseType" />
1093 <xs:complexType name="QueryResponseType">
1094   <xs:sequence>
1095     <xs:element ref="Status" />
1096     <xs:element ref="ResourceOffering" minOccurs="0" maxOccurs="unbounded" />
1097     <xs:element name="Credentials" minOccurs="0">
1098       <xs:complexType>
```

```
1099         <xs:sequence>
1100             <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
1101         </xs:sequence>
1102     </xs:complexType>
1103 </xs:element>
1104 </xs:sequence>
1105 <xs:attribute name="id" type="xs:ID" use="optional" />
1106 </xs:complexType>
1107 <xs:complexType name="InsertEntryType">
1108     <xs:sequence>
1109         <xs:element ref="ResourceOffering" />
1110         <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
1111     </xs:sequence>
1112 </xs:complexType>
1113
1114 <xs:complexType name="RemoveEntryType">
1115     <xs:attribute name="entryID" type="IDReferenceType" use="required" />
1116 </xs:complexType>
1117
1118 <xs:element name="Modify" type="ModifyType" />
1119 <xs:complexType name="ModifyType">
1120     <xs:sequence>
1121         <xs:group ref="ResourceIDGroup" />
1122         <xs:element name="InsertEntry" type="InsertEntryType" minOccurs="0" maxOccurs="unbounded" />
1123         <xs:element name="RemoveEntry" type="RemoveEntryType" minOccurs="0" maxOccurs="unbounded" />
1124     </xs:sequence>
1125     <xs:attribute name="id" type="xs:ID" use="optional" />
1126 </xs:complexType>
1127 <xs:complexType name="DirectiveType">
1128     <xs:attribute name="descriptionIDRefs" type="xs:IDREFS" use="optional" />
1129 </xs:complexType>
1130 <xs:element name="AuthenticateRequester" type="DirectiveType" />
1131 <xs:element name="AuthorizeRequester" type="DirectiveType" />
1132 <xs:element name="AuthenticateSessionContext" type="DirectiveType" />
1133 <xs:element name="EncryptResourceID" type="DirectiveType" />
1134 <xs:element name="ModifyResponse" type="ModifyResponseType" />
1135 <xs:complexType name="ModifyResponseType">
1136     <xs:sequence>
1137         <xs:element ref="Status" />
1138     <xs:element ref="Extension" minOccurs="0" maxOccurs="1" />
1139 </xs:sequence>
1140     <xs:attribute name="id" type="xs:ID" use="optional" />
1141     <xs:attribute name="newEntryIDs" use="optional">
1142         <xs:simpleType>
1143             <xs:list itemType="IDReferenceType" />
1144         </xs:simpleType>
1145     </xs:attribute>
1146 </xs:complexType>
1147 </xs:schema>
1148
```

1149 10. Version 1.2 Extension XSD

```
1150 <?xml version="1.0" encoding="UTF-8"?>
1151 <xs:schema targetNamespace="urn:liberty:disco:2004-04"
1152   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1153   xmlns:md="urn:liberty:metadata:2003-08"
1154   xmlns:dsl1="urn:liberty:disco:2004-04"
1155   xmlns:disco="urn:liberty:disco:2003-08"
1156   elementFormDefault="qualified"
1157   attributeFormDefault="unqualified">
1158
1159   <xs:include schemaLocation="liberty-idwsf-utility-v1.1.xsd"/>
1160   <xs:import namespace="urn:liberty:metadata:2003-08"
1161     schemaLocation="liberty-metadata-v1.1.xsd"/>
1162   <xs:import namespace="urn:liberty:disco:2003-08"
1163     schemaLocation="liberty-idwsf-disco-svc-v1.2.xsd"/>
1164   <xs:annotation>
1165     <xs:documentation>
1166
1167       XML Schema from Liberty Discovery Service Specification.
1168
1169       ### NOTICE ###
1170
1171       Copyright (c) 2004-2005 Liberty Alliance participants, see
1172       http://www.projectliberty.org/specs/idwsf_1_1_copyrights.php
1173
1174     </xs:documentation>
1175   </xs:annotation>
1176   <!-- For use in ModifyResponse Extension field -->
1177   <xs:element name="Keys" type="dsl1:KeysType"/>
1178   <xs:complexType name="KeysType">
1179     <xs:sequence>
1180       <xs:element ref="md:KeyDescriptor" minOccurs="1" maxOccurs="unbounded"/>
1181     </xs:sequence>
1182   </xs:complexType>
1183   <xs:element name="SendSingleLogout" type="disco:DirectiveType"/>
1184   <xs:element name="GenerateBearerToken" type="disco:DirectiveType"/>
1185 </xs:schema>
1186
```

1187 11. WSDL

