



# Liberty Metadata Description and Discovery Specification

Version: 2.0-02

**Editors:**

Peter Davis, NeuStar, Inc.

**Contributors:**

Scott Cantor, Internet2

Darryl Champagne, IEEE-ISTO

Jeff Hodges, Sun Microsystems, Inc.

Bronislav Kavsan, RSA Security, Inc.

Paul Madsen, Entrust, Inc.

**Abstract:**

This document details the metadata schema and methods of resolution for discovering the location of metadata instances for the Liberty Identity Federation Framework

**Filename:** draft-liberty-metadata-v2.0-02.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 ActivCard; America Online, Inc.; American Express Travel Related Services; Axalto; Bank of  
16 America Corporation; Bell Canada; Cingular Wireless; Cisco Systems, Inc.; Communicator, Inc.; Deloitte & Touche  
17 LLP; Earthlink, Inc.; Electronic Data Systems, Inc.; Entrust, Inc.; Epok, Inc.; Ericsson; Fidelity Investments; France  
18 Telecom; Gemplus; General Motors; Hewlett-Packard Company; i2 Technologies, Inc.; Internet2; Intuit Inc.;  
19 MasterCard International; NEC Corporation; Netegrity, Inc.; NeuStar, Inc.; Nextel Communications; Nippon  
20 Telegraph and Telephone Corporation; Nokia Corporation; Novell, Inc.; NTT DoCoMo, Inc.; OneName Corporation;  
21 Openwave Systems Inc.; Phaos Technology; Ping Identity Corporation; PricewaterhouseCoopers LLP; RegistryPro,  
22 Inc.; RSA Security Inc; Sabre Holdings Corporation; SAP AG; SchlumbergerSema; Sigaba; SK Telecom; Sony  
23 Corporation; Sun Microsystems, Inc.; Symlabs, Inc.; Trustgenix; United Airlines; VeriSign, Inc.; Visa International;  
24 Vodafone Group Plc; Wave Systems. 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. Metadata Schema ..... 5  
34 3. Publishing the Metadata ..... 20  
35 4. Metadata Resolution and Retrieval ..... 24  
36 5. Post Processing of the Metadata Document ..... 26  
37 6. Security Considerations ..... 28  
38 7. Metadata XSD ..... 29  
39 References ..... 33

## 40 1. Introduction

41 Within **ID-FF** version 1.1 specification [[LibertyProtSchema1.1](#)] of the Liberty Alliance protocols [[LibertyProtSchema](#)], basic metadata were exchanged out-of-band between entities. This specification more formally  
42 describes metadata, as well as protocols to facilitate real-time requests for this data allowing for more spontaneous  
43 conversations between Liberty enabled entities.  
44

45 There are three primary functions for this metadata:

- 46 • declarations of entity metadata for providers, principals and devices, and affiliations
- 47 • entity trust metadata, which enables entities to cast business decisions based on the characteristic trust information  
48 provided in this class, conveyed through document signature(s), server authenticated protected channel delivery of  
49 the instance using TLS [[RFC2246](#)] as amended by [[RFC3546](#)], DNS zone signatures, and, optionally, additional  
50 material that publishers may convey within the `Extension` and `AdditionalMetaLocation` elements
- 51 • origin and document verification through signature use in (server authenticated) HTTPS retrieval of the instance  
52 documents, DNS signatures, and document level signatures

53 This document presents extensions to the model for metadata described in Liberty ID-FF versions 1.1 to better support  
54 ad-hoc interactions between entities. The location of cryptographic keys in a distributed-computing architecture that  
55 contains "arms-length" peer domains presents an opportunity for some fresh thinking. Conventional solutions to this  
56 problem fail to fully exploit the potential of the evolving Web Services architecture to minimize administrative costs.  
57 Liberty ID-FF version 1.2 [[LibertyProtSchema](#)], ID-WSF and ID-SIS set of specification [[LibertyIDFFOverview](#)]  
58 operations between previously un-introduced parties will benefit from any mechanisms that simplify how keying  
59 material and service interface points can be discovered, leading to mechanisms for trust establishment and services  
60 invocations in both direct and indirect means.

### 61 1.1. Notation and Conventions

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

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

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

69 Within this document, a *publisher* is the subject of, or authorized representing party for, the subject of the instance  
70 document, as referenced by `providerID` and a *consumer* is the entity resolving, retrieving, or otherwise processing  
71 the instance as a relying party to its information.

### 72 1.2. Overview

73 The metadata protocols and schemas specified in this document will enable two Liberty-enabled entities to exchange  
74 or request cryptographic keys, service endpoints information, and protocol and profile support in real time, allowing  
75 dynamic interactions between these parties, eliminating the need for out-of-band negotiations to have occurred a-priori.  
76 The addition of interactions between separate authentication authorities and identity chaining in the Liberty **ID-WSF**  
77 will depend upon this exchange, as portions of a principle's identity may be previously established outside the range  
78 of providers established agreements.

## 79 **2. Metadata Schema**

80 The metadata schema allows several methods of representation:

- 81 • A document expressing metadata describing an entity, referenced by a `providerID`, acting in the role of a Service  
82 Provider or an Identity Provider, or both, within the `EntityDescriptor` Node
- 83 • As a single instance document expressing metadata describing multiple entities, each referenced by a  
84 `providerID`, each entity acting as declared above. The metadata for each entity is contained within separate  
85 `EntityDescriptor` nodes, each being an immediate descendant of the plural `EntitiesDescriptor`  
86 node
- 87 • As a single instance document describing an affiliation (a set of entities) collectively identified as `providerID`  
88 (located with the `EntityDescriptorparent`), which in turn enumerates each entity member by its own  
89 `providerID` and maintained by an entity referenced by its `affiliationOwnerID`. Each member's metadata  
90 is then located by the methods provided in this specification.

91 The first two forms may also be expressed as multiple documents, involving additional metadata, which MAY  
92 be of a namespace `urn:liberty:metadata:2003-08` (the default), or another namespace, as specified by the  
93 element `Location`'s corresponding `namespace` attribute. Additionally, the document location(s) may be identified  
94 by multiple NAPTR resource records.

### 95 **2.1. Schema Declarations**

96 The metadata schema is constructed to allow an entity, referenced by one or more `providerID`'s, to publish single or  
97 multiple schema instances to describe their identity services architecture.

98 The primary container for a published document is either `EntityDescriptor` or the plural form  
99 `EntitiesDescriptor` (used when an affiliated set of entities chooses to publish a consolidated set of meta-  
100 data documents as one).

101 The expected immediate child nodes of `EntityDescriptor` are one or more of:

- 102 • `SPDescriptor`
- 103 • `IDPDescriptor`

104 or one of:

- 105 • `AffiliationDescriptor`

106 which are described below. Additionally, an extension point `Extension` is provided in order to convey additional  
107 metadata.

#### 108 **2.1.1. Namespaces in Metadata**

109 The following namespace declarations are used to complete the metadata schema:

- 110 • `ds:` is described by the W3C XML Signature [\[XMLDsig\]](#) schema

111 In addition, the Liberty Utility Schema is included allowing the common `Extension` element that is used throughout  
112 the Liberty Specifications suite

113 Schema Fragment:

```
114 <xs:schema targetNamespace="urn:liberty:metadata:2004-12 "  
115   xmlns="urn:liberty:metadata:2004-12"  
116   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
117   xmlns:xs="http://www.w3.org/2001/XMLSchema"  
118   elementFormDefault="qualified"  
119   attributeFormDefault="unqualified" version="1.0">  
120   <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"  
121     schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd"/>  
122   <xs:import namespace="http://www.w3.org/XML/1998/namespace"  
123     schemaLocation="http://www.w3.org/2001/xml.xsd"/>  
124   <xs:include schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>  
125  
126
```

## 127 2.1.2. Datatype `entityIDType`

128 The datatype `entityIDType` restricts the XML data to a length of 1024 bytes.

129 Additionally, the `entityIDType` structure is defined by the following BNF, derived from *URI Specification*  
130 [\[RFC2396\]](#) as modified by [\[RFC2732\]](#)

```
131 BNF for Liberty entityIdentifiers  
132 # constraint on absoluteURI  
133 entityID    = absoluteURI [ "#" fragment ]  
134 absoluteURI = scheme ":" ( hier_part | opaque_part )  
135  
136 # constraint on hier_part (net_path only)  
137 hier_part   = net_path [ "?" query ]  
138 opaque_part = uric_no_slash *uric  
139  
140 uric_no_slash = unreserved | escaped | ";" | "?" | ":" | "@" |  
141                "&" | "=" | "+" | "$" | ","  
142  
143 net_path      = "://" authority [ abs_path ]  
144 abs_path     = "/" path_segments  
145  
146 ; pragmatically, scheme SHOULD be an officially IANA registered URI scheme  
147 ; http://www.iana.org/assignments/uri-schemes  
148 scheme       = alpha *( alpha | digit | "+" | "-" | "." )  
149  
150 authority    = server | reg_name  
151  
152 reg_name     = 1*( unreserved | escaped | "$" | "," |  
153                   ";" | ":" | "@" | "&" | "=" | "+" )  
154  
155 server       = [ [ userinfo "@" ] hostport ]  
156 userinfo     = *( unreserved | escaped |  
157                   ";" | ":" | "&" | "=" | "+" | "$" | "," )  
158  
159 hostport     = host [ ":" port ]  
160 ; constraint on host (no ipAddress)  
161 host         = hostname  
162 hostname     = *( domainlabel "." ) toplabel [ "." ]  
163 domainlabel  = alphanum | alphanum *( alphanum | "-" ) alphanum  
164 toplabel     = alpha | alpha *( alphanum | "-" ) alphanum  
165 port         = *digit  
166  
167 path         = [ abs_path | opaque_part ]  
168 path_segments = segment *( "/" segment )  
169 segment      = *pchar *( ";" param )
```

```
170 param      = *pchar
171 pchar       = unreserved | escaped |
172             ":" | "@" | "&" | "=" | "+" | "$" | ","
173
174 query       = *uric
175
176 fragment    = *uric
177
178 uric        = reserved | unreserved | escaped
179 reserved    = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
180             "$" | ","
181 unreserved  = alphanum | mark
182 mark        = "-" | "_" | "." | "!" | "~" | "*" | "'" |
183             "(" | ")"
184
185 escaped      = "%" hex hex
186 hex         = digit | "A" | "B" | "C" | "D" | "E" | "F" |
187             "a" | "b" | "c" | "d" | "e" | "f"
188
189 alphanum    = alpha | digit
190 alpha       = lowalpha | upalpha
191
192 lowalpha    = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
193             "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
194             "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
195 upalpha     = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
196             "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
197             "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
198 digit       = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
199             "8" | "9"
200
201
```

202 The schema fragment for `entityIDType`:

```
203 <xs:simpleType name="entityIDType">
204   <xs:restriction base="xs:anyURI">
205     <xs:maxLength value="1024" id="maxlengthid" />
206   </xs:restriction>
207 </xs:simpleType>
208
209
```

### 210 2.1.3. Common Attributes

211 Several common attributes are defined and generally used throughout the schema:

- 212 • `libertyPrincipalIdentifier` of type `entityIDType` used to provide a pointer to contact an entity's  
213 metadata which MAY be dereferencable
- 214 • `providerID` of type `entityIDType` indicates the `providerID` of the entity described by the descendants of the  
215 node
- 216 • `validUntil` of type `dateTime` indicates the expiration date and time of the node (and its descendants). If  
217 `dateTime` expressions evaluate to nonequivalent values, parsers MUST adhere to the most restrictive value (the  
218 earliest `dateTime`).
- 219 • `cacheDuration` of type `duration` indicates the maximum elapsed time a consumer may cache the metadata  
220 document (or fragment). Consistent with the `validUntil` attribute, the most restrictive value MUST be used  
221 when conflicting cache directives occur

222 Publishers **MUST** provide either a `validUntil` or `cacheDuration` attribute when publishing metadata. Since this  
223 directive is available at both the top-level `EntityDescriptor` and its immediate descendants, care should be taken in selecting  
224 expiration settings. It is **RECOMMENDED** that publishers express document expiration at the `EntityDescriptor`  
225 or `AffiliationDescriptor` element only, and not on the child nodes.

226 All Liberty time values have the type `dateTime`, which is built in to the W3C XML Schema Datatypes specification  
227 [Schema2]. Liberty time values **MUST** be expressed in UTC form, indicated by a "Z" immediately following the time  
228 portion of the value.

229 Liberty entities **SHOULD NOT** rely on other applications supporting time resolution finer than seconds, as imple-  
230 mentations **MAY** ignore fractional second components specified in timestamp values. Implementations **MUST NOT**  
231 generate time instants that specify leap seconds.

232 The consumer **MAY** reset the retrieval `dateTime`, effectively resetting the duration clock (see Section 5.2) if consumers  
233 send an *HTTP (1.1) [RFC2616]* request to the publisher URL with a header *If-Modified-Since: [last retrieval*  
234 *dateTime]*, the publisher server returns a *304 Not-Modified* response, and the publisher expresses the expiration as  
235 a `cacheDuration`,

236 The schema fragment for the common attributes:

```
237 <!--  
238 <xs:attribute name="libertyPrincipalIdentifier" type="entityIDType"/>  
239 <xs:attribute name="providerID" type="entityIDType"/>  
240 <xs:attribute name="validUntil" type="xs:dateTime"/>  
241 <xs:attribute name="cacheDuration" type="xs:duration"/>  
242 -->  
243  
244
```

## 245 2.1.4. Common DataTypes

246 There are several common datatypes defined globally, and used throughout the schema:

### 247 2.1.4.1. organizationType Data Type

248 The `organizationType` datatype provides some basic information consumers may require when interacting with a  
249 principal:

- 250 • `OrganizationName` of type `string` [Required, 1-many]: a localizable ([XML] Section 2.12 Language Identifi-  
251 cation) Organizational Name of the entity, generally the complete Organization Legal name
- 252 • `OrganizationDisplayName` of type `string` [Required, 1-many]: a localizable organization name suitable for  
253 display to a principal
- 254 • `OrganizationURL` of type `anyURI` [Required, 1-many]: a localizable URL of the organization suitable for  
255 dereferencing by a user-agent, which may be used for directing a principal for additional information on the entity



256 Localized strings SHOULD be used when present in the metadata instance, and the preferred language of the target  
257 entity is known by the consumer

```
258 <xs:complexType name="additionalMetadataLocationType">
259   <xs:simpleContent>
260     <xs:extension base="xs:anyURI">
261       <xs:attribute name="namespace" type="xs:anyURI"/>
262     </xs:extension>
263   </xs:simpleContent>
264 </xs:complexType>
265
266 <xs:complexType name="organizationNameType">
267   <xs:simpleContent>
268     <xs:extension base="xs:string">
269       <xs:attribute ref="xml:lang"/>
270     </xs:extension>
271   </xs:simpleContent>
272 </xs:complexType>
273
274 <xs:complexType name="organizationDisplayNameType">
275   <xs:simpleContent>
276     <xs:extension base="xs:string">
277       <xs:attribute ref="xml:lang" use="required"/>
278     </xs:extension>
279   </xs:simpleContent>
280 </xs:complexType>
281
282 <xs:complexType name="organizationType">
283   <xs:sequence>
284     <xs:element name="OrganizationName" type="organizationNameType" maxOccurs="unbounded"/>
285     <xs:element name="OrganizationDisplayName" type="organizationDisplayNameType"
286 e" maxOccurs="unbounded"/>
287     <xs:element name="OrganizationURL" type="localizedURIType" maxOccurs="unbounded"/>
288     <xs:element ref="Extension" minOccurs="0"/>
289   </xs:sequence>
290 </xs:complexType>
291
292 <xs:complexType name="localizedURIType">
293   <xs:simpleContent>
294     <xs:extension base="xs:anyURI">
295       <xs:attribute ref="xml:lang" use="required"/>
296     </xs:extension>
297   </xs:simpleContent>
298 </xs:complexType>
299
300
```

#### 301 2.1.4.2. contactType Data Type

302 The contactType data type conveys general contact information for human-to-human contact regarding an entity. It is  
303 defined with the following attributes:

- 304 • libertyPrincipalIdentifier [Optional]: a Principal's dereferencable nameIdentifier of type entityIDType  
305 which may point to an online instance of the person's PIP profile
- 306 • contactType [Required]: the type of contact, which may be one of: technical, administrative, billing,  
307 or other. The default value is technical

308 The elements defined by this type:

- 309 • **Company** [Optional, 0-1]: The company name of type `xs:string`, by which the cited individual is employed for  
310 the purposes relating to the instance document
- 311 • **GivenName** [Optional, 0-1]: The given name of the contact of type `xs:string`
- 312 • **SurName** [Optional, 0-1]: The surname of the contact of type `xs:string`
- 313 • **EmailAddress** [Optional, 0-many]: The email address of the contact of type `xs:anyURI`
- 314 • **TelephoneNumber** [Optional, 0-many]: The contact's telephone number of type `xs:string`

315 The schema fragment for `contactType`:

```
316 <xs:complexType name="contactType">
317   <xs:sequence>
318     <xs:element name="Company" type="xs:string" minOccurs="0"/>
319     <xs:element name="GivenName" type="xs:string" minOccurs="0"/>
320     <xs:element name="SurName" type="xs:string" minOccurs="0"/>
321     <xs:element name="EmailAddress" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
322     <xs:element name="TelephoneNumber" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
323     <xs:element ref="Extension" minOccurs="0"/>
324   </xs:sequence>
325   <xs:attribute name="libertyPrincipalIdentifier" type="entityIDType" use="optional"/>
326   <xs:attribute name="contactType" type="attr.contactType" use="required"/>
327 </xs:complexType>
328 <xs:simpleType name="attr.contactType">
329   <xs:restriction base="xs:string">
330     <xs:enumeration value="technical"/>
331     <xs:enumeration value="administrative"/>
332     <xs:enumeration value="billing"/>
333     <xs:enumeration value="other"/>
334   </xs:restriction>
335 </xs:simpleType>
336
337
```

### 338 2.1.4.3. **keyDescriptorType Complex Type**

339 The elements of type `keyDescriptorType` convey to a consumer two cryptographic metadata statements:

- 340 • encryption preferences described by `EncryptionMethod` [Optional, 0-1], whose valid URI values are defined in  
341 [\[xmlesc-core\]](#), and  
342 `KeySize` [Optional, 0-1] which may optionally constrain the length of keys used by the consumer when interacting  
343 with another entity
- 344 • Key Material information located in `ds:KeyInfo` [Optional, 0-1] as described by [\[XMLDsig\]](#)

345 The schema fragment for `keyDescriptorType`:

```
346 <xs:element name="KeyDescriptor" type="keyDescriptorType"/>
347 <xs:complexType name="keyDescriptorType">
348   <xs:sequence>
349     <xs:element name="EncryptionMethod" type="xs:anyURI" minOccurs="0"/>
350     <xs:element name="KeySize" type="xs:integer" minOccurs="0"/>
351     <xs:element ref="ds:KeyInfo" minOccurs="0"/>
352     <xs:element ref="Extension" minOccurs="0"/>
353   </xs:sequence>
354   <xs:attribute name="use" type="keyTypes" use="optional"/>
355 </xs:complexType>
356
357
```

358 The `KeyDescriptor` includes the required attribute `use` of type `keyTypes`. `use` may have values of encryption  
359 or signing:

```
360 <xs:simpleType name="keyTypes">
361   <xs:restriction base="xs:string">
362     <xs:enumeration value="encryption"/>
363     <xs:enumeration value="signing"/>
364   </xs:restriction>
365 </xs:simpleType>
366
367
```