```
1188 <?xml version="1.0"?>
1189 <definitions name="disco-svc"
1190   targetNamespace="urn:liberty:disco:2003-08"
1191   xmlns:typens="urn:liberty:disco:2003-08"
1192   xmlns="http://schemas.xmlsoap.org/wsdl/"
1193   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
1194   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
1195   xmlns:sb="urn:liberty:sb:2003-08"
1196   xmlns:disco="urn:liberty:disco:2003-08">
1197
1198   <!-- Abstract WSDL for Liberty Discovery Service Specification -->
1199
1200   <xsd:documentation>
1201
1202     XML Schema from Liberty Discovery Service Specification.
1203
1204     ### NOTICE ###
1205
1206     Copyright (c) 2004-2005 Liberty Alliance participants, see
1207     http://www.projectliberty.org/specs/idwsf_1_1_copyrights.php
1208
1209   </xsd:documentation>
1210
1211   <types>
1212     <xsd:schema>
1213       <xsd:import schemaLocation="liberty-idwsf-disco-svc-exts-v1.2.xsd"/>
1214       <xsd:import schemaLocation="liberty-idwsf-soap-binding-exts-v1.2.xsd"/>
1215       <xsd:import schemaLocation="liberty-idwsf-soap-binding-v1.2.xsd"/>
1216     </xsd:schema>
1217   </types>
1218
1219   <message name="Query">
1220     <part name="body" element="disco:Query"/>
1221   </message>
1222
1223   <message name="QueryResponse">
1224     <part name="body" element="disco:QueryResponse"/>
1225   </message>
1226
1227   <message name="Modify">
1228     <part name="body" element="disco:Modify"/>
1229   </message>
1230
1231   <message name="ModifyResponse">
1232     <part name="body" element="disco:ModifyResponse"/>
1233   </message>
1234
1235   <message name="CorrelationHeader">
1236     <part name="Correlation" element="typens:Correlation"/>
1237   </message>
1238
1239   <portType name="DiscoveryPort">
1240
1241     <operation name="DiscoveryLookup">
1242       <input message="typens:Query"/>
1243       <output message="typens:QueryResponse"/>
1244     </operation>
1245
1246     <operation name="DiscoveryUpdate">
1247       <input message="typens:Modify"/>
1248       <output message="typens:ModifyResponse"/>
1249     </operation>
1250
1251   </portType>
1252
```

```
1253 <!--
1254 An example of a binding and service that can be used with this
1255 abstract service description is provided below.
1256 -->
1257
1258 <binding name="DiscoveryBinding" type="typens:DiscoveryPort">
1259
1260   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
1261
1262   <operation name="DiscoveryLookup">
1263     <soap:operation soapAction="http://example.com/DiscoveryLookup"/>
1264
1265     <input>
1266       <soap:header message="typens:CorrelationHeader" part="Correlation" use="literal"/>
1267       <soap:body use="literal"/>
1268     </input>
1269
1270     <output>
1271       <soap:header message="typens:CorrelationHeader" part="Correlation" use="literal"/>
1272       <soap:body use="literal"/>
1273     </output>
1274
1275   </operation>
1276
1277   <operation name="DiscoveryUpdate">
1278     <soap:operation soapAction="http://example.com/DiscoveryUpdate"/>
1279
1280     <input>
1281       <soap:header message="typens:CorrelationHeader" part="Correlation" use="literal"/>
1282       <soap:body use="literal"/>
1283     </input>
1284
1285     <output>
1286       <soap:header message="typens:CorrelationHeader" part="Correlation" use="literal"/>
1287       <soap:body use="literal"/>
1288     </output>
1289
1290   </operation>
1291
1292 </binding>
1293
1294 <service name="DiscoveryService">
1295
1296   <port name="DiscoveryPort" binding="typens:DiscoveryBinding">
1297
1298     <!-- Modify with the REAL SOAP endpoint -->
1299
1300     <soap:address location="http://example.com/discovery"/>
1301
1302   </port>
1303
1304 </service>
1305
1306 </definitions>
1307
```

1308 References

1309 Normative

- 1310 [RFC2396] Berners-Lee, T., Fielding, R., Masinter, L., eds. (August 1998). "Uniform Resource Identifiers (URI):
1311 Generic Syntax," RFC 2396, The Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2396.txt>
1312 [August 1998].
- 1313 [SAMLCore] Hallam-Baker, Phillip, Maler, Eve, eds. (05 November 2002). "SAML Core Assertions and Protocols,"
1314 OASIS Standard, version 1.0, Organization for the Advancement of Structured Information Standards
1315 <http://www.oasis-open.org/committees/security/#documents>
- 1316 [Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May
1317 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium
1318 <http://www.w3.org/TR/xmlschema-1/>
- 1319 [WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Meredith,
1320 Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001).
1321 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- 1322 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 1.1, Liberty
1323 Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 1324 [LibertySecMech] Ellison, Gary, eds. "Liberty ID-WSF Security Mechanisms," Version 1.2, Liberty Alliance Project
1325 (14 December 2004). <http://www.projectliberty.org/specs>
- 1326 [LibertySecMech11] Ellison, Gary, eds. "Liberty ID-WSF Security Mechanisms," Version 1.1, Liberty Alliance
1327 Project (18 April 2004). <http://www.projectliberty.org/specs>
- 1328 [wss-sms] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (January, 2004). "Web
1329 Services Security: SOAP Message Security," OASIS Standard V1.0 [OASIS 200401], Organization for the
1330 Advancement of Structured Information Standards [http://docs.oasis-open.org/wss/2004/01/oasis-200401-
1331 wss-soap-message-security-1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)
- 1332 [wss-sms-draft] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (June 30,
1333 2003). "Web Services Security: SOAP Message Security," Draft WSS-SOAPMessageSecurity-14-
1334 06-2003, Organization for the Advancement of Structured Information Standards [http://www.oasis-
1335 open.org/committees/download.php/2757/WSS-SOAPMessageSecurity-14-063003-merged.pdf](http://www.oasis-open.org/committees/download.php/2757/WSS-SOAPMessageSecurity-14-063003-merged.pdf)
- 1336 [xmlenc-core] Eastlake, Donald, Reagle, Joseph, eds. (December 2002). "XML Encryption Syntax and Processing,"
1337 W3C Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlenc-core/>

1338 Informative

- 1339 [LibertyPAOS] Aarts, Robert, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version 1.1, Liberty
1340 Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 1341 [LibertyClientProfiles] Aarts, Robert, eds. (14 December 2004). "Liberty ID-WSF Profiles for Liberty enabled User
1342 Agents and Devices," version 1.1, Liberty Alliance Project <http://www.projectliberty.org/specs/>