#### 368 2.1.4.4. `providerDescriptorType` Complex Type

369 The `providerDescriptorType` is a utility type, which describes generic metadata for any Liberty-enabled entity with  
370 attributes that include:

- 371 • `id` [Optional]: The fragment identifier of the instance node (required if the node is signed as described in  
372 [Section 5.1](#)).
- 373 • `validUntil` [Optional]: The `dateTime` the fragment expires. Processing rules are described in [Section 2.1.3](#) [7].
- 374 • `cacheDuration` [Optional]: The maximum duration a consumer may cache the fragment. Processing rules are  
375 described in [Section 2.1.3](#) [7].
- 376 • `protocolSupportEnumeration` [Required] describes the protocol release supported by the entity de-  
377 scribed by `providerID`. It's value is a whitespace delimited enumeration of Liberty ID-FF protocol  
378 releases which the interfaces described within MUST support. The datatype of the tokens MUST be  
379 URIs (presently <http://projectliberty.org/schemas/core/2002/12> for release ID-FF 1.1 and  
380 <urn:liberty:iff:2003-08> for release ID-FF 1.2). Subsequent releases of ID-FF shall express protocol  
381 support using the defined `namespace` attribute of the corresponding ID-FF schema.  
382 When an entity supports both ID-FF 1.1 and ID-FF 1.2 protocols, it SHOULD publish a ID-FF 1.1 valid in-  
383 stance and make reference to it within `AdditionalMetaLocation`, using the appropriate corresponding names-  
384 pace identifier for that schema. Metadata consumers MUST retrieve (or otherwise obtain) this instance if they  
385 intend to use the protocols of ID-FF 1.1. If publisher entities support both protocols on the same `SoapEndPoint`,  
386 they MAY publish one document which describes both protocols uniformly, citing both protocols in the attribute  
387 `protocolSupportEnumeration` or they MAY make reference to it using `AdditionalMetaLocation`. Con-  
388 sumers in possession of an ID-FF 1.1 provider's metadata obtained in an "Out-of-Band" manner as described in  
389 that version of the specification, MAY continue to use this instance, but SHOULD check for a newer version  
390 whenever possible.

391 The elements describing the entity include:

- 392 • `KeyDescriptor` [Optional, 0-many] expresses a set of keying material and key metadata which the coresponding  
393 entity `providerID` will use within Liberty protocols and interactions.
  
- 394 • `SoapEndpoint` [Required, 1] The provider's SOAP endpoint URI.
  
- 395 • `SingleLogoutServiceURL` [Optional, 1] The URL used for user-agent-based Single Logout Protocol profiles.
  
- 396 • `SingleLogoutServiceReturnURL` [Optional, 0-1] The URL to which the provider redirects at the end of user-  
397 agent-based Single Logout Protocol profiles.
  
- 398 • `FederationTerminationServiceURL` [Optional, 0-1] The URL used for user-agent-based Federation Termi-  
399 nation Notification Protocol profiles.
  
- 400 • `FederationTerminationServiceReturnURL` [Optional, 0-1] The URL to which the provider redirects at the  
401 end of user-agent-based Federation Termination Notification Protocol profiles.
  
- 402 • `FederationTerminationNotificationProtocolProfile` [Optional, 0-many] The Federation Termination  
403 Notification Protocol profiles supported by the provider. Each value of the element MUST contain a valid  
404 Federation Termination Notification Protocol profile identification URI as defined in [\[LibertyBindProf\]](#). The  
405 absence of this element SHALL mean that provider does not support any profile of the Federation Termination  
406 Notification Protocol.
  
- 407 • `SingleLogoutProtocolProfile` [Optional, 0-many] The Single Logout Protocol profiles supported by the  
408 provider. Each element MUST contain a valid Single Logout Protocol profile identification URI. The absence of  
409 this element SHALL mean that the provider does not support any profile of the Single Logout Protocol.
  
- 410 • `RegisterNameIdentifierProtocolProfile` [Optional, 0-many] The provider's preferred Register Name  
411 Identifier Protocol profile, which should be used by other providers when registering a new identifier. Each element  
412 MUST contain a valid Register Name Identifier Protocol profile identification URI as defined in [\[LibertyBindProf\]](#).  
413 The absence of this element SHALL mean that the provider does not support any profile of the Register Name  
414 Identifier Protocol.
  
- 415 • `RegisterNameIdentifierServiceURL` [Optional, 0-1] The URL used for user-agent-based Register Name  
416 Identifier Protocol profiles.
  
- 417 • `RegisterNameIdentifierServiceReturnURL` [Optional, 0-1] The provider's redirecting URL for use after  
418 HTTP name registration has taken place.
  
- 419 • `NameIdentifierMappingProtocolProfile` [Optional, 0-many] of type `anyURI`, which indicates the profile  
420 of the `NameIdentifierMapping` protocol supported by the Provider. This subject entity of the metadata instance  
421 should be a provider who administers identifiers for a subject across multiple namespaces.
  
- 422 • `NameIdentifierMappingEncryptionProfile` [optional, 0-many] of type `anyURI`, which indicates the en-  
423 cryption profiles supported by the provider as a recipient of an encrypted `NameIdentifier`.
  
- 424 • `Organization` [Optional, 0-1] The Organization (see [Section 2.1.4.1](#)) information about the provider.
  
- 425 • `ContactPerson` [Optional, 0-many] A Container expressing one or more contacts responsible for technical,  
426 administrative, billing, or other information concerning an identity service implementation expressed in the  
427 metadata (see [Section 2.1.4.2](#))
  
- 428 • `AdditionalMetaLocation` [Optional, 0-many] The location of other relevant metadata about the provider which  
429 MAY contain the attribute `namespace`, indicating the namespace of the target document.
  
- 430 • `Extension` [Optional, 0-1] Provides for metadata extensions describing an *SP* or *IdP*
  
- 431 • `ds:Signature` [Optional, 0-1] An optional signature of the provider metadata (see [Section 5.1](#))

432 Each of these elements is optional. The schema fragment for providerDescriptorType:

```

433 <xs:complexType name="providerDescriptorType">
434   <xs:sequence>
435     <xs:element name="KeyDescriptor" type="keyDescriptorType"
436       minOccurs="0" maxOccurs="unbounded"/>
437     <xs:element name="SoapEndpoint" type="xs:anyURI" minOccurs="0"/>
438     <xs:element name="SingleLogoutServiceURL" type="xs:anyURI" minOccurs="0"/>
439     <xs:element name="SingleLogoutServiceReturnURL"
440       type="xs:anyURI" minOccurs="0"/>
441     <xs:element name="FederationTerminationServiceURL"
442       type="xs:anyURI" minOccurs="0"/>
443     <xs:element name="FederationTerminationServiceReturnURL"
444       type="xs:anyURI" minOccurs="0"/>
445     <xs:element name="FederationTerminationNotificationProtocolProfile"
446       type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
447     <xs:element name="SingleLogoutProtocolProfile"
448       type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
449     <xs:element name="RegisterNameIdentifierProtocolProfile"
450       type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
451     <xs:element name="RegisterNameIdentifierServiceURL"
452       type="xs:anyURI" minOccurs="0"/>
453     <xs:element name="RegisterNameIdentifierServiceReturnURL"
454       type="xs:anyURI" minOccurs="0"/>
455     <xs:element name="NameIdentifierMappingProtocolProfile"
456       type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
457     <xs:element name="NameIdentifierMappingEncryptionProfile"
458       type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
459     <xs:element name="Organization" type="organizationType" minOccurs="0"/>
460     <xs:element name="ContactPerson" type="contactType"
461       minOccurs="0" maxOccurs="unbounded"/>
462     <xs:element name="AdditionalMetaLocation"
463       type="additionalMetadataLocationType" minOccurs="0" maxOccurs="unbounded"/>
464     <xs:element ref="Extension" minOccurs="0"/>
465     <xs:element ref="ds:Signature" minOccurs="0"/>
466   </xs:sequence>
467   <!--xs:attribute ref="providerID" use="required"/-->
468   <xs:attribute name="protocolSupportEnumeration" type="anyURILISTType" use="required"/>
469   <xs:attribute name="id" type="xs:ID" use="optional"/>
470   <xs:attribute name="validUntil" type="xs:dateTime"/>
471   <xs:attribute name="cacheDuration" type="xs:duration"/>
472 </xs:complexType>
473
474 <xs:simpleType name="anyURILISTType">
475   <xs:list itemType="xs:anyURI"/>
476 </xs:simpleType>
477
478

```

## 479 2.1.5. Descriptors for Entities

### 480 2.1.5.1. SPDescriptor Element

481 SPDescriptor extends providerDescriptorType with the following elements:

- 482 • AssertionConsumerServiceURL [Required, 1-many] One or more URI(s) of the SP for receiving Authentica-  
483 tion Assertions from an authenticating party. When an SP sends an *AuthNRequest* to the IdP, it may indicate the  
484 preferred AssertionConsumerServiceURL using the provided id (QNAME) attribute to direct the principal to  
485 for consumption of the *AuthNResponse*.  
486 IdP's should inspect the Service Providers metadata for the appropriate URL, or the default (indicated  
487 by the isDefault attribute) location, if no id is provided. Publishers MUST express only one default  
488 AssertionConsumerServiceURL. AssertionConsumerServiceURL requires the following attributes:

- 489       • `id` [Required]. The fragment identifier of the `AssertionConsumerServiceURL` used as a reference in an  
490        `AuthNRequest`.
- 491       • `isDefault` [Required]. A boolean indicator for the default `AssertionConsumerServiceURL` value to use  
492        when no identifier is provided in the request.
- 493       • `AuthnRequestsSigned` [Required, 1] boolean element indicating whether the Service Provider will always  
494        signed it's `AuthNRequests`

495 the schema fragment for `SPDescriptor`:

```
496 <xs:complexType name="SPDescriptorType">
497   <xs:complexContent>
498     <xs:extension base="providerDescriptorType">
499       <xs:sequence>
500         <xs:element name="AssertionConsumerServiceURL" maxOccurs="unbounded">
501           <xs:complexType>
502             <xs:simpleContent>
503               <xs:extension base="xs:anyURI">
504                 <xs:attribute name="id" type="xs:ID" use="required"/>
505                 <xs:attribute name="isDefault" type="xs:boolean" default="false"/>
506               </xs:extension>
507             </xs:simpleContent>
508           </xs:complexType>
509         </xs:element>
510         <xs:element name="AuthnRequestsSigned" type="xs:boolean"/>
511       </xs:sequence>
512     </xs:extension>
513   </xs:complexContent>
514 </xs:complexType>
515
516
```

### 517 2.1.5.2. IDPDescriptor Element

518 `IDPDescriptor` extends `providerDescriptorType` with the following elements:

- 519       • `SingleSignOnServiceURL` [Required, 1]. The identity provider's URL for accepting authentication requests for  
520        the Single Sign-On and Federation Protocol.
- 521       • `SingleSignOnProtocolProfile` [Required, 1-many]. The Single Sign-On Protocol profiles supported by the  
522        provider. Each element MUST contain a valid Single Sign-On Protocol profile identification URI.
- 523       • `AuthnServiceURL` [Optional, 0-1] of type `anyURI` describes the SOAP Endpoint supporting the ID-FF au-  
524        thentication by the identity provider as defined in [[LibertyAuthn](#)] and supports the relevant profile(s) cited in  
525        `SingleSignOnProtocolProfile`. IF the IDP supports SOAP-based IDFF authentication, indicated by the as-  
526        sociated `SingleSignOnProtocolProfile`, and there is no `AuthnServiceURL` provided, then the IDP supports  
527        this profile at the URL identified by `SingleSignOnServiceURL`.

528 The schema fragment for IDPDescriptor:

```
529 <xs:complexType name="IDPDescriptorType">
530   <xs:complexContent>
531     <xs:extension base="providerDescriptorType">
532       <xs:sequence>
533         <xs:element name="SingleSignOnServiceURL" type="xs:anyURI"/>
534         <xs:element name="SingleSignOnProtocolProfile" type="xs:anyURI" maxOccurs="unbounded"/>
535         <xs:element name="AuthnServiceURL" type="xs:anyURI" minOccurs="0"/>
536       </xs:sequence>
537     </xs:extension>
538   </xs:complexContent>
539 </xs:complexType>
540
541
```

### 542 2.1.5.3. EntityDescriptor Element

543 The element `EntityDescriptor` is used to contain one or more descriptor types for a given organization. Publishers  
544 MUST NOT convey metadata for other unaffiliated organizations within this node. Representations of multiple,  
545 unaffiliated providers within a single instance document MUST be done using the plural node form `EntitiesDescriptor`  
546 (Section 2.1.5.4) instead. Publishers MUST publish all relevant roles in this single document, or indirectly through  
547 `AdditionalMetaLocation`.

548 Entities describing a single `providerID`, but wish to publish different metadata for two implementations protocol  
549 support (e.g. IDFF 1.1 services vs. IDFF 1.2 services) may use the plurality of `IDPDescriptor` and `SPDescriptor`  
550 to convey this.

551 Note that it is possible for a single organization to be represented by more than one `providerID`, by indicating  
552 different `providerID` attributes for each entity descriptor, and publishing the document as `EntitiesDescriptor`.

553 `EntityDescriptor` may contain **either**: zero or more `IDPDescriptors` and zero or more `SPDescriptors`, **or**  
554 exactly one `AffiliationDescriptor` followed by any of: `ContactPerson`, `Organization`, `ds:Signature`,  
555 and `Extension`.

556 Attributes for `EntityDescriptor`:

- 557 • `providerID` [Required]: the `providerID` of the entity whose metadata is represented by all descendants of  
558 `EntityDescriptor`
- 559 • `id` [Optional] fragment identifier which is required if `ds:Signature` is present.
- 560 • `validUntil` [Optional] The expiration `dateTime` of the metadata.
- 561 • `cacheDuration` [Optional] The cache duration period for the metadata.

562 Elements contained in `EntityDescriptor`:

- 563 • `IDPDescriptor` Metadata describing an entity acting as an Identity Provider.
- 564 • `SPDescriptor` Metadata describing an entity acting as a Service Provider.
- 565 • `AffiliationDescriptor` Metadata describing a set of entities identified by their respective `providerIDs`  
566 collectively referred to as an affiliation Section 2.1.5.5
- 567 • `ContactPerson` [Optional, 0-1] Contact information for the overall entity (see Section 2.1.4.2).

- 568 • `Organization` [Optional, 0-1] Organizational information about the entity (see [Section 2.1.4.1](#)).
- 569 • `Extension` [Optional, 0-1] provides extension point for additional entity metadata
- 570 • `ds:Signature` [Optional, 0-1] An XML Signature on the entire entity metadata instance.

571 The schema fragment for `entityDescriptorType`:

```
572 <xs:element name="EntityDescriptor" type="entityDescriptorType"/>
573 <xs:group name="providerGroup">
574   <xs:sequence>
575     <xs:element name="IDPDescriptor" type="IDPDescriptorType"
576       minOccurs="0" maxOccurs="unbounded"/>
577     <xs:element name="SPDescriptor" type="SPDescriptorType"
578       minOccurs="0" maxOccurs="unbounded"/>
579   </xs:sequence>
580 </xs:group>
581
582 <xs:complexType name="entityDescriptorType">
583   <xs:sequence>
584     <xs:choice>
585       <xs:group ref="providerGroup"/>
586       <xs:element name="AffiliationDescriptor" type="affiliationDescriptorType"/>
587     </xs:choice>
588     <xs:element name="ContactPerson" type="contactType" minOccurs="0"/>
589     <xs:element name="Organization" type="organizationType" minOccurs="0"/>
590     <xs:element ref="Extension" minOccurs="0"/>
591     <xs:element ref="ds:Signature" minOccurs="0"/>
592   </xs:sequence>
593   <xs:attribute name="providerID" type="entityIDType" use="required"/>
594   <xs:attribute name="id" type="xs:ID" use="optional"/>
595   <xs:attribute name="validUntil" type="xs:dateTime"/>
596   <xs:attribute name="cacheDuration" type="xs:duration"/>
597 </xs:complexType>
598
599
```

#### 600 2.1.5.4. EntitiesDescriptor

601 The element `EntitiesDescriptor` describes more than one organization in a single instance document. It consists  
602 of 2 or more `EntityDescriptors`.

603 The schema fragment for `EntitiesDescriptor` element:

```
604 <xs:element name="EntitiesDescriptor" type="entitiesDescriptorType"/>
605 <xs:complexType name="entitiesDescriptorType">
606   <xs:sequence>
607     <xs:element ref="EntityDescriptor" minOccurs="2" maxOccurs="unbounded"/>
608   </xs:sequence>
609 </xs:complexType>
610
611
```

#### 612 2.1.5.5. AffiliationDescriptor

613 The `AffiliationDescriptor` element describes a group of entities, identified collectively by `providerID` (located  
614 within `EntityDescriptor`), as an enumeration of `providerID`'s. The uniqueness constraints for `providerID` also  
615 apply for `providerID` in this context, such that it **MUST** be unique across all Liberty entities with which the affiliation  
616 expects to interact, including other affiliations and providers therefore, it **MUST NOT** be the `providerID` of any of  
617 the members of the affiliation, and **SHOULD** be unique across the set of `providerID`'s with which the affiliation  
618 expects to interact. It is the responsibility of the entity represented by `affiliationOwnerID` to administer this  
619 identifier, and thus, its members and uniqueness.



620 AffiliationDescriptor element contains the following attributes:

- 621 • affiliationOwnerID [Required] the providerID of the owner or parent operator of the affiliation, from  
622 which, additional metadata may be derived. This DOES NOT indicate affiliation membership of entity described  
623 as affiliationOwnerID. Thus if a member is both the owner of and a member of the affiliation, they must  
624 indicate both in the instance (e.g. the entities providerID appears in both affiliationOwnerID AND  
625 AffiliateMember.
- 626 • validUntil [Optional] The expiration dateTime of the metadata.
- 627 • cacheDuration [Optional] The cache duration period for the metadata.
- 628 • id [Optional].

629 and the following elements:

- 630 • AffiliateMember [Required, 1-many] One or more providers who are members of the affiliation. The value  
631 MUST be a providerID who's metadata MUST be obtained via methods described in [Section 3](#)
- 632 • Extension [Optional, 0-1] provides an extension point to convey additional metadata concerning the affiliation
- 633 • KeyDescriptor [Optional, 0-many] Zero or more public key material reference that is the property of the  
634 affiliation. This keying material SHOULD be separate from the keying material of the providerID who may  
635 be referenced as the affiliateOwnerID and MAY be used for encryption or signing, as indicated by its  
636 corresponding use attribute.
- 637 • ds:Signature [Optional, 0-1] An XML Signature of the metadata node AffiliationDescriptor.

638 The schema fragment for the AffiliationDescriptor element:

```
639 <xs:complexType name="affiliationDescriptorType">
640   <xs:sequence>
641     <xs:element name="AffiliateMember" type="entityIDType" maxOccurs="unbounded"/>
642     <xs:element ref="Extension" minOccurs="0"/>
643     <xs:element name="KeyDescriptor" type="keyDescriptorType" minOccurs="0"
644 maxOccurs="unbounded"/>
645     <xs:element ref="ds:Signature" minOccurs="0"/>
646   </xs:sequence>
647   <!-- <xs:attribute name="affiliationID" type="entityIDType" use="required"/> -->
648   <xs:attribute name="affiliationOwnerID" type="entityIDType" use="required"/>
649   <xs:attribute name="validUntil" type="xs:dateTime"/>
650   <xs:attribute name="cacheDuration" type="xs:duration"/>
651   <xs:attribute name="id" type="xs:ID" use="optional"/>
652 </xs:complexType>
653
654
```

## 655 2.1.6. WSDL Usage

656 A WSDL [[WSDLv1.1](#)] document MAY be used to describe the web services available at the location SoapEndpoint  
657 in addition to the metadata itself. Following is the abstract WSDL describing the ID-FF services:

```
658 <?xml version="1.0" encoding="UTF-8"?>
659 <definitions name="IDFF" targetNamespace="urn:liberty:md:IDFF:wSDL"
660   xmlns="http://schemas.xmlsoap.org/wsdl/"
661   xmlns:iff="urn:liberty:iff:2003-08"
662   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
663   xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
664   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

```
665     xmlns:tns="urn:liberty:md:IDFF:wSDL"
666     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
667 <import location="oasis-sstc-saml-schema-assertion-1.1.xsd"
668     namespace="urn:oasis:names:tc:SAML:1.0:assertion"/>
669 <import location="liberty-idff-protocols-schema-v1.2.xsd"
670     namespace="urn:liberty:iff:2003-08"/>
671 <import location="oasis-sstc-saml-schema-protocol-1.1.xsd"
672     namespace="urn:oasis:names:tc:SAML:1.0:protocol"/>
673 <types/>
674 <!--annotation>
675     <documentation>WSDL from Metadata description and discovery protocols</documentation>
676     <documentation>The source code in this WSDL file was excerpted verbatim from:
677 Liberty Metadata Description and Discovery Specification
678 Version 1.0 errata 3.0
679 21th October 2004
680 Copyright (c) 2004 Liberty Alliance participants, see
681 https://www.projectliberty.org/specs/idff_copyrights.html
682 </documentation>
683 </annotation-->
684 <message name="authenticationResponse">
685     <part name="body" element="iff:AuthnResponseEnvelope"/>
686 </message>
687 <message name="NameIdentifierMappingResponse">
688     <part name="body" element="saml:Assertion"/>
689 </message>
690 <message name="artifactResponse">
691     <part name="body" element="samlp:Response"/>
692 </message>
693 <message name="LogoutRequest">
694     <part name="body" element="iff:LogoutRequest"/>
695 </message>
696 <message name="registerNameIdentifierRequest">
697     <part name="body" element="iff:RegisterNameIdentifierRequest"/>
698 </message>
699 <message name="LogoutResponse">
700     <part name="body" element="iff:LogoutResponse"/>
701 </message>
702 <message name="NameIdentifierMappingRequest">
703     <part name="body" element="samlp:Request"/>
704 </message>
705 <message name="FederationTermination">
706     <part name="body" element="iff:FederationTerminationNotification"/>
707 </message>
708 <message name="authenticationRequest">
709     <part name="body" element="iff:AuthnRequest"/>
710 </message>
711 <message name="artifactRequest">
712     <part name="body" element="samlp:Request"/>
713 </message>
714 <message name="registerNameIdentifierResponse">
715     <part name="body" element="iff:RegisterNameIdentifierResponse"/>
716 </message>
717 <portType name="IDPPort">
718     <operation name="registerNameIdentifier">
719         <input message="tns:registerNameIdentifierRequest"/>
720         <output message="tns:registerNameIdentifierResponse"/>
721     </operation>
722     <operation name="FederationTermination">
723         <input message="tns:FederationTermination"/>
724     </operation>
725     <operation name="Logout">
726         <input message="tns:LogoutRequest"/>
727         <output message="tns:LogoutResponse"/>
728     </operation>
729 </portType>
```

```
732 <operation name="NameIdentifierMapping">
733   <input message="tns:NameIdentifierMappingRequest" />
734   <output message="tns:NameIdentifierMappingResponse" />
735 </operation>
736 <operation name="authentication">
737   <input message="tns:authenticationRequest" />
738   <output message="tns:authenticationResponse" />
739 </operation>
740 <operation name="artifact">
741   <input message="tns:artifactRequest" />
742   <output message="tns:artifactResponse" />
743 </operation>
744 </portType>
745 <service name="IDFF">
746   <port binding="tns:IDPBinding" name="IDFFPort">
747     <soap:address location="http://localhost:8000/ccx/IDFF" />
748   </port>
749 </service>
750 </definitions>
751
752
```

## 753 3. Publishing the Metadata

754 Two mechanisms are provided for entities to publish metadata document locations: via the DNS and via a "well-  
755 known-location" by directly dereferencing the entities' `providerIDs`.

756 When retrieval requires network transport of the document, in both cases above the transport SHOULD be protected  
757 with *TLS/SSL* [RFC2246] as amended by [RFC3546]. This will ensure the integrity of the metadata document, as  
758 among other information within the document, trust establishment may be based in part on information provided  
759 within the metadata. Relying parties of this metadata are RECOMMENDED to authenticate the server via *TLS/SSL*  
760 validation procedures.

761 Trust establishment of the Metadata will be based on one or more of the following: DNS signatures (RECOM-  
762 MENDED); TLS server authentication (RECOMMENDED); and Metadata `ds:Signature` (STRONGLY RECOM-  
763 MENDED) evaluations. Publishers MAY implement additional trust mechanisms, in conjunction with the required  
764 suggested server authentication. Additional trust metadata content, if supplied, MUST be placed in the extension  
765 points provided.

### 766 3.1. Instance Publication Forms

767 If separate documents are used, references to each MUST be made, either through one or more additional PID2MD  
768 NAPTR record(s), or by using the `AdditionalMetaLocation` element within a document which has an associated  
769 NAPTR RR, or which is situated at the "well-known location" (see Section 3.3).

### 770 3.2. Using the DNS to Publish Metadata Location(s)

771 In order to ensure that all providers have accessible metadata locations, entities are STRONGLY RECOMMENDED  
772 to publish their metadata document locations in a zone of their corresponding DNS [RFC1034]. As *providerIDs* are  
773 flexible identifiers, publication and resolution is determined by an entity's URI scheme and fully qualified name part  
774 of the identifier. URI locations for metadata will subsequently be derived through queries of the NAPTR Resource  
775 Record (RR) as defined in [RFC2915] and [RFC3403].

776 It is RECOMMENDED that entities publish their resource records in signed zone files using [RFC2535] such that  
777 relying parties may establish the validity of the published location and authority of the zone, and integrity of the DNS  
778 response. If DNS zone signatures are present, relying parties MUST properly validate the signature.

#### 779 3.2.1. Publication of Metadata Locations

780 This specification makes use of the resource record described in [RFC2915] and [RFC3403]. Familiarity with these  
781 documents is encouraged.

782 Dynamic Delegation Discovery System (DDDS) [RFC3401] is a general purpose system for the retrieval of infor-  
783 mation based on an application-specific input string and the application of well known rules to transform that string  
784 until a terminal condition is reached requiring a look-up into an application-specific defined database or execution of  
785 a URL based on the rules defined by the application. DDDS defines a specific type of DNS Resource Record, NAPTR  
786 records, for the storage of information in the DNS necessary to apply DDDS rules.

787 Entities MAY publish separate URL's when the metadata documents need to be distributed, or when different metadata  
788 documents are required due to multiple Authentication Domain memberships which require separate keying material,  
789 or when service interfaces require separate metadata declarations. This may be accomplished through the use of the  
790 optional `AdditionalMetaLocation` attribute in the core or other subordinate metadata document, or through the  
791 `regexp` facility and multiple service definition fields in the NAPTR resource record itself.

792 If `providerID` is a URN, resolution of the `MetadataLocation` proceeds as specified in [RFC3404]. Otherwise, the  
793 resolution of the metadata location proceeds as specified in this specification.

794 Following are the application-specific descriptions for the DDDS application for the Liberty Metadata resolution  
795 protocols.

### 796 3.2.1.1. Application Unique String

797 Liberty metadata resolution shall begin with the application unique string of `providerID`.

### 798 3.2.1.2. First Well Known Rule

799 The "first well-known-rule" for processing Liberty Alliance Metadata resolution is to parse the `providerID` URI and  
800 extract the fully qualified domain name (subexpression 3) as described in section [Section 4.1.1](#).

### 801 3.2.1.3. The Order Field

802 The order field indicates the order for processing each NAPTR resource record returned. Publishers MAY provide  
803 multiple NAPTR resource record's which MUST be processed by the resolver application in the order indicated by  
804 this field.

### 805 3.2.1.4. The Preference Field

806 For terminal NAPTR resource records, the publisher expresses the preferred order of use to the resolving application.  
807 The resolving application MAY ignore this order, in cases where the service field value does not meet the resolver's  
808 requirements (e.g.: the resource record returns a protocol the application does not support).

### 809 3.2.1.5. The Flag Field

810 Liberty Metadata resolution twice makes use of the "U" flag, which is terminal, and the null value (implying additional  
811 resource record's are to be processed). The "U" flag indicates that the output of the rule is a URI.

### 812 3.2.1.6. The Service Field

813 The Liberty specific service fields shall include:

```
814  
815  
816 servicefield = 1("PID2U" / "NID2U") "+" proto [*( ":" class ) *( ":" servicetype )]  
817 proto = 1("https" / "uddi ")  
818 class = 1[ "entity" / "entitygroup" ]  
819 servicetype = 1(si / "sp" / "idp" / "authn" / alphanum )  
820 si = "si" [ ":" alphanum ] [ ":" endpoint ]  
821 alphanum = 1*32(ALPHA / DIGIT)  
822
```

823 where:

- 824 • `PID2U` resolves a `providerID` identifier to metadata URL
- 825 • `NID2U` resolves a `nameIdentifier` (principal) metadata URL
- 826 • `proto` describes the retrieval protocol (https or uddi). In the case of UDDI, the resulting URI will be an http(s)  
827 URI referencing a WSDL document.
- 828 • `class` identifies which indicates whether the referenced metadata document describes a single provider, or  
829 multiple. In the latter case, the referenced document MUST contain the entity defined by `providerID` as a  
830 member of a group of entities within the document itself.

- 831 • `servicetype` allows a publishers to publish service provider, identity provider and service instance metadata  
832 locations as separate documents. Resolvers who encounter multiple `servicetype` declarations will dereference the  
833 appropriate URI, depending on which service type required for an operation (e.g.: a provider operating both and  
834 IdP and an SP service, may publish SP and IdP metadata at different locations).
- 835 • the `si` component (with optional endpoint component) allows the publisher to either directly publish the metadata  
836 for a service instance, or by articulating the soap endpoint (using `endpoint`)

837 For example:

- 838 • `PID2U+https:entity` - represents the complete entity metadata document via the `https` protocol
- 839 • `PID2U+https:entity:si:pip` - returns the PIP metadata URL for the entity described by `providerID` via the `https`  
840 protocol profile
- 841 • `PID2U+uddi:entity:si:foo` - returns the WSDL document location which describes a service instance "foo"
- 842 • `PID2U+https:entitygroup:idp` - returns the metadata for a group of entities, of which `providerID` is a member. the  
843 referenced document describes (one or more) IdPs in the group
- 844 • `NID2U+https:idp` - returns an IdP `providerIDs`, who can provider authentication services for a principal
- 845 • `NID2U+https:authn` - returns a URL to attempt to authenticate the principal against

### 846 3.2.1.7. The Regex and Replacement Fields

847 The expected output after processing the application-unique string through the regex MUST be a valid `https` URL or  
848 UDDI node (`http` references `wsdl` document) address.

## 849 3.2.2. NAPTR Examples

### 850 3.2.2.1. Provider Metadata NAPTR Examples

851 Entities publish metadata URLs in the following manner:

```
852  
853  
854 $ORIGIN provider.biz  
855  
856 ;; order pref f service regexp or replacement  
857  
858 IN NAPTR 100 10 "U" PID2U+https:entity  
859 "!.^.*$!https://host.provider.biz/some/directory/trust.xml!" ""  
860 IN NAPTR 110 10 "U" PID2U+https:entity:trust  
861 "!.^.*!https://foo.provider.biz:1443/mdtrust.xml!" ""  
862 IN NAPTR 125 10 "U" PID2U+https:"  
863 IN NAPTR 110 10 "U" PID2U+uddi:entity "!.^.*$!https://this.uddi.node.provider.biz/libmd.wsdl" ""  
864
```

### 865 3.2.2.2. Name Identifier Examples

866 Principals employer example.int operates an IdP which may be used by a office supply company to authenticate  
867 authorized buyers. The supplier takes users email address `buyer@example.int` as input to the resolution process,  
868 and parses the email address to extract the FQDN (`example.int`). The employer publishes the following NAPTR in  
869 `example.int`:

870  
871

```
872 $ORIGIN
873
874 example.int.
875
876     IN NAPTR 100 10 "U" NID2U+https:authn
877         "!(^[^@]+)@ (.*)$!https://serv.example.int:8000/cgi-bin/getmd?\1! " ""
878     IN NAP TR 100 10 "U" NID2U+https:idp
879         "!(^[ ^@]+)@(.*)$!https://auth.example.int/app/auth?\1" ""
880
881
```

### 882 3.3. Publication via Well-Known Location

883 Entities MAY publish their metadata documents at a well known location. The core metadata document location in  
884 this profile simply involves directly dereferencing the providerID and obtaining the document directly (or through  
885 schema-specific means of indirection)

886 For well known location documents, the XML document MUST describe the metadata for the **providerID** entity only.  
887 If other entities need to be described, the **AdditionalMetaLocation** MUST be used. Thus the **entitiesDescriptor**  
888 MUST NOT be used in documents published at a well know location, since entities as a group, are not defined by such  
889 an identifier.

## 890 4. Metadata Resolution and Retrieval

891 Metadata publication is provided in two fashions: via a "well-known-location" and via queries on the DNS. Both  
892 mechanisms depend upon processing of the `providerID` element (see [ Section 3]), which is the primary identifier  
893 for Liberty-enabled entities. Consumers are **STRONGLY RECOMMENDED** to attempt DNS-based resolution prior  
894 to performing a direct dereferencing of a `providerID`.

895 The `providerID` is defined as a restricted form of `anyURI` Section 2.1.2; therefore, it shall be parsed as in Section 4.1.1  
896 for these resolution profiles.

### 897 4.1. Resolving Locations and Retrieving Metadata

898 The summarized steps for retrieving metadata from a given `providerID` is as follows:

- 899 • If the `providerID` is a URN, proceed with the resolution steps as defined in [RFC3404]
- 900 • parse the `providerID` to obtain the *FQDN*
- 901 • query the DNS for NAPTR resource records of the `domain` name iteratively until a terminal resource record is  
902 returned  
903 (optionally, or if DNS-based resolution fails) attempt locating the metadata document(s) via the *well known*  
904 *location* profile by directly dereferencing the `providerID` (end if a document was located, validated and fulfills  
905 the metadata requirements for the present operations).
- 906 • identify which resource record to use based on the service fields, then order fields, then preference fields of the  
907 result set.
- 908 • obtain the document(s) at the provided location(s) as required by the application

#### 909 4.1.1. Parsing the ProviderID

910 To initiate the resolution of the location of the target metadata elements, it will be necessary in some cases to  
911 decompose the `ProviderID` (expressed as a URI) into one or more atomic elements.

912 The following regular expression should be used when initiating the decomposition process:

```
913  
914  
915 ^([^\s:/?#]+:)?/*(?:[^\s:/?#]*@)?(((?:[^\s:/?#]*\.)*)((?:[^\s/?#:\.]+)\.([^\s/?#:\.]+)))(:\d+)?(  
916 [^\s?#]*)(\?([^\s#]*)?)(#[^\s]*)?$  
917 1 2 34 56 7 8 9 10 ←  
918 11  
919  
920
```

921 Subexpression 3 **MUST** result in a Fully Qualified Domain Name (FQDN), which will be the basis for retrieving  
922 metadata locations from this zone.

#### 923 4.1.2. Obtaining Metadata via the DNS

924 Upon completion of the parsing of the `providerID`, the application then performs a DNS query for the resulting  
925 domain (subexpression 5) for NAPTR resource records; it should expect 1 or more responses. Applications **MAY**  
926 exclude from the result set any service definitions which do not concern the present request operations. Should the  
927 DNS not produce a valid response, the consumer **MUST ALWAYS** attempt direct dereferencing of the `providerID`.



928 Resolving applications **MUST** subsequently order the result set according to the order field, and **MAY** order the result  
929 set based on the preference set. Resolvers are **NOT REQUIRED** to follow the ordering of the preferences field.

930 The resulting NAPTR resource record(s) are operated on iteratively (based on the order flag) until a terminal NAPTR  
931 resource record is reached.

932 The result will be a well formed, fully qualified URL, which will then be used to retrieve the metadata document.

#### 933 **4.1.2.1. Post Processing Operations**

934 When service specific metadata is sought, resolvers **MAY** filter the NAPTR result set based on more specific resource  
935 records with service identifiers which match the service(s) sought.

#### 936 **4.1.3. Obtaining Metadata via the "Well-Known Location Method"**

937 Consumers of published metadata **MAY** attempt retrieval via the well-known-location method by directly dereferenc-  
938 ing the providerID. Other forms of well-known location **MAY** be agreed upon by a group of Liberty entities, however,  
939 it is **STRONGLY SUGGESTED** that publication in the DNS be employed as well, to allow for interactions with other  
940 Liberty implementations. The resulting XML document **MUST** describe the metadata for the providerID entity  
941 only. If other entities need to be described, the AdditionalMetaLocation **MUST** be used. There may be only one  
942 location, although this document **MAY** point to other document locations using the AdditionalMetaLocation  
943 element.

## 944 **5. Post Processing of the Metadata Document**

### 945 **5.1. Processing of ds:Signature and General Trust Processing**

946 Metadata processing provides several mechanisms for trust negotiation for both the metadata itself and for the trust  
947 ascribed to the entity described by such metadata:

- 948 • Trust derived from the signature of the zone from which the metadata location URI was resolved, ensuring accuracy  
949 of the metadata document location(s)
- 950 • Trust derived from signature processing of the metadata document itself, ensuring the integrity of the XML  
951 document
- 952 • Trust derived from the SSL/TLS negotiation of the metadata delivery URI, ensuring the identity of the publisher  
953 of the metadata.

954 Post processing of the metadata document **MUST** include the signature processing at the XML-document level  
955 and **MAY** include one of the other two processes. Specifically, the relying party **MAY** choose to trust any of the  
956 cited authorities in the resolution and parsing process. Publishers of metadata **MUST** employ a document-integrity  
957 mechanism and **MAY** employ any of the other two processing profiles to establish trust of the subject of the metadata  
958 document, governed by implementation policies.

#### 959 **5.1.1. Processing Signed DNS Zones**

960 Verification of zone signature **SHOULD** be processed, if present, as described in [\[RFC2535\]](#)

#### 961 **5.1.2. Processing Signed Documents and Fragments**

962 Published metadata documents **SHOULD** be signed, as described in [\[XMLDsig\]](#), either by a certificate issued to the  
963 subject of the document, or by another trusted party. Publishers **MAY** consider signatures of other parties as a means  
964 of trust conveyance.

965 Consumers **MUST** validate signatures, when present, on the metadata document on initial retrieval as described by  
966 [\[XMLDsig\]](#).

#### 967 **5.1.3. Processing Server Authentication in Metadata Retrieval via TLS/SSL**

968 It is **STRONGLY RECOMMENDED** that publishers implement TLS URL's; therefore, consumers **SHOULD** consider  
969 the trust inherited from the issuer of the TLS/SSL certificate. Publication URLs may not always be located in  
970 the domain of the provider of the subject of the metadata document; therefore, consumers **SHOULD NOT** expect  
971 certificates whose subject is the provider, as it may be hosted at another trusted party.

972 As the basis of this trust may not be available against a cached document, other mechanisms **SHOULD** be used under  
973 such circumstances.

## 974 **5.2. Metadata Location and Document Caching**

975 Location caching based on DNS profiles **MUST NOT** exceed the TTL of the DNS zone from which the location was  
976 derived. Resolvers **MUST** obtain a fresh copy of the Metadata location upon reaching the expiration of the TTL of the  
977 zone.

978 A publishers of Metadata documents should carefully consider the TTL of the zone when making updates to its  
979 metadata document location. Should such a location change occur, a publisher **MUST** either keep the document at  
980 both the old and new location until all conforming resolvers are certain to have the updated location (e.g.: time of zone  
981 change + TTL), or provide an HTTP `Redirect` [\[RFC2616\]](#) to the new location.

982 Document caching MUST NOT exceed the `validUntil` attribute of the subject element(s) and the `cacheDuration`  
983 attribute. If fragments have parents which contain caching policies, the parent fragment ALWAYS takes precedence.

984 To properly process the `cacheDuration` attributes on fragments and documents consumers MUST retain the  
985 `dateTime` when the document was retrieved.

986 When a document or fragment has expired the consumer MUST retrieve a fresh copy, which may require a refresh of  
987 the document location(s). Consumers SHOULD process document cache processing according to [\[RFC2616\]](#) section  
988 13, and MAY request the Last-Modified `dateTime` from the HTTPS server. Publishers SHOULD ensure acceptable  
989 cache processing as described in [\[RFC2616\]](#) (Section 10.3.5 304 Not Modified).

### 990 **5.3. Handling of HTTPS Redirects**

991 Publishers MAY issue an HTTP Redirect (301 Moved Permanently, or 307 Temporary Redirect) [\[RFC2616\]](#), and user  
992 agents MUST follow the specified URL in the Redirect response. Redirects SHOULD be to a TLS/SSL protected  
993 resource, and SHOULD be of the same protocol as the initial request.

## 994 **6. Security Considerations**

### 995 **6.1. Trust Establishment**

996 Cryptographic signatures are used to establish identity and tamper evidence in several locations within the metadata  
997 specification. While valid signatures convey some level of trust in the resulting document, extreme care should be  
998 taken as to the validity of the URIs described within the document itself. Relying parties should carefully inspect  
999 agreements and statements made by the signing authorities of the subject certificates or keys.

## 1000 7. Metadata XSD

```
1001
1002 <?xml version="1.0" encoding="UTF-8"?>
1003 <xs:schema targetNamespace="urn:liberty:metadata:2004-12"
1004   xmlns="urn:liberty:metadata:2004-12"
1005   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1006   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1007   elementFormDefault="qualified"
1008   attributeFormDefault="unqualified" version="1.0">
1009   <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
1010     schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd"/>
1011   <xs:import namespace="http://www.w3.org/XML/1998/namespace"
1012     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
1013   <xs:include schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
1014
1015   <xs:annotation>
1016     <xs:documentation>
1017       XML Schema fom Metadata description and discovery protocols
1018     </xs:documentation>
1019   </xs:annotation>
1020
1021   The source code in this XSD file was excerpted verbatim from:
1022
1023   Liberty Metadata Description and Discovery Specification
1024   Version 2.0-02
1025   25 November 2004
1026
1027       Copyright (c) 2004 Liberty Alliance participants, see
1028       http://www.projectliberty.org/specs/idwsf_2_0_copyrights.php
1029
1030   </xs:documentation>
1031 </xs:annotation>
1032
1033 <xs:simpleType name="entityIDType">
1034   <xs:restriction base="xs:anyURI">
1035     <xs:maxLength value="1024" id="maxlengthid"/>
1036   </xs:restriction>
1037 </xs:simpleType>
1038
1039 <!--
1040 <xs:attribute name="libertyPrincipalIdentifier" type="entityIDType"/>
1041 <xs:attribute name="providerID" type="entityIDType"/>
1042 <xs:attribute name="validUntil" type="xs:dateTime"/>
1043 <xs:attribute name="cacheDuration" type="xs:duration"/>
1044 -->
1045
1046 <xs:complexType name="additionalMetadataLocationType">
1047   <xs:simpleContent>
1048     <xs:extension base="xs:anyURI">
1049       <xs:attribute name="namespace" type="xs:anyURI"/>
1050     </xs:extension>
1051   </xs:simpleContent>
1052 </xs:complexType>
1053
1054 <xs:complexType name="organizationNameType">
1055   <xs:simpleContent>
1056     <xs:extension base="xs:string">
1057       <xs:attribute ref="xml:lang"/>
1058     </xs:extension>
1059   </xs:simpleContent>
1060 </xs:complexType>
1061
1062 <xs:complexType name="organizationDisplayNameType">
1063   <xs:simpleContent>
1064     <xs:extension base="xs:string">
1065       <xs:attribute ref="xml:lang" use="required"/>

```

```
1066     </xs:extension>
1067   </xs:simpleContent>
1068 </xs:complexType>
1069
1070 <xs:complexType name="organizationType">
1071   <xs:sequence>
1072     <xs:element name="OrganizationName" type="organizationNameType" maxOccurs="unbounded" />
1073     <xs:element name="OrganizationDisplayName" type="organizationDisplayNameType"
1074 maxOccurs="unbounded" />
1075     <xs:element name="OrganizationURL" type="localizedURIType" maxOccurs="unbounded" />
1076     <xs:element ref="Extension" minOccurs="0" />
1077   </xs:sequence>
1078 </xs:complexType>
1079
1080 <xs:complexType name="localizedURIType">
1081   <xs:simpleContent>
1082     <xs:extension base="xs:anyURI">
1083       <xs:attribute ref="xml:lang" use="required" />
1084     </xs:extension>
1085   </xs:simpleContent>
1086 </xs:complexType>
1087
1088 <xs:complexType name="contactType">
1089   <xs:sequence>
1090     <xs:element name="Company" type="xs:string" minOccurs="0" />
1091     <xs:element name="GivenName" type="xs:string" minOccurs="0" />
1092     <xs:element name="SurName" type="xs:string" minOccurs="0" />
1093     <xs:element name="EmailAddress" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded" />
1094     <xs:element name="PhoneNumber" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
1095     <xs:element ref="Extension" minOccurs="0" />
1096   </xs:sequence>
1097   <xs:attribute name="libertyPrincipalIdentifier" type="entityIDType" use="optional" />
1098   <xs:attribute name="contactType" type="attr.contactType" use="required" />
1099 </xs:complexType>
1100 <xs:simpleType name="attr.contactType">
1101   <xs:restriction base="xs:string">
1102     <xs:enumeration value="technical" />
1103     <xs:enumeration value="administrative" />
1104     <xs:enumeration value="billing" />
1105     <xs:enumeration value="other" />
1106   </xs:restriction>
1107 </xs:simpleType>
1108
1109 <xs:simpleType name="keyTypes">
1110   <xs:restriction base="xs:string">
1111     <xs:enumeration value="encryption" />
1112     <xs:enumeration value="signing" />
1113   </xs:restriction>
1114 </xs:simpleType>
1115
1116 <xs:complexType name="providerDescriptorType">
1117   <xs:sequence>
1118     <xs:element name="KeyDescriptor" type="keyDescriptorType"
1119 minOccurs="0" maxOccurs="unbounded" />
1120     <xs:element name="SoapEndpoint" type="xs:anyURI" minOccurs="0" />
1121     <xs:element name="SingleLogoutServiceURL" type="xs:anyURI" minOccurs="0" />
1122     <xs:element name="SingleLogoutServiceReturnURL"
1123 type="xs:anyURI" minOccurs="0" />
1124     <xs:element name="FederationTerminationServiceURL"
1125 type="xs:anyURI" minOccurs="0" />
1126     <xs:element name="FederationTerminationServiceReturnURL"
1127 type="xs:anyURI" minOccurs="0" />
1128     <xs:element name="FederationTerminationNotificationProtocolProfile"
1129 type="xs:anyURI" minOccurs="0" maxOccurs="unbounded" />
1130     <xs:element name="SingleLogoutProtocolProfile"
1131 type="xs:anyURI" minOccurs="0" maxOccurs="unbounded" />
1132     <xs:element name="RegisterNameIdentifierProtocolProfile"
```

```
1133     type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
1134 <xs:element name="RegisterNameIdentifierServiceURL"
1135     type="xs:anyURI" minOccurs="0"/>
1136 <xs:element name="RegisterNameIdentifierServiceReturnURL"
1137     type="xs:anyURI" minOccurs="0"/>
1138 <xs:element name="NameIdentifierMappingProtocolProfile"
1139     type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
1140 <xs:element name="NameIdentifierMappingEncryptionProfile"
1141     type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
1142 <xs:element name="Organization" type="organizationType" minOccurs="0"/>
1143 <xs:element name="ContactPerson" type="contactType"
1144     minOccurs="0" maxOccurs="unbounded"/>
1145 <xs:element name="AdditionalMetaLocation"
1146     type="additionalMetadataLocationType" minOccurs="0" maxOccurs="unbounded"/>
1147 <xs:element ref="Extension" minOccurs="0"/>
1148 <xs:element ref="ds:Signature" minOccurs="0"/>
1149 </xs:sequence>
1150 <!--xs:attribute ref="providerID" use="required"/-->
1151 <xs:attribute name="protocolSupportEnumeration" type="anyURIListType" use="required"/>
1152 <xs:attribute name="id" type="xs:ID" use="optional"/>
1153 <xs:attribute name="validUntil" type="xs:dateTime"/>
1154 <xs:attribute name="cacheDuration" type="xs:duration"/>
1155 </xs:complexType>
1156
1157 <xs:simpleType name="anyURIListType">
1158   <xs:list itemType="xs:anyURI"/>
1159 </xs:simpleType>
1160
1161 <xs:element name="KeyDescriptor" type="keyDescriptorType"/>
1162 <xs:complexType name="keyDescriptorType">
1163   <xs:sequence>
1164     <xs:element name="EncryptionMethod" type="xs:anyURI" minOccurs="0"/>
1165     <xs:element name="KeySize" type="xs:integer" minOccurs="0"/>
1166     <xs:element ref="ds:KeyInfo" minOccurs="0"/>
1167     <xs:element ref="Extension" minOccurs="0"/>
1168   </xs:sequence>
1169   <xs:attribute name="use" type="keyTypes" use="optional"/>
1170 </xs:complexType>
1171
1172 <xs:element name="EntityDescriptor" type="entityDescriptorType"/>
1173 <xs:group name="providerGroup">
1174   <xs:sequence>
1175     <xs:element name="IDPDescriptor" type="IDPDescriptorType"
1176       minOccurs="0" maxOccurs="unbounded"/>
1177     <xs:element name="SPDescriptor" type="SPDescriptorType"
1178       minOccurs="0" maxOccurs="unbounded"/>
1179   </xs:sequence>
1180 </xs:group>
1181
1182 <xs:complexType name="entityDescriptorType">
1183   <xs:sequence>
1184     <xs:choice>
1185       <xs:group ref="providerGroup"/>
1186       <xs:element name="AffiliationDescriptor" type="affiliationDescriptorType"/>
1187     </xs:choice>
1188     <xs:element name="ContactPerson" type="contactType" minOccurs="0"/>
1189     <xs:element name="Organization" type="organizationType" minOccurs="0"/>
1190     <xs:element ref="Extension" minOccurs="0"/>
1191     <xs:element ref="ds:Signature" minOccurs="0"/>
1192   </xs:sequence>
1193   <xs:attribute name="providerID" type="entityIDType" use="required"/>
1194   <xs:attribute name="id" type="xs:ID" use="optional"/>
1195   <xs:attribute name="validUntil" type="xs:dateTime"/>
1196   <xs:attribute name="cacheDuration" type="xs:duration"/>
1197 </xs:complexType>
1198
1199 <xs:complexType name="SPDescriptorType">
```

```
1200 <xs:complexContent>
1201 <xs:extension base="providerDescriptorType">
1202 <xs:sequence>
1203 <xs:element name="AssertionConsumerServiceURL" maxOccurs="unbounded">
1204 <xs:complexType>
1205 <xs:simpleContent>
1206 <xs:extension base="xs:anyURI">
1207 <xs:attribute name="id" type="xs:ID" use="required"/>
1208 <xs:attribute name="isDefault" type="xs:boolean" default="false"/>
1209 </xs:extension>
1210 </xs:simpleContent>
1211 </xs:complexType>
1212 </xs:element>
1213 <xs:element name="AuthnRequestsSigned" type="xs:boolean"/>
1214 </xs:sequence>
1215 </xs:extension>
1216 </xs:complexContent>
1217 </xs:complexType>
1218
1219 <xs:complexType name="IDPDescriptorType">
1220 <xs:complexContent>
1221 <xs:extension base="providerDescriptorType">
1222 <xs:sequence>
1223 <xs:element name="SingleSignOnServiceURL" type="xs:anyURI"/>
1224 <xs:element name="SingleSignOnProtocolProfile" type="xs:anyURI" maxOccurs="unbounded"/>
1225 <xs:element name="AuthnServiceURL" type="xs:anyURI" minOccurs="0"/>
1226 </xs:sequence>
1227 </xs:extension>
1228 </xs:complexContent>
1229 </xs:complexType>
1230
1231 <xs:element name="EntitiesDescriptor" type="entitiesDescriptorType"/>
1232 <xs:complexType name="entitiesDescriptorType">
1233 <xs:sequence>
1234 <xs:element ref="EntityDescriptor" minOccurs="2" maxOccurs="unbounded"/>
1235 </xs:sequence>
1236 </xs:complexType>
1237
1238 <xs:complexType name="affiliationDescriptorType">
1239 <xs:sequence>
1240 <xs:element name="AffiliateMember" type="entityIDType" maxOccurs="unbounded"/>
1241 <xs:element ref="Extension" minOccurs="0"/>
1242 <xs:element name="KeyDescriptor" type="keyDescriptorType" minOccurs="0"
1243 maxOccurs="unbounded"/>
1244 <xs:element ref="ds:Signature" minOccurs="0"/>
1245 </xs:sequence>
1246 <!-- <xs:attribute name="affiliationID" type="entityIDType" use="required"/> -->
1247 <xs:attribute name="affiliationOwnerID" type="entityIDType" use="required"/>
1248 <xs:attribute name="validUntil" type="xs:dateTime"/>
1249 <xs:attribute name="cacheDuration" type="xs:duration"/>
1250 <xs:attribute name="id" type="xs:ID" use="optional"/>
1251 </xs:complexType>
1252
1253 </xs:schema>
1254
1255
```



# References

1256

## Normative

1257

- 1258 [LibertyAuthn] Aarts, Robert, Hodges, Jeff, Madsen, Paul, eds. "Liberty ID-WSF Authentication Ser-  
1259 vice and Single Sign-On Service Specification ," 2.0-02, Liberty Alliance Project (25 Nov 2004).  
1260 <http://www.projectliberty.org/specs/>
- 1261 [LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version  
1262 1.2-errata-v2.0, Liberty Alliance Project (12 September 2004). <http://www.projectliberty.org/specs>
- 1263 [LibertyProtSchema1.1] Beatty, John, Kemp, John, eds. "Liberty Protocols and Schema Specification," Version 1.1,  
1264 Liberty Alliance Project (January 2003). <http://www.projectliberty.org/specs> [January 2003].
- 1265 [LibertyBindProf] Cantor, Scott, Kemp, John, Champagne, Darryl, eds. "Liberty ID-FF Bindings and  
1266 Profiles Specification," Version 1.2-errata-v2.0, Liberty Alliance Project (12 September 2004).  
1267 <http://www.projectliberty.org/specs>
- 1268 [RFC1034] Mockapetris, P., eds. (November 1987). "DOMAIN NAMES - CONCEPTS AND FACILITIES," RFC  
1269 1510, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc1034.txt> [November 1987].
- 1270 [RFC2119] Bradner, S., eds. "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet  
1271 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt> [March 1997].
- 1272 [RFC2246] Dierks, T., Allen, C., eds. (January 1999). "The TLS Protocol," Version 1.0 RFC 2246, Internet  
1273 Engineering Task Force <http://www.ietf.org/rfc/rfc2246.txt> [January 1999].
- 1274 [RFC2396] Berners-Lee, T., Fielding, R., Masinter, L., eds. (August 1998). "Uniform Resource Identifiers (URI):  
1275 Generic Syntax," RFC 2396, The Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2396.txt>  
1276 [August 1998].
- 1277 [RFC2535] Eastlake, D., eds. (March 1999). "Domain Name System Security Extensions," RFC 2535, The Internet  
1278 Engineering Task Force <http://www.ietf.org/rfc/rfc2535.txt> [March 1999].
- 1279 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., eds. (June  
1280 1999). "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, The Internet Engineering Task Force  
1281 <http://www.ietf.org/rfc/rfc2616.txt> [June 1999].
- 1282 [RFC2732] Hinden, R., Carpenter, B., Masinter, L., eds. (December 1999). "Format for Literal IPv6 Addresses in  
1283 URL's," RFC 2732, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2732.txt> [December 1999].
- 1284 [RFC2915] Mealling, M., Daniel, R., eds. (September 2000). "The Naming Authority Pointer (NAPTR) DNS Re-  
1285 source Record," RFC 2915, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2915.txt> [September  
1286 2000].
- 1287 [RFC3401] Mealling, M., eds. (October 2002). "Dynamic Delegation Discovery System (DDDS) - Part One: The  
1288 Comprehensive DDDS," RFC 3401, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc3401.txt>  
1289 [October 2002].
- 1290 [RFC3403] Mealling, M., eds. (October 2002). "Dynamic Delegation Discovery System (DDDS) - Part  
1291 Three: The Domain Name System (DNS) Database," RFC 3403, Internet Engineering Task Force  
1292 <http://www.ietf.org/rfc/rfc3403.txt> [October 2002].
- 1293 [RFC3404] Mealling, M., eds. (October 2002). "Dynamic Delegation Discovery System (DDDS) - Part Four: The  
1294 Uniform Resource Identifiers (URI) Resolution Application," RFC 3404, Internet Engineering Task Force  
1295 <http://www.ietf.org/rfc/rfc3404.txt> [October 2002].

- 1296 [RFC3546] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., Wright, T., eds. (June 2003).  
1297 "Transport Layer Security (TLS) Extensions," RFC 3546, The Internet Engineering Task Force  
1298 <http://www.ietf.org/rfc/rfc3546.txt> [June 2003].
- 1299 [Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May  
1300 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium  
1301 <http://www.w3.org/TR/xmlschema-1/>
- 1302 [Schema2] Biron, Paul V., Malhotra, Ashok, eds. (May 2002). "XML Schema Part 2: Datatypes," Recommendation,  
1303 World Wide Web Consortium <http://www.w3.org/TR/xmlschema-2/>
- 1304 [WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Meredith,  
1305 Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001).  
1306 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- 1307 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C.M., Maler, Eve, eds. (Oct 2000). "Extensible  
1308 Markup Language (XML) 1.0 (Second Edition)," Recommendation, World Wide Web Consortium  
1309 <http://www.w3.org/TR/2000/REC-xml-20001006>
- 1310 [XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and  
1311 Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlsig-core>
- 1312 [xmlesc-core] Eastlake, Donald, Reagle, Joseph, eds. (December 2002). "XML Encryption Syntax and Processing,"  
1313 W3C Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlesc-core/>
- 1314 **Informative**
- 1315 [LibertyIDFFOverview] Wason, Thomas, eds. "Liberty ID-FF Architecture Overview," Version 1.2-errata-v1.0,  
1316 Liberty Alliance Project (12 September 2004). <http://www.projectliberty.org/specs>